Warum brauchen wir Viterbi in Wortart-tagger?

Wozu wird Viterbi verwendet?

Welche Werte(Parameter) brauchen wir um eine Viterbi-WK zu berechnen?

Warum brauchen wir Viterbi in Wortart-tagger?

$$\hat{t}_1^n = \arg\max_{t_1^n} p(t_1^n | w_1^n)$$

Problem: Es gibt zuviele mögliche Tagfolgen für eine gegebene Wortfolge, um alle aufzählen und ihre Wahrscheinlichkeit berechnen zu können.

Beobachtung: Wenn zwei mögliche Wortartfolgen $T = t_1, ..., t_m$ und $S = s_1, ..., s_m$ (für die ersten m Wörter eines Satzes) in den letzten k Tags übereinstimmen, dann kann bei einem HMM k-ter Ordnung die weniger wahrscheinliche der beiden Tagfolgen nicht ein Präfix der besten Gesamttagfolge sein, und wir können sie ignorieren.

Suche

Viterbi berücksichtigt die WK der ganzen Tag-Folge, bevor es den besten Tag bestimmt.

Diese Eigenschaft erlaubt eine effiziente Verarbeitung, hat aber zur Folge, dass manche Ambiguitäten nicht korrekt aufgelöst werden können:

The	horse	raced	past	the	barn	(fell)	.
DT	NN	VBD	IN	DT	NN		
DT	NN	VBN	IN	DT	NN		

Ein HMM 2. Ordnung würde das Wort "raced" gleich taggen, egal ob "fell" nachfolgt oder nicht. Erst ein HMM 4. Ordnung kann weit genug zurückschauen, um korrekt zu desambiguieren.

Anmerkung: Man könnte das Problem lösen, indem man das Tagset verfeinert und bspw. alle Tags, die nach einem finiten Verb folgen, mit einem Apostroph markiert.

The	horse	raced	past	the	barn	(fell)	.
DT	NN	VBD	IN'	DT'	NN'		$.
DT	NN	VBN	IN	DT	NN	VBD	$.

Wozu wird Viterbi verwendet?

Um die beste Tag-Folge für eine gegebene Wort-Folge zu berechnen.
Anstatt alle möglichen Tag-Folgen t1..tn aufzulisten und p(t1..tn, w1,..wn) zu berechenen und argmax, benutzen wir Viterbi-Algorithmus.
Also, wir berechnen Viterbi-Wk für Tags/Tagpaare an jeden Positionen und extrahiert die beste Tag-Folge.

$$\hat{t}_1^n = \arg\max_{t_1^n} p(t_1^n | w_1^n)$$

Welche Werte(Parameter) brauchen wir um eine Viterbi-WK zu berechnen?

Welche Werte(Parameter) brauchen wir um eine Viterbi-WK zu berechnen?

Viterbi-Algorithmus (Bigramm-Tagger)

1. Initialisierung: $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

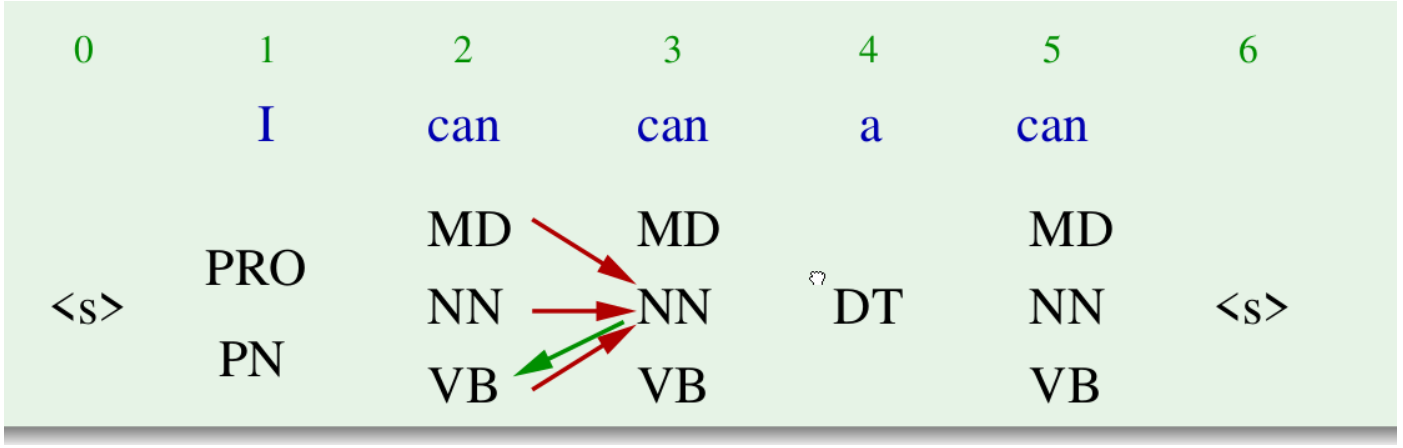
2. Berechnung: (für $0 < k \leq n+1$)

$$\begin{aligned} \delta_t(k) &= \max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \\ \psi_t(k) &= \arg\max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \end{aligned}$$

3. Ausgabe: (für $0 < k \leq n$)

$$\begin{aligned} t_{n+1} &= \langle s \rangle \\ t_k &= \psi_{t_{k+1}}(k+1) \end{aligned}$$

t, t' sind Tags, k sind Wortpositionen, $\delta_t(k)$ sind die Viterbiwahrscheinlichkeiten und $\psi_t(k)$ sind die besten Vorgängertags

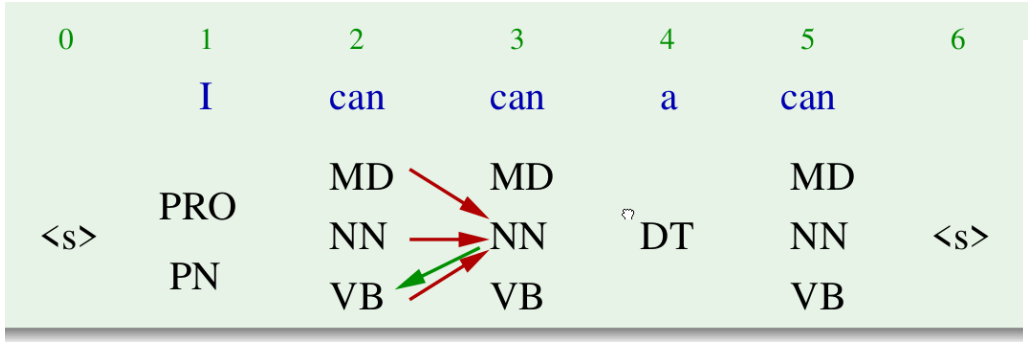


(Bigramm-Tagger)

Schreibt die Formel für Bigramm-Viterbi.

Berechnet alle Viterbi-WK für Position 0, 1, und 6.

vit_PRO (1) = 1* p(pro | <s>) p(I | pro)
vit_PN (1) = 1* p(PN | <s>) p(I | PN)



Viterbi-Algorithmus (Bigramm-Tagger)

1. Initialisierung: $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für $0 < k \leq n+1$)

$$\delta_t(k) = \max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t)$$

$$\psi_t(k) = \arg\max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t)$$

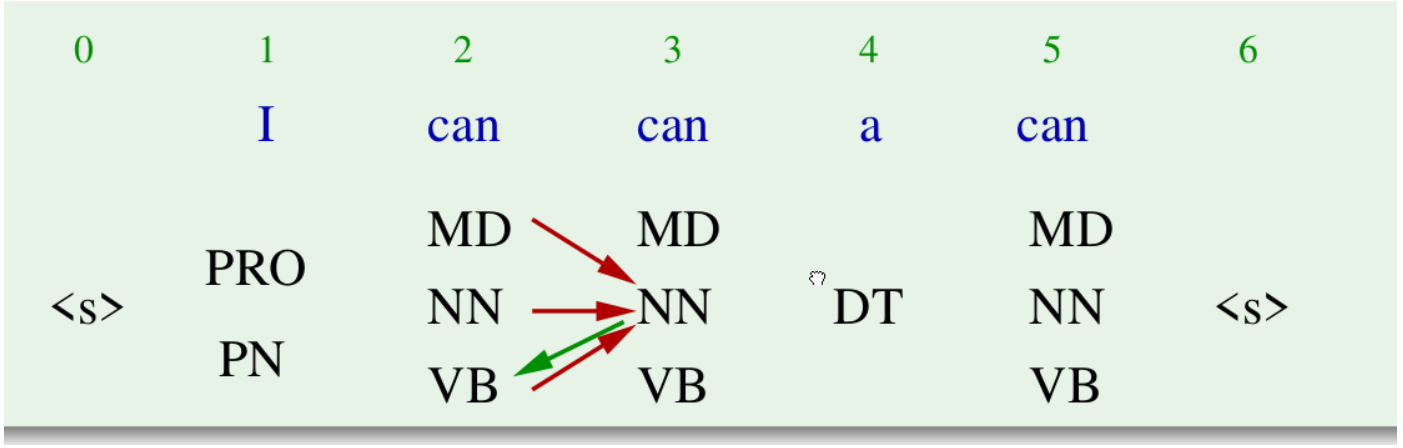
(Bigramm-Tagger)

Welche Positionen müssen wir dafür die Viterbi-WK berechnen?

Schreibt die Formel für Bigramm-Viterbi.

Berechnet alle Viterbi-WK für Position 0,1, 4, und 6.

Wie berechnet man den besten Tag für das Wort an Position 5? (Schriebt die Formel)



(Trigram-Tagger)

Welche Positionen müssen wir dafür die Viterbi-WK berechnen?

Schreibt die Formel für Trigram-Viterbi.

Berechnet alle Viterbi-WK für Position 0,1, 5.

Wie berechnet man den besten Tag für das Wort an Postion 5? (Schriebt die Formel)

Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modelles nicht einem einzelnen Tag, sondern einem Tagpaar.
- Übergänge gibt es nur zwischen Zuständen (t, t') und (t'', t''') mit $t' = t''$

1. Initialisierung: $\delta_{t',t}(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \text{ und } t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$
2. Berechnung: (für $0 < k \leq n+1$)

$$\delta_{t',t}(k) = \max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t)$$
$$\psi_{t',t}(k) = \arg\max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t)$$

Wie berechnet man den Tag an der letzten Position?

Sie müssen dazu zunächst über alle Einträge in der letzten Spalte maximieren. Dann können Sie mit der psi-Variablen die beste Tagfolge finden.

Die Alternative dazu ist, k Endetags hinzuzufügen. Dann müssen Sie nicht
maximieren. Das ist eigentlich die einfachere Lösung. (Vielleicht überarbeite ich die Folien dazu.)

```python
def viterbi(self, words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words =  [''] + words + ['']  # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitscore = [dict() for _ in range(len(words))]  # speichert logarithmierte Werte
    bestprev = [dict() for _ in range(len(words))]  # speichert die besten Vorgänger
    vitscore[0][('<s>','<s>')] = 0.0  # =log(1)
    for i in range(1, len(words)):
        lexprobs = self._lex_probs(words[i])  # die möglichen Tags nachschlagen
        for tag, lexprob in lexprobs:
            for tagpair in vitscore[i-1]:
                tag1, tag2 = tagpair  # Kontext-Tags
                p = self._context_prob(tagpair, tag) * lexprob / self._apriori_tag_prob[tag]
                p = vitscore[i-1][tagpair] + log(p)
                newtagpair = (tag2, tag)
                if newtagpair not in vitscore[i] or vitscore[i][newtagpair] < p:
                    vitscore[i][newtagpair] = p
                    bestprev[i][newtagpair] = tagpair
    # in der letzten Spalte das Tagpaar mit der höchsten Bewertung suchen
    tagpair = max(vitscore[-1], key=vitscore[-1].get)
    # beste Tagfolge extrahieren
    result_tags = []
    for i in range(len(words)-1, 1, -1):
        result_tags.append(tagpair[0])
        tagpair = bestprev[i][tagpair]
    return reversed(result_tags)  # Tagfolge umdrehen
```

Aufgabe 5) Erklären Sie detailliert und mit Formeln, wie ein Bigramm-Tagger auf Basis von HMMs effizient die beste Tagfolge berechnet. (4 Punkte)