# Hidden-Markov-Modelle: Wortart-Tagging

# Hidden-Markov-Modelle: Wortart-Tagging

- Given a sequence of words, we want to find the part-of-speech (POS) of every word in the sequence.
- Some words can have more than one possible POS-tags such as “can”.
- Some words such as “I” and “a” have only one possible POS. To do POS-tagging, we can use this knowledge to narrow down the set of possible POS for these words.
- For words like “can” we still need to disambiguate their POS.
- We will use the HMM to do that.

| I | can | can | a | can |
|---|---|---|---|---|
| **PRO** | PRO | PRO | PRO | PRO |
| PN | PN | PN | PN | PN |
| MD | MD | MD | MD | MD |
| NN | NN | NN | NN | NN |
| VB | VB | VB | VB | VB |
| DT | DT | DT | **DT** | DT |

HMM gets a sequence of words and POS-tag sequence candidates as input and computes the best POS-tag sequence for the input.

| I | want | to | shop | for | a | new | dress |
|---|---|---|---|---|---|---|---|
| PRP | VBP | TO | VB | IN | DT | JJ | NN |
| | | | NN | | | | VB |

Suppose this is our input text and the possible POS tags for each word. Here we have to disambiguate “shop” and “dress” so we will list all possible tag sequences and pass them to the HMM model.

| I | want | to | shop | for | a | new | dress |
|---|---|---|---|---|---|---|---|
| PRP | VBP | TO | VB | IN | DT | JJ | NN |
| PRP | VBP | TO | VB | IN | DT | JJ | VB |
| PRP | VBP | TO | NN | IN | DT | JJ | NN |
| PRP | VBP | TO | VB | IN | DT | JJ | VB |

HMM computes p(tag seq | word seq) for every tag seq candidate and does argmax to get the best tag seq, then returns it as output.

model

PRP VBP TO VB IN DT JJ NN

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

p(t1,t2,t3 | w1,w2,w3) = 0.5
p(t1,t2,t3 | w1,w2,w3) = 0.2
p(t1,t2,t3 | w1,w2,w3) = 0.2
p(t1,t2,t3 | w1,w2,w3) = 0.1

| I | want | to | shop | for | a | new | dress |
|---|---|---|---|---|---|---|---|
| PRP | VBP | TO | VB | IN | DT | JJ | NN |

The model estimates p(t1,t2,t3 | w1,w2,w3), based on two things.

- Lexical probability
  - how likely it is that "I" would have the tag "PRP" ?
  - how likely it is that "shop" would have the tag "VB" ?
- Context probability (Let's say here we consider the previous 2 tags)
  - if the previous 2 tags are "VBP, TO" what is the probability that VB will be the tag at position 4( shop)? (is it more than having NN?)
- These probabilities (=model parameters) are estimated by looking in a corpus.
- If these probabilities are high, the total probability will be high.

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i | t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i | t_i)}_{\text{lexikalische Wk.}}$$

For every position i to i=n+1, we will compute the product of the context prob and the lexical prob regarding that position. Then, multiply every result together to get the final probability.

| I | want | to | shop | for | a | new | dress |
|---|---|---|---|---|---|---|---|
| PRP | VBP | TO | VB | IN | DT | JJ | NN |
| 1.0 | 1.0 | 1.0 | 0.6 | 1.0 | 1.0 | 1.0 | 0.5 |

This is just the theoretical concept. In practice, we do not compute the best tag seq this way because it is inefficient -- > we will use the Viterbi algorithm instead

Model Herleitung (1/2)

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

$$\arg\max_{t_1^n} \frac{p(t_1^n, w_1^n)}{p(w_1^n)} :$$ bedingte Wahrscheinlichkeit: $p(x|y) = \frac{p(x,y)}{p(y)}$

$$\arg\max_{t_1^n} p(t_1^n, w_1^n)$$ Die Konstante $p(w_1^n)$ hat keinen Einfluss auf das Maximierungsergebnis.

$$p(t_1^n, w_1^n) = \left( \prod_{i=1}^{n+1} p(t_i | t_1^{i-1}) \right) \prod_{i=1}^{n+1} p(w_i | t_1^{n+1}, w_1^{i-1})$$ Zerlegung in ein Produkt bedingter Wahrscheinlichkeiten:

Kettenregel $p(x, y, z) = p(x)p(y|x)p(z|xy)$

Wir fügen hier ein Endetag $t_{n+1} = \langle /s \rangle$ und ein Endetoken $w_{n+1} = \epsilon$ hinzu,
mit $p(\epsilon | \langle /s \rangle) = 1$.

**Example**

p(t1,t2,t3,w1,w2,w3) = p(t1) p(t2|t1) p(t3|t1,t2) p(</s>|t1,t2,t3)
p(w1|t1,t2,t3, </s>) p(w2 | t1,t2,t3, </s>, w1) p(w3|t1,t2,t3,</s>, w1,w2) p(leer | t1,t2,t3,</s>, w1,w2, w3)

| w1 | w2 | w3 | e |
|---|---|---|---|
| t1 | t2 | t3 | </s> |

# Model Herleitung (2/2)

$$p(t_1^n, w_1^n) = \left(\prod_{i=1}^{n+1} p(t_i|t_1^{i-1})\right) \prod_{i=1}^{n+1} p(w_i|t_1^{n+1}, w_1^{i-1})$$

just switch the position

move the product symbol to the front

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i|t_1^{i-1})\ p(w_i|w_1^{i-1}, t_1^{n+1})$$

Wir machen nun die folgenden vereinfachenden Annahmen:

- Das Wortart-Tag $t_i$ hängt nur von den $k$ vorherigen Tags ab.
- Das Wort $w_i$ hängt nur von seiner Wortart $t_i$ ab.
- Die Wahrscheinlichkeiten sind unabhängig von der Position.

the final formula

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i|t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i|t_i)}_{\text{lexikalische Wk.}}$$

Dieses Modell heißt **Hidden-Markow-Modell**, weil die Zustände (Tags bzw. Tagpaare beim Trigramm-Tagger) nicht direkt beobachtbar sind.

Wir fügen am Anfang $k$ Starttags hinzu, damit $p(t_1|t_{1-k}^0)$ definiert ist.

p(t1,t2,t3,w1,w2,w3) = p(t1) p(t2|t1) p(t3|t1,t2) p(</s>|t1,t2,t3)
p(w1|t1,t2,t3, </s>) p(w2 | t1,t2,t3, </s>, w1)
p(w3|t1,t2,t3,</s>, w1,w2) p(leer | t1,t2,t3,</s>, w1,w2, w3)

p(t1,t2,t3,w1,w2,w3) = p(t1|<s>,<s>) p(t2|<s>, t1) p(t3|t1,t2) p(</s>|t2,t3)
p(w1|t1,~~t2,t3, </s>~~) p(w2 | ~~t1,~~t2,~~t3, </s>, w1~~)
p(w3|~~t1,t2,~~t3,~~</s>, w1,w2~~)
p(leer ~~| t1,t2,t3~~,</s>~~, w1,w2, w3~~)

k=2

| e | e | w1 | w2 | w3 | e |
|---|---|---|---|---|---|
| <s> | <s> | t1 | t2 | t3 | </s> |

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

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i | t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i | t_i)}_{\text{lexikalische Wk.}}$$

## Zwei Typen von Parametern

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i | t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i | t_i)}_{\text{lexikalische Wk.}}$$

Die Kontextwahrscheinlichkeiten erfassen die syntaktischen Abhängigkeiten der Wortart-Tags vom linken Satzkontext.

Die lexikalischen Wahrscheinlichkeiten erfassen die syntaktischen Eigenschaften des aktuellen Wortes.

# Beispiel

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1})p(w_i|t_i)$$

Bei einem HMM 1. Ordnung (k=1) gilt:

$$p(\text{NE VVFIN, Peter liest}) = p(\text{NE}|\langle s\rangle)\ p(\text{Peter}|\text{NE}) \cdot p(\text{VVFIN}|\text{NE})\ p(\text{liest}|\text{VVFIN}) \cdot p(\langle/s\rangle|\text{VVFIN})\ p(\epsilon|\langle/s\rangle)$$

Beim HMM 2. Ordnung (k=2) gilt:

$$p(\text{NE VVFIN, Peter liest}) = p(\text{NE}|\langle s\rangle,\langle s\rangle)\ p(\text{Peter}|\text{NE}) \cdot p(\text{VVFIN}|\langle s\rangle,\text{NE})\ p(\text{liest}|\text{VVFIN}) \cdot p(\langle/s\rangle|\text{NE,VVFIN})\ p(\epsilon|\langle/s\rangle)$$

| pos 0 | 1 | 2 | 3 |
|---|---|---|---|
| e | Peter | liest | e |
| <s> | NE | VVFIN | </s> |

1

pos1:
- -lexical: p(Peter| NE)
- -context: p(NE|<s>)

pos2
- - lexical: p(liest|VVFIN)
- -context: p(VVFIN|NE)

pos3
- -lexical: p(leer|</s)
- -context: p(</s>|VVFIN)

2

**Final answer**

p(Peter| NE)p(NE|<s>) *
p(liest|VVFIN)p(VVFIN|NE) *
p(leer|</s)p(</s>|VVFIN)

3

**Aufgabe 3)** Wie berechnen Sie bei einem **Hidden-Markowmodell** 2. Ordnung die gemeinsame Wahrscheinlichkeit der Wortfolge *Bayern, schlägt, Manchester* und der Tagfolge *NE, VVFIN, NE*? (2 Punkte)

K=2

e e Bayern, schlägt, Manchester e
<s> <s> NN, VVFIN, NE </s>

p(Bayern, schlägt, Manchester, NN, VVFIN, NE) = p(NN| <s>, <s>) p(Bayern|NN)
p(VVFIN |<s>, NN) p(schlägt | VVFIN)
p(NE|NN, VVFIN) p(Manchester |NE)
p(</s>|VVFIN, NE) p(e | </s> )

context: p(NN| <s>, <s>)
lexical: p(Bayern|NN)

**Aufgabe 2)** Wie berechnen Sie bei einem **Markowmodell** 2. Ordnung (Trigramm-Modell) über Buchstaben die Wahrscheinlichkeit der Buchstabenfolge *CIS*? (2 Punkte)

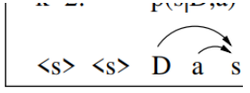
k=2

<s> <s> C I S </s>

p(C,I,S) = p(C| <s>,<s>) * p(I | <s>, C) * p(S| C, I ) p(</s> | I, S)

example

$$p_L(T) = \prod_{i=1}^{n+1} p_L(a_i|a_{i-k}...a_{i-1})$$




Ein solches Modell heißt **Markowmodell** der Ordnung $k$.

Für $k = 2$ bekommen wir:
p(Er ölt) =
p(E|⟨s⟩,⟨s⟩) p(r|⟨s⟩,E) p(_ |E,r) p(ö|r,_) p(l|_,ö) p(t|ö,l) p(⟨/s⟩|l,t)

⇒ Es werden $k$ Startsymbole $\langle s \rangle$ und ein Endesymbol $\langle /s \rangle$ hinzugefügt:
$a_{-1} = a_0 = \langle s \rangle$ und $a_{n+1} = \langle /s \rangle$

note: in the character-level model, **spaces** should also be seen as a symbol.

Bei einem HMM 1. Ordnung (k=1) gilt:

$$p(\text{NE VVFIN, Peter liest}) = p(\text{NE}|\langle s\rangle)\, p(\text{Peter}|\text{NE}) \cdot p(\text{VVFIN}|\text{NE})\, p(\text{liest}|\text{VVFIN}) \cdot p(\langle/s\rangle|\text{VVFIN})\, p(\epsilon|\langle/s\rangle)$$

## Parameterschätzung

$p(t|t_1, ..., t_k)$ : $p(w|t)$

Zur Schätzung der Parameter braucht man ein **Trainingskorpus**, in dem jedes Wort mit seiner Wortart annotiert ist.

Man zählt die k+1-Gramm-Häufigkeiten $f(t_1, ..., t_{k+1})$ und die Tag-Wort-Häufigkeiten $f(t, w)$ und schätzt dann die Wahrscheinlichkeiten wie folgt:

$$p(w|t) = \frac{f(t,w)}{\sum_{w'} f(t,w')} \qquad p(t|t_1, ..., t_k) = \frac{f(t_1, ..., t_k, t)}{\sum_{t'} f(t_1, ..., t_k, t')}$$

Beispiel:

$$p(\text{Haus}|\text{NN}) = \frac{f(\text{NN, Haus})}{\sum_{w'} f(\text{NN}, w')} \qquad p(\text{NN}|\text{ART}) = \frac{f(\text{ART, NN})}{\sum_{t'} f(\text{ART}, t')}$$

Die Kontextwahrscheinlichkeiten $p(t|t_1, ..., t_k)$ müssen aber noch geglättet werden.

Training data (each word is annotated with its POS)
Ich wohne in meinem Haus.
POS, ... ART NN
Er hat ein Haus gekauft.
.... ART NN.....
Ein Hund ist gut.
ART NN V ADJ

Extract the frequency of (tag, word) and tag n-gram (tag1, tag2)

f(Tag, word)

NN, Haus: 100
NN,Hund: 150

f(Tag1, Tag2)

NN, V : 9
NN,ART : 3
ART, NN : 8

Estimate the probabilities

$p(w|t)$ $p(t|t_1, ..., t_k)$ :

#Zusammenfassung

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

Formulate the problem (compute the best tag seq between tag sequence candidates)

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i | t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i | t_i)}_{\text{lexikalische Wk.}}$$

- Define how to compute the prob p(tag_seq | word_seq)
- Here we choose HMM

$$p(w|t) = \frac{f(t,w)}{\sum_{w'} f(t,w')} \qquad p(t|t_1,...,t_k) = \frac{f(t_1,...,t_k,t)}{\sum_{t'} f(t_1,...,t_k,t')}$$

- Choose a method to estimate the probs
- Here we choose the relative frequency method

Better : apply a smoothing method such as backoff smoothing

## Model training

After we defined how p should be computed, we can collect relevant information to compute them. E.g. choose k=1, then we extract tag bigrams.

**Training data (each word is annotated with its POS)**
Ich wohne in meinem Haus.
POS, ... ART NN
Er hat ein Haus gekauft.
.... ART NN.....
Ein Hund ist gut.
ART NN V ADJ

Extract the frequency of (tag, word) and tag n-gram (tag1, tag2)

f(Tag, word)

NN, Haus: 100
NN,Hund: 150

f(Tag1, Tag2)

NN, V : 9
NN,ART : 3
ART, NN : 8

Estimate the probabilities

$p(w|t)$ $p(t|t_1,...,t_k)$ :

## Prediction

Use the model to predict the tag sequence of a new text.

„Peter, liest“

list all possible tags for „Peter“ and „liest“

compute p(tags|text) for every possible tag seq, return the best tag seq

# Search for the best tag seq

## Suche

**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.

## Widerspruchsbeweis

Angenommen S ist weniger wahrscheinlich als T, aber ein Präfix der besten Tagfolge. Dann hat die beste Tagfolge die Form SV, wobei V eine weitere Tagfolge ist. In diesem Fall hätte aber die Tagfolge TV eine höhere Wahrscheinlichkeit als SV, weil T wahrscheinlicher als S ist und die Wahrscheinlichkeiten, die jeweils für V hinzumultipliziert werden, bei SV und TV identisch sind. Also kann S kein Präfix der besten Tagfolge sein. □

**Beispiel:** k=2 T=X,A,B S=Y,A,B V=C,D
TV=X,A,B,C,D SV=Y,A,B,C,D

C hängt hier nur von A und B ab. D hängt nur von B und C ab.
Also sind die Tags C,D nach X,A,B genauso wahrscheinlich wie nach Y,A,B.

## Suche

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 | $. |

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

The initial way to determine the best tag seq is not optimal because

1. there can be too many tag seq candidates (require a lot of computing)
2. The way HMM computes p(tags|words) is not good because it can assign low prob to good tag seq. This can happen when, e.g., two candidates have the same tag suffix.

   Suppose, Y,A,B,C,D is the best tag. Another tag seq X,A,B,C,D can have higher prob than the best tag if its prefix X,A,B has a higher prob than Y,A,B

Solution: Viterbi

## 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 $0 < k \leq n+1$)

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

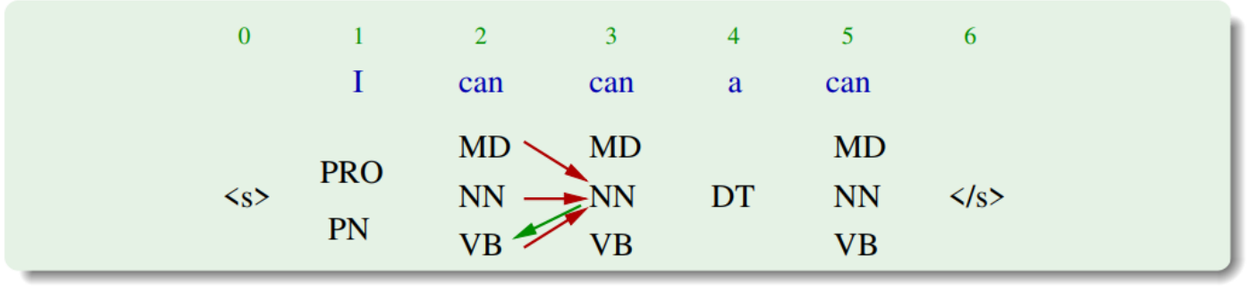
3. Ausgabe: (für $0 < k \leq n$)

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

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

(I will talk about it in the next session)

## Beispiel: Viterbi-Algorithmus




$$\delta_{NN}(3) = \max \begin{pmatrix} \delta_{MD}(2)\, p(NN|MD)\, p(can|NN), \\ \delta_{NN}(2)\, p(NN|NN)\, p(can|NN), \\ \delta_{VB}(2)\, p(NN|VB)\, p(can|NN) \end{pmatrix}$$

# Smoothing

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

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i | t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i | t_i)}_{\text{lexikalische Wk.}}$$

$$p(w|t) = \frac{f(t,w)}{\sum_{w'} f(t,w')} \qquad p(t|t_1,...,t_k) = \frac{f(t_1,...,t_k,t)}{\sum_{t'} f(t_1,...,t_k,t')}$$

Lexikal. WK Kontext WK

Now, we want to smooth these probabilities

- Choose a method to estimate the probs
- Here we choose the relative frequency method

Better : apply a smoothing method such as backoff smoothing

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i|t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i|t_i)}_{\text{lexikalische Wk.}}$$

1) use the backoff smoothing to smooth the context prob

See the detail in Übung7

- Kontextwahrscheinlichkeiten $p(t|t', t'')$

  Diese Wahrscheinlichkeiten sollen mit Kneser-Ney-Glättung geglättet werden. Sie müssen hier nur die relativen Häufigkeiten $p^*(t|t', t'')$, $p^*(t|t'')$ und $p^*(t)$ (vgl. Übung 3) berechnen. $p(t|t', t'')$ wird erst im eigentlichen Taggerprogramm berechnet.

2) smooth the lex. prob

$$p(t_1^n, w_1^n) = \prod_{i=1}^{n+1} \underbrace{p(t_i|t_{i-k}^{i-1})}_{\text{Kontextwahrsch.}} \underbrace{p(w_i|t_i)}_{\text{lexikalische Wk.}}$$

adjust the formula

$$p(w|t) = \frac{p(t|w)p(w)}{p(t)}$$

p(w) is constant, so it gets canceled out.

Die Apriori-Wahrscheinlichkeit $p(t) = f(t)/N$ (N = Korpusgröße) der Tags kann leicht aus dem Trainingskorpus geschätzt werden.

backoff p(t| w ) by using the suffix and the case (Groß-/Kleinschreibung) of the word.

für Suffixlänge l=5:

$$\begin{aligned} p(\text{NN}|\text{schwärzlich}) = {} & r(\text{NN}|\text{schwärzlich}) + \alpha(\text{schwärzlich}) \; ( \\ & r_{suff}(\text{NN}|\text{zlichK}) + \alpha_{suff}(\text{zlichK}) \; ( \\ & r_{suff}(\text{NN}|\text{lichK}) + \alpha_{suff}(\text{lichK}) \; ( \\ & r_{suff}(\text{NN}|\text{ichK}) + \alpha_{suff}(\text{ichK}) \; ( \\ & r_{suff}(\text{NN}|\text{chK}) + \alpha_{suff}(\text{chK}) \; ( \\ & r_{suff}(\text{NN}|\text{hK}) + \alpha_{suff}(\text{hK}) \; p(\text{NN}|\text{K}))))) \end{aligned}$$

the recursion ends here

This helps prevent zero prob when the new text contains unseen words or tags

## Behandlung unbekannter Wörter

Die Apriori-Wahrscheinlichkeit $p(t) = f(t)/N$ (N = Korpusgröße) der Tags kann leicht aus dem Trainingskorpus geschätzt werden.

Die bedingte Wahrscheinlichkeit $p(t|w)$ kann anhand des Wortsuffixes (z.B. der Länge $l$) geschätzt werden:

$$p(t|w) = p(t|a_1 a_2 ... a_n) \approx p_{suff}(t|a_{n-l+1} ... a_n)$$

$$p(\text{NN}|\text{schwärzlich}) \approx p_{suff}(\text{NN}|\text{zlich}) \quad \text{falls } l = 5$$

Die Suffix-Tag-Wahrscheinlichkeit ($\alpha$ ist hier eine beliebige Buchstabenfolge.)

$$p_{suff}(t|s_1 ... s_l) = \frac{f_{suff}(s_1 ... s_l, t)}{\sum_{t'} f_{suff}(s_1 ... s_l, t')} \qquad f_{suff}(s_1 ... s_l, t) = \sum_{w=\alpha s_1 ... s_l} f(w, t)$$

wird aus Trainingsdaten geschätzt und mit einem Backoff-Verfahren geglättet.

Für groß- und kleingeschriebene Wörter schätzt man am besten separate Parameter. Dafür kann man einfach einen Buchstaben (z.B. "G" für Groß- und "K" für Kleinschreibung) an das Wortsuffix anhängen: $p_{suff}(\text{NN}|\text{zlichK})$

## Behandlung unbekannter Tags

neue Formel zum Taggen mit unbekannten Wörtern:

$$\hat{t}_1^n = \arg\max_{t_1^n} \prod_{i=1}^{n+1} p(t_i|t_{i-k}^{i-1}) \frac{p(t_i|w_i)}{p(t_i)}$$

Neben unbekannten Wörtern stellen auch ungesehene Tags bekannter Wörter ein Problem dar (bspw. wenn das englische Wort "indicate" nur als Infinitiv aber nicht als finites Verb aufgetaucht ist.)

Die suffixbasierten Tag-Wahrscheinlichkeiten geben meist auch den ungesehenen Tags eine kleine Wahrscheinlichkeit.

Daher ist es sinnvoll, wortbasierte und suffixbasierte Wahrscheinlichkeiten in einem Backoff-Modell zu kombinieren:

$$p(t|w) = r(t|w) + \alpha(w)\, p_{suff}(t|suffix(w))$$

mit $r(t|w) = max(0, \frac{f(w,t)-\delta}{\sum_{t'} f(w,t')})$

**Hidden Markov Model** (**HMM**) is a statistical Markov model in which the system being modeled is assumed to be a Markov process — call it $X$ — with unobservable ("*hidden*") states. As part of the definition, HMM requires that there be an observable process $Y$ whose outcomes are "influenced" by the outcomes of $X$ in a known way. Since $X$ cannot be observed directly, the goal is to learn about $X$ by observing $Y$. HMM has an additional requirement that the outcome of $Y$ at time $t = t_0$ may be "influenced" exclusively by the outcome of $X$ at $t = t_0$ and that the outcomes of $X$ and $Y$ at $t < t_0$ must not affect the outcome of $Y$ at $t = t_0$.

Hidden Markov models are known for their applications to thermodynamics, statistical mechanics, physics, chemistry, economics, finance, signal processing, information theory, pattern recognition - such as speech, handwriting, gesture recognition,[1] part-of-speech tagging, musical score following,[2] partial discharges[3] and bioinformatics.[4][5]