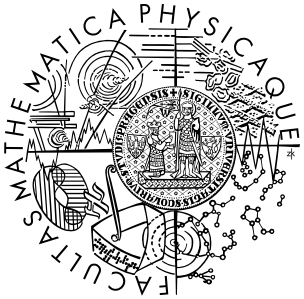
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Kypta

Rozšířený HMM Tagger a jeho aplikace na morfologické značkování češtiny

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Jiří Mírovský
Studijní program: Informatika, programování

2007

Rád bych poděkoval vedoucímu bakalářské práce, Mgr. Jiřímu Mírovskému, za poskytnutí cenných rad, námětů, konzultací, potřebné literatury a také testovacích dat.

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

V Praze dne 7. 8. 2007 Tomáš Kypta

Obsah

Název práce: Rozšířený HMM tagger a jeho aplikace na morfologické značkování češtiny
Autor: Tomáš Kypta
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Jiří Mírovský
e-mail vedoucího: mirovsky@ufal.ms.mff.cuni.cz

Abstrakt: V předložené práci studuji možnosti morfologického značkování češtiny při použití statistického značkovače založeného na skrytých Markovových modelech (HMM taggeru). Zejména pak ověřuji vliv: různě velkých trénovacích dat, délky tagovací historie, nastavení parametru 'n' ve *variantě výběru n nejlepších průchodů* (varianta n-best) a omezení sady tagů v historii značek na úspěšnost značkovače. Text je doplněn řadou tabulek s výsledky běhu značkovače včetně porovnání s předchozími výsledky jiných značkovačů. V příloze se na kompaktním disku nachází testovací data a program, jehož výsledky jsou zde prezentovány.

Klíčová slova: skryté Markovovy modely, Viterbiho algoritmus, morfologické značkování češtiny

Title: Enhanced HMM Tagger and Its Application for Czech Morphological Tagging
Author: Tomáš Kypta
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Jiří Mírovský
Supervisor's e-mail address: mirovsky@ufal.ms.mff.cuni.cz

Abstract: In the present work I study possibilities of Czech morphological tagging by using statistical tagger based on hidden Markov models (HMM tagger). I especially intend to verify an influence of various size of training data, length of tagging history, setting n-parameter in n-best variant and reduction of tag set in history of tags to the successfulness of tagging. Text is completed with tables with results of tagger including comparison with previous results of other taggers. There is also a supplementary CD with test data and the program, which results are presented here.

Keywords: hidden Markov models, Viterbi algorithm, Part-of-speech tagging

Kapitola 1

Úvod

1.1 Motivace

Rychlý rozvoj výpočetní techniky umožnil nejen zkoumání mnoha dříve těžko řešitelných problémů, ale také přinesl možnost potenciálního využití počítačů v dříve nemyslitelných oblastech vědy.

Jednou z těchto oblastí se stala i lingvistika. Objevila se idea počítačového zpracování přirozeného jazyka. Počítačová lingvistika má široké spektrum nejrůznějších aplikací. Jako příklad můžeme uvést automatické překlady, kontrolu pravopisu nebo možnost komunikace s počítačem mluveným slovem. Bohužel některé lingvistické aplikace s sebou přináší i řadu nových dosud neřešených problémů. Ukazuje se, že implementace takovýchto aplikací fungujících opravdu dobře a v rozumném čase je zatím spíše hudbou budoucnosti a otázkou dalšího výzkumu.

Problematika počítačového zpracování přirozeného jazyka se rozpadla na řešení elementárních problémů. Jedním z nejdůležitějších je morfologická analýza.

Morfologická analýza usiluje o analýzu vstupních slov z hlediska jejich gramatického významu. Podrobněji jde o přiřazení základního tvaru (*lemmatu*) a informace o tvaru (*tagu*) vstupnímu slovu. K takovému rozboru nám ve většině případů nestačí samotné slovo, neboť to samo o sobě může mít více různých významů. Příkladem může být např. slovo 'moci', které je jednak slovesem, jednak i množným číslem podstatného jména 'moc'; podobně slovo 'se' může být zájmenem nebo předložkou, atd. Pro analýzu je tedy potřeba určitý kontext.

V následujícím textu popíši jeden konkrétní přístup k této problematice i s dosaženými výsledky.

1.2 Analýza založená na skrytých Markovových modelech

K problému morfologické analýzy lze přistupovat několika různými způsoby, které se dělí na dvě hlavní větve: statistické taggery a taggery založené na ručně psaných pravidlech. Má práce popisuje jeden ze statistických přístupů založený na *skrytých Markovových modelech* (detailněji popsáno dále v sekci 2.1.2). Tento značkovač se nazývá *HMM tagger* (název je odvozen od zkratky anglického názvu - Hidden Markov Models).

HMM tagger umožňuje celkem efektivní zpracování, jeho nevýhodou jsou potom větší paměťové i časové nároky. Jistým ulehčením práce HMM taggeru je proto použití vstupních dat obsahujících výstup morfologického analyzátoru, který každému slovu přiřazuje množinu potenciálních tagů a lemmat.

1.3 Přesné zadání

Vytvoření HMM taggeru - aplikace na morfologické značkování češtiny. Umožnění volby 'n' nejlepších průchodů ve Viterbiho algoritmu, dále ověření vlivu různě velkých trénovacích dat a vlivu nastavení parametru 'n' na úspěšnost tagování. Možnost dalších experimentů, např. omezení sady tagů v historii značek a podobně.

Kapitola 2

Teoretická část

2.1 Markovovy modely

Skryté Markovovy modely jsou využívány v mnoha různých systémech. Jsou například jedním z úspěšných přístupů k rozpoznávání řeči počítačem. Nás ale zajímá jejich využití pro morfologické značkování přirozeného jazyka, kde se jejich aplikace staly rovněž úspěšnými.

HMM je pravděpodobnostní funkcí Markovova procesu. Teorie Markovových procesů/řetězců/modelů byla vyvinuta A. Markovem a to původně k lingvistickým účelům. Později byla teorie rozvinuta v obecný statistický nástroj (viz. [1]).

Teorie Markovových modelů je založena na myšlence, že náhodné veličiny v sekvenci nemusí být nezávislé. Je zde použit předpoklad, že další veličina v sekvenci by mohla být závislá na hodnotě aktuální veličiny. Předpokládá se však také, že bude nezávislá na všech předchozích veličinách v sekvenci.

Pro sekvenci $X = (X_1, \ldots, X_T)$ náhodných veličin nabývajících hodnot z množiny $S = \{s_1, \ldots, s_N\}$ tedy dostáváme teorii založenou na dvou *Markovových vlastnostech* (podle [1]):

1. **Omezený rozhled**

$$P(X_{t+1} = s_k | X_1, \ldots, X_t) \quad = \quad P(X_{t+1} = s_k | X_t)$$

2. **Neměnnost v čase**

$$P(X_{t+1} = s_k | X_t) \quad = \quad P(X_2 = s_k | X_1)$$

Sekvenci X splňující tyto dvě podmínky poté nazveme *Markovovým řetězcem.*

Markovův řetězec lze tedy specifikovat maticí A přechodových pravděpodobností mezi stavy s_i a s_j, $i, j \in \{1, \ldots, N\}$. Pro prvky a_{ij} matice A platí:

- $a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$
- $a_{ij} \geq 0, \forall i, j$
- $\sum_{j=1}^{N} a_{ij} = 1, \forall i$

Dále je ještě potřeba specifikovat matici Π počátečních pravděpodobností π_i pro jednotlivé stavy s_i. Případně lze místo použití matice Π definovat speciální počáteční stav s_0 a rozšířit matici A o tento nový stav. Toto druhé řešení jsem použil v přiloženém programu (více v sekci 2.3).

Markovovy modely lze dobře aplikovat pro modelování pravděpodobností lineárních sekvencí.

Jinou reprezentací Markovových modelů může být stavový diagram (graf), kde vrcholy značí stavy a hrany značí pravděpodobnosti přechodů mezi stavy (příslušnými k dané hraně). Přechody mezi stavy s nulovou pravděpodobností se většinou z diagramů vynechávají. Podle vlastností prvků matice A platí, že součet pravděpodobností výchozích hran z každého stavu musí být roven jedné.

Z možné repreprezentace stavovým diagramem vyplývá, že Markovovy modely jsou vlastně *nedeterministickým konečným automatem*[1].

2.1.1 Viditelné Markovovy Modely

V případě viditelných Markovových modelů (Visible Markov Models) víme, kterými stavy konečný automat při výpočtu prochází. Jako výstup tedy můžeme mít sekvenci stavů či nějakou deterministickou funkci těchto stavů. Takovou funkcí může být i pravděpodobnost výskytu nějaké sekvence $X_1, \ldots, X_T$ v prostoru možných sekvencí.

[1]Pro tuto konstrukci nedeterministického konečného automatu jsou podstatné Markovovy vlastnosti

Tato pravděpodobnost může být snadno spočítána:

$$\begin{aligned} P(X_1,\ldots,X_T) &= P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\cdots P(X_T|X_1,\ldots,X_{T-1}) \\ &= P(X_1)P(X_2|X_1)P(X_3|X_2)\cdots P(X_T|X_{T-1}) \\ &= \pi_{X_1}\prod_{t=1}^{T-1} a_{X_tX_{t+1}} \end{aligned}$$

2.1.2 Skryté Markovovy Modely

Při použití skytých Markovových modelů (HMM) neznáme přímo sekvenci stavů (Markovův řetězec), přes které konečný automat prochází, lze pouze zjistit pravděpodobnosti možných sekvencí stavů.

Formálně (podle [1]) je HMM pětice (S, K, Π, A, B), kde S je množina stavů, K množina výstupních symbolů (výstupní abeceda), Π vektor pravděpodobností počátečních stavů, A matice přechodových pravděpodobností a B je matice výstupních pravděpodobností.

Množina stavů	$S = \{s_1, \ldots, s_N\}$
Výstupní abeceda	$K = \{k_1, \ldots, k_M\}$
Pravděpodobnosti počátečních stavů	$\Pi = \{\pi_i\}, i \in S$
Přechodové pravděpodobnosti	$A = \{a_{ij}\}, i, j \in S$
Výstupní pravděpodobnosti	$B = \{b_{ik}\}, i \in S, k \in K$

Náhodná veličina X_t mapuje prostor stavů do prostoru výstupní abecedy. V této verzi předpokládáme výstupy symbolů ze stavů (obecně lze uvažovat přechodové výstupy[2]).

2.1.3 Proč HMM?

Při řešení problému morfologické disambiguace se snažíme vytvořit jazykový model s plnou historií, jež by obsáhl veškeré závislosti přirozeného jazyka). Pro příliš velké paměťové a výpočetní nároky není však možné takový model vytvořit. Překážkou je zde i nutnost mít obrovská trénovací data pro zjištění jazykových závislostí. Proto se uchylujeme k zjednodušeným modelům, které lze efektivně implementovat. Jedním z těchto modelů je i HMM.

[2]HMM se stavovými výstupy lze snadno převést na HMM s přechodovými výstupy tak, že $\forall k_1, k_2, b_{ijk_1} = b_{ijk_2}$, kde b_{ijk} je výstupní pravděpodobnost při přechodu ze stavu i do stavu j s výstupním symbolem k.

Motivací k použití skrytých Markovových modelů (HMM) je myšlenka, že určitá sekvence byla generována z nějakého Markovova řetězce. K této sekvenci modelujeme HMM, abychom získali pravděpodobnosti možných sekvencí stavů.

Přihlédneme-li k Markovovým vlastnostem a vlastnostem přirozených jazyků, je použití HMM pro morfologickou disambiguaci vcelku opodstatněné.

2.1.4 Tři základní otázky pro HMM

1. Mějme daný model $\mu = (A, B, \Pi)$. Otázkou je jak účinně spočítat pravděpodobnost pozorované sekvence O, tedy $P(O|\mu)$?

2. Mějme dánu pozorovanou sekvenci O a model μ. Jak najdeme sekvenci stavů $(X_1, \ldots, X_{T+1})$, která nejpravděpodobněji emituje pozorovanou sekvenci O?

3. Mějme dánu pozorovanou sekvenci O a prostor možných modelů (možných hodnot parametrů modelu) $\mu = (A, B, \Pi)$. Jak najdeme model, který nejpravděpodobněji emituje pozorovanou sekvenci?

Takto formulované lze tyto otázky najít v [1]. V této práci se budu zabývat pouze 2. otázkou, která je podstatou morfologické disambiguace přirozeného jazyka při použití HMM.

Tuto otázku pro HMM dále zobecníme tak, že budeme řešit problém nalezení $n, n \in N$ nejpravděpodobnějších sekvencí[3]. Tato varianta se nazývá *n-best*. Abychom předešli zmatku ve značení dále v dokumentu, budeme počet nejlepších variant pro variantu n-best označovat písmenem m (tedy m-best)[4].

2.2 Druhá otázka pro HMM

Při řešení 2. otázky pro HMM (pro $m = 1$) řešíme problém (viz. [1]):

$$arg \max_X P(X|O, \mu)$$

[3]Zde se myslí až n nejlepších sekvencí, neboť různých sekvencí může existovat méně než n.

[4]Použití písmene n v tomto odstavci bylo pouze kvůli konvenci ve značení.

Pro pevné O:

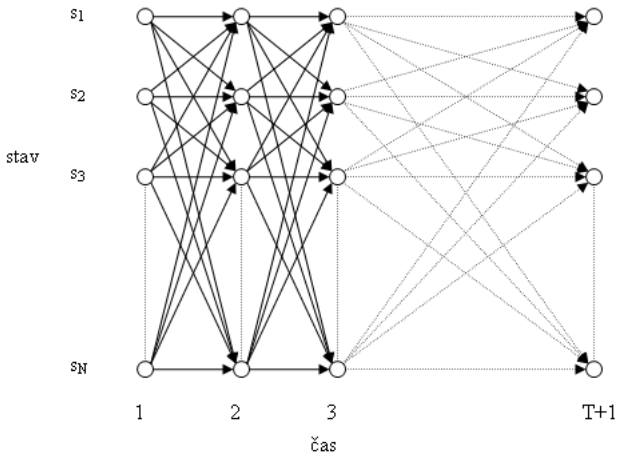
$$arg \max_X P(X, O|\mu)$$

Aplikací tohoto vzorce na řešení problému morfologické disambiguace vznikne vzorec vyjadřující závislost nejlepšího tagování $t_{1,n}$ na dané větě $w_{1,n}$[5].

$$\begin{aligned} arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) &= arg\max_{t_{1,n}} \frac{P(w_{1,n}|t_{1,n})P(t_{1,n})}{P(w_{1,n})} \\ &= arg\max_{t_{1,n}} P(w_{1,n}|t_{1,n})P(t_{1,n}) \end{aligned}$$

Pro efektivní řešení 2. otázky byl vyvinut *Viterbiho algoritmus*, který je podrobně popsán dále v textu.

2.2.1 Trellis



Obrázek 2.1: Trellis

Pro efektivní výpočet Viterbiho algoritmu je důležitá datová struktura zvaná *trellis*. Jde o vrstevnatou strukturu, kde každá vrstva obsahuje možné stavy pro element v sekvenci. Přechody existují pouze ze stavů aktuální vrstvy do stavů vrstvy následující (viz. obrázek 2.1 převzaný z [1]). Stavy v *trellis* mají význam tagů, případně n-tic tagů[6], a to i v případě varianty m-best, se kterou se setkáme dále v textu.

[5]Použijeme Bayesovo pravidlo pro podmíněnou pravděpodobnost

[6]N-tice jsou potřebné pro možnost použití delší historie. V tomto případě se předchozí tagy shlukují do jednoho stavu.

2.2.2 Viterbiho algoritmus

Definujme:

$$\delta_j(t) = \max_{X_1 \cdots X_{t-1}} P(X_1 \cdots X_{t-1}, o_1 \cdots o_{t-1}, X_t = j|\mu)$$

Tato proměnná tedy obsahuje pravděpodobnost nejpravděpodobnější cesty z počátku do stavu t, který leží v j-té vrstvě. Definujeme ještě další proměnnou $\psi_j(t)$, která bude obsahovat uzel[7], z kterého vede poslední hrana této nejpravděpodobnější cesty do stavu t.

Nyní stačí použít principů dynamického programování, abychom získali *Viterbiho algoritmus* (pro $m = 1$, jak je prezentován v [1]):

```
// mějme větu délky n
// inicializace
for all tags t^j do
    δ_1(t^j) = π_j

// indukce
for i:=1 to n do
    for all tags t^j do
        δ_{i+1}(t^j) := max_{1≤k≤T} [δ_i(t^k) · P(w_{i+1}|t^j) · P(t^j|t^k)]
        ψ_{i+1}(t^j) := arg max_{1≤k≤T} [δ_i(t^k) · P(w_{i+1}|t^j) · P(t^j|t^k)]
    end
end

// ukončení a přečtení cesty
X_{n+1} = arg max_{1≤j≤T} [δ_{n+1}(t^j)]
for j:=n downto 1 do
    X_j = ψ_{j+1}(X_{j+1})
end
P(X_1, ..., X_n) = max_{1≤j≤T} [δ_{n+1}(t^j)]
```

Pro zápis algoritmu varianty *m-best* ($m > 1$) potřebujeme předefinovat následující proměnné. Proměnná $\delta_{i+1}(t, l)$ má nyní význam pravděpodobnosti l-té položky ve stavu t ve vrstvě i v *trellis*. Proměnná $\psi_{i+1}(t, l)$ potom obsahuje dvojici proměnných (k, l), které znamenají po řadě stav v předchozí vrstvě a položku v tomto stavu. Z této dvojice vede nejpravděpodobnější přechod do položky (t, l) v aktuální vrstvě. Proměnná $m_{i+1}(t)$ obsahuje počet položek varianty m-best (může jich být méně než

[7] Tento uzel bude ležet v předchozí (t-1)-ní vrstvě v *trellis*.

m) v tagu t ve vrstvě i. Proměnná m_c obsahuje maximálně m různých nejpravděpodobnějších cest z počátku přes celou sekvenci. *Viterbiho algoritmus* pro obecné m potom vypadá následovně:

// mějme větu délky n
// inicializace
for all tags t^j **do**
    $\delta_1(t^j, 1) := \pi_j$
    $m_1(t^j) := 1$
    for $i := 2$ **to** $n + 1$ **do**
        $m_i(t^j) := m$
    end
end

// indukce
for i:=1 **to** n **do**
    for all tags t^j **do**
        for $s := 1$ **to** $m_{i+1}(t^j)$ **do**
            $U := \{(k, l)|1 \leq k \ \leq T, 1 \leq l \leq m_i(t^k), (k, l) \neq \psi_{i+1}(t^j, r),\ r < s\}$
            if $U = \emptyset$ **then**
                $m_{i+1}(t^j) := s - 1$
                break
            end
            $\delta_{i+1}(t^j, s) := \max\limits_{(k,l)\in U} \left[\delta_i(t^k, l) \cdot P(w_{i+1}|t^j) \cdot P(t^j|t^k)\right]$
            $\psi_{i+1}(t^j, s) := \arg \max\limits_{(k,l)\in U} \left[\delta_i(t^k, l) \cdot P(w_{i+1}|t^j) \cdot P(t^j|t^k)\right]$
        end
    end
end

// ukončení a přečtení cesty
$U = \{(k, l)|1 \leq k \ \leq T, 1 \leq l \leq m_{n+1}(t^k)\}$
$m_c := \min\{|U|, m\}$
for $s := 1$ **to** m_c **do**
    $X_{n+1,s} = \arg \max\limits_{\substack{(j,l)\in U \\ (j,l)\neq X_{n+1,r},\ r<s}} \left[\delta_{n+1}(t^j, l)\right]$
end
for j:=n **downto** 1 **do**
    for $s := 1$ **to** m_c **do**
        $X_{j,s} = \psi_{j+1}(X_{j+1,s})$
end
for $s := 1$ **to** m_c **do**
    $P_s(X_1, \ldots, X_n) = \max\limits_{\substack{(j,l)\in U \\ (j,l)\neq X_{n+1,r},\ r<s}} \left[\delta_{n+1}(t^j, l)\right]$
end

2.3 Použití HMM pro morfologickou disambiguaci

Pro použití HMM a Viterbiho algoritmu pro morfologickou disambiguaci přirozeného jazyka potřebujeme nějakým způsobem získat přechodové a výstupní pravděpodobnosti a také pravděpodobnosti počátečních stavů.

K tomuto účelu použijeme sadu *trénovacích dat*, která jsou ručně otagována, abychom se naučili závislosti mezi tagy (přechodové pravděpodobnosti), závislosti mezi slovy a jim přidělenými tagy (výstupní praděpodobnosti) a také počty tagů prvních slov ve větách (pravděpodobnosti počátečních stavů). Postup, který použijeme, se nazývá *maximum likelihood estimation* (MLE). Jde o jednoduchý výpočet podmíněných pravděpodobností. Základem pro výpočet je:

- $C(t^j, t^k)$ - četnost dvojic po sobě následujících tagů t^j a t^k
- $C(t^j)$ - četnost tagů t^j
- $C(w^l, t^j)$ - četnost tagů t^j emitujících slovo w^l

Pro přechodové pravděpodobnosti poté vypočítáme:

$$P(t^k|t^j) = \frac{C(t^j, t^k)}{C(t^j)}$$

Pro výstupní pravděpodobnosti:

$$P(w^l|t^j) = \frac{C(w^l, t^j)}{C(t^j)}$$

Pravděpodobnosti počátečních stavů lze počítat jako podíl četností jednotlivých tagů na prvním místě ve větě ku celkovému počtu tagů na prvním místě ve větě (tedy počtu vět). V následujícím vzorci je $C_1(t^j)$ četnost tagu t^j na prvním místě ve větě a C_1 počet vět.

$$\pi_j = \frac{C_1(t^j)}{C_1}$$

Většinou se ale používá trik, který spočívá v zavedení falešného tagu, který má význam počátku věty. Pravděpodobnosti počátečních stavů se potom počítají stejně jako přechodové pravděpodobnosti (zde FAKE označuje falešný tag):

$$\pi_j = \frac{C(FAKE, t^j)}{C(FAKE)}$$

Toto řešení jsem také v programu použil.

2.3.1 Elementární trénovací algoritmus

Podle výše uvedených vzorců lze snadno sestavit elementární trénovací algoritmus[8] jak je prezentován v [1]:

```
for all tags t^j do
    for all tags t^k do
        P(t^k|t^j) := C(t^j,t^k) / C(t^j)
    end
end
for all tags t^j do
    for all words w^l do
        P(w^l|t^j) := C(w^l,t^j) / C(t^j)
    end
end
```

Četnosti tagů a slov $C(t^j, t^k)$, $C(t^j)$ a $C(w^l, t^j)$ předtím spočítáme z trénovacích dat.

2.4 N-gramové modely

Markovovy modely definované v předchozích kapitolách počítají s omezeným rozhledem. Díky tomu jsme si definovali rychlý *Viterbiho algoritmus*. Omezený rozhled nás ovšem limituje v tom, že můžeme používat pouze podmíněnou pravděpodobnost jednoho tagu na jiném. Velmi užitečné ale může být použití podmíněné pravděpodobnosti jednoho

[8] Dále v textu ještě potkáme úplný trénovací algoritmus, který obsahuje *lineární interpolaci* podmíněných pravděpodobností s historií delší než 1.

tagu na několika předchozích tazích v sekvenci. Jinak řečeno je vhodné použít delší historii pro HMM Tagger.

Pro n-gramový model platí:

$$P_n(t^m|t^{m-n+1}, \ldots, t^{m-1}),\ n > 1$$

Klasický HMM tedy tvoří bigramový model. Otázkou je, jak obecný n-gramový model ($n > 2$) počítat pomocí HMM. Možným řešením je shlukování více stavů do jednoho.

Pokud například chceme vyjádřit trigramový model, stačí shlukovat stavy tak, že nové stavy budou obsahovat dva původní stavy.

Místo

$$P(A|B, C)$$

potom počítáme

$$P((BA)|(CB)).$$

Obdobně lze použít HMM i pro vyšší n než 3. Cenou za tuto úpravu bude exponenciální růst počtu stavů[9]. O *Markovovu modelu* použitém na *n-gramovém modelu* mluvíme jako o *Markovovu modelu* řádu $n - 1$.

2.4.1 Použití n-gramových modelů pro morfologickou disambiguaci

Při letmém pohledu na možnost použití n-gramových modelů nás může napadnout použití delší historie (vyššího n) v HMM. To zlepšuje schopnost vystihnout řadu jazykových závislostí. Bigramový model umožňuje odvozování tagu pouze z jednoho předchozího tagu. N-gramový model nám otvírá možnost odvozování tagu od více předchozích tagů. S těmito novými vlastnostmi se ale pojí i několik problémů.

Prvním z nich je již zmíněné zvýšení počtu stavů mající za následek zvětšení paměťové a časové náročnosti programu.

Dalším problémem je řídkost dat (data sparseness). Pro správné natrénování HMM potřebujeme použít velké množství trénovacích dat, které roste společně s rostoucím počtem stavů.

Další problém je možnost vzniku nulových pravděpodobností některých přechodů a výstupů. Důvod je prozaický. Vlivem řídkosti trénovacích dat

[9]Musíme uvážit, že pokud máme T různých stavů, můžeme v n-gramovém modelu získat až T^{n-1} různých stavů.

se program během trénování nesetkal s určitými závislostmi tagů mezi sebou nebo se závislostmi výstupních slov na tazích. Proto při výpočtu přechodových nebo výstupních pravděpodobností podle MLE mohou po dosazení nulových četností vyjít nulové pravděpodobnosti, a to i přesto, že v reálu jsou tyto pravděpodobnosti nenulové. Řešením je použití různých vyhlazovacích metod, o kterých bude řeč dále.

Experimentálně bylo zjištěno, že nemá význam neomezeně zvyšovat délku historie. V praxi běžně stačí trigramový model.

2.5 Lineární interpolace

Jedním ze způsobů řešení či omezení problému řídkosti dat je použití *lineární interpolace* ([1]) různých n-gramových pravděpodobností až do určitého n. Jde o lineární kombinaci různých délek historie a absolutního členu, který řeší problém nulových pravděpodobností, neboť je vždy nenulový[10].

Absolutním členem pro přechodové pravděpodobnosti může být $1/T$, kde T znamená počet různých tagů v trénovacích datech či případně počet všech druhů tagů. Lineární interpolace přechodových pravděpodobností potom vypadá následovně (pro $n = 3$):

$$P(t^j|t^{j-2}, t^{j-1}) = \lambda_3 P_3(t^j|t^{j-2}, t^{j-1}) + \lambda_2 P_2(t^j|t^{j-1}) + \lambda_1 P_1(t^j) + \lambda_0 \frac{1}{T}$$

Přičemž musí platit: $\sum_{i=0}^{n} \lambda_i = 1$, $\forall i\ \lambda_i > 0$.

Absolutním členem lineární interpolace výstupních pravděpodobností může být $1/W$, kde W znamená počet slov ve slovníku (v trénovacích datech). Zde nemá smysl jako W uvažovat počet všech možných slov, neboť toto číslo ani neznáme. Lineární interpolace výstupních pravděpodobností vypadá následovně:

$$P(w^l|t^j) = \lambda_2 P_2(w^l|t^j) + \lambda_1 P_1(w^l) + \lambda_0 \frac{1}{W}$$

Zde opět musí platit: $\sum_{i=0}^{2} \lambda_i = 1$, $\forall i\ \lambda_i > 0$.

[10] Jiným řešením je použití vyhlazovacích metod. Několik z nich lze nalézt v [1].

2.6 EM algoritmus

Nejjednodušším způsobem, jak nastavit lambda koeficienty pro členy lineární interpolace, je uniformní rozdělení (zde $k+1$ je počet členů lineární interpolace):

$$\lambda_i = \frac{1}{k+1} \tag{2.1}$$

Lepší nastavení koeficientů nám ovšem může poskytnout *EM algoritmus* (EM od Expectation Maximization, viz. [1]). Jde o numerickou metodu. K jejímu použití potřebujeme balík dat, která nebyla použita pro trénování. Tato data se nazývají *held-out data*, označme je H. EM algoritmus, jak byl převzat z [2], poté vypadá následovně ($\tilde{p}(t_1, \ldots, t_k)$ je relativní četnost k-gramu v heldout datech):

```
// dána přesnost ε
// dán počet členů interpolace k

// inicializace
for i := 0 to k do
    λ_i := 1/(k+1)

// iterace algoritmu
do
    for i := 0 to k do
        φ_i := λ_i
    for i := 0 to k do
        c_i := Σ_{t_1,...,t_k ∈ H} p̃(t_1,...,t_k) · λ_i p_i(t_k|t_{k-i+1},...,t_{k-1}) / Σ_{j=0}^{k} λ_j p_j(t_k|t_{k-j+1},...,t_{k-1})
    for i := 0 to k do
        λ_i := c_i / Σ_{j=0}^{k} c_j
while ∃i |λ_i − φ_i| > ε
```

Jako held-out data H jsou v případě výpočtu lambd pro přechodové pravděpodobnosti použity pouze tagy. V případě výstupních pravděpodobností se používají tagy i slova a pravděpodobnosti v algoritmu jsou v odpovídajících variantách (např. místo $P(t_j|t_{j-1})$ se v algoritmu nachází $P(w^l|t_j)$).

2.7 Poznámky k trénování

Jak bylo zmíněno v sekci 2.3.1, během trénování je vhodné spočítat četnosti tagů, slov a jejich závislostí. Ty se potom ukládají do datových struktur. Není vhodné rovnou počítat pravděpodobnosti, jak jsou popsané v trénovacím algoritmu, nebo jejich lineární interpolace. Důvod je prostý - zbytečně by se počítalo mnoho pravděpodobností, které nikdy nebudou využity a ukládala by se tak zbytečně velká data. Místo toho je vhodné uložit si jen četnosti tagů, slov a jejich závislostí a potřebné pravděpodobnosti počítat, až budou potřeba přímo ve **Viterbiho algoritmu**. Mírně se tak zvýší časová náročnost Viterbiho algoritmu, ale podstatně se sníží celková paměťová náročnost programu.

Held-out data nesmí být předtím použita pro trénování, jinak rozvržení vah pro lambdy nedopadne správně a veškerá váha připadne jediné lambdě (důkaz lze nalézt v [5]). Data použitá v EM algoritmu je ale následně možné použít k dodatečnému dotrénování.

2.8 Měření úspěšnosti tagování

Účelem morfologické analýzy je přiřadit slovům ve větě správné tagy. Existuje několik základních měřítek úspěšnosti tagování. Definujme nejprve několik pojmů: t jako počet tokenů v testovacích datech, h jako počet tokenů, jejichž manuálně přiřazený tag je přítomen ve výstupu tagování a c jako počet všech tagů přiřazených tokenům.

Jedním sledovaným hlediskem je *precision*

$$p = \frac{h}{c}$$

dále *recall*

$$r = \frac{h}{t}$$

posledním hlediskem je *F-measure*[11]

$$f = \frac{2 * p * r}{p + r}$$

[11] Obecně se F-measure počítá jako $F_\alpha = \frac{(1+\alpha)\cdot(precision\cdot recall)}{\alpha\cdot precision+recall}$ pro dané α. V případě $\alpha = 1$ jde o "tradiční" F-measure, kde má precision i recall stejnou váhu.

V případě základní verze Viterbiho algorimu, kdy se každému tokenu přiřadí právě jeden tag, platí $p = r = f$. Tato veličina se potom nazývá *accuracy*.

Kapitola 3

Program

Program popisovaný v této kapitole a přiložený na kompaktním disku k této práci je implementací skrytých Markovových modelů použitou pro morfologické značkování češtiny. Účelem programu (HMM Taggeru) je umožnit otestování vlivu různých parametrů na úspěšnost tagování.

HMM Tagger je implementován v programovacím jazyku Java 5.0 (podle starého číslování Java 1.5). Pro použití Ant je potřeba verze 1.6.2.

Důraz při tvorbě programu byl kladen na variabilitu parametrů HMM Taggeru. Program umožňuje nejen specifikovat délku historie tagování, ale lze také určit velikost m pro verzi *m-best* a použít pro tagování pouze podmnožinu znaků v tagovací značce.

Podstatným omezením tohoto programu je potřeba větší paměti (řádově stovky MB až jednotky GB), při velké historii tagování či velkém m při větších trénovacích datech[1]. Při nedostatku paměti se může objevit výjimka *java.lang.OutOfMemory*.

Umožnění velké variability programu negativně ovlivňuje rychlost celého programu. Dochází k mírnému zpomalení běhu oproti pevně daným parametrům. Vzhledem k typu aplikace programu je ale takovéto zpomalení celkem irelevantní. Přesto by si měl být uživatel vědom toho, že běh může u některých parametrů trvat i několik hodin.

Informace k programu lze nalézt v této kapitole, v nápovědě vygenerované při spuštění programu s parametrem `-h` a také v JavaDoc dokumentaci nacházející se na CD přiloženém k této práci.

[1] Velikost heapu pro Java Virtual Machine lze nastavovat pomocí přepínačů `-Xms` a `-Xmx`. Například počáteční velikost heapu na 64 MB a maximální velikost na 768 MB lze nastavit pomocí `-Xms64m -Xmx768m`.

3.1 Uživatelská dokumentace

Běh programu je stavový, podle parametrů jej lze spouštět v různých módech. Pod pojmem stavový je myšleno, že HMM Tagger musí být natrénován, než může být použit k tagování. Program se tedy nejprve spustí s trénovacími parametry a poté s tagovacími. Po natrénování si program uloží natrénovaná data na disk. Trénování a tagování jsou pouze dva základní módy. Program navíc obsahuje i testování výsledků tagování a EM algoritmus (viz. 2.6), pomocí kterého se počítají koeficienty pro lineární interpolaci přechodových a výstupních pravděpodobností.

Uložená natrénovaná data mohou být velká desítky až stovky MB, jejich načítání a ukládání je časově náročné. Z důvodu odstranění tohoto problému byl do programu přidán speciální *makro* mód. V *makro* módu lze sekvenčně provádět všechny ostatní módy bez nutnosti ukládání a načítání natrénovaných dat. Stačí specifikovat makro soubor, který obsahuje prováděné příkazy (podrobněji viz. 3.1.4).

3.1.1 Spouštění programu

Program neobsahuje žádné grafické prostředí a spouští se výhradně z příkazové řádky. Parametrů příkazové řádky je značné množství, detailně jsou popsány v tabulce.

`-tr`	Označuje trénovací mód. Potřebuje parametry `-n` a `-i` nebo `-di`. Vylučuje se s parametry `-tg`, `-ts`, `-m`, `-h`, `-o`, `-do`, `-dc`.
`-tg`	Označuje tagovací mód. Potřebuje parametry `-n` a `-i`, `-o` nebo `-di`, `-do`. Vylučuje se s parametry `-tr`, `-ts`, `-m`, `-h`.
`-em`	Označuje mód pro EM algoritmus. V dalším parametru může následovat číslo od 0 do 3 určující, zda EM algoritmus bude počítat přechodové a/nebo výstupní pravděpodobnosti. Pokud numerická hodnota v dalším parametru chybí použije se defaultní hodnota 3 (více viz. 3.1.3). Potřebuje parametr `-ei` nebo `-ed`. Vylučuje se s parametry `-ts`, `-m`, `-h`, `-x`.

`-ts`	Označuje mód pro testování výsledků tagování. Potřebuje parametry `-i` nebo `-di`. Vylučuje se s parametry `-tr`, `-tg`, `-m`, `-em`, `-en`, `-ei`, `-ed`, `-n`, `-x`, `-h`, `-b`, `-o`, `-do`, `-dc`.
`-m`	Označuje makro mód. V dalším parametru musí následovat název makro souboru. Vylučuje se se všemi ostatními parametry.
`-h`	Určuje mód pro nápovědu. Tento parametr je silnější než všechny ostatní a v případě, že je zadán, se zobrazí nápověda.
`-n`	Specifikuje délku trénovací a tagovací historie. V dalším parametru musí následovat číslo určující délku historie. Používá se s parametrem `-tr` nebo `-tg`.
`-b`	Specifikuje m pro verzi m-best. V dalším parametru musí následovat číslo určující m. Pokud parametr chybí, defaultní hodnota je 1. Používá se s parametrem `-tg`.
`-sp`	Specifikuje speciální trénovací a tagovací historii. Specifikuje speciální historii použitou v EM algoritmu. Specifikuje i speciální hodnotu tagu testovanou v testovacím módu. V dalším parametru musí následovat speciální sekvence hexadecimálních čísel. Používá se s parametry `-tr`, `-tg`, `-ts`, `-em`.
`-i`	Specifikuje vstupní datové soubory. V dalších parametrech musí následovat názvy souborů. Používá se s parametry `-tr`, `-tg`, `-ts`.
`-o`	Specifikuje výstupní datové soubory. V dalších parametrech musí následovat názvy souborů. Používá se s parametrem `-tg`.
`-di`	Specifikuje adresáře se vstupními datovými soubory. V dalších parametrech musí následovat názvy adresářů. Používá se s parametry `-tr`, `-tg`, `-ts`.
`-do`	Specifikuje adresáře s výstupními datovými soubory. V dalších parametrech musí následovat názvy adresářů. Používá se s parametry `-tg`.

`-dc`	Specifikuje suffix jmen souborů (vkládaný před příponu), které jsou obsažené v adresářích specifikovaných za `-di`. Otagované soubory jsou vytvářeny v adresářích specifikovaných za `-do`. V dalším parametru musí následovat vkládaný suffix.
`-en`	Specifikuje délku historie pro EM algoritmus. V dalším parametru musí následovat číslo určující délku historie. Používá se s parametrem `-em`.
`-ei`	Specifikuje vstupní datové soubory pro EM algoritmus. V dalších parametrech musí následovat názvy souborů. Používá se s parametry `-em`.
`-ed`	Specifikuje adresáře se vstupními datovými soubory pro EM algoritmus. V dalších parametrech musí následovat názvy adresářů. Používá se s parametry `-em`.
`-ln`	Specifikuje logovací soubor. V dalším parametru musí následovat název logovacího souboru. Používá se s parametry `-tr`, `-tg`, `-ts`, `-em`.
`-ll`	Specifikuje úroveň logování. V dalším parametru musí následovat číslo od 1 (základní úroveň) do 3 (detailní úroveň) určující úroveň logování. Pokud tento parametr chybí, je použita defaultní hodnota 2. Používá se s parametrem `-ln`.
`-x`	Specifikuje mazání datových struktur. Vylučuje se s parametry `-ts`, `-em`, `-m`, `-h`. Používá se s parametrem `-tr` nebo `-tg`.
`-c`	Specifikuje název kódování souboru. V dalším parametru musí následovat název kódování. Používá se s parametry `-i`, `-o`, `-di`, `-do`, `-ei`, `-ed`.

3.1.2 Přepínače příkazové řádky

V případě chybně zadaných parametrů se zobrazí chybová hláška a nápověda.

Přepínače programu musí začínat znakem '-'. Ostatní parametry nesmí začínat znakem '-'. Pro zrušení speciálního významu znaku '-' (např. pokud chceme použít v parametru `-dc` suffix "-x") lze použít zpětné lomítko ('\'), tento znak ruší speciální význam pouze počátečního znaku. Pokud je tedy na počátku parametru backslash, použije se parametr až od druhého znaku. Pro použití zpětného lomítka na počátku parametru je nutné jej zdvojit (pozor na případné shellovské escapování).

Příklady:

Požadovaný řetězec	**Řetězec, který je třeba zadat**
`-x`	`\-x`
`-x`	`\\-x (v Bashi)`

Každý přepínač lze na příkazové řádce použít pouze jednou.

3.1.3 Běh EM algoritmu

Mód pro běh EM algoritmu obsahuje určité specifikum v tom, že může být spuštěn samostatně nebo může být přiřazen k trénovacímu či tagovacímu módu. Pokud je přiřazen k trénování, proběhne nejprve trénování a potom se spustí EM algoritmus. Pokud je přiřazen k tagování, nejprve proběhne EM algoritmus a až potom tagování.

Parametrem následujícím za `-em` může být číselná hodnota od 0 do 3 specifikující, zda EM algoritmus proběhne pro přechodové a/nebo výstupní pravděpodobnosti. Jde o kombinaci dvou bitů. První bit je pro výstupní, druhý pro přechodové pravděpodobnosti. Pokud je daný bit nastaven jako pravdivý (1), proběhne daný výpočet. Pokud chybí parametr s číselnou hodnotou, použije se defaultní hodnota 3 a výpočet proběhne pro přechodové i výstupní pravděpodobnosti. Hodnoty lze nalézt v tabulce.

Hodnota	**Význam**
0	žádný výpočet neproběhne
1	výpočet proběhne pouze pro přechodové pravděpodobnosti
2	výpočet proběhne pouze pro výstupní pravděpodobnosti
3	defaultní hodnota, výpočet proběhne pro přechodové i výstupní pravděpodobnosti

Podstatné pro běh EM algoritmu je, jak bylo zmíněno v sekci 2.7, že held-out data nesmí být předtím použita k trénování. Mohou být ovšem použita k dodatečnému dotrénování po doběhnutí EM algoritmu.

3.1.4 Makro soubor

Pro snazší používání a rychlejší běh programu lze použít makro soubor. Makro soubor by měl na každém řádku obsahovat jeden příkaz. Běh v makro režimu je rychlejší díky zrušení potřeby ukládání (serializaci) natrénovaných dat.

Příkazem v makro souboru může být řádek s parametry pro jednotlivý běh programu. Soubor může také obsahovat komentáře. Komentář začíná znakem '`#`' a pokračuje až na konec řádku. Speciální význam znaku '`#`' lze zrušit escapováním pomocí znaku '\' nebo uzavřením mezi uvozovky '`"`'. Víceslovné parametry lze použít buď uzavřené mezi uvozovky nebo musí být bílé znaky (mezera, tabulátor) escapovány pomocí zpětného lomítka. Uvnitř uvozovek pozbývá backslash svého speciálního významu.

V makro režimu nelze použít parametr '`-h`' pro zobrazení nápovědy. Počet příkazů není nijak omezen.

Příklady:

Požadovaný řetězec	**Řetězec, který je třeba zadat**
`hello world`	`hello\world`
`hello world`	`"hello world"`
`hello world`	`"hello "world`
`hello world`	`"hello"" "wor\ld`
`/home/wombat`	`"/home/wombat"`
`/home/wombat`	`/home/wombat`
`d:\data`	`d:\\data`
`hello "Peter"`	`hello\ \"Peter\"`
`hello "Peter"`	`"hello "\"Peter\"`

3.1.5 Formátování datových souborů

Program je uzpůsoben pro použití vstupních souborů ve formátu CSTS (data z [6]). Program soubory parsuje a pro trénování používá tagy <`t`>, <`f`> a <`d`>. Pro tagování navíc tagy <`MMt`> a <`MMl`> obsahující výstup morfologické analýzy. Výstupní soubor vzniklý při tagování se od vstupního liší pouze přidáním tagů <`MDt`> a <`MDl`>, které obsahují nalezené tagy a lemmata. V případě verze m-best jsou navíc přidány atributy *src* a to následujícím způsobem: <`MDt src="n1"`>, <`MDl src="n1"`>, <`MDt src="n2"`> atd.

3.1.6 Zadávání speciální historie

Parametrem následujícím za přepínačem `-sp` musí být sekvence hexadecimálních čísel oddělených středníky a uzavřená v kulatých závorkách. Každé hexadecimální číslo musí mít 4 číslice. První číslice může nabývat pouze hodnot z rozmezí 0 až 7 a definuje speciální hodnotu tagu, který je počítán; druhá číslice hodnotu předchozího tagu; atd. Hexadecimální číslice mají význam binárního zápisu použitých pozic v tagu (tagy použité v datech z PDT 2.0 mají 15 znaků (viz. [6])). Ve výsledku je možné použít libovolné podmnožiny znaků všech tagů v historii.

Při použití pro testování výsledků tagování bude sekvence obsahovat pouze jedno hexadecimální číslo o 4 znacích (zde nemají tagy v historii žádný význam).

Při použití je třeba dát pozor na to, aby měly všechny příkazy v sekvenci[2] stejné hodnoty speciální historie.

Příklady:

Zadaný parametr	**Význam**
`(7000)`	Z aktuálně počítaného tagu se vezmou pouze první 3 znaky
`(7f00;7000;7000)`	Z aktuálně počítaného tagu se vezme prvních 7 znaků a ze dvou předchozích první 3 znaky.

Při použití speciální historie jsou během tagování ve Viterbiho algoritmu počítány tagy nabízené v morfologické analýze. Rozdíl oproti klasickému tagování je v tom, že praděpodobnosti přechodu a výstupní pravděpodobnosti jsou počítány pouze z dané podmnožiny znaků v počítaném n-gramu.

3.1.7 Logovací soubory

Logovací soubor, jehož název je definovaný za parametrem `-ln`, se vždy vytváří nový. Pokud existoval soubor se stejným jménem, je smazán a je vytvořen soubor nový (pokud to dovolí operační systém). V případě, že se

[2]Sekvencí příkazů myslíme posloupnost příkazů na stejných datech, tzn. příkazy od jednoho vymazání natrénovaných dat do dalšího vymazání natrénovaných dat. Všechny tyto příkazy musí mít stejnou speciální historii.

nepodaří soubor vytvořit nebo není jeho název za parametrem `-ln` specifikován, vypisují se logovací informace na standartní výstup.

Program nabízí tři úrovně logování: základní, normální a detailní.

Každý mód kromě makro módu zapisuje do logů různá data. Pokud je mód pro EM algoritmus přidružen k trénování či tagování, potom jsou i logovací výstupy EM agloritmu přidruženy k trénovacím resp. tagovacím logům.

Informace, které do logů zapisují všechny módy jsou: čas začátku a konce běhu programu v daném módu, příkazová řádka, kódování souborů a úroveň logování.

Data zapisovaná v jednotlivých módech:

1. **trénování**
   (a) základní
      - trénovací historie
      - speciální historie
      - počet vět, počet tagů a počet druhů tagů v trénovacích datech

   (b) normální
      - počty jednotlivých druhů tagů

   (c) detailní
      - počty jednotlivých lemmat

2. **EM mód**
   (a) základní
      - historie pro běh EM algoritmu
      - speciální historie pro EM algoritmus
      - počet vět, počet tagů a počet druhů tagů v datech pro EM algoritmus
      - počty iterací EM algoritmu při výpočtu koeficientů pro přechodové a výstupní pravděpodobnosti
      - vypočítané koeficienty pro přechodové a výstupní pravděpodobnosti

   (b) normální

- počty jednotlivých druhů tagů v datech pro EM algoritmus

(c) detailní

- počty jednotlivých lemmat v datech pro EM algoritmus

3. **tagování**

(a) základní

- tagovací historie
- speciální historie
- m-best
- počet vět v tagovaných datech
- počet tokenů v tagovaných datech
- průměrný počet vypočítaných tagů (<MDt>) na token
- počet vypočítaných tagů a lemmat a počty druhů vypočítaných tagů a různých lemmat
- koeficienty pro přechodové a výstupní pravděpodobnosti

(b) normální

- počty jednotlivých druhů vypočítaných tagů

(c) detailní

- počty jednotlivých vypočítaných lemmat

4. **testování**

(a) základní

- speciální hodnota testovaných tagů
- úspěšnost tagování (accuracy-tags)
- úspěšnost přiřazení lemmat (accuracy-lemmas)
- precision testovaných dat
- recall testovaných dat
- f-measure testovaných dat
- počet správně vypočítaných tagů (ručně přiřazený tag se nachází mezi vypočítanými tagy)
- počet špatně vypočítaných tagů (jeden na token)

- počet špatně vypočítaných tagů s ohledem na verzi m-best (může jich být na jeden token víc)
- počet druhů špatně vypočítaných tagů s ohledem na verzi m-best
- počet špatně přiřazených lemmat (jedno na token)
- počet špatně přiřazených lemmat s ohledem na verzi m-best
- počet různých špatně přiřazených lemmat s ohledem na verzi m-best
- počet vět a tokenů v testovaných datech
- počet tokenů se špatnou morfologickou analýzou (ručně přiřazený tag se nenachází mezi tagy nabízenými morfologicku analýzou)
- recall morfologického analyzátoru (vstupních dat)
- průměrný počet vypočítaných tagů na token
- průměrný počet tagů nabízených morfologickou analýzou na jeden token
- počet druhů tagů v testovaných datech
- počet různých lemmat v testovaných datech
- počet vypočítaných tagů
- počet druhů vypočítaných tagů
- počet přiřazených lemmat
- počet různých přiřazených lemmat

(b) normální

- upravená úspěšnost tagování - vynechané tokeny se špatnou morfologickou analýzou
- upravená úspěšnost tagování[3]
- upravená úspěšnost přiřazení lemmat[3]
- upravený počet špatně vypočítaných tagů[3]
- upravený počet druhů špatně vypočítaných tagů[3]
- upravený počet špatně vypočítaných lemmat[3]
- upravený počet různých špatně vypočítaných lemmat[3]
- upravený počet vět[3]

[3]Pod pojmem upravená úspěšnost je myšleno, že byly vynechány všechny věty obsahující alespoň jeden token se špatnou morfologickou analýzou.

- upravený počet tagů[3]
- upravený počet druhů tagů[3]
- upravený počet lemmat[3]
- upravený počet různých lemmat[3]
- upravený počet vypočítaných tagů[3]
- upravený počet druhů vypočítaných tagů[3]
- upravený počet přiřazených lemmat[3]
- upravený počet různých přiřazených lemmat[3]
- špatná přiřazení tagů
- špatná přiřazení lemmat
- počty špatných přiřazení tagů vzhledem k pozici ve větě (místo ve větě - počet - procento ze všech těchto míst ve větě)
- počty špatných přiřazení lemmat vzhledem k pozici ve větě (místo ve větě - počet - procento ze všech těchto míst ve větě)
- upravené počty špatných přiřazení tagů vzhledem k pozici ve větě (místo ve větě - počet - procento ze všech těchto míst ve větě)[3]
- počty špatných přiřazení lemmat vzhledem k pozici ve větě (místo ve větě - počet - procento ze všech těchto míst ve větě)[3]

(c) detailní

- počty vět různých délek v testovaných datech (délka - počet)
- upravené počty vět různých délek v testovaných datech (délka - počet)[3]
- počty jednotlivých druhů tagů v testovaných datech
- počty jednotlivých lemmat v testovaných datech
- počty jednotlivých druhů vypočítaných tagů v testovaných datech
- počty jednotlivých přiřazených lemmat v testovaných datech

3.1.8 Poznámky k trénování a tagování

Pokud není v programu použita speciální historie, lze program natrénovat na delší historii, než jaká je následně použita během tagování. Tato možnost je důsledkem návrhu datových struktur (podrobněji popsaných v sekci 3.2.2).

Uživatel by si měl pouze dát pozor na použití koeficientů pro lineární interpolaci. Koeficienty vypočítané pro určitou délku historie nebudou vhodné pro použití na jiné délce historie.

3.1.9 Další poznámky k parametrům

Přepínač `-x` lze použít při trénování nebo tagování. Pokud je použit při trénování, nejprve se vymažou stará natrénovaná data a teprve poté se přejde k vlastnímu trénování. Pokud je použit při tagování, nejprve proběhne tagování a poté se vymažou natrénovaná data.

3.1.10 Příklady vstupních parametrů

Trénování

```
-tr -n 3 -ln "HMM_train_h3.log" -ll 2 -i "soubor1.csts"
-tr -x -n 2 -di "/home/wombat/adr1" "/home/wombat/adr2"
```

Trénování s EM algoritmem

```
-tr -n 3 -ln "HMM_train_em_h3.log" -ll 2 -em -en 3 -ei "soubor_pro_em.csts"
   -i "soubor1.csts"
-tr -x -n 2 -di "/home/wombat/adr1" "/home/wombat/adr2" -em -ed
   "/home/wombat/adr_pro_em"
```

EM algoritmus

```
-em -en 3 -ei "soubor_pro_em1.csts" "soubor_pro_em2.csts" -ln
-em 2 -en 4 -ed "/home/wombat/adr_pro_em1" "/home/wombat/adr_pro_em1" -ln
   "HMM_em2_h4.log"
```

Tagování

```
-tg -n 3 -i "soubor1.csts" "soubor2.csts" -o "vystup1.csts"
   "vystup2.csts" -ln "HMM_tag_h3.log" -ll 3
```

```
-tg -x -n 2 -di "/home/wombat/adr1" "/home/wombat/adr2" -do
   "/home/wombat/vystup_adr1" "/home/wombat/vystup_adr2" -dc ".tagged"
   -ln "HMM_tag_h2.log" -ll 1
```

Tagování s EM algoritmem

```
-tg -n 3 -i "soubor1.csts" "soubor2.csts" -o "vystup1.csts"
   "vystup2.csts" -em -ei "em_soubor1.csts" -ln "HMM_tag_em_h3.log" -ll 1
-tg -x -n 2 -di "/home/wombat/adr1" "/home/wombat/adr2" -do
   "/home/wombat/vystup_adr1" "/home/wombat/vystup_adr2" -dc ".tagged"
   -em 1 -en 2 -ed "em_adr1.csts" "em_adr2.csts" -ei "em_soubor1.csts"
```

Testování

```
-ts -i "soubor1.csts" "soubor2.csts -ln "HMM_test.log"
-ts -di "/home/wombat/adr1" "/home/wombat/adr2" -ln -ll 1
```

Makro

```
-m "makro_soubor.csts"
```

Nápověda

```
-h
```

3.1.11 Vzorový běh programu

V předchozí kapitole jsou ukázány příklady jednotlivých běhů programu v různých módech. Pro lepší pochopení práce programu zde uvádím i sekvence příkazů, které lze použít jako vzorový běh programu (až na případné odchylky v adresářových cestách).

Tagování s historií délky 3

```
-tr -x -n 3 -ln "/home/user1/logs/Train_p1.log" -ll 3 -di "data/train-1/"
-em -en 3 -ln "/home/user1/logs/Heldout.log" -ll 3 -ed "data/train-8/"
-tr -n 3 -ln "/home/user1/logs/Train_p2.log" -ll 3 -di "data/train-8/"
-tg -n 3 -ln "/home/user1/logs/Tag.log" -ll 3 -di "data/morph/etest/" -do
   "data_tagged/etest/" -dc ".tagged"
-ts -ln "/home/user1/logs/Test.log" -ll 3 -di "data_tagged/etest/"
```

Tagování s historií délky 2 ve verzi m-best, kde m = 5

```
-tr -x -n 2 -ln "/home/user1/logs/Train_p1.log" -ll 3 -di "data/train-1/"
-em -en 2 -ln "/home/user1/logs/Heldout.log" -ll 3 -ed "data/train-8/"
-tr -n 2 -ln "/home/user1/logs/Train_p2.log" -ll 3 -di "data/train-8/"
-tg -n 2 -b 5 -ln "/home/user1/logs/Tag_b5.log" -ll 3 -di
   "data/morph/etest/" -do "data_tagged/etest/" -dc ".tagged"
-ts -ln "/home/user1/logs/Test_b5.log" -ll 3 -di "data_tagged/etest/"
```

Tagování s historií délky 3 se speciální historií používající prvních 5 znaků z každého tagu

```
-tr -x -n 3 -sp (7600;7600;7600) -ln "/home/user1/logs/Train_p1_sp.log"
   -ll 3 -di "/home/user1/data/train-1/"
-em -en 3 -sp (7600;7600;7600) -ln "/home/user1/logs/Heldout_sp.log" -ll
   3 -ed "/home/user1/data/train-8/"
-tr -n 3 -sp (7600;7600;7600) -ln "/home/user1/logs/Train_p2_sp.log" -ll 3
   -di "/home/user1/data/train-8/"
-tg -n 3 -sp (7600;7600;7600) -ln "/home/user1/logs/Tag_sp.log" -ll 3 -di
   "/home/user1/data/morph/etest/" -do "/home/user1/data_tagged/etest/"
   -dc ".tagged"
-ts -sp (7600) -ln "/home/user1/logs/Test_sp.log" -ll 3 -di
   "/home/user1/data_tagged/etest/"
```

3.2 Programátorská dokumentace

Jak již bylo zmíněno výše, program je implementován v jazyku Java 5.0 a je dělen do několika souborů. Při tvorbě byl kladen důraz na variabilitu parametrů HMM Taggeru a případnou rozšiřitelnost programu v budoucnu.

Program využívá nových vlastností jazyku Java 5.0, zejména pak generických typů. Ty jsou použity jako základ datových struktur, do kterých si program ukládá natrénovaná data.

V následujícím textu se nachází pouze stručný popis některých tříd z hlediska jejich účelu. Detailnější popis tříd lze nalézt v JavaDoc dokumentaci, která se nachází na CD přiloženém k této práci.

3.2.1 Datové soubory

Program vyžaduje datové soubory ve formátu CSTS. K jejich zpracování používá implementaci interface *IAbstractDataFile*, konkrétně třídu *CDataFile_CSTS*.

Interface IAbstractDataFile, který obsahuje rozhraní ke čtení souborů a k zapisování do souborů. Je navržen tak, aby se v budoucnu daly snadno napsat implementace i pro jiné datové soubory.

Program navíc obsahuje třídu *CDataFileElement*, která je určena k práci s jednotlivými tokeny v datech. Tato třída má položky pro jednotlivé elementy (tagy) a je primárně navržena pro práci se soubory CSTS, takže by se při eventuelním přidání dalších formátů souborů mohlo ukázat, že nemusí být úplně vhodná (toto ale zatím nenastalo).

Datové struktury popsané v další části jsou nezávislé na formátu vstupních souborů. Stejně tak jsou na formátu souborů nezávislé hlavní algoritmy v programu - tagovací Viterbiho algoritmus, jeho verze m-best i EM algoritmus.

3.2.2 Datové struktury

Trénování HMM Taggeru obnáší vytvoření informací o četnostech tagů (jak bylo zmíněno v sekci 2.7). Je tedy potřeba mít v programu datové struktury schopné efektivně ukládat četnosti n-tic tagů (n-gramů).

Pro datové struktury je použita stromovitá hashovací struktura založená na kolekci *java.util.HashMap*. Rozhraní struktury je obsažené v interface *IProbabilityStruct*. Třída *CProbabilityStruct* implementuje zmíněný interface pomocí generických typů.

Stromovitá hashovací struktura byla zvolena z důvodu rychlého vkládání a hledání prvků. Její uzly jsou tvořeny hashovacími tabulkami (*java.util.HashMap*), které mapují tagy na objekty typu *CProbabilityStructItem*. Tyto objekty obsahují jednak četnosti sekvencí tagů vedoucích do těchto uzlů, dále pak odkaz na další strukturu typu *CProbabilityStruct*, která tvoří podstrom začínající v daném uzlu.

Tabulky jsou postavené na generických typech, v programu jsou potom používány instance typu String (tagy v datových souborech jsou řetězce o délce 15 znaků).

V programu se používají 3 struktury - jedna pro přechodové pravděpodobnosti, druhá pro emisní pravděpodobnosti a třetí pro pravděpodobnosti slov.

Struktura pro přechodové pravděpodobnosti má k vrstev (kde k je délka trénovací historie). V této struktuře se ukládají k-tice tagů (k-gramy). V první úrovni se ukládají aktuální tagy. Následně se přejde do další úrovně pod vloženým tagem, kam se vloží předchozí tag z k-gramu. Stejně se postupuje při ukládání celého k-gramu[4]. Jak vyplývá z tohoto popisu v j-té vrstvě[5] ($0 < j \leq k$) si struktura pamatuje počty viděných sekvencí délky j.

Stromová struktura není vytvářena celá pro všechny tagy, její větve jsou vytvářeny až v momentě, kdy jsou potřeba.

Struktura pro výstupní pravděpodobnosti má dvě vrstvy. V první se nacházejí emitující tagy a ve druhé emitovaná slova.

Struktura pro pravděpodobnosti slov je jednovrstvá. Zde by se dalo případně použít i obyčejné mapování pomocí *java.util.HashMap*.

Pro potřebu uložení struktur do souborů implementují třídy *CProbabilityStruct* a *CProbabilityStructItem* interface *java.io.Serializable*.

V programu se vyskytuje ještě jedna sada těchto tří struktur používaná v EM algoritmu. Tyto struktury se ovšem neserializují, jsou pouze jednorázově použity v EM agloritmu.

3.2.3 Trénování a tagování

Program si do struktury pro přechodové pravděpodobnosti ukládá počty sekvencí tagů. Sledují se sekvence tagů ve větách. Na počátcích vět je použito falešných tagů (zmíněno v sekci 2.3).

Tagování pomocí skrytých Markovových modelů je implementováno ve třídě *CTagging*. Pravděpodobnosti jsou ukládány v proměnných typu double. To má za následek potenciální možnost vzniku podtečení double (double underflow). Řešení se zde nabízela dvě: vytvoření vlastní třídy zpracovávající aritmetiku nebo úpravu pravděpodobností celé hladiny v trellis. První řešení

[4]Ukládání n-gramu směrem od aktuálního tagu je výhodnější, neboť jsou k výpočtu lineární interpolace potřeba četnosti všech k-gramů, kde $1 \leq k \leq n$. Pro výpočet lineární interpolace přechodové pravděpodobnosti jsou potom zapotřebí teoreticky jen 2 přístupy do stuktury četností n-gramů.

[5]Vrstvou v dané struktuře je myšlena množina uzlů, do kterých vedou stejně dlouhé cesty od kořene.

by bylo přesnější ale pomalejší. Proto jsem použil úpravu celé hladiny v trellis. Pokud pravděpodobnost nějakého stavu v trellis klesne pod určitou mez, potom se pravděpodobnosti všech stavů v další hladině vynásobí konstantou, a tím se pravděpodobnosti vrátí do příslušných mezí.

3.2.4 Zdrojové soubory

Program se skládá z více než dvacítky zdrojových souborů. Seznam souborů a popis jejich obsahu se nachází v následující tabulce.

Main.java	funkce main, řízení programu, spojuje všechny ostatní soubory dohromady
CParameterAnalyzer.java	parsování příkazové řádky a řádek z makro souboru
Subsidiary.java	různé pomocné třídy, třída pro statistiky (*HMMStats*)
CTraining.java	trénování, výpis trénovacích logů
CTagging	tagování, jednoduchý Viterbiho algoritmus i jeho varianta pro verzi m-best, výpis tagovacích logů
CTesting	testování, výpis testovacích logů
CHeldout	EM algoritmus, výpis logů EM algoritmu
CProbabilityStruct.java	třída datových struktur, které jsou trénovány
IProbabilityStruct.java	interface datových struktur
CProbabilityStructItem.java	pomocná třída datových struktur
CHMM_Probabilities.java	výpočet přechodových a výstupních pravděpodobností a výpočet jejich lineárních interpolací
CHMM_Statistics.java	pomocné třídy pro položky, jejichž statistiky jsou počítány
CHMM_Logging.java	třída zpracovávající logy
IAbstractN_Gram.java	interface n-gramu
CN_Gram.java	implementace interface *IAbstractN_Gram* n-gramu
DataManip.java	pomocné třídy pro serializaci a deserializaci natrénovaných dat

CSpecialHistory.java	třída pomocných funkcí používaných pro úpravu tagů speciální historie
CTrellisElementNBest.java	element trellis používaný ve Viterbiho algoritmu verze m-best
CTrellisElement.java	část elementu trellis používaná ve Viterbiho algoritmu verze m-best
IAbstractDataFile.java	interface pro čtení z datových souborů a zápis do nich
CDataFile_CSTS.java	implementace interface *IAbstractDataFile* pro datové soubory typu CSTS
CDataFileElement.java	element načítaný z datových souborů
_C_CSTS_Line.java	řádek datového souboru typu CSTS

3.2.5 Makro

Běh programu v makro režimu je rychlejší z důvodu vynechání serializace a deserializace natrénovaných dat. Program běží ve vnitřní smyčce dokud nezpracuje všechny příkazy v makro souboru. Serializace či deserializace proběhne jedině na začátku nebo na konci celé sekvence příkazů zadaných v makro souboru.

3.2.6 Serializovaná data

Serializovaná data jsou ukládána do čtyř souborů, které jsou vytvářeny v adresáři s programem. Struktura obsahující přechodové pravděpodobnosti je ukládána v souboru TransferProb.dat, struktura obsahující výstupní pravděpodobnosti v souboru EmissionProb.dat a struktura obsahujicí pravděpodobnosti slov v souboru WordProb.dat. V souboru OtherData.dat jsou ukládány další potřebné informace jako trénovací historie, koeficienty pro lineární interpolaci a parametry speciální historie.

3.3 Zjištěné nedostatky

Největším nedostatkem programu je jeho relativně pomalý běh, jenž může být způsoben několika faktory. Prvním je například použití jazyka Java, který je překládán do bytekódu. Ten je následně interpretován. Navzdory just-in-time kompilaci zůstává Java pomalejší než jazyky, které jsou rovnou překládány do nativního kódu.

Druhým důvodem pomalého běhu aplikace je již zmíněná orientace na variabilitu programu, a tím i použití konstrukcí vyžadujících více instrukcí. Třetím problémem mohou být různé skryté neefektivnosti ve zdrojovém kódu, které se nevyhnou téměř žádnému programu.

Kapitola 4

Výsledky programu

Testování programu probíhalo na datech z Pražského závislostního korpusu verze 2.0 (Prague Dependency Treebank 2.0, [6]). Data před použitím prošla morfologickým analyzátorem ([6]), který přidal ke každému tokenu tagy <MMl> a <MMt>, které výrazně omezují množinu počítaných tagů v každé úrovni trellis.

Tagy obsažené v PDT 2.0 mají 15 znaků, jejichž význam je v následující tabulce.

Pozice	Název	Popis
1	POS	Slovní druh
2	SUBPOS	Detailní určení slovního druhu
3	GENDER	Jmenný rod
4	NUMBER	Číslo
5	CASE	Pád
6	POSSGENDER	Přivlastňovací rod
7	POSSNUMBER	Přivlastňovací číslo
8	PERSON	Osoba
9	TENSE	Čas
10	GRADE	Stupeň
11	NEGATION	Negace
12	VOICE	Aktivum/pasívum
13	RESERVE1	Nepoužito
14	RESERVE2	Nepoužito
15	VAR	Varianta, stylový příznak apod.

Testování probíhalo jednak na dtest datech, která jsou používána pro ladění programů, jednak na etest datech, která jsou používána pro konečné testování taggerů.

Hlavním cílem testů bylo ověřit rozdíl úspěšnosti při různých nastaveních parametrů programu, nikoliv překonat jiné taggery. Proto jsou v testech často používána dtest data.

4.1 Porovnání s jinými taggery

Pro porovnání úspěšnosti zde prezentovaného HMM Taggeru jsem použil data I (viz. sekce A.1), na kterých byly testovány i ostatní zde zmíněné taggery. Oficiálně se jedná o dtest data, přesto jsou to ale jiná data, než na kterých byl vytvořený program laděn, z tohoto pohledu jsou to pro něj vlastně etest data. Výsledky na etest datech jiných taggerů pro lepší srovnání se nepodařilo získat.

Ostatními taggery jsou Morče[1], Feature-based a jiný (vylepšený) HMM (viz. [4]). Prezentovaný tagger nedosahuje takových výsledků jako 3 zde zmíněné taggery, přesto jeho výsledky nebyly o tolik horší. Nastavení taggeru bylo následující: historie délky 3, EM algoritmus byl použit pro výpočet koeficientů lineární interpolace přechodových i výstupních pravděpodobností, speciální historie nebyla použita. Podrobněji jsou výsledky popsány v následující tabulce.

Tagger	accuracy
HMM (cizí - vylepšený)	95.13 %
Feature-based	94.27 %
Morče	95.43 %
HMM (zde prezentovaný)	93.83 %

[1] Jméno Morče je zkratkou z názvu MORfologie ČEštiny.

4.2 Další testování

Pro testování programu bylo použita data A až I (viz. sekce A.1). U některých z těchto dat byla held-out data použita k dodatečnému dotrénování datových struktur (viz. tabulka informací o datech v sekci A.1).

Byly provedeny různé testy, ověřující vliv velikosti tagovací historie (tabulky v sekci A.2), varianty m-best (tabulky v sekci A.4) a speciální historie (tabulky v sekci A.3) na úspěšnost tagování. Testy byly prováděny na 5 strojích, které jsou popsány v tabulce. Velikost heapu pro 32-bitové stroje a na jednom 64-bitovém (stroj 4) šla nastavit pouze do 2 GB. Zbylý 64-bitový stroj (stroj 5) byl použit k otestování úspěšnosti tagování s historií délky 5 (nutnost velkého množství operační paměti). Veškeré tabulky s dosaženými výsledky jsou umístěny v přílohách v kapitole A.

Stroj	Architektura	Operační paměť	Počet CPU	Frekvence CPU
1	32	1264 MB	1	1.4 GHz
2	32	4 GB	4	2.4 GHz
3	32	4 GB	4	2.4 GHz
4	64	16 GB	8	1.5 GHz
5	64	32 GB	4	2.2 GHz

4.3 Dosažené výsledky

Výsledky programu ukázaly, že pro tagování je nejvhodnější historie délky 3. Použití kratší historie dosahuje výrazně horších výsledků. Delší historie zase nepřináší větší zlepšení z důvodu řídkosti trénovacích dat (data sparseness) (viz. tabulky v sekci A.2.1), naopak přináší výrazný nárůst časové složitosti (viz. tabulka v sekci A.2.3).

Odhlédneme-li od výsledků při použití speciální historie a verze m-best, bylo nejlepších výsledků dosaženo při použití historie délky 4 (historii délky 5 zde neuvažuji z důvodu nedostatečného otestování) a EM algoritmu pro výpočet koeficientů jen přechodových pravděpodobností (viz. tabulky v sekci

A.2.1). Konkrétní nejlepší výsledek počítaný na etest datech (data B) za zmiňovaných parametrů je 92.859 %[2].

4.3.1 Velikost trénovacích dat

Při použití menších trénovacích dat a zachování jejich poměru ku tagovacím datům se více projevuje jejich řídkost (data sparseness). Výsledkem je nižší růst úspěšnosti při použití delší tagovací historie (viz. tabulky v sekci A.2.1. Při použití asi 200 000 trénovacích tagů a 25 000 tagů k označkování se úspěšnost při použití historie délky 4 pohybuje pouze kolem 90 % oproti více než 92 % při použití asi sedmkrát větších dat.

4.3.2 Speciální historie

Výrazné zlepšení úspěšnosti může přinést použití speciální historie. Při použití prvních dvou znaků z tagu lze dosáhnout úspěšnosti i více než 98 % (viz. tabulky v sekci A.3.1). Na takto tagovaných datech je při delší tagovací historii (3 a více) znát i přetrénování programu (viz. první sloupec v tabulce v sekci A.3.1), které má za následek propad úspěšnosti tagování. Použitím speciální historie bohužel ztrácíme velkou část informace, kterou v sobě tagy skrývají. Speciální historie by se dala použít v aplikacích, kde nejsou podstatné všechny informace obsažené v tazích.

4.3.3 Varianta m-best

Při testování úspěšnosti tagování ve variantě m-best testuje program precission, recall a f-measure [3] (viz. 2.8). Při zvyšujícím se n dochází ke snižování precission v důsledku zvyšování počtu přiřazených tagů (viz. tabulky v sekci A.4.1). Ačkoli je zvoleno $n \geq 1, n \in N$, není počet tagů přiřazených tokenům prakticky nikdy rovný n. Průměrný počet tagů na token vychází mírně větší než 1.

[2]Výsledek na datech I při použití historie délky 3 neuvažuji, neboť data I byla určena pouze k porovnání výsledků s jinými taggery. Program nebyl na těchto datech více testován a nejsou známy výsledky pro jiná nastavení.

[3]Pro $n = 1$ nabývají tyto veličiny stejných hodnot a v tomto případě mluvíme o tzv. accuracy.

Recall se zvyšuje i na více než 97 % pro $n = 20$. Ve výsledku ale dochází ke snižování F-measure, a to až k 75 % pro $n = 20$.

4.3.4 EM algoritmus

Program byl testován při různém použití EM algoritmu. Testovalo se současné použití pro přechodové i výstupní pravděpodobnosti, použití samostatně pro každý typ pravděpodobnosti a také použití uniformního rozdělení pro oba typy pravděpodobností (viz. tabulky v sekcích A.2.1 a A.4.1).

Podle předpokladů zvýšilo použití EM algoritmu pro přechodové pravděpodobnosti úspěšnost tagování. Překvapením bylo mírné zhoršení výsledků při jeho použití pro výstupní pravděpodobnosti. Dodatečným laděním nebyla zjištěna nějaká chyba v programu, naopak výsledné hodnoty lambd přibližně odpovídaly jiným výsledkům (viz. [3]). Možným vysvětlením by mohla být špatná velikost held-out dat v poměru k trénovacím datům.

Tabulky s vypočítanými lambda koeficienty přechodových a výstupních pravděpodobností pro různé délky historie a různá data jsou v sekci A.2.2. Tabulky s vypočítanými lambda koeficienty přechodových pravděpodobností pro různé délky historie a různé speciální historie jsou v sekci A.3.2.

Kapitola 5

Shrnutí

Účelem programu bylo otestovat vliv nastavení parametrů na úspěšnost tagování. Z tohoto důvodu bylo provedeno velké množství různých testů. Některé z nich proběhly na dtest datech pro ukázku vlivu změn parametrů na nastavení programu, jiné byly provedeny na etest datech pro korektní posouzení výsledků. Výsledky testů jsou shrnuty v sekci 4, podrobné výsledky jsou v podobě tabulek obsaženy v sekci A. Některá možná vylepšení, která by měla mít za následek zlepšení úspěšnosti programu, jsou obsažena v části 5.1.

Z důvodu velké časové náročnosti bohužel neproběhly všechny zamýšlené testy. Šlo hlavně o testy s historií délky 5 (z nich stačil proběhnout pouze jediný - viz. tabulka v sekci A.2.1, test na datech H).

Přínosem programu je možnost větší variability při volbě parametrů HMM Taggeru. Jediným omezením programu jsou paměťová a časová náročnost.

5.1 Náměty do budoucna

Jedním z možných rozšíření v budoucnu by mohlo být přidání bucketingu (popsáno např. v [5]), který dokáže zlepšit výsledky o desetiny procent.

Dalším vhodným rozšířením je výpočet výstupních pravděpodobností z bigramu místo z unigramu, který je v prezentovaném programu použit. Místo $P(w^l|t^j)$ se potom počítá $P(w^l|t^j, t^{j-1})$. Takto upravený výpočet výstupních pravděpodobností přináší další zlepšení (viz. [3], [4]).

Rozvinutím předchozí myšlenky do obecné podoby lze přejít od používání unigramů až ke k-gramům, kde $k > 1$. Tedy počítat $P(w^l|t^j, \ldots t^{j-k+1})$.

5.1.1 Implementační vylepšení

Velkým problémem prezentovaného programu je již zmíněná časová náročnost. Pro budoucí rozšíření by bylo vhodné optimalizovat implementaci Viterbiho algoritmu. Výpočet by se jistě dal vylepšit při použití speciální historie. V současné verzi jsou v trellis používány celé tagy, speciální historie je použita pouze pro výpočet přechodových a výstupních pravděpodobností ve Viterbiho algoritmu. Celé tagy jsou v trellis ponechány z důvodu volnosti při používání speciální historie (možnost použít libovolnou podmnožinu znaků ve všech tazích v historii). Zmíněný přístup by se dal vylepšit optimalizací založenou na omezení počtu stavů v trellis podle maximální podmnožiny znaků v tagu. Jako příklad uveďme speciální historii `(7000;7f00;7f60)`. V tomto případě stačí použít podmnožinu `(7f60)` pro znaky tagů v trellis, která se tímto může razantně zmenšit.

Z programu by se dala odstranit i spousta dalších (často skrytých) neefektivností. Například by nejspíše šlo nahradit některé proměnné, které jsou kolekcemi jazyka Java, za pole a tím urychlit běh programu.

Užitečnou vlastností, kterou by bylo vhodné doprogramovat je možnost tagování ve směru od konců vět k jejich začátkům.

Dodatek A

Tabulky

V této sekci se nacházejí výsledky experimentů shrnuté v tabulkách. Z důvodu časové náročnosti se nepodařilo provést veškeré zamýšlené testy.

A.1 Použitá data

Zde je prezentována velikost dat použitých k otestování programu. Jedná se o několik sad dat různé velikosti. Počty tagů, vět a zdrojových souborů jednotlivých dat jsou shrnuty v tabulce.

Data	**A**	**B**	**C**	**D**
Počet tokenů v trénovacích datech	1336944	1336944	568895	575668
Počet vět v trénovacích datech	79153	79153	33452	34231
Počet souborů s trénovacími daty	4992	4992	2139	2139
Počet tokenů v held-out datech	202297	202297	63061	49071
Počet vět v held-out datech	11896	11896	3778	2894
Počet souborů s held-out daty	713	713	257	164
Held-out data použita k dodatečnému dotrénování	ano	ano	ne	ne
Počet tokenů v tagovacích datech	201651	219765	77002	68139
Počet vět v tagovacích datech	11880	13136	4580	4003
Počet souborů s tagovacími daty	712	712	256	168
Typ tagovacích dat	dtest	etest	etest	etest

Data	**E**	**F**	**G**	**H**	**I**
Počet tokenů v trénovacích datech	192381	188297	193556	1336944	1393616
Počet vět v trénovacích datech	11470	11152	11509	79153	82277
Počet souborů s trénovacími daty	714	713	713	4992	1
Počet tokenů v held-out datech	21191	27107	23328	202297	145625
Počet vět v held-out datech	1295	1530	1362	11896	8772
Počet souborů s held-out daty	99	93	89	713	1
Held-out data použita k dodatečnému dotrénování	ne	ne	ne	ano	ano
Počet tokenů v tagovacích datech	24071	26983	24238	55924	201651
Počet vět v tagovacích datech	1411	1597	1378	3224	11880
Počet souborů s tagovacími daty	103	93	85	194	1
Typ tagovacích dat	etest	etest	etest	dtest	dtest

Data I slouží k porovnání úspěšnosti prezentovaného taggeru s jinými taggery. Jde o taggery zmíněné v sekci 4.1.

A.2 Tagovací historie a velikost dat

Závislost úspěšnosti tagování na délce historie a velikosti dat je shnuta v této sekci. Jsou zde navíc zapsány i vypočítané koeficienty lineární interpolace a přibližné časy nutné k tagování.

Výsledky testů jsou rozděleny do 4 tabulek podle využití EM algoritmu.

A.2.1 Úspěšnost tagování - accuracy

Použití EM algoritmu pro výpočet lambd pro přechodové i výstupní pravděpodobnosti:

Data	A	B	C	D	E
Historie 1	82.304 %	82.017 %	81.051 %	81.093 %	80.486 %
Historie 2	91.825 %	91.423 %	90.495 %	90.32 %	89.759 %
Historie 3	92.648 %	92.348 %	91.187 %	91.041 %	90.266 %
Historie 4	92.733 %	92.398 %	91.259 %	91.088 %	90.316 %

Data	F	G	H	I
Historie 1	80.776 %	80.2 %	82.253 %	-
Historie 2	89.615 %	88.909 %	91.89 %	-
Historie 3	90.123 %	89.454 %	92.699 %	93.83 %
Historie 4	90.141 %	89.574 %	92.806 %	-
Historie 5	-	-	92.831 %	-

Použití EM algoritmu pouze pro přechodové pravděpodobnosti:

Data	A	B	C	D	E
Historie 1	82.321 %	82.032 %	80.391 %	81.141 %	80.59 %
Historie 2	92.299 %	91.94 %	89.448 %	90.991 %	90.552 %
Historie 3	93.122 %	92.815 %	90.979 %	91.608 %	90.964 %
Historie 4	93.203 %	92.859 %	91.224 %	91.72 %	91.022 %

Data	F	G
Historie 1	80.843 %	80.245 %
Historie 2	90.408 %	89.805 %
Historie 3	90.879 %	90.242 %
Historie 4	90.886 %	90.312 %

Použití EM algoritmu pouze pro výstupní pravděpodobnosti:

Data	A	B	C	D	E
Historie 1	81.66 %	81.35 %	80.352 %	80.494 %	79.884 %
Historie 2	90.08 %	89.755 %	88.58 %	88.499 %	87.752 %
Historie 3	91.512 %	91.155 %	89.971 %	89.829 %	88.941 %
Historie 4	91.98 %	91.616 %	90.306 %	90.315 %	89.464 %

Data	F	G
Historie 1	80.257 %	79.325 %
Historie 2	87.766 %	87.383 %
Historie 3	88.918 %	88.398 %
Historie 4	89.295 %	88.856 %

Použití uniformního rozdělení:

Data	A	B	C	D	E	F	G
Historie 1	81.683 %	81.361 %	-	-	-	-	-
Historie 2	90.804 %	90.404 %	-	-	-	-	-
Historie 3	92.325 %	91.945 %	-	-	-	-	-
Historie 4	92.69 %	-	-	-	-	-	-

A.2.2 Koeficienty lineární interpolace

Koefiecienty lineární interpolace výstupních pravděpodobností:

	λ_0	λ_1	λ_2
A,B,H	0.0882	0.0001	0.9117
C	0.1267	0.0002	0.8731
D	0.1204	0.0004	0.8792
E	0.2074	0.0002	0.7924
F	0.1934	0.0003	0.8063
G	0.1985	0.0002	0.8013
I	0.0863	0.0001	0.9136

Koeficienty lineární interpolace přechodových pravděpodobností:

Historie 1	λ_0	λ_1
A,B,H	0.0028	0.9972
C	0.0035	0.9965
D	0.0056	0.9944
E	0.0085	0.9915
F	0.0078	0.9922
G	0.0079	0.9921

Historie 2	λ_0	λ_1	λ_2
A,B,H	0.0052	0.0457	0.9491
C	0.0069	0.0653	0.9278
D	0.0094	0.0721	0.9185
E	0.0149	0.1041	0.8810
F	0.0167	0.1022	0.8811
G	0.0158	0.1083	0.8759

Historie 3	λ_0	λ_1	λ_2	λ_3
A,B,H	0.0060	0.0427	0.4539	0.4974
C	0.0079	0.0615	0.5168	0.4138
D	0.0105	0.0670	0.5484	0.3741
E	0.0158	0.0995	0.6175	0.2672
F	0.0178	0.0970	0.6310	0.2542
G	0.0166	0.1042	0.6043	0.2749
I	0.0064	0.0388	0.4543	0.5005

Historie 4	λ_0	λ_1	λ_2	λ_3	λ_4
A,B,H	0.0062	0.0410	0.4624	0.4019	0.0885
C	0.0080	0.0603	0.5240	0.3586	0.0491
D	0.0106	0.0663	0.5542	0.3264	0.0425
E	0.0157	0.0989	0.6209	0.2415	0.0230
F	0.0178	0.0967	0.6340	0.2358	0.0157
G	0.0166	0.1038	0.6076	0.2429	0.0291

Historie 5	λ_0	λ_1	λ_2	λ_3	λ_4	λ_5
H	0.0062	0.0410	0.4629	0.4023	0.0802	0.0074

A.2.3 Rychlost tagování (časová náročnost)

Následující hodnoty časové náročnosti programu jsou pouze orientační, skutečné výsledky se mohou lišit v závislosti na daném hardware počítače a momentálním zatížení CPU. Časová náročnost závisí hlavně na délce tagovací historie, vliv ostatních parametrů je marginální. Při stejných vstupních

parametrech bylo dosaženo podobných časů na všech testovacích strojích (viz. sekce 4.2). Pro výsledky zde prezentované byla použita maximální velikost heapu JVM 2 GB pro historii kratší než 5 (pro historii menší či rovnou 3 lze vystačit s 512 MB), 24 GB pro historii rovnou 5. Testováno na datech A.

Historie	1	2	3	4	5
Časová náročnost	30 s	2 min	40-60 min	40-70 hod	desítky dnů

A.3 Speciální historie

Testování účinků speciální historie na úspěšnost tagování probíhalo na datech B. EM algoritmus byl použit pouze pro určení koeficientů přechodových pravděpodobností.

V této sekci jsou v tabulkách shrnuty výsledky programu při použití speciální historie a hodnoty koeficientů pro lineární interpolaci přechodových pravděpodobností vypočítané v EM algoritmu.

A.3.1 Úspěšnost tagování - accuracy

Speciální historie	(6000)	(7600)	(7f00)	(7f60)
Historie 1	97.89 %	82.38 %	82.0 %	81.995 %
Historie 2	98.057 %	92.147 %	92.178 %	92.138 %
Historie 3	97.933 %	93.111 %	93.069 %	93.011 %
Historie 4	97.657 %	93.161 %	93.174 %	93.1 %

A.3.2 Koeficienty lineární interpolace

Přechodové pravděpodobnosti pro různé délky historie při použití různých typů speciální historie.

Historie 1	λ_0	λ_1
(6000)	0.0003	0.9997
(7600)	0.0002	0.9998
(7f00)	0.0005	0.9995
(7f60)	0.0008	0.9992

Historie 2	λ_0	λ_1	λ_2
(6000)	0	0.4758	0.5242
(7600)	0.0003	0.0044	0.9953
(7f00)	0.0010	0.0080	0.9910
(7f60)	0.0016	0.0128	0.9856

Historie 3	λ_0	λ_1	λ_2	λ_3
(6000)	0	0.0005	0.0233	0.9762
(7600)	0.0010	0.0033	0.1567	0.8390
(7f00)	0.0024	0.0108	0.2715	0.7153
(7f60)	0.0033	0.0169	0.3193	0.6605

Historie 4	λ_0	λ_1	λ_2	λ_3	λ_4
(6000)	0	0	0.0012	0.0546	0.9442
(7600)	0.0019	0.0071	0.2226	0.4952	0.2732
(7f00)	0.0035	0.0191	0.3408	0.4820	0.1546
(7f60)	0.0046	0.0238	0.3746	0.4636	0.1334

A.4 Verze m-best

V této sekci jsou shrnuty výsledky tagování ve variantě m-best Viterbiho algoritmu v závislosti na m, délce tagovací historie a použití EM algoritmu. Za m byly zvoleny hodnoty 1, 2, 3, 5, 10 a 20.

Koeficienty lineární interpolace jsou shrnuty v sekci A.2.2.

A.4.1 Úspěšnost tagování - precision, recall, F-measure

Úspěšnost tagování je rozdělena do tabulek podle délky tagovací historie a použití EM algoritmu. Testy proběhly pro délku historie od 1 do 3, přičemž testy historie délky 3 neproběhly všechny.

V závěru této sekce jsou prezentovány i výsledky, které vznikly experimentováním s EM algoritmem pro výstupní pravděpodobnosti[1].

Historie 1

Použita byla data B a EM algoritmus pro přechodové i výstupní pravděpodobnosti.

m	1	2	3	5	10	20
Precision	82.017 %	79.134 %	76.408 %	72.97 %	68.132 %	62.944 %
Recall	82.017 %	83.724 %	85.164 %	86.698 %	88.5 %	90.15 %
F-measure	82.017 %	81.364 %	80.548 %	79.244 %	76.991 %	74.13%
Tag/token	1	1.058	1.1145	1.1881	1.2989	1.4322

[1] V tabulkách v celé cekci A lze pozorovat větší úspěšnost při vynechání EM algoritmu pro výpočet výstupních pravděpodobností.

Použita byla data B a EM algoritmus pro přechodové pravděpodobnosti.

m	1	2	3	5	10	20
Precision	82.032 %	79.148 %	76.415 %	72.956 %	68.016 %	62.675 %
Recall	82.032 %	83.74 %	85.172 %	86.729 %	88.558 %	90.22 %
F-measure	82.032 %	81.379 %	80.556 %	79.248 %	76.939 %	73.966%
Tag/token	1	1.058	1.1145	1.1887	1.302	1.4394

Použita byla data B a EM algoritmus pro výstupní pravděpodobnosti.

m	1	2	3	5	10	20
Precision	81.35 %	78.486 %	75.738 %	72.219 %	67.119 %	61.772 %
Recall	81.35 %	83.04 %	84.417 %	85.944 %	87.717 %	89.383 %
F-measure	81.35 %	80.698 %	79.842 %	78.485 %	76.047 %	73.055%
Tag/token	1	1.058	1.1145	1.19	1.3068	1.4469

Použita byla data B a uniformní rozdělení koeficientů pro lineární interpolaci.

m	1	2	3	5	10	20
Precision	81.361 %	78.51 %	75.776 %	72.219 %	67.037 %	61.555 %
Recall	81.361 %	83.065 %	84.46 %	85.968 %	87.757 %	89.436 %
F-measure	81.361 %	80.723 %	79.882 %	78.495 %	76.01 %	72.922%
Tag/token	1	1.058	1.1145	1.1903	1.309	1.4529

Historie 2

Použita byla data B a EM algoritmus pro přechodové i výstupní pravděpodobnosti.

m	1	2	3	5	10	20
Precision	91.423 %	87.122 %	83.195 %	78.027 %	70.491 %	62.671 %
Recall	91.423 %	93.201 %	94.252 %	95.173 %	96.051 %	96.7 %
F-measure	91.423 %	90.059 %	88.379 %	85.751 %	81.309 %	76.052%
Tag/token	1	1.0697	1.1328	1.2197	1.3625	1.5429

Použita byla data B a EM algoritmus pro přechodové pravděpodobnosti.

m	1	2	3	5	10	20
Precision	91.94 %	87.524 %	83.512 %	78.139 %	70.172 %	61.997 %
Recall	91.94 %	93.714 %	94.779 %	95.69 %	96.512 %	97.126 %
F-measure	91.94 %	90.513 %	88.789 %	86.028 %	81.26 %	75.683%
Tag/token	1	1.0707	1.1349	1.2246	1.3753	1.5666

Použita byla data B a EM algoritmus pro výstupní pravděpodobnosti.

m	1	2	3	5	10	20
Precision	89.755 %	85.744 %	82.041 %	77.107 %	69.962 %	62.55 %
Recall	89.755 %	91.484 %	92.634 %	93.641 %	94.743 %	95.563 %
F-measure	89.755 %	88.521 %	87.016 %	84.573 %	80.488 %	75.61%
Tag/token	1	1.0669	1.1291	1.2144	1.3542	1.5277

Použita byla data B a uniformní rozdělení koeficientů pro lineární interpolaci.

m	1	2	3	5	10	20
Precision	90.404 %	86.338 %	82.573 %	77.501 %	70.017 %	62.136 %
Recall	90.404 %	92.2 %	93.374 %	94.407 %	95.468 %	96.27 %
F-measure	90.404 %	89.172 %	87.641 %	85.122 %	80.785 %	75.525%
Tag/token	1	1.0678	1.1307	1.2181	1.3634	1.5493

Historie 3

Použita byla data B a EM algoritmus pro přechodové i výstupní pravděpodobnosti.

m	1	2	3	5	10	20
Precision	92.348 %	87.748 %	83.646 %	78.246 %	70.288 %	62.031 %
Recall	92.348 %	94.045 %	95.017 %	95.828 %	96.587 %	97.107 %
F-measure	92.348 %	90.787 %	88.969 %	86.149 %	81.365 %	75.703%
Tag/token	1	1.0717	1.1359	1.2247	1.3741	1.5654

Použita byla data B a EM algoritmus pro přechodové pravděpodobnosti.

m	1	2	3	5	10	20
Precision	92.815 %	88.101 %	83.904 %	78.267 %	69.947 %	-
Recall	92.815 %	94.558 %	95.497 %	96.276 %	96.989 %	-
F-measure	92.815 %	91.215 %	89.325 %	86.342 %	81.277 %	-
Tag/token	1	1.0732	1.1381	1.2301	1.3866	-

Další výsledky

Následující výsledky vznikly při testování úspěšnosti tagování ve verzi m-best na datech A při použití historie délky 3. EM algoritmus byl pozměněn vynecháním relativních četností k-gramů v held-out datech při výpočtu c_i (viz. sekce 2.6) pro výstupní pravděpodobnosti. Výpočet lambda proběhl pro přechodové i výstupní pravděpodobnosti.

m	1	2	3	5	10
Precision	92.951 %	88.382 %	84.247 %	78.793 %	70.676 %
Recall	92.951 %	94.7 %	95.623 %	96.375 %	97.08 %
F-measure	92.951 %	91.431 %	89.575 %	86.701 %	81.8 %
Průměrný počet tagů na token	1	1.071	1.135	1.223	1.374

Dodatek B

Kompaktní disk

Přílohou této práce je i kompaktní disk obsahující prezentovaný program, JavaDoc dokumentaci a testovací data.

Literatura

[1] Christopher D. Manning, Hinrich Schutze: *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge, Massachusetts, 1999. ISBN 0-26213-360-1.

[2] Jiří Mírovský: *Morfologické značkování textu: automatická disambiguace*, Master's thesis, ÚFAL, Faculty of Mathematics and Physics, Charles University, Prague, 1998.

[3] Jan Hajič, Pavel Krbec, Pavel Květoň, Karel Oliva, Vladimír Petkevič: *Serial Combination of Rules and Statistics: A Case Study in Czech Tagging*, In Proceedings of the ACL, 2001

[4] Drahomíra Spoustová, Jan Hajič, Jan Votrubec, Pavel Krbec, Pavel Květoň: *The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech.*

[5] Jan Hajič: Introduction to Natural Language Processing, http://www.cs.jhu.edu/∼hajic/courses/cs465/cs46506/ppframe.htm

[6] Hajič J. et al.: Prague Dependency Treebank 2.0, CD-ROM, LDC2006T01, LDC, Philadelphia, 2006