# Übung 8

## Wortart-Tagger (Anwendung)

Schreiben Sie ein Programm, welches die Parameterdatei, welche Sie in der letzten Übung erzeugt haben, einliest, die Backoff-Faktoren berechnet und dann einen tokenisierten Text (ein Satz pro Zeile, Tokens durch Leerzeichen getrennt) aus einer Datei liest und satzweise annotiert. Verwenden Sie zum Annotieren den Viterbi-Algorithmus. Zur Vermeidung von Underflow arbeiten Sie mit logarithmierten Viterbi-Wahrscheinlichkeiten.

Sie können folgendermaßen vorgehen: Zur Initialisierung setzen Sie `logvitprob[0][('<s>','<s>')] = log(1)`. Dann iterieren Sie über alle Wortpositionen $i = 1, ..., n + 1$ und berechnen zunächst für alle Tags $t$ im Tagset die Wahrscheinlichkeit $p(t|w_i)$ (Formel siehe ÜB 6). Für jedes Tag t mit $p(t|w_i) > 0.001$ und alle Tag-Paare c (Zustände) in `logvitprob[i-1]` berechnen Sie nun das Produkt aus der geglätteten Kontextwahrscheinlichkeit $p(t|c)$ und der geglätteten lexikalischen Wahrscheinlichkeit $p(t|w_i)$ und teilen durch die Apriori-Wahrscheinlichkeit $p(t)$. Dann berechnen Sie davon den Logarithmus und addieren ihn zu `logvitprob[i-1][c]`. Sie vergleichen Werte, die Sie für alle möglichen Vorgänger-Tagpaare erhalten haben, und speichern den höchsten Wert in `logvitprob[i][t]`. Das beste Vorgänger-Tagpaar speichern Sie in `bestprev[i][(t2,t)] = (t1,t2)`.

Auf diese Art können Sie die Viterbiwahrscheinlichkeiten von links nach rechts berechnen. Als Letztes sollten Sie noch die Viterbiwahrscheinlichkeit der beiden Endetags `</s>` berechnen. Dann können Sie rückwärts mit Hilfe der Variablen `bestprev` die beste Tagfolge extrahieren und ausgeben. Sie beginnen damit, dass Sie mit `t,_ = bestprev[n+1][("</s>","</s>")]` das Tag $t$ des letzten Wortes extrahieren.

Das Ergebnis sollte im folgenden Format mit Tabulatoren als Trennzeichen auf den Bildschirm ausgegeben werden:

| | |
|---|---|
| Das | PDS |
| ist | VAFIN |
| ein | ART |
| Satz | NN |
| . | $. |

**Programmaufruf:** python3 tagger.py params.pickle input.txt

Schritte:

- Speichern Sie zwei deutsche Beispielsätze in einer Datei (tokenisiert, 1 Satz pro Zeile).

- Schreiben Sie eine Funktion `backoff_factors`, welche ein Dictionary of Dictionaries mit Wahrscheinlichkeiten als Eingabe erhält und ein Dictionary mit Backoff-Faktoren zurückliefert.

  backoff[context] = 1 - $\sum_t$ prob[context][t]

- Einlesen der Parameterdatei und Berechnung der Backoff-Faktoren mit Hilfe der Funktion `backoff_factors`.

- Funktion `get_suffix(word)` aus der letzten Übungsaufgabe kopieren.

- Funktion `compute_suffix_prob(suff,tag)`, welche für ein Suffix suff=$a_1, ..., a_k g$ und ein Tag tag rekursiv die Wahrscheinlichkeit berechnet:

$$\begin{aligned} p(tag|a_1, ..., a_k, g) &= p^*(tag|a_1, ..., a_k) + \alpha(a_1, ..., a_k, g)\ p(tag|a_2, ..., a_k, g) \\ p(tag|g) &= p^*(tag|g) \end{aligned}$$

  $g$ ist hier ein Buchstabe für die Groß-/Kleinschreibung des Wortes und $p^*$ sind die Suffix-Tag-"Wahrscheinlichkeiten" aus der Parameterdatei.

- Funktion `compute_word_prob(word,tag)`, welche mit Hilfe der Funktionen `get_suffix` und `compute_suffix_prob` für ein Wort und ein Tag die bedingte Wahrscheinichkeit $p(tag|word)$ berechnet.

  $p(tag|word) = p^*(tag|word) + \alpha(word)\ p(tag|get\_suffix(word))$

- Funktion `lex_probs`, welche unter Verwendung der Funktion `compute_word_prob` für ein Wort die Menge der Tags mit einer Wahrscheinlichkeit größer als 0.001 ermittelt und die entsprechenden Tags und ihre Wahrscheinlichkeiten in einem Dictionary speichert und zurückgibt.

- Funktion `context_prob`, welche für ein Tag-Trigramm seine geglättete Kontextwahrscheinlichkeit liefert.

- Funktion `viterbi`, welche für einen als Token-Liste gespeicherten Satz die wahrscheinlichste Tagfolge liefert

- Hauptfunktion, welche erst die Parameterdatei einliest und dann die Eingabedatei Zeile für Zeile annotiert.