Chapter 8

Part-of-Speech Tagging

8.1 Problem

Let's turn to a new task: Given a sentence, assign a part-of-speech (POS) tag (noun, verb, etc.) to each word. Many words have a unique POS tag, but some words are ambiguous: for example, *short* can be an adjective (*short vowel*), a noun (direct a *short*), an adverb (to throw a ball *short*) or a verb (to *short* an appliance). Figuring out which POS is the correct one depends on the context, including the POS tags of the neighboring words.

This is a sequence labeling task, just like the first version of the typo correction problem we looked at, which involved only character substitutions and no insertions or deletions. Other examples of sequence labeling tasks are:

- Word segmentation: Given a representation of a sentence without any word boundaries, reconstruct the word boundaries. In some languages, like Chinese, words are written without any spaces in between them. (Indeed, it can be difficult to settle on the definition of a "word" in such languages.) This situation also occurs with any spoken language.
- Word sense disambiguation: Given a dictionary which gives one or more *word senses* to each word (for example, a *bank* can either be a financial institution or the sloped ground next to a river), and given a sentence, guess the sense of each word in the sentence.
- Named entity detection: Given a sentence, identify all the proper names (Notre Dame, Apple, etc.) and classify them as persons, organizations, places, etc. The typical way to set this up as a sequence-labeling problem is called *BIO tagging*. Each word is labeled *B* (beginning) if it is the first word in a named entity, *I* (inside) if it is a subsequent word in a named entity, and *O* (outside) otherwise. Other encodings are possible as well.

8.2 Hidden Markov Models

Previously, we talked about Hidden Markov Models in the context of typo correction. Now the reason for the choice of variable names $\mathbf{t}$ and $\mathbf{w}$ emerges – part-of-speech tagging is the classic application of HMMs in NLP, and $\mathbf{w}$ stands for "words" and $\mathbf{t}$ stands for "tags." The same methods we discussed previously can be used to do supervised part-of-speech tagging fairly well.

What about unsupervised part-of-speech tagging? Given just plain text, can an HMM automatically figure out what the parts of speech are? With some caveats...kind of.

The procedure at a high level would look like this:

1. Choose a set of possible tags.
2. Start with an initial model (estimates for the $p(t' \mid t)$ and $p(w \mid t)$).
3. E step: For each sentence in the trianing data, use the Forward-Backward algorithm to compute a distribution over possible taggings; or (for hard EM) use the Viterbi algorithm to find the best tagging.
4. M step: Count up the tag bigrams and tag-word pairs, and restimate $p(t' \mid t)$ and $p(w \mid t)$.
5. Go to 3.

The first problem is with step 1. You can choose a tag set like {N,V}, but there's absolutely no way for EM to know that you want N to be the nouns and V to be the verbs. So there's no point in giving the tags names/meanings; just number them, and try to assign names/meanings afterwards.

A second, related, problem is with step 2: if you initialize the model to uniform probabilities, then the model will be perfectly symmetric between the tags. So EM will get stuck, not doing anything. (This didn't happen to us before, because we assumed a pre-trained language model that breaks the symmetry.) Fortunately, the solution is very simple: just add some random noise to the initial probabilities.

The third problem, also related to the first, is that there's nothing here that tells EM to look for *parts of speech*; it could go off and look for something else instead, like words about finance versus words about politics. Whatever helps the model give the highest probability to the observed sentences is what EM will try to find, and that might or might not be parts of speech.

I'm not aware of a satisfying solution to this problem. There's a restricted version of the task, commonly known as the Merialdo task, that solves all three problems by assuming the availability of a dictionary that says, for each word, what all the possible tags for that word are. Under this restriction, EM is able to do something resembling POS tagging.

Yet another problem is overfitting. If you choose the tag set to be too large (say, one million), then EM could simply give each word *token* its own tag, which would allow the model to give the observed sentences a very high probability. (Why?) Even if you choose a tag set with a reasonable size (45), though, this problem can crop up in subtle ways.

8.3 Conditional Random Fields

Returning to supervised part-of-speech tagging, the state-of-the-art method for most sequence labeling tasks is a *bidirectional LSTM plus conditional random field*, and before that, it was just a *conditional random field* or CRF (Lafferty, McCallum, and Pereira, 2001). CRFs consistently outperform HMMs for sequence labeling tasks, and generally do as well or better than other methods. The downside of CRFs is that they have to be trained using numerical optimization, which is a lot slower than (supervised) HMMs.

8.3.1 Definition

In a HMM, the parameters to be learned are the $p(t' \mid t)$ and $p(w \mid t)$, which are required to sum to one. In a CRF, these parameters are no longer probabilities; they are now just numbers that can be positive or

negative, big or small. There are two kinds of parameters: $\lambda(t, t')$ for every tag/tag pair, and $\mu(t, w)$ for every tag/word pair. There's no definition of what the values of these parameters should be; instead, you initialize them to all zeros and optimize them to maximize the likelihood of the training data.

Recall that a HMM is (when written out using log-probabilities):

$$\log P(\mathbf{t}, \mathbf{w}) = \log p(t_1 \mid \texttt{<s>}) + \log p(w_1 \mid t_1) + \sum_{i=2}^{n} \Big(\log p(t_i \mid t_{i-1}) + \log p(w_i \mid t_i)\Big) + \log p(\texttt{</s>} \mid t_n). \quad (8.1)$$

The new model replaces all the log-probabilities with parameters that can be positive or negative, big or small:

$$s(\mathbf{t}, \mathbf{w}) = \lambda(\texttt{<s>}, t_1) + \mu(t_1, w_1) + \left(\sum_{i=2}^{n} \lambda(t_{i-1}, t_i) + \mu(t_i, w_i)\right) + \lambda(t_n, \texttt{</s>}). \quad (8.2)$$

And our goal is to adjust all these parameters to maximize the *conditional* likelihood:

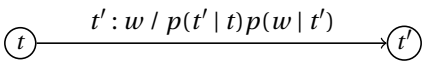
$$L = \sum_{(\mathbf{t}, \mathbf{w}) \in \text{data}} \log P(\mathbf{t} \mid \mathbf{w}) \quad (8.3)$$

$$P(\mathbf{t} \mid \mathbf{w}) = \frac{\exp s(\mathbf{t}, \mathbf{w})}{\sum_{\bar{\mathbf{t}}} \exp s(\bar{\mathbf{t}}, \mathbf{w})} \quad (8.4)$$

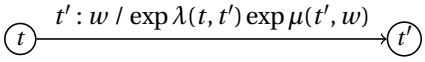
where $\bar{\mathbf{t}}$ ranges over all possible tag sequences of length n. (See below for how to compute this sum efficiently.)

8.3.2 As finite transducers

Recall that we wrote an HMM as a finite transducer, where the edges looked like



We can write a CRF as a finite transducer too:



If we run this transducer on tags $\mathbf{t}$ and words $\mathbf{w}$, then the weight of the path is $\exp s(\mathbf{t}, \mathbf{w})$, which is the numerator of (8.4). How do we compute the denominator?

8.3.3 Summing over all tag sequences

The denominator in equation (8.4) is a summation over all possible tag sequences $\bar{\mathbf{t}}$ – that's a lot of tag sequences! The way that we perform the summation is similar to how the Viterbi algorithm finds the maximum over all possible tag sequences. The only difference is that when a state has multiple incoming edges, we add the weights instead of taking their maximum.

forward$[q_0] \leftarrow 1$
forward$[q] \leftarrow 0$ **for** $q \neq q_0$
for each state q' in topological order **do**
  for each incoming transition $q \rightarrow q'$ with weight p **do**
    forward$[q'] \leftarrow$ forward$[q'] +$ forward$[q] \times p$
  end for
end for

8.3.4 Training

As always, we want to maximize the log-likelihood:

$$\log L = \sum_{\mathbf{t},\mathbf{w} \text{ in data}} \log P(\mathbf{t} \mid \mathbf{w}) \tag{8.5}$$

We do this using stochastic gradient descent, looping over all the sentences, and for each sentence $\mathbf{w}$ with correct tags $\mathbf{t}$, take a step uphill on $\log P(\mathbf{t} \mid \mathbf{w})$. The gradient can be computed by automatic differentiation.

Bibliography

Lafferty, John, Andrew McCallum, and Fernando Pereira (2001). “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *ICML*, pp. 282–289.