Hidden Markov Models

Markov Models

A Markov model is a probabilistic process over a finite set, $\{S_1, ..., S_k\}$, usually called its *states*. Each state-transition generates a character from the *alphabet* of the process. We are interested in matters such as the probability of a given state coming up next, $pr(x_t=S_i)$, and this may depend on the prior history to t-1.

In computing, such processes, if they are reasonably complex and interesting, are usually called Probabilistic Finite State Automata (PFSA) or Probabilistic Finite State Machines (PFSM) because of their close links to deterministic and non-deterministic finite state automata as used in formal language theory.

Examples are (hidden) Markov Models of biased coins and dice, formal languages, the weather, etc.; Markov models and Hidden Markov Models (HMM) are used in Bioinformatics to model DNA and protein sequences.

Order 0 Markov Models

An order 0 Markov model has no "memory": $pr(x_t = S_i) = pr(x_{t'} = S_i)$, for all points t and t' in a sequence.

An order 0 Markov model is equivalent to a multinomial probability distribution. The issue of the *accuracy* with which the model's parameters should be stated, and hence the model's complexity, was investigated by Wallace and Boulton (1968, appendix).

Order 1 Markov Models

An order 1 (first-order) Markov model has a memory of size 1. It is defined by a table of probabilities $pr(x_t=S_i \mid x_{t-1}=S_j)$, for $i = 1..k$ & $j = 1..k$. You can think of this as k order 0 Markov models, one for each S_j.

Order m Markov Models

The *order* of a Markov model of fixed order, is the length of the history or *context* upon which the probabilities of the possible values of the next state depend. For example, the next state of an order 2 (second-order) Markov Model depends upon the two previous states.

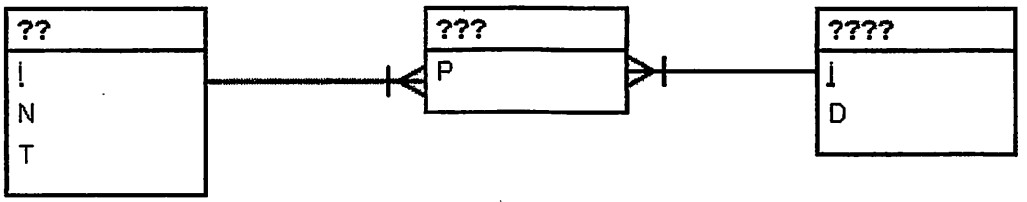
Notes about scientific notation

Suppose you are reading a document which includes this terse scientific notation:

$$N_i, \quad T_i, \quad D_j, \quad P_{ij}$$

To understand it better you can reverse engineer a data model where:

- Each subscript becomes an identifying descriptor, as appropriate, attribute or link.
  - A subscript such as "i", which is by itself with "N" maps to an identifying attribute, identifying an independent entity. Likewise for "j".
  - The "i" in P_{ij} maps to an identifying link, partially identifying an intersection or dependent entity. [In this case it's an intersection entity
- Each variable becomes a non-identifying attribute of an entity with the variable's subscript(s) serving as the entity's identifier.



To make the data model well-formed and understandable, you need to:

- Determine each entity name, which may be explicit in the document or be implicit, requiring you to pick the name
- Substitute a human-understandable name for each terse attribute.

In this case, pick entity names and make these substitutions:

- i → C_id; j → S_code;
  N → C_name; T → C_type;
  D → S_description; P → Proficiency.

Then, voila, the Creature-Achievement-Skill fragment appears.

This is a quite empowering aspect of mastering data modeling. You can understand really hard material by applying your content-neutral skills. You can do this with something you are generating or material written by others that you are reading. For example, you can find holes in notation, which can manifest as descriptor misplacement. You can ask if the subscripts, which generally have integer values, are nominal, ordinal or numeric scale, that is, is it a set or a sequence, and can one do sensible arithmetic on the values or not. You can practice the mantras "what do we mean by one of these", "one or many", and "anchor understanding with instances.

Cool, eh.