Course on Artificial Intelligence and Intelligent Systems

Hidden Markov Models and their implementation in the PRISM system

Henning Christiansen
Roskilde University, Computer Science Dept.
 Version 18 sep 2008

Contents

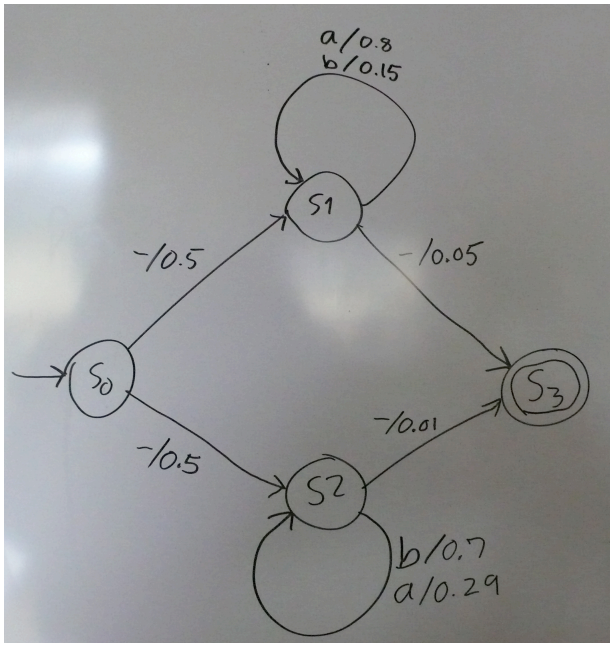
1 Introduction

In this note we show examples of how Hidden Markov Models (HMMs) can be implemented and used in the PRISM system. With PRISMs facilities, we can ask for the most probable series of state transitions that can explain a given observed sequence, and we can also adapt a HMM in PRISM to search for more abstract criteria.

2 A simple example

We consider a HMM with four states, s0–3. From s0 it can move to either of s1 and s2 with equal chance and not emitting any symbol. States s1 and s2 behave in similar ways, but with different probability: they can emit an `a` or a `b` and stay in the same state, or it can go to the final state s3 without emitting any symbol.

The following drawing shows a graphical representation of this HMM. The arrow pointing from nowhere to s1 indicates that s1 is the initial state, and the double circle around s3 (as well as no arrows away from it) indicates that it is a final state. Each arrow indicates state transition with indication of emitted symbol; symbol before the slash and probability after; as "–" indicates that no symbol is emitted.



Notice that this way of implementing a step to a final state by a probability gives a so-called geometric distribution of string lengths which may not always be what we want; in fact, here the lengths are governed by the sum of two geometric distributions. Here we chose it for simplicity of implementation. This HMM can be implemented in PRISM as follows; we use here a top-level predicated called `hmm/1`, and for each state, one predicate and one random variable to characterize next event.

```
values(s0,[s1,s2]).
values(s1,[s1a,s1b,s3]).
values(s2,[s2a,s2b,s3]).

set_ss:-
  set_sw(s0, [0.5, 0.5]),
  set_sw(s1, [0.8, 0.15, 0.05]),
  set_sw(s2,  [0.29, 0.7, 0.01]).

target(hmm,1).

hmm(S):- hmm0(S).

hmm0(S):-
    msw(s0,Next),
    (Next=s1 -> hmm1(S)
     ; Next=s2 -> hmm2(S)).

hmm1(S):-
    msw(s1,Next),
    (Next=s1a -> S=[a|Sx], hmm1(Sx)
     ; Next=s1b -> S=[b|Sx], hmm1(Sx)
     ; Next=s3  -> hmm3(S)).

hmm2(S):-
    msw(s2,Next),
    (Next=s2a -> S=[a|Sx], hmm2(Sx)
     ; Next=s2b -> S=[b|Sx], hmm2(Sx)
     ; Next=s3  -> hmm3(S)).

hmm3([]).
```

We can illustrate some of PRISM's facilities tha are useful for HMMs by the following queries and answers. We do not explain everything in full detail, so you may in some cases need to consult the PRISM manual.

We may use it for generating samples in the following way (notice we must remember to set the probabilities), which may not be that interesting in this case, but we show it anyhow.

```
?- set_ss.
yes
?- hmm(S).
S = [a,a,b,a,a,b,a,a,a,a,a,a,a,a,a,a,a,b,a,a,a,a,a,a,b,a,
a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,b,a,b,a,a,a,b,a,a,a,a,a,a,a,
a,a,a,a,a,a,a,a,a,a,a,a,b] ?
```

The following query asks for the probability that the model produces a list of one `a` and nothing more.

```
?- prob(hmm([a])).
Probability of hmm([a]) is: 0.021450000000000
```

It is interesting to compare this probability with the viterbi probability for the same thing.

```
?- viterbi(hmm([a])).
Viterbi_P = 0.02
```

This is slightly lower that the overall probability, as it is defined as the probability of the most probably way that the model can generate this sequence. We may guess that this is via s1 (and that the difference is due to the less probable history that the `a` was produced through s2). We can confirm this by the following query which gives an explanation of this "most probable way".

```
?- viterbif(hmm([a])).

hmm([a])  <= hmm0([a])
hmm0([a])  <=
     hmm1([a]) & msw(s0,s1)
hmm1([a])  <=
     hmm1([]) & msw(s1,s1a)
hmm1([])  <= msw(s1,s3)

Viterbi_P = 0.02
```

3 Providing more concise information

In case the HMM or the string is larger, this way of using the model may not be that interesting. So we should extend the implementation of the model, i.e., the `hmm/1` predicate, with an argument that indicates which property we are interested in in a more abstract way. It this simple model we have considered so far, we may find it sufficient to know whether we went through the state s1 or s2 to produce a given output. To do this, we modify the program in the previous section as follows.

```
target(hmm,1).

hmm(S):- hmm(S,_).

hmm(S,X):- hmm0(S,X).

hmm0(S,Next):-
    msw(s0,Next),
    ...
```

The rest of the program is unaltered, so the modification was to add an extra argument to `hmm` that holds the abstract explanation. Thus, we have now both a predicate `hmm/1` which gives the same result as before, and `hmm/2` which includes the abstract explanation.

Let us first test it by generating a sample.

```
?- hmm(S,A).
S = [a,a,a,a,a,a,b,a,a,a,a,a,a,b,a,a,a,a,a,a,b,b]
A = s1 ?
```

It seems quite reasonable that this list, with many `a`'s, was generated via state s1, although it in principle it could also have been produced via s2, likely (we may believe) with a smaller probability.

4 Using the HMM to produce abstract explanations of observations

We can think of the list of symbol produced by the HMM as being produced over time by some process in reality, and we want to predict the most likely explanation why it produced this sequence. Assume, for example, we have observed the list

L = `[a,a,a,a,a,a,b,a,a,a,a,a,a,b,a,a,a,a,a,a,a,b,b]`.

So we can consider the observation of L as a fact, and we face the abductive question of what caused L. More precisely, what are the (most) probably explanation(s) and the probabilities for the different possible explanation.

In other words, we are interested in knowing the conditional probabilities, P(E|L), for the observed L and possible E. Rephrasing this in terms of the implemented model, we are interested to know probabilities P(`hmm(L,s1)`|`hmm(L)`) and P(`hmm(L,s2)`|`hmm(L)`). For this purpose, we can use PRISMs predicates for conditional hindsight probabilities.

```
?- L=[a,a,a,a,a,a,b,a,a,a,a,a,a,b,a,a,a,a,a,a,a,b,b],
   chindsight_agg(hmm(L),hmm(_,query)).

conditional hindsight probabilities:
  hmm(*,s1): 0.999998891844173
  hmm(*,s2): 0.000001108155827
```

As expected the probability for s1 is very high and the one for s2 is very small but different from zero.

With this long sequence observed, there is intuitively a very high evidence for a particular cause. With less evidence, the answers is expected to be less clear, as the following example seems to confirm.

```
?- S=[a,b], chindsight_agg(hmm(S),hmm(S,query)).

conditional hindsight probabilities:
  hmm([a,b],s1): 0.747198007471980
  hmm([a,b],s2): 0.252801992528020
```

So if the task is to identify, say, a fault inside a machine from listening to the sounds it produces, it may be a good idea to have in run for a longer time, as in the example above, there is 25% chance of being wrong if we state that s1 is the case.

Finally, the following combination of viterbi computation and hindsight probability may be useful in those cases where there are many different possible, abstract explanations. We get the most probable explanation together with a probability showing how much we can trust it.

```
?- viterbig(hmm([a,b],A)), chindsight_agg(hmm([a,b]), hmm([a,b],A)).

conditional hindsight probabilities:
  hmm([a,b],s1): 0.747198007471980
A = s1 ?
```

5 A generic program structure for HMMs searching for the final state

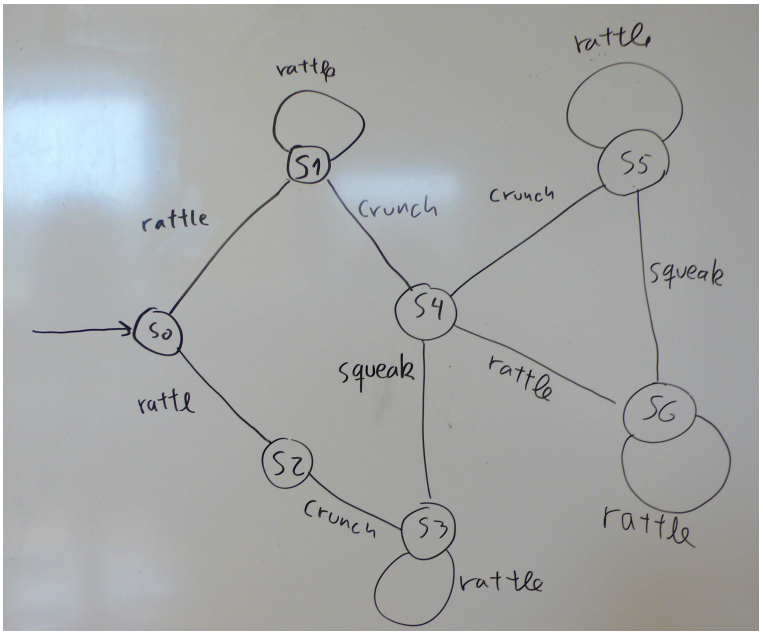
By means of an example, we indicate how a generic program structure may be used for a particular sort of problems.

Consider a old house with an obscure and dark cellar; the floor is just soil and there is a lot of strange rubbish that one will run into and make noise when moving around down there. The possible sounds are: `rattle`, `crunch`, and `squeak`. Furthermore, if someone is digging in the floor, this will be heard as a sequence of one or more `rattle` sounds. If you are in the ground floor of the house and someone is in the cellar, you can hear these sounds, but with no sense of direction.

It is known that a treasure is buried somewhere in that cellar, but it is not known where exactly. Imagine now, that we are in the house and we see a sinister-looking person with a shovel entering the cellar from outside; we expect it is a criminal who wants to find the treasure. As it is dark in the cellar and the person did not carry also a lamp, we expect that he or she will walk around randomly and dig a bit here and there, perhaps even the same place several times.

Now we listen to the sequence of sounds, and we are speculating about the question: Where in the cellar is this mysterious person.

This is a question for which a HMM is appropriate. The following diagram is a map of the cellar, with s0 being the entry. Each line corresponds to a corridor and is marked by the sound it makes when someone passes it. The nodes are places where corridors meet and, for some of them, it is relevant to dig; the latter is indicated by the circular line labeled `rattle`. The diagram should not be read in the same way as the previous diagram, as the corridors can be passed in both directions.



In the following, we show an implementation of a HMM for this problem, and we will structure the code in a generic fashion, so it can solve similar problems by just exchanging the `msw` definitions.

For each state, we indicate the set of possible next state plus sound emitted in each case. For each state, we indicate that it can move into one called `stop`, that does not corresponds to a

location or state in the usual sense, but just indicates the position in the sequence of sounds we have experienced until the given time.

```
values(s0,[s1+rattle,s2+rattle,stop+nothing]).
values(s1,[s0+rattle,s1+rattle,s4+crunch,stop+nothing]).
values(s2,[s0+rattle,s3+crunch,stop+nothing]).
values(s3,[s2+crunch,s3+rattle,s4+squeak,stop+nothing]).
values(s4,[s1+crunch,s3+squeak,s5+crunch,s6+rattle,stop+nothing]).
values(s5,[s4+crunch,s5+rattle,s6+squeak,stop+nothing]).
values(s6,[s4+rattle,s5+squeak,s6+rattle,stop+nothing]).
```

We were interesting in the position of the person at a given time, so we will make the implementation return the state in which the person is at the given moment, i.e., the one which experiences the `stop`. This is expressed as follows. Notice that there only one rule that takes care of all possible state transitions.

```
target(hmm,1).

hmm(S):- hmm(S,_).

hmm(S,Where):- hmm(S,s0,Where).

hmm(S,From,Final):-
    msw(From,Next+Sound),
    (Next=stop -> S=[], Final=From
     ;
     S=[Sound|Sx], hmm(Sx,Next,Final)).
```

We have no expectations about different probabilities, so we assume an even distribution which is PRISM's default when no probabilities are given or learned. Notice that the probability of `stop` really does not matter when the probabilities we seek are always conditioned by a given observed sequence. To test, we may generate an arbitrary sample.

```
?- hmm(S,A).
S = [rattle,crunch,crunch,crunch,crunch,crunch,rattle,squeak,crunch,rattle,squeak,squeak]
A = s5 ?
```

However, the sort of problems, we want to solve with this HMM, is to estimate probabilities for the different possible positions of the thief, given the observed sequence of sounds. The following query tests a specific sequence, uses viterbi to find the most probable explanation, and hindsight calculation to find the probability of it.

```
?- S = [rattle,crunch,crunch,crunch,crunch,crunch,rattle,squeak,
       crunch,rattle,squeak,squeak],
       viterbig(hmm(S,A)), chindsight_agg(hmm(S), hmm(S,A)).

conditional hindsight probabilities:
  hmm([rattle,....,squeak],s5): 0.536480686695279
S = [rattle,....,squeak]
A = s5 ?
```

In this particular case, we see that the conditional probability is not close to 1.0, so we may consider to check the other possible, final states (i.e., present positions of the thief). To do this, we can drop the call to `viterbig` in the query.

```
?- S = [rattle,crunch,crunch,crunch,crunch,crunch,rattle,squeak,
       crunch,rattle,squeak,squeak],
       chindsight_agg(hmm(S), hmm(S,query)).

conditional hindsight probabilities:
  hmm([rattle,....,squeak],s5): 0.536480686695279
  hmm([rattle,....,squeak],s6): 0.463519313304721
S = [rattle,....,squeak] ?
```

So the conclusion is that there are two possible positions of almost identical probability. If we had known more about the thief's preferences (in terms of probabilities for state transitions), we might have got a more significant difference in the probabilities.

The interesting part of this example is the program structure, where we expressed the Markov condition in just a single clause, and with details given by the `values` declarations only.

In general, it is an advantage to use such generic programs, but it will be a bit more difficult to suggest a generic structure, if the abstract explanations, we were looking for, are expressed in more complicated was.

6 Using learning to train HMMs (sketch)

In the example above, we used a well-defined abstract explanation criteria, namely the final state after a given sequence, which is uniquely determined by the sequence of state transition.

For applications in bioinformatics, HMMs are also used for searching more subtle patterns, e.g., searching for "important" substrings such as "Where is the gene?" So the explanation may be something like "There is a gene from position 124 to position 137".[1]

To express this, we can describe a HMM which has one state for skipping arbitrary letters and another set of states for recognizing the gene. Such a HMM will be highly ambiguous in the sense that there will be quite many ways that a given sequence can be interpreted as a series of state transitions.

However, if we are given a set of sequences S that have been studied in the laboratory, so that competent researchers have pointed out the positions of genes in those sequences, we can use S as training data. When the net is trained in this way, it can give suggestions weighted according to the probabilities for positions of genes in new sequences, so that the biochemists get hypotheses for where to focus their laboratory investigations.

For such purposes, a first-order HMM may often be too simple, and higher-order or hierarchical HMMs, or combinations thereof, may be suggested, or other sorts of extensions to HMMs may be developed.

The advantage of using PRISM in such a case, is that it is fairly easy to program and test different variations.

[1] These figures seem to indicate that the sequence is taken from a very primitive bacteria.

7 Exercise: Spellchecking without a dictionary

A file with the name `words.dat` will be made available to you which contain a raw listing of all words that appear in the LaTeX source file for the note on Prolog and Artificial Intelligence that we have used in the course. Word is understood in a broad sense so that it also includes names of LaTeX macros, and words or names that contain special characters may be chopped into two, as any special character have been as a separator. It looks as follows:

```
word("documentclass").
word("pt").
word("article").
word("usepackage").
...
word("prolog").
word("is").
word("a").
word("programming").
word("language").
word("that").
word("is").
word("very").
word("easy").
word("to").
word("learn").
...
word("end").
word("document").
```

It contains a total of 34729 words which include duplicates; all capital letters have been replaced by small ones; one-letter words except "i" and "a" have been deleted but otherwise everything has been kept.

Recall that a string such as `"word"` is syntactic sugar for a list of character values, i.e., `"word"=[119,111,114,100]`.

You may use a predicate `write_string/1` which prints a list of character values so it looks as a string.

7.1 Question

Write a PRISM program that implements a first-order HMM for recognizing words that are "similar" to the ones in the text.

Train the model with the file, and try to generate different samples of "probable words". Can you suggest a way to measure the "goodness" of a given proposed word using the trained HMM?

7.2 Question

Modify you model so it becomes a second-order HMM, and consider whether it works better than the previous one.

(We did not show second-order HMMs in this note; we recall that the next letter in a second-order HMM depends on the previous two letters, i.e., the probability for letter n is conditioned by letters $n-1$ and $n-2$.)

7.3 Question

Consider possible extensions to third or even higher order, and give your oppinion about space and time complexity as the order increases. Will you expect any accuracy.

7.4 Post scriptum

whistser folu annitininventionstra tofyintionfo verpow endra noinci