Name : Tang Sophearin

Affiliation : Department of Intelligence Science and Technology, Nishida Laboratory

Student ID : 6930-30-3963

Part of Speech Transcription Using HMM

Part of speech tagging is the process of annotating part of speech or word category to each word in a sentence. Although it sounds trivial, but it happens to be not just a look-up table problem because each word may associate with different part of speeches. Therefore, a sophisticated approach should be done to disambiguate the part of speech of each word in a sentence depending on context.

In the 1980s, hidden Markov models was introduced in the part of speech transcription or tagging problem. Given a sequence of observable words, we want to find the best estimate of part of speech of current word. The ideas behind HMM base on the assumption on how human brain generate a sentence. To generate a sentence, it is assumed that human will start by formulate part of speech order of the to-be-generated sentence. Depending on the previous part of speech(es), the next part of speech will be selected by some POS probability distribution. Then, each word of the sentence will be sampled from the distribution of each part of speech respectively. Therefore, the probability of a word appearing depends only on its own tag and the probability of a tag is dependent only on the previous tag. Thus, the basic equation of HMM tagging is:

$$\hat{t}_c^n = \underset{t_c^n}{\operatorname{argmax}} P(t_c^n|w_c^n) \approx \underset{t_c^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Both emission probability $P(w_i|t_i)$ and transition probability $P(t_i|t_{i-1})$ can be easily computed using tagged training data. One example of tagged training data is WSJ corpus[1]. The calculations follow:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$C(x)$ is a counting function. For example, $C(t_{i-1}, t_i)$ is a number of times tag (POS) t_{i-1} and t_i appearing in training corpus. Assuming that there are $N + 2$ tags (including starting and ending symbols) and T words, two matrixes of probability distribution will be obtained after the calculation ($N + 1$ by N table and N by T).

Last but not least, we can employ '*Viterbi Algorithm*' to be our searching (decoding) algorithm for finding optimal sequence of tags. *Viterbi* is a kind of dynamic programming which makes use of dynamic programming trellis. It can solve above problem with complexity $O(T \times |N|^2)$.

The *Viterbi* algorithm sets up a probability matrix with one column for each observation t (word) and one row for each state in the state graph (one row corresponds to one transition of states

[1] WSJ corpus: https://catalog.ldc.upenn.edu/ldc2000t43

of one sentence). In the initialization step (first iteration), each cell of the first column contains *Viterbi* value as a product of transition probability of current state (tag) given starting symbol and emission (observation) probability of first word given the state. These probabilities can be looked up from the previously calculated matrixes. Then, in recursion step, cell values are obtained by taking the maximum of all previously calculated *Viterbi* values multiplying by the transition probability to the current state and so on. At the end of the algorithm, we can backtrack and reconstruct the optimal state sequence of the sentence. Pseudo code of *Viterbi* algorithm is shown below[2]:

function VITERBI(*observations* of len *T*,*state-graph* of len *N*) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
  $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step *t* **from** 2 **to** *T* **do** ; recursion step
  for each state *s* **from** 1 **to** *N* **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Although HMM is originally built upon bigram assumption, we can further extend the capability and accuracy of part of speech transcription by using more history or context; for instance, we can use the trigram model: $P(t_i|t_{i-1},t_{i-2})$ can be used in place of $P(t_i|t_{i-1})$. One issue with HMM trigram tagger is data sparsity.

$$P(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})}$$

As a result of extending the context dependency, tag sequence t_{i-2},t_{i-1},t_i might never occur in our training data! To answer this problem, we can use a technique called linear interpolation. We estimate the probability $P(t_i|t_{i-1},t_{i-2})$ by a weighted sum of unigram, bigram and trigram probabilities:

$$P(t_i|t_{i-1},t_{i-2}) = \lambda_3\hat{P}(t_i|t_{i-1},t_{i-2}) + \lambda_2\hat{P}(t_i|t_{i-1}) + \lambda_1\hat{P}(t_i)$$

Unknown word is another issue when doing POS tagging. Standard method to deal with this problem is to consider morphology such as word suffixes. Combining all the features, a state-of-the-art trigram HMM of Brants (2000)[3] has a tagging accuracy of 96.7% on the Penn Treebank corpus.

[2] Figure 10.8 https://web.stanford.edu/~jurafsky/slp3/10.pdf
[3] Brants, T. (2000). TnT: A statistical part-of-speech tagger. In ANLP 2000, Seattle, WA, pp. 224–231.