Last Class:

1. Intro to part-of-speech tagging

Today:

1. Why it's hard
2. Hidden Markov Model Tagger

HMM Tagger

Given $W = w_1, \ldots, w_n$, find $T = t_1, \ldots, t_n$ that maximizes

$$P(t_1, \ldots, t_n | w_1, \ldots, w_n)$$

Restate using Bayes' rule:

$$(P(t_1, \ldots, t_n) * P(w_1, \ldots, w_n | t_1, \ldots, t_n))/P(w_1, \ldots, w_n)$$

Ignore denominator...
Make independence assumptions...

1. General problem: given a sequence of words, want to find the sequence of lexical categories that maximizes...As before, this means that we'll want to maximize the numerator (since common denominator).
2. Unfortunately, we **can't use direct methods** (i.e., can't compute from first principles) because it would take much too much data to get reasonable estimates for sequences of that length.
3. Can use approximation techniques. First, restate the problem using Bayes' rule...
4. Still no effective methods for calculating the probability of such long sequences accurately. Requires too much data.
5. Approximated by probabilities that are simpler to collect if we first make some important independence assumptions. These aren't really valid, but the estimates made using them work well in practice.

Independence Assumptions (factor 1)

$P(t_1, \ldots, t_n)$: approximate using **n-gram model**

bigram $\prod_{i=1,n} P(t_i \,|\, t_{i-1})$

trigram $\prod_{i=1,n} P(t_i \,|\, t_{i-2} t_{i-1})$

1. Approximate each factor of numerator.
2. Remind of independence assumption: probability is product of individual probabilities — $P(t_1) * P(t_2) \dots P(t_n)$. prob that category in first position is t_1, etc.
3. Probability of sequence of categories ($P(t_1, \dots, t_n)$): can be approximated by a series of probabilities based on a limited number of previous categories. Most common assumptions involve either one or two previous categories.
4. **bigram model**: looks at pairs of categories (or words) and uses the conditional probability that a category t_i will follow a category t_{i-1}.
5. **trigram model**; uses conditional probability that one category/word will appear next, given the two preceding categories/words.
6. Good thing is that we can actually estimate these probabilities.
7. Posit a pseudo category ϕ at position 0 as the value of t_0.
8. Trigram model will produce better results. But we'll use bigram for examples.

Independence Assumptions (factor 2)

$P(w_1, \dots, w_n \,|\, t_1, \dots, t_n)$: approximate by assuming that a word appears in a category independent of its neighbors

$$\prod_{i=1,n} P(w_i \,|\, t_i)$$

Assuming bigram model:

$$P(t_1, \dots, t_n) * P(w_1, \dots, w_n | t_1, \dots, t_n) \approx$$
$$\prod_{i=1,n} P(t_i | t_{i-1}) * P(w_i | t_i)$$

Slide CS674–5

1. Second probability can be approximated by assuming that a word appears in a category **independent of the words in the preceding or succeeding categories**.
2. Take product of the probability that each word occurs in the indicated part of speech...
3. With these two approximations, the original equation that we wanted to maximize becomes...
4. We know the words in the sequence, so our task is now to find the sequence of categories that maximizes the value of...
5. Advantage of this equation is that each of the probabilities involved can be readily estimated from a corpus of text labeled with parts of speech.
6. Can estimate these probabilities from the corpus as we've done before: (1) **n-gram frequencies**; (2) **lexical-generation probabilities**— probability that a given category is realized by a specific word.

Hidden Markov Models

Equation can be modeled by an HMM.

- **states**: represent a possible lexical category
- **transition probabilities**: bigram probabilities
- **observation probabilities**, **lexical generation probabilities**: indicate, for each word, how likely that word is to be selected if we randomly select the category associated with the node.

Slide CS674–6

1. **Equation can be modeled by a hidden Markov model.**
2. **states**: represent a possible lexical category
3. **transition probabilities**: bigram probabilities
4. Allow each node to have an **output probability** — probability associated with each node that indicates, for each word, how likely that word is to be selected if we randomly select the category associated with the node.
5. *hidden* indicates that for a specific sequence of words, it's not clear what state the markov model is in. "spring" could be generated from N state with particular probability; could be generated from V state with another probability.
6. Given a sequence of words and their corresponding p-o-s categories, can traverse the network to compute probability that the HMM would generate the sentence with that pos sequence. Multiply the transition probabilities times the lexical generation probabilities.
7. Example.
8. Still need an algorithm for computing the most likely sequence of categories for a given sequence of words. Don't want to enumerate all possible sequences and run through network to find one with maximum value.
9. Example: *students need another break*. Assume 4 possible pos categories. $4^4 = 256$ different sequences of length four. Brute force algorithm would have to generate all of them and compare the results. Infeasible for reasonable tagsets and sentence lengths. [show full graph on board.]
10. Key idea: because of the Markov assumptions we're making, we don't have to enumerate all the possible sequences.
11. Example. Sweep forward, one word at a time, finding the most likely tag sequence ending in each category. E.g. find the four most likely sequences for the two words *students need*: the best ending with *need* as a V, the best as an N, the best as a P, and the best as an ART. Using this information, you then find the four best sequences for the three words *students need a*, each one ending in a different category. Keep going until all of the words have been accounted for.
12. Called the Viterbi algorithm...uses dynamic programming.

Viterbi Algorithm

c: number of lexical categories

$P(w_t|t_i)$: lexical generation probabilities

$P(t_i|t_j)$: bigram probabilities

Find most likely sequence of lexical categories $T_1, \ldots, T_n$ for word sequence.

Initialization

```
For i = 1 to c do
   SCORE(i,1) = $P(t_i|\phi) * P(w_1|t_i)$
   BPTR(i,1) = 0
```

Slide CS674–7

Iteration

```
For t = 2 to n
  For i = 1 to c
     SCORE(i,t) =
        $MAX_{j=1..c}(SCORE(j,t-1) * P(t_i|t_j)) * P(w_t|t_i)$
     BPTR(i,t) = index of j that gave max
```

Identify Sequence

```
T(n) = i that maximizes SCORE(i,n)
For i = n-1 to 1 do
   T(i) = BPTR( T(i+1), i+1 )
```

Slide CS674–8

p. 203, figures 7.10-7.12.

1. For a problem involving n words and c lexical categories, it is guaranteed to find the most likely sequence using $k * n * c^2$ steps (for some constant k), significantly better than the c^n steps required by brute force search.
2. Use 2 cxn arrays to keep track of best sequence leading to each possible category at each position: **seqscore**: probability for the best sequence up to position t that ends with a word in a particular category. **backptr**: for each category in each position, indicates what the category is in the best sequence at preceding position.
3. Algorithm also assumes that we've analyzed some corpus and obtained the bigram probabilities and lexical-generation probabilities. Any bigram not occurring in corpus is assumed to have a probability of 0.0001.
4. Go to algorithm.
5. **initialization**; for each category...
6. **main part: creating markov chain** proceed through input words, proceed through categories...

Results

- Effective if probability estmates are computed from a large corpus
- Effective if corpus is of the same style as the input to be classified
- Consistently achieve accuracies of 96% or better using trigram model
- Cuts error rate in half vs. naive algorithm (90% accuracy rate)
- Can be smoothed using backoff or deleted interpolation...

Extensions

- Can train HMM tagger on unlabeled data using the EM algorithm, starting with a dictionary that lists which tags can be assigned to which words.
- EM then learns the word likelihood function for each tag, and the tag transition probabilities.
- Merialdo (1994) showed, however, that a tagger trained on even a small amount hand-tagged data works better than one trained via EM.