CSCI 4152/6509 — Natural Language Processing

21-Oct-2009

Lecture 17: Hidden Markov Model

Room: FASS 2176
Time: 11:35 – 12:25

Previous Lecture

- N-gram model (continued):
  - N-gram model assumption,
  - graphical representation,
  - use of log probabilities;
- Markov chain:
  - stochastic process,
  - Markov process,
  - Markov chain;
- Perplexity and evaluation of N-gram models,
- Text classification using language models

10.1 Smoothing

- Add-one smoothing
- Bell-Witten smoothing
- Good-Turing smoothing

Add-one smoothing

The *add-one smoothing* is also known the *Laplace smoothing*. We start with count of 1 for all events, and thus prevent any events to end up with the probability 0. In a unigram example, it would mean that all tokens $w \in V$, where V is the vocabulary, start with count 1. If $|V|$ is the vocabulary size, and N is the number of tokens in a text, the smoothed unigram probabilities end up to be

$$P(w) = \frac{\#(w) + 1}{N + V}$$

where $\#(w)$ denotes the number of occurrences of the token w in the training text.
Similarly, for bigram smoothing for example, the estimated probability would be:

$$P(a|b) = \frac{\#(ba) + 1}{\#(b) + V}$$

If the vocabulary size is very large compared to $\#(b)$, which happens with words, for example, or if b is relatively rare, then this kind of smoothing takes too much of the probability distribution from the seen events and assigns it to the unseen events.

Witten-Bell Discounting

In the context of data compression, Witten and Bell (1991) analyzed several smoothing methods, under the title "The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression". They analyzed the methods A, B, and C, and then, based on a Poisson process modelling, the methods P, X, and XC. It is interesting to note that the method X uses the same, or very similar, idea as in the Good-Turing smoothing.

The method C is what is usually referred to as the Witten-Bell smoothing. It uses an intuitive idea from data compression. Let us assume that we use a data compression method, which uses a dictionary of tokens w_1, w_2, $\ldots$, w_r, so far. As long as the new tokens are from this set, we can encode them in some way, but whenever, a new token appears, we need a special 'escape' code to introduce this token to the vocabulary. In this way, we can think of new tokens appearing as a new event, beside the events of seeing existing tokens. This is supported in practice by seeing approximately constant rate of new words appearing in a text, after some initial reading. By using the frequency of such 'escape' code, we can make an estimate of the probability of seeing previously unseen events, and make sure that we allocate that much probability distribution mass for the smoothing purposes.
This means that if we have seen r distinct tokens in a text of length n, then we have actually seen n events and r 'escape' events, so the probability of seeing new tokens is $\frac{r}{n+r}$. Hence the unigram probability for seen tokens:

$$P(w) = \frac{\#(w)}{n+r}$$

and the total probability for unseen tokens is:

$$\frac{r}{n+r}$$

A convenient fact is that we do not have to know the vocabulary size so far. If we know the vocabulary size, we can now divide the probability for unseen tokens equally, and obtain:

$$P(w) = \frac{r}{(n+r)(|V|-r)}$$

for unseen tokens w.
The probabilities for bigrams and higher-order n-grams are smoothed in a similar way:

$$P(a|b) = \frac{\#(ba)}{\#(b)+r_b}$$

for seen bigrams ba, where r_b is the number of different tokens following b. The remaining probability mass for unseen events:

$$\frac{r_b}{\#(b)+r_b}$$

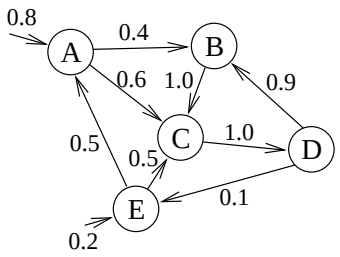
is used for unseen bigrams that start with b, and is usually divided according to lower-order n-grams; which would be unigrams in this case:

$$P(a|b) = \frac{r_b}{\#(b)+r_b} \cdot P(a)$$

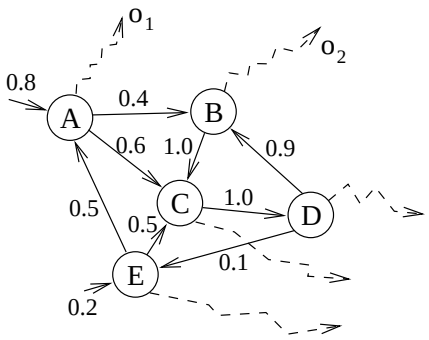
for unseen bigrams ba.

11 Hidden Markov Model (HMM)

Let us take a look again at the concept of Markov chain, as described before:



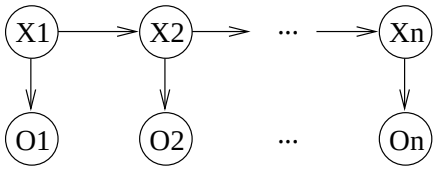
If we assume that the states in such model are not observable, i.e., that they are "hidden," and we can actually observe only an "emitted" symbol, based on another distribution for producing observable symbols, we obtain the *Hidden Markov Model* (HMM). An example of such model is represented in the following figure:



HMMs are successfully used for acoustic modeling in speech recognition. They are also successfully applied to language modeling and POS tagging, among many applications in NLP.

In more precise terms, an HMM includes a finite set of hidden states $Q = \{q_1, q_2, \ldots, q_N\}$. A probability distribution $\pi(q)$ specifies for each state q the probability that it will be the initial state. $\sum_s \pi(s) = 1$ holds. Instead of making transitions from state to state deterministically, for each pair of states q_i and q_j there is a probability of the transition from state q_i to q_i $a_{ij} = a(q_i, q_j)$, so that $\sum_s a(q, s) = 1$ for each state q. From each visited state an observable o is generated, where $o \in V$, V is a finite vocabulary. For an arbitrary state $q \in Q$ and any observable symbol $o \in V$, the output probability $b(q, o)$ that the observable symbol o will be generated is defined, so that for all q $\sum_o b(q, o) = 1$.

Given an HMM, we can generate samples by generating an initial state, producing an observable, and then creating another state, another observable, and so on. For a particular length n, the following graph can be used to illustrate operation of an HMM:



(Note: This is an example of another Bayesian Network, which will be defined later.) The value of X_1 is the initial state of the HMM, and the value of each consecutive variable X_i is the consecutive state of HMM. The values of variables O_1, O_2, ..., are the produced observables.

The HMM assumption formula is:

$$P(X_1, O_1, \ldots, X_n, O_n) = P(X_1) \cdot P(O_1|X_1) \cdot P(X_2|X_1) \cdot P(O_2|X_2) \cdot \ldots \cdot P(X_n|X_{n-1}) \cdot P(O_n|X_n)$$