Übung 7

Wortart-Tagger (Training)

Laden Sie das manuell getaggte Korpus an der Adresse `www.cis.uni-muenchen.de/~schmid/lehre/data/tiger.txt.gz` herunter und dekomprimieren Sie es. Die erste Spalte der Datei enthält die Wörter, die zweite Spalte die Tags und auf jeden Satz folgt eine Leerzeile.

Schreiben Sie ein Programm, welches das Korpus einliest und die Parameter eines Trigramm-HMMs schätzt. Das verwendete Modell habe folgende Form

$$p(w_1^n, t_1^n) \propto \prod_{i=1}^{n+1} p(t_i|t_{i-2}, t_{i-1}) p(t_i|w_i)/p(t_i)$$

mit $t_{-1} = t_0 = \langle s \rangle$ und $t_{n+1} = \langle /s \rangle$ (wobei $\propto$ "proportional zu" bedeutet).

Die lexikalischen Wahrscheinlichkeiten wurden also (wie in der Vorlesung beschrieben) mit der Bayes'schen Regel ,,umgedreht".

Schätzen Sie die Parameter zu folgenden Wahrscheinlichkeiten:

- Apriori-Tagwahrscheinlichkeiten $p(t) = f(t)/\sum_{t'} f(t')$
- 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.
- lexikalische Wahrscheinlichkeiten $p(t|w)$

  Glätten Sie diese Wahrscheinlichkeiten ebenfalls mit Kneser-Ney-Glättung, wobei suffixbasierte Wahrscheinlichkeiten als Backoff-Wahrscheinlichkeiten genommen werden. Falls $w = a_1...a_n$ (d.h. w besteht aus den Buchstaben $a_1...a_n$), dann:

$$\begin{aligned} p(t|w) &= \frac{f(w,t) - \delta_6}{\sum_{t'} f(w,t')} + \alpha(w) p(t|a_{n-4}^n g(w)) \\ p(t|b_1^k g) &= \frac{f(b_1^k g, t) - \delta_k}{\sum_{t'} f(b_1^k g, t')} + \alpha(b_1^k g) p(t|b_2^k g) \text{ für } 0 < k < 5 \\ p(t|g) &= \frac{f(g,t)}{\sum_{t'} f(g,t')} \end{aligned}$$

$g(w)$ ist eine Funktion, welche bei kleingeschriebenen Wörtern ,,k", bei großgeschriebenen Wörtern ,,g", und bei allen übrigen Wörtern ,,0" liefert. $f(t, b_1^k g)$ ist die Zahl der

Wörter mit Endung b_1^k und Groß-/Kleinschreibung g, die mit t getaggt sind. Wenn ein Wort kürzer als die maximale Suffixlänge ist, füllen Sie es am Anfang mit Leerzeichen auf.

Schritte:

- Einlesen der Trainingsdaten aus einer Datei und Extraktion der Häufigkeiten.
  - Wort-Tag-Häufigkeiten.
    Wird das Wort w mit dem Tag t gesehen, so wird seine Häufigkeit mit dem Befehl word_tag_freq[w][t] += 1 erhöht. word_tag_freq ist also ein Dictionary of Dictionaries.
  - Tag-Trigramm-Häufigkeiten
    Hier müssen Sie bei jedem Satz am Beginn und Ende der Tagfolge 2 Grenztags hinzufügen.[1] Wenn Sie das Trigramm t1,t2,t3 sehen, erhöhen Sie mit dem Befehl tag_ngram_freq[t1,t2][t3] += 1 seine Häufigkeit. Auch hier verwenden Sie also ein Dictionary of Dictionaries.
- Funktion *get_suffix*, welche beispielsweise für das Wort *rot* und Suffixlänge n=5 das Ergebnis "_ _rotk" liefert, wobei _ für ein Leerzeichen steht und "k" die Kleinschreibung repräsentiert.
- Berechnung der Suffix-Tag-Häufigkeiten durch Summation von Wort-Tag-Häufigkeiten, wobei n die maximale Länge der Suffixe ist. Der Parameter n wird zu Programmbeginn z.B. auf den Wert 5 gesetzt. Speichern Sie auch hier in ein Dictionary of Dictionaries.
- Funktion *compute_discount*, welche eine Häufigkeitstabelle $freq[context][tag]$ als Argument erhält und den Discount berechnet und zurückgibt.
  Spezialfall: Wenn die Zahl der zweimal aufgetretenen Tupel gleich oder größer der Zahl der einmal aufgetretenen Tupel ist, soll der Discount 0.5 zurückgegeben werden.
- Funktion *estimate_probs*, welche eine Häufigkeitstabelle $freq[context][tag]$ als Argument erhält, die Funktion *compute_discount* aufruft und eine Tabelle mit relativen Häufigkeiten $p^*(tag|context) = (f(context, tag) - \delta) / \sum_{t'} f(context, t')$ zurückgibt
- Funktion *reduced_context_freqs*, welche eine Häufigkeitstabelle context_tag_freq als Argument erhält, und die Häufigkeiten kleinerer Kontexte berechnet. Sie iteriert über alle Kontexte c und alle Tags t und erhöht dann die Häufigkeit mit red_context_tag_freq[c[1:]][t] += 1 (Kneser-Ney-Backoff)

  Bei Verwendung von Standard-Backoff würde der Befehl stattdessen lauten: red_context_tag_freq[c[1:]][t] += context_tag_freq[c][t]

[1] Wir fügen hier auf beiden Seiten gleich viele Grenz-Tags hinzu, weil dies die Implementierung des Viterbi-Algorithmus vereinfacht.

- Berechnung der Wort-Tag-Wahrscheinlichkeiten (unter Verwendung von estimate_probs) und der Suffix-Tag-Wahrscheinlichkeiten (in einer Schleife über alle Suffixlängen unter Verwendung von estimate_probs und reduced_context_freqs zur Berechnung der nächstkleineren Suffixe). Bei leerem Suffix (wenn nur noch das Symbol für die Groß-/Kleinschreibung übrig ist) sollte nicht mehr geglättet werden.

- Berechnung der Apriori-Tag-Wahrscheinlichkeiten $p(t) = freq(t) / \sum_{t'} freq(t')$
  Zunächst werden die Taghäufigkeiten durch Summation von Tag-Trigramm-Häufigkeiten berechnet: freq[tag] += tag_ngram_freq[context][tag]

- Berechnung der Tag-nGramm-Wahrscheinlichkeiten in einer Schleife über alle n-Gramm-Größen unter Verwendung von estimate_probs und reduced_context_freqs zur Berechnung der nächstkleineren n-Gramme. Bei den Unigrammen darf nicht geglättet werden.

- Speicherung aller fürs Taggen benötigten Parameter in einer Datei mit pickle.