Practical 1: Parts of speech in the Universal Dependencies treebanks

This practical is worth 50% of the coursework credits for this module. Its due date is Friday 12th of March 2021, at 21:00. The usual penalties for lateness apply, namely Scheme B, 1 mark per 8 hour period or part thereof.

The purpose of this assignment is to gain understanding of the Viterbi algorithm, and its application to part-of-speech (POS) tagging.

You will also get to see the Universal Dependencies treebanks. The main purpose of these treebanks is dependency parsing (to be discussed later in the module), but here we mainly use their part-of-speech tags.

Getting started

We will be using Python3. On the lab (Linux) machines, you need the full path `/usr/local/python/bin/python3`, which is set up to work with NLTK. (Plain `python3` won't be able to find NLTK.)

If you run `python` on your personal laptop, be sure it is Python3 rather than Python2; in case of doubt, do `python --version`. Next to NLTK (`https://www.nltk.org/`), you will also need to install the `conllu` package (`https://pypi.org/project/conllu/`).

To help you get started, download `P1getstarted.py`, download the zip file with corpora from this directory, unzip it, and run `/usr/local/python/bin/python3 P1getstarted.py`. You may, but need not, use parts of this code in your submission.

The three corpora come from Universal Dependencies. You can download the entire set of treebanks from `https://universaldependencies.org/`.

Viterbi algorithm

You will develop a first-order HMM (Hidden Markov Model) for POS (part of speech) tagging in Python. This involves:

- counting occurrences of one part of speech following another in a training corpus,
- counting occurrences of words together with parts of speech in a training corpus,
- relative frequency estimation with smoothing,
- finding the best sequence of parts of speech for a list of words in the test corpus, according to a HMM model with smoothed probabilities,
- computing the accuracy, that is, the percentage of parts of speech that is guessed correctly.

As discussed in the lectures, smoothing is necessary to avoid zero probabilities for events that were not witnessed in the training corpus. Rather than implementing a form of smoothing yourself, you can for this assignment take the implementation of Witten-Bell smoothing in NLTK (among the forms of smoothing in NLTK, this seems to be the most robust one). An example of use for emission probabilities is in file `smoothing.py`; one can similarly apply smoothing to transition probabilities.

Run your application on the English (EWT) training and testing corpora. You should get an accuracy above 89%. If your accuracy is much lower, then you are probably doing something wrong.

Comparisons between languages

Investigate, by visual inspection and by computational means, the `upos` parts of speech in different treebanks from Universal Dependencies. (Take a few languages based on your own interests, but no more than about 10. Go for the quality of your submission, not quantity!) Two examples of specific questions you could address:

- Which of the chosen languages have a rich morphology and which have a poor morphology?
- How similar are the chosen languages, in terms of bigram models of their parts of speech?

For the first question, know that you can access the lemma of a token by `token['lemma']`. What can you say about the relation between forms and lemmas in the case of languages with rich morphology?

For the second question, consider that the transition probabilities of two related languages may be very similar, even though the emission probabilities may be incomparable due to the mostly disjoint vocabularies. How could we measure the similarity between two bigram models trained from corpora?

Feel free to think of further questions to address. It is worth noting that next to the ('universal') `upos` tags, the Universal Dependencies treebanks sometimes also contain language-specific (`xpos`) tags.

Requirements

Submit your Python code, with a README file telling me how to run it, and a report describing your findings, including experimental results and their analysis. You would include the treebanks needed to run the code, **but please do not include the entire set of Universal Dependencies treebanks, because this would be a huge waste of disk space and band width for the marker**.

Marking is in line with the General Mark Descriptors (see pointers below). Evidence of an acceptable attempt (up to 7 marks) could be code that is not functional but nonetheless demonstrates some understanding of the Viterbi algorithm. Evidence of a competent attempt addressing most requirements (up to 13 marks) could be fully correct code in good style, but without exploration of the differences between languages.

Evidence of exceptional achievements (19-20 marks) would be thorough investigation of the differences between languages as it relates to parts of speech, by means of experiments and with awareness of the linguistic background discussed in the lectures. Results of the experiments would typically be reported using graphs, tables, and so on. Keep in mind that not all treebanks are the same size, and a common pitfall is making unfair comparisons between languages by not correcting for differences in corpus size.

Hints

Even though this module is not about programming per se, a good programming style is expected. Choose meaningful variable and function names. Break up your code into small functions. Avoid cryptic code, and add code commenting where it is necessary for the reader to understand what is going on. Do not overengineer your code; a relatively simple task deserves a relatively simple implementation. If you feel the need for Python virtual environments, then you are probably overdoing it. The code that you upload would typically consist of one or two `.py` files.

You cannot use any of the POS taggers already implemented in NLTK. However, you may use general utility functions in NLTK such as `ngrams` from `nltk.util`, and `FreqDist` and `WittenBellProbDist` from `nltk`.

There is a chance that you will experience underflow when running the Viterbi algorithm. For this reason, you may omit consideration of sentences longer than 100 tokens.

When you are reporting the outcome of experiments, the foremost requirement is reproducibility. So if you give figures or graphs in your report, explain precisely what you did, and how, to obtain those results.

Pointers

- Marking
  `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`
- Lateness
  `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`
- Good Academic Practice
  `https://www.st-andrews.ac.uk/students/rules/academicpractice/`