Ročníkový projekt - specifikace

HMM Tagger

Autor: Tomáš Kypta
Vedoucí projektu: Mgr. Jiří Mírovský

Programovací jazyk: Java 5.0
Platforma: Java-kompatibilní

1. Motivace

Jedním ze základních cílů zpracování přirozeného jazyka je rozklad a pochopení jazyka. Existuje mnoho různých metod, které k tomu slouží. Jednou z nich je morfologické tagování. Zde jde o označení každého slova ve větě vhodnou morfologickou značkou.

2. Zadání

Implementace obecného skrytého Markovova modelu a Viterbiho algoritmu.
Následné použití na morfologickou disambiguaci textu s volitelnou délkou historie (řešeno pomocí zvyšování počtu stavů, jeden stav reprezentuje více značek). Přechodové a výstupní pravděpodobnosti budou trénovány na ručně anotovaných datech, s použitím vyhlazování, např. linearní interpolace; v případě vyhlazování pomocí linearní interpolace budou koeficienty trénovány pomocí EM-algoritmu.
Testovací data budou předzpracována pomocí existujícího programu pro morfologickou analýzu.

3. Úvod do problematiky

Často se chceme zabývat sekvencí (např. v čase) náhodných proměnných, které nejsou nezávislé, ale hodnota každé proměnné závisí na předchozích prvcích v sekvenci. Pro spoustu takových systémů je důvod předpokládat, že vše, co potřebujeme k předpovězení budoucí proměnné je hodnota současné proměnné a ne hodnoty všech minulých proměnných v sekvenci. Tedy budoucí prvky v sekvenci jsou podmínečně nezávislé na několika minulých prvcích.
Tak je tomu i v tomto projektu, metoda, která bude dále popsána, je založena na statistickém zpracování závislostí morfologických značek slov na značkách slov předchozích.

Tento systém bude tedy zalozen na 2 vlastnostech (obecně nazývany Markovovy vlastnosti):

1. **Omezený rozhled**

$$P(X_{t+1} = s_k \mid X_1,...,X_t) = P(X_{t+1} = s_k \mid X_t)$$

2. **Neměnnost v čase**

$$= P(X_2 = s_k \mid X_1)$$

X nazýváme Markovův řetězec. Markovův řetězec můžeme popsat maticí přechodových pravděpodobností A:

$$a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i)$$

Zde $a_{ij} \geq 0$, pro každé i,j a současně $\sum_{j=i}^{N} a_{ij} = 1$ pro každé i.
Potřebujeme také specifikovat Π - pravděpodobnosti různých počátečních stavů Markovova řetězce:

$$\pi_i = P(X_1 = s_i)$$

Zde platí $\sum_{j=i}^{N} \pi_i = 1$.

3.1. HMM

Nástrojem zde budou Hidden Markov Models (skyté Markovovy modely, zkráceně HMM). Je to pravděpodobnostní funkce Markovova procesu. Jde o obecný statistický nástroj operující na vyšším stupni abstrakce pomocí doplňkové "skryté" struktury.

HMM je pětice (S, K, Π, A, B), kde:

$S = \{s_1,...,s_N\}$	množina stavů
$K = \{k_1,...,k_M\} = \{1,...,M\}$	výstupní abeceda
$\Pi = \{\pi_i\}$, i z S	počáteční stavové pravděpodobnosti
$A = \{a_{ij}\}$, i,j z S	přechodové pravděpodobnosti
$B = \{b_{ijk}\}$, i,j z S, k z K	emisní pravděpodobnosti
$X = (X_1,....,X_{T+1})$ X_t: S → {1,...,N}	sekvence stavů
$O = (o_1,...,o_T)$ o_t z K	výstupní sekvence

3.1.1. Tři základní otázky pro HMM:

1. Mějme dán model μ = (A,B,Π). Jak efektivně spočítat, jak pravděpodobný je sledovaný výstup (tedy P(O|μ))?
2. Mějme dánu výstupní sekvenci O a model μ. Jak zvolíme sekvenci stavů $(X_1,...,X_{T+1})$, která nejlépe vysvětluje výstupní sekvenci?
3. Mějme dánu výstupní sekvenci O a prostor možných modelů založený na modifikaci parametrů μ = (A,B,π). Jak nalezneme model, který nejlépe vysvětluje sledovaný výstup?

V tomto projektu se budu zabývat pouze 2. otázkou.

3.2. Označení:

w_i	slovo na pozici i v korpusu
t_i	tag slova i
$w_{i,i+m}$	slova na pozicích i až i+m
$t_{i,i+m}$	tagy ti až ti+m slov wi až wi+m
w^l	l-té slovo ve slovníku
t^j	j-tý tag ve množině tagů
$C(w^l)$	počet výskytů slova wl v trénovacích datech
$C(t^j)$	počet výskytů tagu tj v trénovacích datech
$C(t^j,t^k)$	počet výskytů tagu tj následovaném tagem tk
$C(w^l:t^j)$	počet výskytů slova wl otagovaného jako tj

T	počet tagů v množině tagů
W	počet slov ve slovníku
n	délka věty

Úkolem je tedy najít nejpravděpodobnější sekvenci tagů pro určitou sekvenci slov, neboli nejpravděpodobnější sekvenci stavů pro určitou sekvenci slov. Slova zde zapojíme díky tomu, že Markovův model emituje slova pokaždé, když opouští stav.
Tedy

$$P(O_n = k \mid X_n = s_i, X_{n+1} = s_j) = b_{ijk}$$

3.3. MLE

Přechodové a emisní pravděpodobnosti lze jednoše vypočítat pomocí MLE (maximum likehood estimation):

Přechodová pravděpodobnost pro tag t^k, který následuje tag t^j je:

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

Emisní pravděpodobnost pro slovo w^l emitované ze stavu t^j je:

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

3.4. Linearní interpolace

Pravděpodobnosti vypočítané pomocí MLE ovšem přináší velké problémy pro tagy a slova, která nebyla dosud viděna, a tedy takové tagy a slova budou mít nulové pravděpodobnosti. Tyto problémy se dají snadno odstranit pomocí některé metody vyhlazování. Zde použiji linearní interpolaci.
Linearní interpolace je metoda, kdy se řídkost v modelu s delší historií řeší pomocí linearní kombinace tohoto modelu s modely s kratší historií.
Tedy pro model s délkou historie k budou přechodové pravděpodobnosti:

$$\begin{aligned} P_{li}(t^n \mid t^{n-k}, t^{n-k+1},...,t^{n-1}) = {} & \lambda_k P_k(t^n \mid t^{n-k}, t^{n-k+1},...,t^{n-1}) + \\ & \lambda_{k-1} P_{k-1}(t^n \mid t^{n-k+1},...,t^{n-1}) + \\ & \lambda_2 P_2(t^n \mid t^{n-1}) + \\ & \lambda_1 P_1(t^n) + \\ & \lambda_0 * 1/T \end{aligned}$$

a emisní pravděpodobnosti:

$$P_{li}(w^n \mid t^j) = \lambda_2 P_2(w^n \mid t^j) + \lambda_1 P_1(w^n) + \lambda_0 * 1/W$$

Přičemž pro obě rovnosti musí platit $\sum_i \lambda_i = 1$. A navíc pravděpodobnosti Pi pro každé i budou počítány pomocí MLE.

3.5. Trénovací algoritmus

Během trénování stačí uložit do paměti počty výskytů tagů tj = C(tj), počty výskytů tagů tj následovaných tagem tk = C(tj,tk) a počty výskytů slov wl otagovaných jako tj = C(wl,tj). Tato data je potřeba uložit pro každé k: $1 \leq k \leq n$.
Program si tedy vytvoří tabulky četností, pro všechny k-tice stavů, kde $k \leq n$. Z těchto tabulek se potom vypočítají pravděpodobnosti $P_1,..., P_n$ a z těch nakonec pravděpodobnosti P_{li}.
Pravděpodobnosti je možné počítat rovnou během tagovacího algoritmu. Výpočet bude vypadat následovně:

1. **Pro každé n≥0, kde n označuje délku historie (pro n>1 bude t^i znamenat n-tici tagů; vytvoří se tedy tabulky pro každé $1 \leq k \leq n$), spočítáme (pomocí MLE):**

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

2. **Spočítáme pravděpodobnosti P_{li} (pravděpodobnosti P_j znamenají pravděpodobnosti z bodu 1 pro konkrétní j-tice tagů).**

for all tags $t^j = (t^{n-k}, t^{n-k+1},...,t^{n-1})$ do
    for all tags t^m do
        $Pli\ (t^m \mid t^{n-k}, t^{n-k+1},...,t^{n-1}) = \lambda_k P_k(t^m \mid t^{n-k}, t^{n-k+1},...,t^{n-1}) +$
        $\lambda_{k-1} P_{k-1}(t^m \mid t^{n-k+1},...,t^{n-1}) +$
        ...
        $\lambda_2 P_2(t^m \mid t^{n-1}) +$
        $\lambda_1 P_1(t^m) +$
        $\lambda_0 * 1/T$
    end
end
for all tags t^j do
    for all words w^l do
        $Pli\ (w^l \mid t^j) = \lambda_2 P_2(w^l \mid t^j) + \lambda_1 P_1(w^l) + \lambda_0 * 1/W$
    end
end

Parametry λ_i budou určeny pomocí EM (Expectation Maximization) algoritmu.

3.6. Viterbiho algoritmus (tagování)

Snažíme se nalézt nejpravděpodobnější kompletní cestu, tedy:

$$\arg\max P(X \mid O, \mu)$$

$$\underset{X}{}$$

Tedy pro pevné O stačí maximalizovat:

$$\arg\max_{X} P(X, O \mid \mu)$$

Definujme $\delta_i(t) = \max_{X_1 \ldots X_{t-1}} P(X_1 \ldots X_{t-1}, o_1 \ldots o_{t-1}, X_t = j \mid \mu)$

Tato proměnná uchovává pro každý bod v trellis pravděpodobnost nejpravděpodobnější cesty, která vede do uzlu. Odpovídající proměnná $\psi_j(t)$ uchovává uzel, ze kterého vedla tato nejpravděpodobnější cesta.

3.7. Tagovací algoritmus

Mějme větu délky n
Inicializace
for all tags t^j do
    $\delta_1(t^j) := \pi_j$
end
Indukce
for i:= 1 to n step 1 do
    for all tags t^j do
        $\delta_{i+1}(t^j) := \max_{1 \le k \le T} [\delta_i(t^k) * P_{li}(w^{i+1} \mid t^j) * P_{li}(t^j \mid t^k)]$
        $\psi_{i+1}(t^j) := \arg\max_{1 \le k \le T} [\delta_i(t^k) * P_{li}(w^{i+1} \mid t^j) * P_{li}(t^j \mid t^k)]$
    end
end
Ukončení a přečtení cesty (odzadu)
$X_{n+1} = \arg\max_{1 \le k \le T} \delta_{n+1}(j)$
for j:= n to 1 step -1 do
    $X_j = \psi_{j+1}(X_j)$
end
$P(X_1,\ldots,X_n) = \max_{1 \le k \le T} \delta_{n+1}(t^j)$

4. Implementace

Algoritmy použité v programu jsou zmíněné v části Úvod do problematiky.

4.1. Návrh struktury programu

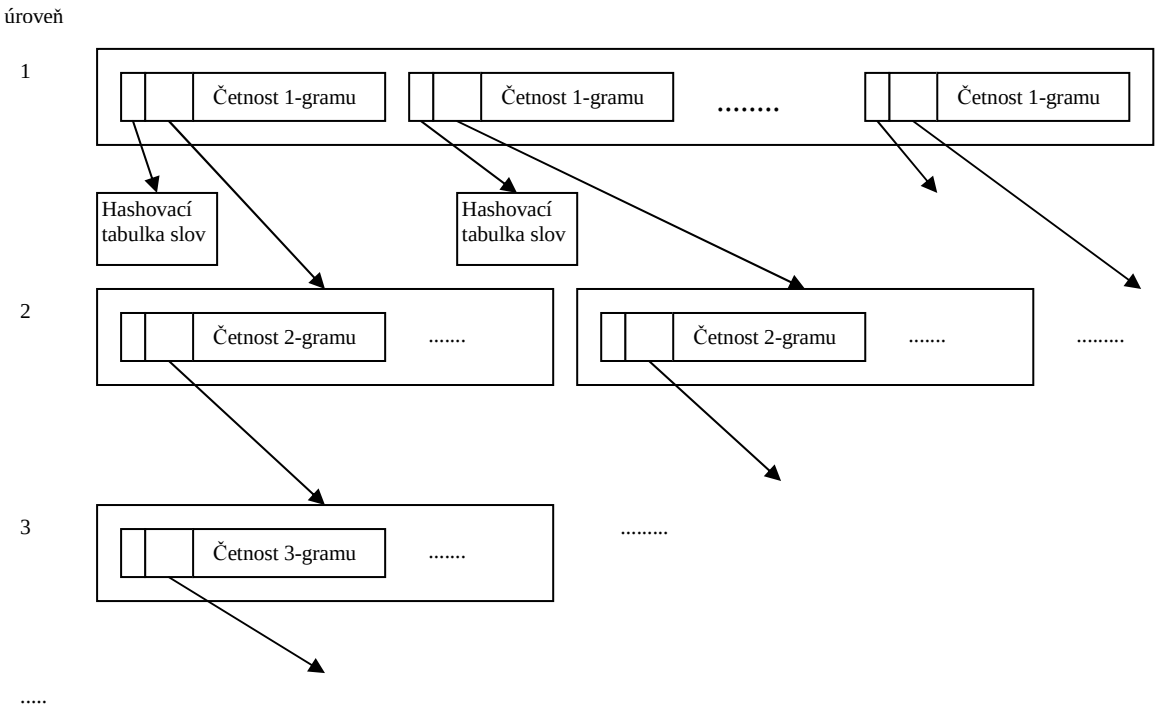
Program by se měl skadat minimalně ze 4 modulu:

1. Logika programu - měl by obsluhovat ovládání programu (načítání parametrů z příkazové řádky, jejich ověřování a zpracování, IO funkce). Program tohoto typu nepotřebuje GUI a stačí tedy načítání parametrů z příkazové řádky.
2. Modul obsahující trénovací algoritmus - vše potřebné pro trénování (algoritmus, vytváření příslušných datových struktur).

3. Modul obsahující tagovací algoritmus - vše potřebné pro tagování (Viterbiho algoritmus, přístup k datovým strukturám).
4. Modul obsahující třídy datových struktur.

4.2. Datové struktury

Pro uložení četností by byla vhodná víceúrovňová hashovací struktura. Prvkem na úrovni k by byla jednak četnost k-gramu, jehož první prvek by byl ten, který je vyhledáván na této úrovni, jednak by zde byl přístup do další hashovací tabulky úrovně k+1. Přístup do hashovací tabulky slov by byl jen z první vrstvy. Vznikla by tedy jakási stromová struktura hashovacích tabulek (každý uzel by měl až tolik následovníků, kolik je různých tagů - viz obr. 1 - první úroveň by navíc obsahovala ještě jednou tolik odkazů na hashovací tabulky slov). Hloubka takové struktury by byla n+1, kde n je délka historie.
Hashovací tabulka slov by obsahovala jen četnosti jednotlivých slov emitovaných daným k-gramem. Poslední úroveň by už odkazy na hashovací tabulky slov neobsahovala.
Vyhledávalo by se podle prvků n-gramu odzadu, tedy např. vyhledání četnosti trigramu (A,B,C) by probíhalo tak, že by se nejprve v hashovací tabulce vyhledal tag C, tím by se vyhledala struktura obsahující jednak četnost tagu C, ale i nová hashovací tabulka, kde by se vyhledal i tag B a tak dále. Pokud by se hledalo ještě slovo X emitované tagem C, tak by se po vyhledání tagu C vyhledalo slovo X v hashovací tabulce slov.



obr. 1

Při výpočtu přechodových a emisních pravděpodobností by tedy stačilo dvakrát projít až do poslední úrovně struktury (přesněji jednou do poslední a jednou do předposlední úrovně).
Tedy pro výpočet přechodové pravděpodobnosti

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

kde jsou tagy například trigramy, tedy $t^j = (t^{j1}, t^{j2}, t^{j3})$ a $t^k = (t^{j2}, t^{j3}, t^{k1})$ by mělo stačit vyhledat četnosti $C(t^{j1}, t^{j2}, t^{j3}, t^{k1})$ a $C(t^{j1}, t^{j2}, t^{j3})$, což jsou 2 průchody.

Obdobně (ale jednodušeji) by probíhal výpočet emisní pravděpodobnosti.

Pro výpočet pravděpodobností P_{li} (přechodových i emisních dohromady) potom stačí také 2 průchody (tím, že se prochází odzadu, by se při průchodu postupně vypočítaly přechodové pravděpodobnosti pro každé k, současně by se vypočítaly i emisní pravděpodobnosti).

Pro výpočet emisní pravděpodobnosti P_{li} je ještě potřeba mít vedle speciální hashovací tabulku četností všech slov (ve výpočtu je potřeba pravděpodobnost $P(w^l)$).

Takto navržená struktura je vhodná i pro případné přetrénování. Pokud bychom chtěli tagovat text pomocí HMM s kratší délkou historie, tak přetrénování není nutné vůbec. Pokud bychom chtěli tagovat s delší délkou historie tak by stačilo přidat další vrstvy do struktury.

Nevýhodou takovéto struktury bude veliká paměťová náročnost.

Jinou možnou variantou datových struktur je oddělit z této struktury emisní pravděpodobnosti a udělat pro ně vlastní 2 vrstvou strukturu, kde v první vrstvě by byly tagy a v druhé vrstvě slova. Tím by se zjednodušila implementace, ale trošku zvětšily paměťové nároky.

Implementace v Javě pomocí kolekce HashMap.

5. Literatura

Christopher D. Manning, Hinrich Schütze: Foundations of Statistical Natural Language Processing