Übung zur Vorlesung "Computerlinguistik II / Sprachtechnologie"

Sommersemester 2018, Prof. Dr. Udo Hahn, Tobias Kolditz
Übungsblatt 3 vom 21.05.2018
Abgabe bis 28.05.2018 per E-Mail (Python-Datei) an
tobias.kolditz@uni-jena.de

Aufgabe 1 : Bigramm-Sprachmodell in Python implementieren **10 pt**

In der letzten Vorlesung wurde die Markov-Annahme vorgestellt. Danach können wir die Wahrscheinlichkeit $P(w_1^n)$ einer Kette von Wörtern $w_1 w_2 \dots w_n$ wie folgt approximieren (Markov-Modell erster Ordnung):

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1}) \tag{1}$$

In der Vorlesung wurde w_0 als spezielles Symbol "<start>" definiert. Im Folgenden gehen wir der Einfachheit halber davon aus, dass w_0 das erste Wort der Kette ist, also z.B. für "Das ist ein Satz." $w_0 =$ "Das", $w_1 =$ "ist", ..., $w_n =$ ".". Damit besteht das Produkt in (1) ausschließlich aus konditionalen Wahrscheinlichkeiten – wir ignorieren $P(w_0)$[1].

Sie werden ein auf Gleichung (1) basierendes Sprachmodell implementieren, das einer beliebigen Wortkette eine Wahrscheinlichkeit zuweist. Nutzen Sie dazu den Starter-Code in `uebung_3.py`.

a) Unigramme und Bigramme zählen **2 pt**

Vervollständigen Sie die `update_counts`-Funktion des `ProbabilityEstimators`. Hier sollen, wie in Übungsblatt 2, Unigramme und Bigramme gezählt werden. Wir iterieren dazu über alle Bigramme und zählen mithilfe zweier Python-`dictionaries` (Hash-Tabellen) alle verschiedenen Bigramme und Unigramme (jeweils das erste Token jedes Bigramms) auf einmal.

Hinweis: Beim Zählen mit `dictionaries` gibt es zwei Fälle zu beachten – der Key existiert bereits, oder er existiert (noch) nicht. Diese beiden Fälle müssen auch bei jedem Zugriff auf im dictionary gespeicherte Werte beachtet werden!

b) Maximum Likelihood Estimates (MLE) **1 pt**

Vervollständigen Sie die `mle_probability`-Funktion des `ProbabilityEstimators`. Hier sollen die Wahrscheinlichkeiten wie in Aufgabe 2, Übungsblatt 2 berechnet werden.

Für ein Unigramm a, das im Trainings-Korpus nicht beobachtet wurde, ist die Wahrscheinlichkeit $P(b|a)$ nach der Formel des letzten Übungsblatts nicht definiert (Division durch Null). In diesem Fall soll die Funktion 0 zurückgeben.

c) Additive Smoothing **2 pt**

In der Vorlesung wurde angesprochen, dass wir mit Gleichung (1) ein Problem bekommen können, wenn wir als Faktoren unsere MLE-Wahrscheinlichkeiten nutzen. (Was ist das Problem?)

Eine Technik zur Umgehung dieses Problems ist das *Smoothing* (die 'Glättung' von Wahrscheinlichkeiten). Die einfachste Art des Smothings ist das *Additive Smoothing* (auch *Laplace Smoothing*):

$$P_{\text{add}}(w_k | w_{k-1}) = \frac{C_2(w_{k-1} w_k) + \delta}{C_1(w_{k-1}) + \delta |V|} \tag{2}$$

wobei C_1 und C_2 definiert sind wie auf Übungsblatt 2, δ eine Konstante und $|V|$ die Vokabulargröße ist. Nutzen Sie Gleichung (2), um die `smoothed_probability`-Funktion des `ProbabilityEstimators` zu vervollständigen.

---

[1] Wir berechnen also $P(w_1^n) \approx P(w_1|w_0) \cdot P(w_2|w_1) \cdot \ldots \cdot P(w_n|w_{n-1})$.

d) Perplexität 3 pt

Die Perplexität $PP(w_1^n)$ ist ein simples Maß dafür, wie gut unser Sprachmodell eine Kette von Wörtern (Testdaten) vorhersagen kann:

$$PP(w_1^n) = \left(\prod_{k=1}^{n} P_{\text{add}}(w_k | w_{k-1}) \right)^{-\frac{1}{n}} \tag{3}$$

Da wir viele sehr kleine Werte multiplizieren, ist es besser, für die Implementierung folgende äquivalente Formel zu verwenden (Vermeidung von Underflow):

$$PP(w_1^n) = \exp(-\frac{1}{n} \sum_{k=1}^{n} \log P_{\text{add}}(w_k | w_{k-1})) \tag{4}$$

Nutzen Sie Gleichung (4), um die `compute_perplexity`-Funktion zu vervollständigen. Vergleichen Sie die Ergebnisse für verschiedene δ-Werte mit der Funktion `compare_deltas`.

Hinweis: Für $\exp$ und $\log$ können Sie die Funktionen `math.exp(...)` und `math.log(...)` aus Pythons `math`-Module nutzen.

e) Wahrscheinlichkeit eines Satzes 2 pt

Vervollständigen Sie die `compute_prob`-Funktion. Diese Funktion berechnet die Wahrscheinlichkeit $P(w_1^n)$ einer Zeichenkette $w_1 w_2 \dots w_n$ nach Gleichung (1). Da wir hier wieder sehr kleine Werte multiplizieren, berechnen Sie statt eines Produkts zunächst die Summe der Log-Wahrscheinlichkeiten und wenden auf das Ergebnis die Exponentialfunktion an:

$$\hat{P}(w_1^n) = \exp(\sum_{k=1}^{n} \log P_{\text{add}}(w_k | w_{k-1})) \tag{5}$$

Führen Sie das Skript aus (`python3 uebung_3.py`) – alle `assertions` sollten erfüllt sein. Es sollten keine Fehler mehr auftreten.