Assignment 3
CSE 490U: Natural Language Processing

University of Washington

Due: February 20, 2017

In this assignment you will design and implement hidden Markov models (HMMs) for part-of-speech (POS) tagging. Given a natural language sentence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_\ell \rangle$, the task is to find the most likely sequence of POS tags $\hat{\boldsymbol{y}} = \langle y_1, y_2, \ldots, y_\ell \rangle$.

The data you need to tag is from Twitter, which introduces some added challenges (as discussed in lecture). The tag set $\mathcal{L}$ comes from Gimpel et al. [2011]; it's considerably smaller than the tag set used in the Penn Treebank (45), which is in turn a reduced tag set from the 87-tag tag set used for the Brown corpus. Take a look at Table 1 in Gimpel et al. [2011] to get a better understanding of the tags. The paper also has some examples.

1 Dataset

For this task, we provide a random set of tweets from April 2016, filtered down to English (as defined by the Twitter API). We POS-tagged them using a state-of-the-art POS tagger [Owoputi et al., 2013]. These are the files (download them via Canvas):

- `twt.train.txt`: data for training your HMMs (50,000 tagged tweets).
- `twt.dev.txt`: development data for you to choose the best smoothing parameters (5,000 tagged tweets)
- `twt.test.txt`: test data for evaluating your HMM models (5,000 tagged tweets)
- `twt.bonus.txt`: bonus data; if you feel like training a model on more training data, you're free to use this file or a subset of it (1,671,785 tagged tweets)

You should immediately note a problem with this protocol: we are asking you to build a POS tagger from data that was labeled by another POS tagger (not humans). If the existing POS tagger is good enough to create gold-standard data to train on, why bother training a new POS tagger? (In fact, that POS tagger isn't perfect.) If we train on imperfect data, won't our new POS tagger's performance be bounded above by the performance of the old tagger? There are some surprising results using "bootstrapping" methods that train new models on the output of old models, but that's not really the focus here. We want you to learn to implement HMMs, and use a reasonable amount of interesting data while using it, and we're willing to work in a somewhat unrealistic setting to enable that.

Each file contains one tagged tweet per line, where each line is a JSON object containing (word, tag) pairs. For example:

```
    [["Spending", "V"], ["the", "D"], ["day", "N"], ["withhh", "P"], ["mommma",
"N"], ["!", ","]]
```

2 Accuracy

To report accuracy on any dataset, count the number of words (excluding start and stop symbols) that you tagged correctly, and divide by the total number of words (excluding start and stop symbols). A good practice, which we recommend, is to cleanly separate your evaluation code from your modeling code. Your standalone script for computing accuracy should read in two files (one contains "correctly" labeled data, the other contains the predictions of a model) and print out the accuracy. Often when people implement evaluation scripts, they include additional statistics, such as the accuracy for different subsets of the data (e.g., by POS tag), the "confusion matrix" of mistakes,[1] and/or some examples that the system did especially well on or especially badly on.

3 Bigram HMM (40%)

Start by estimating $p(X_i \mid Y_i)$ and $p(Y_i \mid Y_{i-1})$ as presented in lectures. As explained there, assume y_0 is a special start symbol and $y_{\ell+1}$ is a special stop symbol. You will need to handle out-of-vocabulary words and smooth your distributions; how you do it is up to you.

Next, implement the Viterbi algorithm so that you can tag new sentences in the development and test sets.

Deliverables

- Describe your design decisions for OOVs and smoothing. Compare the model performance of a few different design choices on the train and development sets. Pick your best model and report its performance on the test set. Only report one model's performance on the test set.
- Error analysis: provide error analysis on the development set, discussing prominent error cases you may have noticed. Show concrete examples and consider showing the confusion matrix.

4 Trigram HMM (60%)

Next, consider the so-called "trigram" HMM, which replaces $p(Y_i \mid Y_{i-1})$ with $p(Y_i \mid Y_{i-1}, Y_{i-2})$. Just like in a trigram language model, you now need two start symbols for y_0 and y_{-1}. Now, your local prefix score will be $s_i(y, y')$ instead of $s_i(y)$. You can interpret $s_i(y, y')$ as "the joint probability of the first i words of the sequence, together with their most likely label sequence ending in $y'\ y$." The recurrence was given in the HMM lecture slides; you will need to work out the special cases for the beginning and end of the sequence.

You need to implement parameter estimation and the Viterbi algorithm for this model.

Deliverables

- Describe your design decisions for OOVs and smoothing. Compare the model performance of a few different design choices on the train and development sets. Pick your best model and report its performance on the test set. Only report one model's performance on the test set.
- Error analysis: provide error analysis on the development set, discussing prominent error cases you may have noticed. Show concrete examples and consider showing the confusion matrix.
- Did the trigram model outperform the bigram model? Significantly?

[1] https://en.wikipedia.org/wiki/Confusion_matrix

Submission Instructions

Submit a single gzipped tarfile (`A3.tgz`) on Canvas.

- **Code**: You will submit your code together with a neatly written README file to instruct how to run your code with different settings. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade.
- **Report** (use the filename `A3.pdf` and include in the tarfile): Your writeup should be two to three pages long, or less, in pdf (one-inch margins, reasonable font sizes, LaTeX-typeset). Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

References

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proc. of ACL*, 2011. URL `http://homes.cs.washington.edu/~nasmith/papers/gimpel+etal.acl11.pdf`.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proc. of NAACL*, 2013. URL `http://homes.cs.washington.edu/~nasmith/papers/owoputi+oconnor+dyer+gimpel+schneider+smith.naacl13.pdf`.