POS Tagger

Tadeusz Teleżyński

27.04.2016

1 Zadanie

Celem projektu było stowrzenie programu tagującego podzbiór korpusu Browna znacznikami części mowy. W bilbiotece NLTK korpus ten jest otagowany dwoma tagsetami – uniwersalnym i domyślnym składającymi się z odpowiednio, 12 i 471 tagów. Z przyczyn obliczeniowych sprawdzanie programu na korpusie rozwojowym dla domyślnego tagsetu było bardzo nieefektywne, dlatego zdecydowałem się przeprowadzić mapowanie tagów domyślnych na zbiór 64 tagów (plik tag_map.py). Obliczeniowo wciąż jest to pewne obciążenie, które zdecydowałem się obejść szacując skuteczność programu dla pierwszych stu zdań korpusu rozwojowego (dla dwunastu tagów różnice w ewaluacji nie przekraczają w tym wypadku 1%). Zdecydowałem się zaimplementować uryty model Markova wraz ze zliczaniem Good-Turinga i Backoffem Katza, co okazało się metodą skuteczną, lecz nie idealną.

2 Implementacja

Głównym składnikiem bilbioteki jest plik POS_tagger.py. Rozwiązanie zamknięte jest w dwóch klasach. Klasa "Korpus" dostarcza korpus czytelny dla taggera. Klasa "Tagger" zawiera główną część implementacji. Konstruktor Taggera przyjmuje jako parametry korpus treningowy, stopień gramów, a także korpus, z którego nauczy się słownictwa tj. rozkładu emitowanego prawdopodobieństwa (rozwiązanie nie do końca eleganckie, ale na potrzeby projektu przyjęte zostało, że Tagger zna ten rozkład skądinąd niż z korpusu treningowego).

Plik test_runner.py zawiera funkcję testującą tagger. Przyjmuje ona jako parametry korpus, który zostanie podzielony na trzy części zgodnie z funckją z poprzedniego zadania domowego, arność gramu (niestety nie zdążyłem zdebugować programu dla n różnego od 3) i ewentualny zakres jeśli chcemy ewaluować nasz korpus dla części korusu rozwojowego lub testowego. Oprócz kanonicznej ewaluacji (procent poprawnie otagowanych słów) z ciekawości postanowiłem sprawdzić jeszcze procent poprawnie otagowanych zdań (funkcja "eval_sents").

3 Problemy i rozwiązania

Główny problem, który napotkałem pojawił się przy implementacji backoffu. Aby go uzyskać koniecznie jest zaaplikowanie algorytmu Good-Turinga dla wszyskich ngramów, gdzie n <= głównemu parametrowi n. W przypadku 12 tagów, zarówno unigramy jak i bigramy, nie dają się w pełni poprawnie przeliczyć, gdyż współczynnik krzywej regresji liniowej używanej do wygładzenia frekwencji frekwencjii jest większy od -1, co oznacza, że mamy niewystarczająco dużo danych, by wygładzanie to było w pełni poprawne. W literaturze nie udało mi się znaleźć sposobu na rozwiązanie tej kwestii. Problem polega na tym, że w efekcie, wygładzanie zwiększa, zamiast zmniejszać liczność poszczególnych frekwencji. Skutkiem tego współczynniki Beta i Alfa używane w algorytmie Katza mogą być ujemne, co powoduje przypisanie ujemnego prawdopodobieństwa niektórym gramom. Postanowiłem w sytuacjach takich ustalić Alfa = 1, co w praktyce oznacza pełne doszacowanie mniejszym n-gramem. Jest to rozwiązanie nieeleganckie, ale jak pokazują wyniki wystarczająco skuteczne.

Alternatywnym rozwiązaniem byłoby np. zrezygnowanie z backoffu i ograniczenie się wyłącznie do trigramów, gdzie niespotkanym okazom przypisywałoby się prawdopodobieństwo w oparciu o liczność gramów widzianych raz. W przypadku tym jednak skuteczność dla tagsetu ograniczonego spada poniżej 70%, co potwierdza, że przeliczanie Good-Turinga zastosowne samotnie jest zdecydowanie słabsze.

4 Test

Tager (n = 3) osiągnął zbliżone wyniki dla korpusów rozwojowego i testowego. Czas pełnego tagowania przy tagsecie uniwersalnym (12 tagów) wynosi 8.5 minuty, a tagowanie stu zdań tagsetu okrojonego (64 tagi) zajmuje 3.3 minuty, czyli pełne tagowanie szacować należy na ok. 3 godziny. Co ciekawe istnieje spora różnica w ilości poprawnie otagowanych zdań (65% dla tagsetu uniwersalnego i 33% dla tagsetu okrojonego).

Tablica 1: Wyniki programu

tagset/korpus	rozwojowy	testowy
uniwersalny	97.9%	97.8%
okrojony	94.3%	94.7%