Übung 06: Automatische Sprachidentifizierung

Programmiertechniken in der Computerlinguistik II, FS 17

Abgabedatum: 23. Mai 2017, 23:59

Hinweise zur Abgabe

- Bitte gib ausschliesslich die von dir vervollständigte Datei `charlm.py` (siehe 1.1) und eine PDF-Datei mit deiner Evaluation (siehe 1.2) ab.
- In dieser Übung musst du relativ wenig – d.h. ungefähr 20–30 Zeilen – Code schreiben. Trotzdem ist sie anspruchsvoll, weil du dich (i) in ein bestehendes Code-Gerüst einarbeiten und (ii) n-Gramm-Modelle und Laplace-Smoothing genau verstehen musst. Es lohnt sich darum, anfangs Zeit in die Analyse des bereitgestellten Codes zu investieren. Zögere bei Unklarheiten keinesfalls, die Tutoren zu kontaktieren.
- Geize nicht mit Kommentaren direkt im Programm-Code, wo Erläuterungen angebracht sind. Umfangreiche Erklärungen werden hingegen besser in einer separaten README-Datei mitgeliefert (vorzugsweise Plain-Text oder PDF).
- Halte dich an die Vorgaben des Python Style Guide! Grobe Verstösse werden mit Punkteabzug geahndet.
- Um das Hochladen der Abgabe auf OLAT zu erleichtern, kannst du die Dateien mit `zip` (oder einem anderen verbreiteten Format) archivieren / komprimieren.

1 Automatische Sprachidentifizierung

N-Gramm-Modelle haben sich insbesondere für die automatischen Sprachidentifierzung bewährt. Hierbei wird für jede Sprache, die später erkannt werden soll, ein Sprachmodell auf Zeichenebene trainiert: Die Symbole in unserem Modell sind also nicht Wörter, sondern einzelne Buchstaben. Die Sequenz aller Buchstaben eines zu klassifizierenden Strings S wird dann mit jedem Modell «bewertet»: Die Wahrscheinlichste Sprache ist diejenige, deren Modell für S die geringste Perplexität aufweist.

Warum funktioniert das? Buchstabensequenzen wie beispielsweise *sch* kommen in deutschen (de) Texten häufiger vor als in englischen (en). Entsprechend ordnen Sprachmodelle solchen Buchstabensequenzen unterschiedliche Wahrscheinlichkeiten zu:

$$P_{de}(h|sc) > P_{en}(h|sc) \tag{1}$$

Deine Aufgabe besteht darin, einen auf Buchstaben-n-Grammen basierenden Sprachklassifikator zu implementieren und zu evaluieren.

1.1 Implementierung

Im Aufgabenordner findest du ein Grundgerüst für n-Gramm-Modelle auf Zeichenebene (`charlm.py`). Implementiere in dieser Datei das Lernen der n-Gramm-Wahrscheinlichkeiten in der `train`-Methode. Die Stelle ist mit `#TODO` im Quellcode markiert.

Wie du in `main.py` siehst, wird für die automatische Sprachklassifikation je ein `CharLM`-Modell pro Sprache trainiert. Diese werden dazu in eine `LanguageIdentifier`-Instanz geladen, welche für einen zu klassifizierenden String die Perplexität aller Sprachmodelle abfragt und dann die wahrscheinlichste Sprache (d.h. geringste Perplexität) zurückgibt.

In den Dateien `main.py` und `identifyer.py` musst (und sollst) du nichts verändern. Du gibst für diesen Teil lediglich deine angepasste `charlm.py`-Datei ab.

Smoothing

Verwende Laplace-Smoothing, um Null-Wahrscheinlichkeiten für unbekannte n-Gramme in der Anwendungsphase zu vermeiden. Für Bigramme ($n = 2$) ergeben sich die bedingten Wahrscheinlichkeiten durch

$$P^*_{Laplace}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + V} \tag{2}$$

V ist die Anzahl verschiedener n–1-Gramme, die in den Trainingsdaten vorkommen. Für Bigramm-Modelle zählen wir also die Anzahl verschiedener Unigramme in den Trainingsdaten (was unserem Vokabular entspricht); für Trigramm-Modelle zählen wir die Anzahl verschiedener Bigramme; usw.

Unbekannte n-Gramme

Im bereitgestellten Code-Gerüst musst du dich nicht um die Ersetzung von unbekannten Zeichen mit Spezialsymbolen kümmern. Wahrscheinlichkeiten für in den Trainingsdaten ungesehene n-Gramme musst Du jedoch berechnen und mit den vorhandenen Methoden speichern.

Wir unterscheiden zwei Fälle. Zur Illustration sei *head* das letzte Zeichen eines n-Gramms, und *history* dessen $n-1$ vorhergehende Zeichen.[1] Die Wahrscheinlichkeit von n-Grammen, deren history nicht in den Trainingsdaten vorkommt, berechnest du als

$$P^*_{Laplace}(\text{head}|\text{history}) = \frac{1}{V} \tag{3}$$

und speicherst sie mit der Methode `self._set_unk_given_unknown_history`. Der zweite Fall trifft auf n-Gramme zu, deren history zwar in den Trainingsdaten vorkommt, aber nie in Kombination mit dem aktuellen head. Gemäss Laplace-Smoothing berechnest du diese Wahrhrscheinlichkeit als

$$P^*_{Laplace}(\text{head}|\text{history}) = \frac{1}{count(\text{history}) + V} \tag{4}$$

und speicherst sie history mit der Methode `self._set_unk_given_known_history`.

Weitere Hinweise

- Deine Implementation soll, wie im Grundgerüst angedeutet, für Modelle beliebiger Ordnungen (bigramme, trigramme, usw.) einsetzbar sein.
- Beachte die Hinweise auf den Vorlesungsfolien zu Methoden und Datenstrukturen im Umgang mit n-Grammen.
- Transformiere alle Wahrscheinlichkeiten mittels `self.log` in den logarithmischen Zahlenraum, bevor du sie speicherst.

[1] Beispiel: Das 5-Gramm *schwe* besteht aus dem head *e* und der history *schw*.

1.2 Evaluation

Wie gut funktioniert unser Sprachidentifikator? Als zweiter Teil deiner Abgabe erwarten wir eine Evaluation mit von dir frei gewählten Modellparametern und Testdaten.

Sprachen

Wähle für deine Evaluation mindestens zwei der fünf Sprachen aus, für welche Trainingsdaten im Aufgabenordner verfügbar sind (cs, de, en, es, fr).

Modelle

Trainiere mindestens zwei unterschiedliche Modelle pro Sprache auf diesen Trainingsdaten, z.B. je ein 3- und ein 5-Gramm-Modell. Du kannst auch andere Parameter anpassen; interessant wäre d.h. die Wahl einer alternativen Smoothing-Methode (s. Jurafsky und Martin, 2017, Kapitel 4.4).

Testdaten

Stelle ein Testset mit mindestens 100 Sätzen pro Sprache zusammen. Diese Sätze entnimmst du *nicht* den Trainingsdaten im `data`-Verzeichnis, sondern einer anderen Quelle deiner Wahl. Der Anteil aller Sprachen sollte im Testset ungefähr gleich gross sein (genaue Anzahl bitte angeben). Die Sätze in den Trainingsdaten stammen aus Zeitungstexten. Interessant sind darum sowohl Testdaten aus der gleichen Domäne (z.B. Sätze aus Zeitungsartikeln) als auch ganz Anderes: Wie gut funktioniert unser Sprachidentifikator z.B. mit Twitter-Statements?

Python-Skripte für die Evaluation

Für die Evaluation empfiehlt es sich, ein kleines Skript zu in der Art von `main.py` zu schreiben, damit die Testsätze nicht manuell in die Konsole eingegeben und die Resultate händisch ausgewertet werden müssen. Die Abgabe deines/deiner für die Evaluation verwendeten Skripte ist freiwillig.

Abgabe

In der als PDF-Datei abzugebenden Evaluation erwarten wir im Minimum

- eine ganz klare Beschreibung deines Experiments
  - Welche Sprachen wurden berücksichtigt?
  - Welche Modellparameter wurden verglichen?
  - Woher stammen deine Testdaten, und welche Charakteristiken weisen sie auf? Wieviele Testsätze hast du pro Sprache verwendet?
- die Ergebnisse
  - Minimum: Angabe, wieviele Sätze des Testsets der korrekten Sprache zugeordnet worden sind (d.h. Accuracy)
  - Optional: Precision-/Recall-/F-Werte
- Eine kurze, persönliche Schlussfolgerung

Reflexion/Feedback

a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen zusammen.

b) Wie viel Zeit hast du in diese Übungen investiert?