Introduction

Often we are interested in finding patterns which appear over a space of time. These patterns occur in many areas; the pattern of commands someone uses in instructing a computer, sequences of words in sentences, the sequence of phonemes in spoken words - any area where a sequence of events occurs could produce useful patterns.

Consider the simple example of someone trying to deduce the weather from a piece of seaweed - folklore tells us that `soggy' seaweed means wet weather, while `dry' seaweed means sun. If it is in an intermediate state (`damp'), then we cannot be sure. However, the state of the weather is not restricted to the state of the seaweed, so we may say on the basis of an examination that the weather is probably raining or sunny. A second useful clue would be the state of the weather on the preceding day (or, at least, its probable state) - by combining knowledge about what happened yesterday with the observed seaweed state, we might come to a better forecast for today. This is typical of the type of system we will consider in this tutorial.

- First we will introduce systems which generate probabalistic patterns in time, such as the weather fluctuating between sunny and rainy.
- We then look at systems where what we wish to predict is not what we observe - the underlying system is hidden. In the above example, the observed sequence would be the seaweed and the hidden system would be the actual weather.
- We then look at some problems that can be solved once the system has been modeled. For the above example, we may want to know
    1. What the weather was for a week given each day's seaweed observation.
    2. Given a sequence of seaweed observations, is it winter or summer? Intuitively, if the seaweed has been dry for a while it may be summer, if it has been soggy for a while it might be winter.

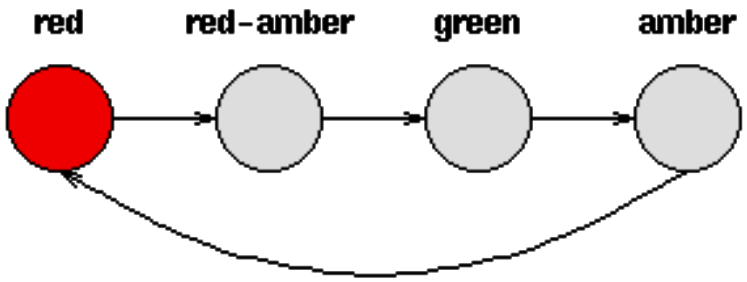
Generating Patterns

Section 1 - Page 1

Deterministic Patterns

Consider a set of traffic lights; the sequence of lights is red - red/amber - green - amber - red. The sequence can be pictured as a state machine, where the different states of the traffic lights follow each other.

e.g.



Notice that each state is dependent solely on the previous state, so if the lights are green, an amber light will always follow - that is, the system is deterministic. Deterministic systems are relatively easy to understand and analyse, once the transitions are fully known.

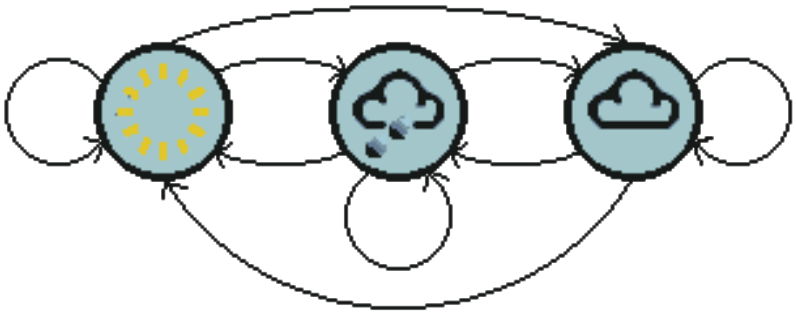
Non-deterministic patterns

To make the weather example a little more realistic, introduce a third state - cloudy. Unlike the traffic light example, we cannot expect these three weather states to follow each other deterministically, but we might still hope to model the system that generates a weather pattern.

One way to do this is to assume that the state of the model depends only upon the previous states of the model. This is called the Markov assumption and simplifies problems greatly. Obviously, this may be a gross simplification and much important information may be lost because of it.

When considering the weather, the Markov assumption presumes that today's weather can always be predicted solely given knowledge of the weather of the past few days - factors such as wind, air pressure etc. are not considered. In this example, and many others, such assumptions are obviously unrealistic. Nevertheless, since such simplified systems can be subjected to analysis, we often accept the assumption in the knowledge that it may generate information that is not fully accurate.

A Markov process is a process which moves from state to state depending (only) on the previous *n* states. The process is called an *order n* model where *n* is the number of states affecting the choice of next state. The simplest Markov process is a first order process, where the choice of state is made purely on the basis of the previous state. Notice this is not the same as a deterministic system, since we expect the choice to be made probabalistically, not deterministically. The figure below shows all possible first order transitions between the states of the weather example.



Notice that for a first order process with M states, there are M^2 transitions between states since it is possible for any one state to follow another. Associated with each transition is a probability called the state transition probability - this is the probability of moving from one state to another. These M^2 probabilities may be collected together in an obvious way into a state transition matrix. Notice that these probabilities do not vary in time - this is an important (if often unrealistic) assumption.

The state transition matrix below shows possible transition probabilities for the weather example;

$$
\begin{array}{cc}
 & \textit{Today} \\
 & \begin{array}{ccc} \text{sun} & \text{cloud} & \text{rain} \end{array} \\
\textit{Yesterday} \begin{array}{c} \text{sun} \\ \text{cloud} \\ \text{rain} \end{array} & \begin{bmatrix} 0.50 & 0.375 & 0.125 \\ 0.25 & 0.125 & 0.625 \\ 0.25 & 0.375 & 0.375 \end{bmatrix}
\end{array}
$$

- that is, if it was sunny yesterday, there is a probability of 0.5 that it will be sunny today, and 0.375 that it will be cloudy. Notice that (because the numbers are probabilities) the sum of the entries for each row is 1.

To initialise such a system, we need to state what the weather was (or probably was) on the day after creation; we define this in a vector of initial probabilities, called the π vector.

Sun	Cloud	Rain
1.0	0.0	0.0

- that is, we know it was sunny on day 1.
We have now defined a first order Markov process consisting of :

- **states** : Three states - sunny, cloudy, rainy.
- **π vector** : Defining the probability of the system being in each of the states at time 0.
- **state transition matrix** : The probability of the weather given the previous day's weather.

Any system that can be described in this manner is a Markov process.

Summary

We are trying to recognise patterns in time, and in order to do so we attempt to model the process that could have generated the pattern. We use discrete time steps, discrete states, and we may make the Markov assumption. Having made these assumptions, the system producing the patterns can be described as a Markov process consisting of a **π vector and a state transition matrix**. An important point about the assumption is that the state transition probabilites *do not* vary in time - the matrix is fixed throughout the life of the system.

Patterns generated by a hidden process

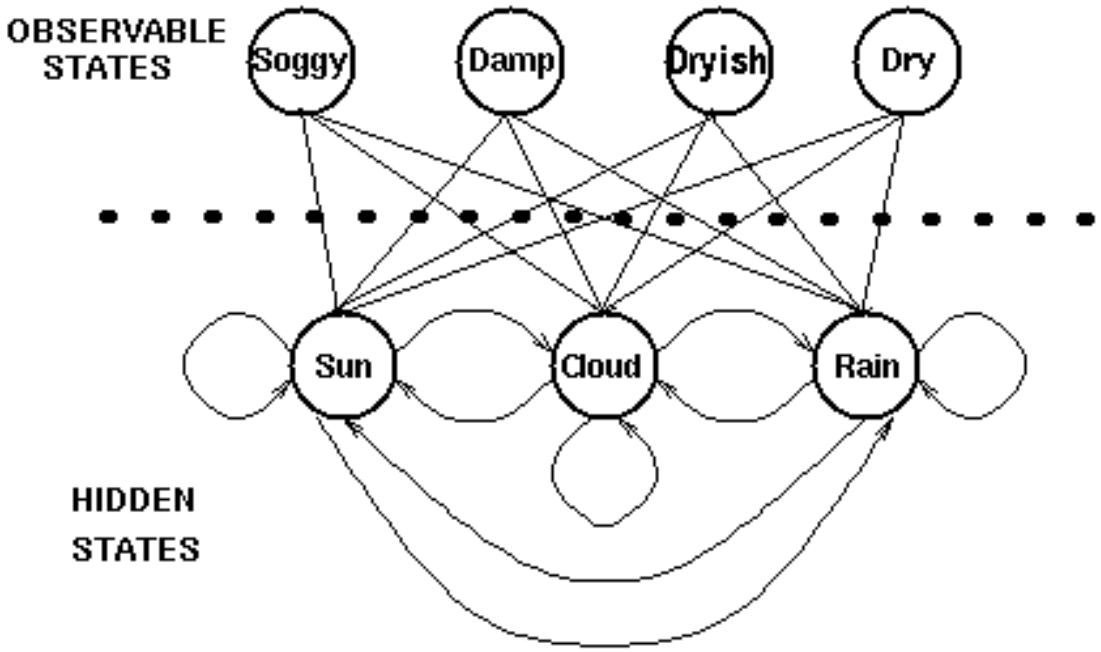
When a Markov process may not be powerful enough

In some cases the patterns that we wish to find are not described sufficiently by a Markov process. Returning to the weather example, a hermit may perhaps not have access to direct weather observations, but does have a piece of seaweed. Folklore tells us that the state of the seaweed is probabalistically related to the state of the weather - the weather and seaweed states are closely linked. In this case we have two sets of states, the observable states (the state of the seaweed) and the hidden states (the state of the weather). We wish to devise an algorithm for the hermit to forecast weather from the seaweed and the Markov assumption without actually ever seeing the weather.
A more realistic problem is that of recognising speech; the sound that we hear is the product of the vocal chords, size of throat, position of tongue and several other things. Each of these factors interact to produce the sound of a word, and the sounds that a speech recognition system detects are the changing sound generated from the internal physical changes in the person speaking.

Some speech recognition devices work by considering the internal speech production to be a sequence of hidden states, and the resulting sound to be a sequence of observable states generated by the speech process that at best approximates the true (hidden) states. In both examples it is important to note that the number of states in the hidden process and the number of observable states may be different. In a three state weather system (sunny, cloudy, rainy) it may be possible to observe four grades of seaweed dampness (dry, dryish, damp,soggy); pure speech may be described by (say) 80 phonemes, while a physical speech system may generate a number of distinguishable sounds that is either more or less than 80.
In such cases the observed sequence of states is probabalistically related to the hidden process. We model such processes using a hidden Markov model where there is an underlying hidden Markov process changing over time, and a set of observable states which are related somehow to the hidden states.

Hidden Markov Models

The diagram below shows the hidden and observable states in the weather example. It is assumed that the hidden states (the true weather) are modelled by a simple first order Markov process, and so they are all connected to each other.



The connections between the hidden states and the observable states represent the probability of generating a particular observed state given that the Markov process is in a particular hidden state. It should thus be clear that all probabilities `entering' an observable state will sum to 1, since in the above case it would be the sum of *Pr*(Obs|Sun), *Pr*(Obs|Cloud) and *Pr*(Obs|Rain).
In addition to the probabilities defining the Markov process, we therefore have another matrix, termed the confusion matrix, which contains the probabilities of the observable states given a particular hidden state. For the weather example the confusion matrix might be;

		Seaweed			
		Dry	**Dryish**	**Damp**	**Soggy**
weather	**Sun**	0.60	0.20	0.15	0.05
	Cloud	0.25	0.25	0.25	0.25
	Rain	0.05	0.10	0.35	0.50

Notice that the sum of each matrix row is 1.

Summary

We have seen that there are some processes where an observed sequence is probabalistically related to an underlying Markov process. In such cases, the number of observable states may be different to the number of hidden states.
We model such cases using a hidden Markov model (HMM). This is a model containing two sets of states and three sets of probabilities;

- **hidden states** : the (TRUE) states of a system that may be described by a Markov process (e.g., the weather).
- **observable states** : the states of the process that are `visible' (e.g., seaweed dampness).

Hidden Markov Models

Section 1 - Page 1

Definition of a hidden Markov model

A hidden Markov model (HMM) is a triple (π,A,B).

$\Pi = (\pi_i)$ the vector of the initial state probabilities;

$A = (a_{ij})$ the state transition matrix; $Pr(x_{i_t} \mid x_{j_{t-1}})$

$B = (b_{ij})$ the confusion matrix; $Pr(y_i \mid x_j)$

Each probability in the state transition matrix and in the confusion matrix is time independent - that is, the matrices do not change in time as the system evolves. In practice, this is one of the most unrealistic assumptions of Markov models about real processes.

Uses associated with HMMs

Once a system can be described as a HMM, three problems can be solved. The first two are pattern recognition problems: Finding the probability of an observed sequence given a HMM (evaluation); and finding the sequence of hidden states that most probably generated an observed sequence (decoding). The third problem is generating a HMM given a sequence of observations (learning).

1. Evaluation

Consider the problem where we have a number of HMMs (that is, a set of (π,A,B) triples) describing different systems, and a sequence of observations. We may want to know which HMM most probably generated the given sequence. For example, we may have a `Summer' model and a `Winter' model for the seaweed, since behaviour is likely to be different from season to season - we may then hope to determine the season on the basis of a sequence of dampness observations.

We use the forward algorithm to calculate the probability of an observation sequence given a particular HMM, and hence choose the most probable HMM.
This type of problem occurs in speech recognition where a large number of Markov models will be used, each one modelling a particular word. An observation sequence is formed from a spoken word, and this word is recognised by identifying the most probable HMM for the observations.

2. Decoding

Finding the most probable sequence of hidden states given some observations

Another related problem, and the one usually of most interest, is to find the hidden states that generated the observed output. In many cases we are interested in the hidden states of the model since they represent something of value that is not directly observable.

Consider the example of the seaweed and the weather; a blind hermit can only sense the seaweed state, but needs to know the weather, i.e. the hidden states.
We use the Viterbi algorithm to determine the most probable sequence of hidden states given a sequence of observations and a HMM.
Another widespread application of the Viterbi algorithm is in Natural Language Processing, to tag words with their syntactic class (noun, verb etc.) The words in a sentence are the observable states and the syntactic classes are the hidden states (note that many words, such as wind, fish, may have more than one syntactical interpretation). By finding the most probable hidden states for a sentence of words, we have found the most probable syntactic class for a word, given the surrounding context. Thereafter we may use the primitive grammar so extracted for a number of purposes, such as recapturing `meaning'.

3. Learning

Generating a HMM from a sequence of obersvations

The third, and much the hardest, problem associated with HMMs is to take a sequence of observations (from a known set), known to represent a set of hidden states, and fit the most probable HMM; that is, determine the (π,A,B) triple that most probably describes what is seen.
The forward-backward algorithm is of use when the matrices A and B are not directly (empirically) measurable, as is very often the case in real applications.

Summary

HMMs, described by a vector and two matrices (π,A,B) are of great value in describing real systems since, although usually only an approximation, they are amenable to analysis. Commonly solved problems are:

1. Matching the most likely system to a sequence of observations -evaluation, solved using the forward algorithm;
2. determining the hidden sequence most likely to have generated a sequence of observations - decoding, solved using the Viterbi algorithm;
3. determining the model parameters most likely to have generated a sequence of observations - learning, solved using the forward-backward algorithm.

Source: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html