

LC-CRF-Wortart-Tagger 2: Optimierung und Tagging

Optimierung des LC-CRF-Tagger-Trainings

In der letzten Aufgabe haben Sie einen CRF-Tagger trainiert. Nun sollen Sie das Trainingsprogramm verbessern durch

- Vermeiden Sie Underflow beim Forward-Backward-Algorithmus durch logarithmierte Darstellung der Zahlen. Wegen $\log(\exp(f)) = f$, berechnen Sie nun $\alpha_t(i-1) + s(t', t, \mathbf{w}, i)$ statt $\alpha_t(i-1) \cdot \exp(s(t', t, \mathbf{w}, i))$. Die Initialisierung müssen Sie entsprechend anpassen. Die Additionen im Forward-Backward-Algorithmus ersetzen Sie durch die logsumexp-Operation $\text{logsumexp}(a, b) = \log(\exp(a - m) + \exp(b - m)) + m$ mit $m = \max(a, b)$. Die γ -Werte berechnen Sie bspw. mit $\gamma_t(i) = \exp(\alpha_t(i) + \beta_t(i) - \text{alpha}_{\setminus s}(n+1))$, wobei n die Zahl der Wörter ohne Grenzsymbole ist.

- Beschleunigung durch separate Berechnung eines lexikalischen Scores und separate Berechnung von erwarteten lexikalischen Merkmalshäufigkeiten.

- Beschleunigung durch Pruning an drei Stellen:

Im Forward-Algorithmus berechnen Sie die lexikalischen Scores aller Tags an der aktuellen Position und iterieren dann nur über die Tags mit einem Score größer dem maximalen Score plus $\log(\text{Schwellwert})$ (wobei der Schwellwert bspw. bei 0.001 liegt).

Wenn Sie über alle Tags an Position i iteriert haben, dann eliminieren Sie in `forward[i]` alle Tags, deren Forward-Score (logarithmierte Darstellung) unter dem maximalen Forward-Score plus $\log(\text{Schwellwert})$ liegt. Schreiben Sie eine Funktion `prune`, die Sie bei beiden Pruning-Aufgaben aufrufen.

Der Backward-Algorithmus iteriert an Position i nur über die Tags, die in `forward[i]` eingetragen sind.

- Beschleunigung durch Caching:

Die Berechnung der Merkmalsvektoren und der Scores ist rechenintensiv und erfolgt mehrfach. Sie können das Programm beschleunigen, indem Sie diese Werte in Caches speichern. Als Key dient jeweils das Argument-Tag(-Paar) plus Position. Die Caches werden nach jedem Satz gelöscht, um den Speicherplatzbedarf zu begrenzen.

- L_1 -Regularisierung

- Evaluation der Tagging-Genauigkeit auf den Developmentdaten nach jeder Epoche und Speicherung des besten Gewichtsvektors

- neuer Aufruf: **crf-train.py train-file dev-file param-file**

Implementierung eines Taggerprogrammes

Außerdem sollen Sie ein Programm schreiben, welches die gespeicherten Parameter einliest und dann Eingabesätze mit dem Viterbi-Algorithmus annotiert. Die Eingabesätze werden

aus einer Datei eingelesen, die ein Wort pro Zeile enthält, wobei auf jeden Satz eine Leerzeile folgt. Die Ausgabe erfolgt im gleichen Format wie die Trainingsdaten auf den Bildschirm. Sie sollten auch hier Pruning anwenden. (Caching ist hier nutzlos.)

Aufruf: **crf-annotate.py param-file text.txt**

Vorüberlegungen

- Wie implementieren Sie die Evaluation?
- Wie kann die L_1 -Regularisierung effizienter gemacht werden?
- Welche Teilaufgaben umfasst das Taggerprogramm?

Optimieren Sie den L_1 -Regularisierungsfaktor und die Lernrate auf Developmentdaten und berechnen Sie zum Schluss die Tagginggenauigkeit auf den Testdaten. Auf den Developmentdaten sind über 97% Genauigkeit möglich. Das Training kann über eine Stunde dauern.

Schicken Sie alle Programme, die optimalen Metaparameter und die erzielte Genauigkeit auf den Testdaten an `schmid@cis.lmu.de`.