6.3 Decoding Sequences of Classifications

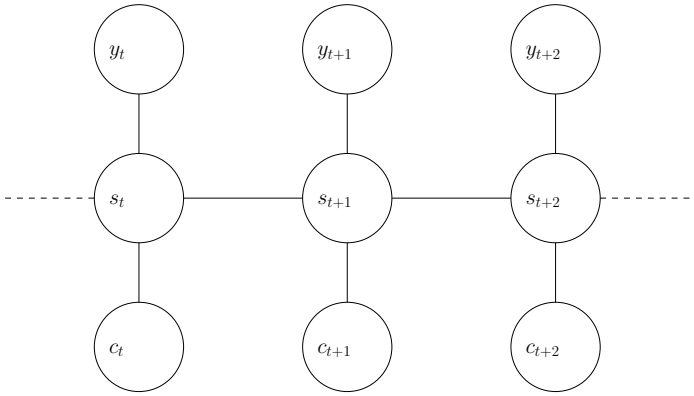
In the book I presented the algorithm described in Fig. 6.6 for finding the *best* classification sequence,

$$\hat{c}_1^T \equiv \underset{c_1^T}{\operatorname{argmax}}\, P(y_1^T, c_1^T),$$

given an observation sequence y_1^T. I claimed that the number of computations required is a linear function of T. In 2013, as I was developing a test suite for a new version of the software, I discovered a configuration for which the algorithm failed to find a *possible* classification sequence even though it was using the model that generated the test data.

Examining the failure, I discovered that for each of the "best" class histories ending in each of the classes at a particular time t, the conditional probability of a particular state s given that class history was 0. However, at time $t+1$, the only state that could produce $y(t+1)$ required that $S(t) = s$. Thus the algorithm calculated $P(y_1^t, \hat{c}_1^t) = 0$ and died.

I had designed the algorithm thinking that the class sequence was Markov and that future evidence could not change what was the best history leading to any class. The following drawing makes my error obvious.



Blocking out s_{t+1} separates the past from the future, but blocking out c_{t+1} doesn't.

Thus to avoid discarding a class history that may at later times become the first portion of the *best* class sequence, one may need to keep a number of class histories that is exponential in sequence length t. That is too many to retain for most realistic applications.

6.3.1 Finding a *Pretty Good Class* Sequence

Rather than finding the *best* class sequence given a sequence of observations y_1^T, I now seek a *good* one using the following ideas. The ideas yield performance similar to that described in the book when applied to the apnea problem. At each time t, I keep a limited collection of class histories which I build in the following three steps:

Step 1, Propagate and Sort: I take the histories I've retained at time $t-1$ and use all possible successors to create a list of histories at time t, then I sort those histories by $P(y_1^t, c_1^t)$.

Step 1, Prune: Initially, I limit the collection using the following criteria:

- If there are two histories ${}^a c_1^t$ and ${}^b c_1^t$ with $P(s_i, y_1^t, {}^b c_1^t) \leq P(s_i, y_1^t, {}^a c_1^t) \forall s_i$, then I drop ${}^b c_1^t$.
- The total number of histories is at most $N_{\max}$
- I don't keep histories c_1^t if $\frac{P(y_1^t, c_1^t)}{P(y_1^t, \bar{c}_1^t)} < R_{\min}$ where $\bar{c}_1^t$ is the best class history in the sorted list.

Step 3, Augment: For each state $s(t)$ that is impossible given the histories saved so far, if there are discarded histories for which $s(t)$ is possible, I save the one that comes first in the sorted list.

Initialize:
  for each c
    $\nu_{\text{next}}(c) = \log\left(\sum_s g(s, C) P_{Y(1),S(1)}\left(y(1), s\right)\right)$

Iterate:
  for t from 1 to T
    Swap $\nu_{\text{next}} \leftrightarrow \nu_{\text{old}}$
    for each c_{next}

      # Find best predecessor
      $c_{\text{best}} = \text{argmax}_{c_{\text{old}}}\left(\nu_{\text{old}}(c_{\text{old}}) + \log\left(\sum_s g(s, c_{\text{best}}) f(t+1, s, c_{\text{best}})\right)\right)$

      # Update ν
      $\nu_{\text{next}}(c_{\text{next}}) = \nu_{\text{old}}(c_{\text{best}}) + \log\left(\sum_s g(s, c_{\text{best}}) f(t+1, s, c_{\text{best}})\right)$

    # Update predecessor array
    Predecessor$[c_{\text{next}}, t] = c_{\text{best}}$

    # Update ϕ
    for s in c_{next}
      Assign $\phi_{\text{next}}(s, c_{\text{next}})$ using Eqn. 6.7

Backtrack:
  $c_1^t = \hat{c}_1^t(\bar{c})$, where $\bar{c} = \text{argmax}_c\, \nu_{\text{next}}(c)$ at $t = T$

Figure 6.6: **This algorithm does not work!** In the book, this figure was entitled *Pseudocode for the Viterbi algorithm for class sequences.*