Schriftliche Wiederholungsprüfung zur Übung
Statistische Methoden in der maschinellen Sprachverarbeitung
WS 2016/17
Dozent: Helmut Schmid

Aufgabe 1) Eine Evaluierung dient dazu, die Genauigkeit eines trainierten Modelles zu bestimmen. Welche Schritte führt man bei einer Evaluierung aus? Was braucht man dafür? Wozu dient eine Baseline? Welche Bewertungsmaße verwendet man bei der Evaluierung (dem Vergleich) von Sprachmodellen, HMM-Taggern bzw. Parsern jeweils? (5 Punkte)

Aufgabe 2) Viele statistische Modelle besitzen Metaparameter (bspw. die Größe des Kontextes bem HMM-Taggen oder die Lernrate beim CRF-Training), die der eigentliche Lernalgorithmus nicht optimieren kann. Erklären Sie detailliert, wie man gute Werte für diese Metaparameter bestimmt. Was braucht man dazu? (3 Punkte)

Aufgabe 3) Bei der Evaluierung berechnet man oft die *Signifikanz* des Ergebnisses. Warum macht man das und was sagt die Signifikanz aus? Beschreiben Sie detailliert, wie man die Signifikanz einer Wortart-Tagger-Evaluierung mit einem Vorzeichentest (Binomialtest) berechnet. Warum kann auch ein signifikant besseres System in Wirklichkeit schlechter sein? (4 Punkte)

Aufgabe 4) Beim unüberwachten Training (Training ohne annotierte Daten) eines HMM-Taggers mit dem EM-Algorithmus wird der Forward-Backward-Algorithmus verwendet.

- Erklären Sie wozu der Forward-Backward-Algorithmus hier dient. Wird er im E-Schritt oder im M-Schritt benutzt?
- Begründen Sie, warum der Viterbi-Algorithmus hier nicht verwendet werden kann.
- Bei welchem Modell wird der Forward-Backward-Algorithmus beim *überwachten* Training eingesetzt? (Es handelt sich dabei nicht um EM-Training.)
- Welcher Algorithmus entspricht dem Forward-Backward-Algorithmus beim unüberwachten Training von PCFGs?
- Welcher Parser setzt diesen Algorithmus beim überwachten Training ein?

(5 Punkte)

Aufgabe 5) Erklären Sie wie man mit Hilfe von Markowmodellen einen Sprachidentifizierer realisieren kann, der die Sprache eines Textes bestimmt. (3 Punkte)

Aufgabe 6) Ein Bigramm-HMM-Tagger berechnet die beste Tagfolge mit dem Viterbi-Algorithmus. Die Viterbi-Wahrscheinlichkeiten sind wie folgt definiert:

$$\begin{aligned}\delta_t(0) &= \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \delta_t(k) &= \max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \text{ für } 0 < k \leq n+1 \end{aligned}$$

Dabei sind t, t' Tags, k eine Wortposition, $\delta_t(k)$ eine Viterbiwahrscheinlichkeit und $\psi_t(k)$ das beste Vorgängertag von t an Position k.

Schreiben Sie eine Funktion *viterbi(words)*, welche die Viterbi-Wahrscheinlichkeiten berechnet. Das Argument *words* ist die Liste der zu annotierenden Wörter. Sie können die Viterbiwahrscheinlichkeiten in einer Liste (Array) von Hashtabellen (Dictionaries) mit Namen *vitprob* speichern. Eine Funktion *lexprobs(word)* liefert eine Hashtabelle (oder Dictionary) mit den möglichen Tags von *word* als Schlüsseln und den entsprechenden Wahrscheinlichkeiten als Werten. Eine Funktion *contextprob(tag1,tag2)* gibt die Wahrscheinlichkeit zurück, dass tag2 auf tag1 folgt. Beide Funktionen sind gegeben und müssen nicht implementiert werden.

Hier ist Pseudocode für die zu implementierende Funktion:

```
Viterbitabelle initialisieren
für alle Positionen i von 0 bis zur Länge des Satzes
    für alle möglichen Tags des i-ten Wortes
        für alle Tags in vitprob[i]
            Berechne das Produkt aus lexikalischer Wahrscheinlichkeit,
            Kontextwahrscheinlichkeit und Viterbiwahrscheinlichkeit
            des Vorgängertags und maximiere.
```

Wie müssten Sie das Programm noch erweitern, um tatsächlich die beste Tagfolge berechnen zu können? (Sie müssen das nicht programmieren.)

(10 Punkte)

(30 Punkte insgesamt)