Machine Learning (CS 726) *29 Nov 2011*

Hidden Markov Models

We now return to supervised learning for a bit. However, we are interested in solving problems whose output has *structure*. We will begin with the simplest non-trival structure: sequences. There are lots of problems from langauge and biology that fit in this paradigm.

Sequence labeling is the following problem: we get a sequence of M inputs and want to produce a sequence of M labels. Training data will be N such input/output pairs.

Example: "The/det man/noun ate/verb a/det sandwith/noun with/prep mayo/noun" We get the sentence and want to produce the part of speech labels. We will usually talk about a *label set* $\mathcal{Y}$ of size L.

Several non-options:

1. Treat it as a classification problem into M^L classes. Doesn't work – too many classes, M not constant.
2. Treat as M separate L-class classification problems. Works okay, but doesn't reflect structure (eg., unlikely to have "verb" follow "det").

We adopt a *Markov* model. This model makes two assumptions:

1. The value of the word depends only on its label and not on surrounding labels/words
2. Conditioned on the K previous labels, the current label is independent of all others

Number 2 is called a K-th order Markov assumption.

The generative model looks something like:

1. Choose a label y_1
2. Generate input x_1 conditioned on y_1
3. For $m = 2 \dots M$,
   (a) Choose a label y_m conditioned on $\{y_{m-1}, \dots, y_{m-K}\}$
   (b) Generate input x_m conditioned on y_m

For simplicity we will stick with $K = 1$; the extension is straightforward.

For this, we need three ingredients:

1. Initial state probabilities π. This is a vector of length L, all positive with sum 1. These are used to choose y_1. π_l is the probability that $y_1 = l$.
2. Emission probabilities β. This is a matrix of size $LtimesV$, where V is the size of the output vocabulary. $\beta_{l,v}$ is the probability of "emitting" word v given that the label is l.
3. Transition probabilities α. This is a matrix of size $LtimesL$, where $\alpha_{p,l}$ is the probability of transitioning to state l given that we were in state p ("p" for previous).

Putting this all together, the joint probability of an input sequence x and state sequence y (both of length M) is:

$$p(x, y \mid \pi, \alpha, \beta) = \pi_{y_1} \beta_{y_1, x_1} \prod_{m=2}^{M} \alpha_{y_{m-1}, y_m} \beta_{y_m, x_m}$$

It becomes frustrating to have to separate out the first element all the time. So what we'll do instead is define y_0 to be a state with a unique label z and define $\alpha_{z,\cdot} = \pi$. Then, we get:

$$\begin{aligned} p(x, y \mid \alpha, \beta) &= \prod_{m=1}^{M} \alpha_{y_{m-1}, y_m} \beta_{y_m, x_m} \\ &= \prod_{m=1}^{M} \prod_{l} \left(\prod_{v} \beta_{l,v}^{\mathbf{1}[x_m = v]} \prod_{p} \alpha_{p,l}^{\mathbf{1}[y_{m-1} = p]} \right)^{\mathbf{1}[y_m = l]} \end{aligned}$$

Given data, we can estimate α and β exactly as in the naive Bayes classification model:

$$\begin{aligned} \hat{\alpha}_{p,l} &= \frac{\text{\# of times } l \text{ follows } p}{\text{\# of times } p \text{ occurs}} \\ \hat{\beta}_{l,v} &= \frac{\text{\# of times } v \text{ has label } l}{\text{\# of times } l \text{ occurs}} \end{aligned}$$

We can generalize this to a simple naive Bayes model over emissions, rather than the "label emits word" part. I.e., if x_m itself is a bunch of features, $x_{m,1}, \ldots, x_{m,D}$, then be_l becomes a vector of length D instead of V, but everything else remains the same.

The key problem in such models is at test time: given a word sequence x, find the best label sequence y. This question is somewhat ill-posed: there are many possible interpretations of "best:"

1. Best = output sequence y that maximizes $p(x, y \mid \alpha, \beta)$
2. Best = output sequence y that maximizes $\prod_m p(x, y_m \mid \alpha, \beta)$

Typically people consider the first because it is "parsimonious."

This problem is most easily considered by picturing a lattice of possible states, with connections between adjacent time steps.

We ask ourselves: what is the probability that the most likely path will pass through state l at time m? We store this as $A_{m,l}$; leaving off conditioning on α and β, we have:

$$A_{m,l} = \max_{y_{1:m-1}} p(y_{1:m-1}, x_{1:m-1}, y_m = l)$$

We can compute this probability recursively:

$$
\begin{aligned}
A_{m+1,l} &= \max_{y_{1:m}} p(y_{1:m}, x_{1:m}, y_m = l) \\
&= \max_{y_{1:m-1}} \max_{p} p(y_{1:m-1}, x_{1:m-1}, y_m = p, x_m, y_m = l) \\
&= \max_{y_{1:m-1}} \max_{p} p(y_{1:m-1}, x_{1:m-1}, y_m = p) p(x_m, y_m = l \mid y_{1:m-1}, x_{1:m-1}, y_m = p) \\
&= \max_{y_{1:m-1}} \max_{p} p(y_{1:m-1}, x_{1:m-1}, y_m = p) p(x_m, y_m = l \mid y_m = p) \\
&= \max_{p} \left[\max_{y_{1:m-1}} p(y_{1:m-1}, x_{1:m-1}, y_m = p) \right] p(x_m, y_m = l \mid y_m = p) \\
&= \max_{p} A_{m,p} p(x_m, y_m = l \mid y_m = p) \\
&= \max_{p} A_{m,p} p(y_m = l \mid y_m = p) p(x_m \mid y_m = l) \\
&= \max_{p} A_{m,p} \alpha_{p,l} \beta_{l,x_m}
\end{aligned}
$$

This analysis leads to an efficient, $\mathcal{O}(ML^2)$ dynamic programming algorithm for finding the most likely state sequences. The only extra thing we need to do is, for each position (m, l), we need to store a "backpointer" to where we came from — i.e., to the value of p that maximized the produce. We store these in a ζ matrix:

1. Initialize $A_{0,z} = 1$ and $A_{0,l} =$ for $l \neq z$ (remember, z is our unique start state).
2. For $m = 0, \ldots, M$,
   (a) Compute $A_{m+1,l} = \max_p A_{m,p} \alpha_{p,l} \beta_{l,x_m}$
   (b) Store $\zeta_{m+1,l} = \operatorname{argmax}_p A_{m,p} \alpha_{p,l} \beta_{l,x_m}$

Now, all we need to do is extract the most likely state sequence from this lattice. We do this by starting at the end and working our way backward. I.e., first we compute the most likely state to be in at time $M + 1$. That is, $\operatorname{argmax}_l A_{M+1,l}$. Call this n. Now, we recurse backward. The label for time step M is $\zeta_{M+1,n}$. This gives us y_M. The label for time step $M - 1$ is ζ_{M,y_M}. This gives us y_{M-1}. In general, we compute $y_m = \zeta_{m+1,y_{m+1}}$.