BM1 Advanced Natural Language Processing
University of Potsdam, Winter 2016/17

Assignment 2: Part-of-Speech Tagging

Tatjana Scheffler, Ph.D.

Due: Friday, November 18, 10:00 a.m.

Problem 1: Part-of-Speech Tagging Using HMMs

Implement a bigram part-of-speech (POS) tagger based on Hidden Markov Models from scratch. Using NLTK is disallowed, except for the modules explicitly listed below. For this, you will need to develop and utilize the following modules:

1. Corpus reader and writer
2. Training procedure, including smoothing
3. Viterbi tagging, including unknown word handling
4. Evaluation

The task is mostly very straightforward, but each step requires careful design. Thus, we suggest you proceed in the following way.

Viterbi algorithm. First, implement the Viterbi algorithm for finding the optimal state (tag) sequence given the sequence of observations (words). We suggest you test your implementation on a small example for which you know the correct tag sequence, such as the Eisners Ice Cream HMM from the lecture. Make sure your Viterbi algorithm runs properly on the example before you proceed to the next step. There are plenty of other detailed illustrations for the Viterbi algorithm on the Web from which you can take example HMMs, even in Wikipedia.

Training. Second, learn parameters for your HMM from *annotated* data, i.e. the initial, transition, and emission probabilities. Implement a maximum likelihood training procedure (with smoothing) for supervised learning of HMMs. You can get a corpus on Moodle. We will be using the Wall Street Journal corpus (part of the Penn Treebank). It is a licensed corpus, so please do not redistribute the files. The zip archive contains a training set, a test set, and an evaluation set. The training set (wsj_tagged_train.tt) and the evaluation set (wsj_tagged_eval.tt) are written in the commonly used CoNLL format. They are text files with two colums; the first column contains the words, the POS tags are in the second column, and empty lines delimit sentences. The test

set (wsj_tagged_test.t) is a copy of the evaluation set with tags stripped, as you should tag the test set using your tagger and then compare your results with the gold-standard ones in the evaluation set. The corpus uses the Penn Treebank tagset.

You are welcome to use any NLTK data structures from the two modules `nltk.corpus.reader` (and submodules) and `nltk.probability`. The latter includes a number of smoothing procedures, which you may want to apply to your corpus frequency counts. Take care to get NLTK to make the smoothed probability distributions sum to one (see the SUM_TO_ONE parameter). Experiment with unsmoothed distributions, Laplace add-one smoothing, and at least one further smoothing procedure.

Evaluation. Once you have trained a model, evaluate it on the unseen data from the test set. Run the Viterbi algorithm with each of your models, and output a tagged corpus in the two-column CoNLL format (*.tt). We will provide an evaluation script. Run it on the output of your tagger and the evaluation set and report your results. Note that your tagger will initially fail to produce output for sentences that contain words you havent seen in training. If you have such a word w appear at sentence position t, you will have $b_j(w) = 0$ for all states/tags j, and therefore $V_t(j) = 0$ for all j. Adapt your tagger by implementing the following crude approach to unknown words. Whenever you get $V_t(j) = 0$ for all j because of an unknown word at position t, pretend that $b_j(w) = 1$ for all j. This will basically set $V_t(j) = max_i V_{t1}(i)a_{ij}$, and allow you to interpolate the missing POS tag based on the transition probabilities alone.

Extra credit. The task is challenging as it stands. However, feel free to go further for extra credit, e.g. by doing one of the following: implement better unknown word handling, use a trigram tagger, plot a learning curve for your tagger (accuracy as a function of training data size), plot a speed vs. sentence length curve.

Please submit your code, instructions for running your tagger and tagging output(s). Document any additional data you submit. With this, you will have implemented your first POS tagger! Happy coding!

Submit your solutions and code via email to `johannsmeier@uni-potsdam.de`