Johannes Choo
October 31, 2017

1 Extending Kneser-Ney to Closure in Unseen N-grams

We build in python2 a HMM tagger employing Interpolated Kneser-Ney smoothing as described by Jurafsky and Martin (August 2017 Draft). The description given by Jurafsky and Martin is not complete, and notably not closed under unobserved tokens and contexts. We extend it to closure over unobserved N-grams with the following probability measure defined over words w with either no context, or a context of arbitrary positive length $c'\mathbf{c}$. We have

$$P_{KN}(w \mid c'\mathbf{c}) = \begin{cases} \frac{\max\left(c(c'\mathbf{c}w)-d,0\right)}{S(c'\mathbf{c})} + \lambda(c'\mathbf{c})P_{KN}(w \mid \mathbf{c}) & \text{if } S(c'\mathbf{c}) > 0, \\ P_{KN}(w \mid \mathbf{c}) & \text{if } S(c'\mathbf{c}) = 0, \end{cases}$$

$$P_{KN}(w) = P_{KN}(w \mid \epsilon) = \begin{cases} \frac{\max(c(w)-d,0)}{S(\epsilon)} + \lambda(\epsilon)\frac{1}{|T|+1} & \text{if } S(\epsilon) > 0, \\ \frac{1}{|T|+1} & \text{if } S(\epsilon) = 0, \end{cases}$$

where we wish to obtain the probability of a token w after observing context $c'\mathbf{c}$. T is the (unigram) vocabulary. For bigrams, we let $\mathbf{c} = \epsilon$, the empty tuple of words, in order that the context is just a single word c'. d is a parameter with $0 \leq d \leq 1$ of the absolute discount to the N-gram counts; it is to be learned by experimentation. $S(\mathbf{c})$ is a normalization term given by

$$S(\mathbf{c}) = \sum_{w \in T} c(\mathbf{c}w),$$

including for $\mathbf{c} = \epsilon$. λ redistributes the discounted probability mass uniformly over all observed tuples the size of $c'\mathbf{c}w$, given by

$$\lambda(\mathbf{c}) = \frac{d}{S(c)}|\{w \in T : c(\mathbf{c}w) > 0\}|,$$

including for $\mathbf{c} = \epsilon$. The count c is defined by

$$c(\mathbf{u}) = \begin{cases} c_t(\mathbf{u}) & \text{if } |\mathbf{u}| = N-1, \\ |\{v : c_t(v\mathbf{u}) > 0\}| & \text{if } |\mathbf{u}| < N-1. \end{cases}$$

including for $\mathbf{u} = \epsilon$ where c_t are the raw observations from the training set and N is the N of our N-gram model (i.e. the number of atoms).

Our implementation uses a straightforward approach where we compute c_t, c, T, S during training, and compute P_{KN} at the time of query. We slide a window of maximum size N across the training corpus, and for each final token w, we record the N tuples that it generates, from the N-gram to the unigram. In our implementation, we perform smoothing separately for the (hidden) Markov chain and for the tag-token emission probabilities. We use $N = 2$ for a bigram HMM model.

2 Implementation of the Viterbi Algorithm

We implement the Viterbi algorithm in a straightforward translation of the description provided by Jurafsky and Martin, with a couple of modifications. The computation of the tag transition and the token emission probabilities are done in probability masses themselves (since we have to add probabilities from different models in KN smoothing) but once computed, they are converted and accumulated as logprobs.

We make a second modification: we only consider candidate tags for which there has been an observed transition from the context tag. This slightly reduces the accuracy when the correct tag involves an unobserved transition from the current context, but greatly speeds up the algorithm and its asymptotic performance if we reasonably assume that the (observed) transition matrix from context to tag is sparse.

3 Training and Running the Tagger

To train the tagger, run:

```
python2 build_tagger.py sents.train sents.devt model_file
```

To run the tagger on a test file, run:

```
python2 run_tagger.py sents.test model_file sents.out
```

In both cases, the `HMMTagger.py` file should be present in the same directory as the script file (`build_tagger.py` or `run_tagger.py`).

4 References

Daniel Jurafsky and James H. Martin. 2017. *Speech and Language Processing (3rd Edition, August Draft).* [online] Available at https://web.stanford.edu/~jurafsky/slp3/ [Accessed 28 Sep. 2017].