Aufgabenblatt 3 – 01.05.2024

Programmieren II: Fortgeschrittene (Python) Abgabe bis 10.05.2024

In diesem Aufgabenblatt lesen wir Textdateien ein, um einen Unigramm-Tagger zu trainieren und diesen schließlich zu evaluieren.

Verwende für das ganze Blatt keine Bibliotheken, die dir "Arbeit abnehmen" (`collections`, `nltk`, `pandas` etc. verboten; `typing`, `abc`, `mylocalmodule` erlaubt).

Definiere lediglich Methoden, keine "normalen" Funktionen.

Tipp: Du arbeitest auf diesem Aufgabenblatt hauptsächlich in Dateien, die du am Ende als Modul benutzen willst. Um deinen Fortschritt zu überprüfen, kannst du die Datei aber als Script ausführen.

Aufgabe 1) Corpus einlesen **Punkte: 2**

Die Datei `train.tsv` ist eine tabulatorgetrennte Datei (TSV), in der jede Zeile ein annotiertes Wort repräsentiert, wobei Wortform und PoS-Tag durch Tabulatoren voneinander getrennt sind. Satzenden sind durch eine Leerzeile markiert.

In der Datei `corpus_utils.py` findest du eine Klasse namens `Corpus` sowie die Klassen `Sentence` und `Token`. Implementiere für `Corpus` die folgende Funktionalität:

- Das Korpus muss natürlich irgendwie eingelesen werden. Überlege selbst, wie genau und wann das ablaufen soll. *(0.5 Punkte)*
- `len(corpus_instance)` sollte die Anzahl der Sätze zurückgeben. *(0.5 Punkte)*
- `corpus_instance[5]` sollte den 6. Satz aus dem Korpus zurückgeben. *(0.5 Punkte)*
- Es sollte möglich sein, über die Sätze im Korpus zu iterieren (`[sent for sent in corpus_instance]`). *(0.5 Punkte)*

Benutze in deiner Implementation die Klassen `Sentence` und `Token` (bzw. Instanzen dieser Klassen), um Tokens und Sätze zu repräsentieren.

Aufgabe 2) Unigramm-Tagger **Punkte: 4.5**

- Schreibe die Methode `train -> None` in der Klasse `UnigramTagger` in der Datei `tagger_models.py`.
  - `train` soll ein `Corpus` erhalten und den Unigramm-Tagger trainieren, sodass

er auch bei vorher ungesehenen Texten Vorhersagen machen kann.

  - Ein Unigramm-Tagger weist einer Wortform während der Inferenz (d. h. nach dem Training) dasjenige PoS-Tag zu, das während des Trainings am häufigsten vorkam.[1]

  - Die Aufgabe von `train` besteht also darin, zu zählen, mit welchen PoS-Tags die Wortformen auftreten und wie häufig diese PoS-Tags sind. Erinnerung: Es sollen keine Bibliotheken importiert werden, die diese Aufgabe erleichtern können.

(2 Punkte)

- Schreibe die Methode `tag(self, sentence) -> Sentence` in der Klasse `UnigramTagger`.

  - Während der Inferenz sind die PoS-Tags nicht bekannt. Daher bekommt diese Methode einen Satz ohne die entsprechenden Label übergeben (bzw. das Label ist `None`).

  - Die Methode soll die Vorhersagen des Modells benutzen, um den annotierten Satz zurückzugeben.

  - Es kann während der Inferenz passieren, dass Wortformen auftreten, die der Tagger während des Trainings nicht gesehen hat. Überlege dir eine (einfache) Strategie, mit diesen Fällen umzugehen.

(2 Punkte)

- Es soll möglich sein, eine Taggerinstanz wie eine Funktion aufzurufen. `tagger_instance(sentence)` sollte also einen annotierten Satz zurückgeben. *(0.5 Punkte)*

Aufgabe 3) Tester **Punkte: 2**

- Schreibe die Methode `evaluate_tagger` in der Klasse `TaggerTester` in der Datei `eval_utils.py`.

  - `evaluate_tagger` nimmt ein `Corpus` und einen trainierten Tagger als Parameter entgegen.

  - Es übergibt dem Tagger jeden einzelnen Satz aus dem Corpus und vergleicht die Vorhersagen mit den Labels.

[1] Angenommen, während des Trainings liegt das Wort "aus" dreimal als `ADP` und einmal als `VERB` vor, dann sagt der Unigramm-Tagger *immer* `ADP` vorher, wenn er das Wort "aus" liest.

– Es berechnet die Accuracy und gibt sie aus.

(1 Punkt)

- Wenn du alles richtig implementiert hast, solltest du nun die Datei `main.py` ausführen können, nachdem du die notwendigen `import`-Anweisungen ergänzt hast. *(1 Punkt)*

Aufgabe 4) MRO **Punkte: 1.5**

Führe die Datei `ex04.py` aus. Erkläre, warum du einen `TypeError` bekommst und was genau das vorliegende Problem ist. Gehe dabei darauf ein, was "MRO" ist. Du kannst für deine Erklärung gerne auf `.mro()` zurückgreifen. *(1.5 Punkte)*

Abgabemodalitäten *Reiche die Lösung pünktlich bis 12:00 Uhr am angegebenen Abgabetermin über Moodle ein. Gültige Abgaben sind Dateien, die nach gesundem Menschenverstand eine sinnvolle Möglichkeit darstellen, den Tutor:innen eine Lösung zu präsentieren (wenn Code abzugeben ist: .py-Datei, .txt-Datei: oft sinnvoll, .pdf-Datei: oft sinnvoll, .txt-Datei auf Arabisch + verschlüsselt: nicht sinnvoll). Gruppenabgaben von genau 2 Personen sind erlaubt. Bei offensichtlichem Abschreiben von anderen Gruppen werden alle beteiligten Abgaben mit 0 Punkten bewertet.*