Vorlesung "Computerlinguistische Techniken"

5. Übung (24.11.2015)

Wintersemester 2015/16 – Prof. Dr. Alexander Koller

In dieser Übung implementieren Sie einen vollständigen Part-of-Speech-Tagger auf der Basis von Hidden Markov Models.

Bitte verwenden Sie in dieser Übung keine Klassen aus NLTK. Damit Sie sich auf die wesentlichen Probleme konzentrieren können, habe ich allerdings schon einige nützliche Python-Funktionen und -Klassen in der Datei `hmm.py` (im Piazza) für Sie vorbereitet.

1 Viterbi-Algorithmus

Wir gehen in drei Schritten vor. Repräsentieren Sie zunächst das HMM aus der Vorlesung (Jason Eisners Eiscreme) in geeigneten Python-Dictionaries. Stellen Sie alle Parameter des Modells (Anfangswahrscheinlichkeiten, Übergangswahrscheinlichkeiten und Ausgabewahrscheinlichkeiten) als log-Wahrscheinlichkeiten dar. Die Basis des Logarithmus können Sie beliebig (aber einheitlich!) wählen.

Implementieren Sie dann den Viterbi-Algorithmus und berechnen Sie damit die wahrscheinlichste Tag-Sequenz für die Beobachtungen (3, 1, 3). Sie können dazu in zwei Teilschritten vorgehen: (a) Berechnung der Viterbi-Wahrscheinlichkeiten $V_t(q)$; (b) Erweiterung Ihrer Datenstruktur um Backpointers und Ausgabe der wahrscheinlichsten Tagsequenz. Beachten Sie, dass Ihr Algorithmus die log-Wahrscheinlichkeiten addieren muss (anstatt, wie in der Vorlesung, Wahrscheinlichkeiten zu multiplizieren).

2 POS-Tagging

Nun wenden wir Ihre Viterbi-Implementierung auf echte Daten an. Lesen Sie den Quelltext von `hmm.py`. Dort wird beschrieben, wie Sie die Klasse `HMM` verwenden können, um mit Maximum-Likelihood-Schätzung die Parameter eines HMM aus annotierten Daten zu schätzen, und wie Sie diese dann aus einem trainierten Modell wieder auslesen können.

Passen Sie Ihre Implementierung des Viterbi-Algorithmus aus Aufgabe 1 so an, dass sie die Parameter aus einem `HMM`-Objekt verwendet. Sie werden mit realen Daten auf das Problem stoßen, dass mehrere Vorzustände (für die Stringposition $t-1$) zum gleichen maximalen Wert von $V_t(q)$ führen können. In diesem Fall können Sie einen beliebigen dieser Vorzustände als Backpointer nehmen. Wenn alle Vorzustände zu einem Wert $V_t(q) = 0$ führen (d.h. $\log V_t(q) = -\infty$, z.B. wegen unbekannter Wörter), setzen Sie einen Backpointer zum Zustand mit der höchsten Unigramm-Wahrscheinlichkeit.

Trainieren Sie dann ein HMM auf allen Sätzen aus dem Brown-Korpus außer den letzten 100 (= Trainingskorpus), und evaluieren Sie die Accuracy Ihres Systems auf den letzten 100 Sätzen (= Testkorpus). Verwenden Sie die geeigneten Methoden aus `hmm.py`, um das Brown-Korpus vorzuverarbeiten und die Evaluation durchzuführen. Rechnen Sie damit, dass die Evaluation eine Weile dauern wird. Geben Sie die Accuracy an, die Ihr System erzielt.

3 Smoothing

Implementieren Sie schließlich Add-One-Smoothing für die Übergangs- und Ausgabewahrscheinlichkeiten. Sie können dazu die Anweisung `pass` in Zeile 78 von `hmm.py` durch Ihren eigenen Code ersetzen, der die Wahrscheinlichkeiten für gesehene und ungesehene Parameter berechnet. Lesen Sie dazu auch in den Funktionen `get_transition_prob` und `get_emission_prob` nach, in welcher Form diese Funktionen diese Daten erwarten.

Schätzen Sie dann Ihr HMM neu (nachdem Sie mit `HMM(True)` ein HMM konstruiert haben, das Smoothing durchführt) und evaluieren Sie neu. Geben Sie die Accuracy an, die Ihr System mit Smoothing erzielt.

Abgabe bis 08.12.2015, vor der Vorlesung, per Mail an `mgerdes@uni-potsdam.de`