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.

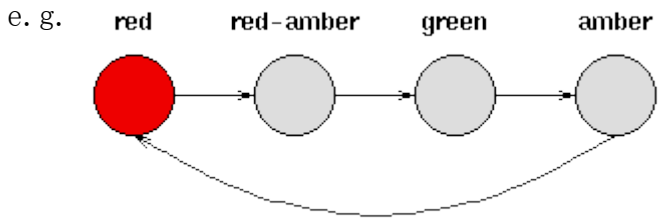
This is typical of the type of system we will consider in this tutorial.

- 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

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.



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

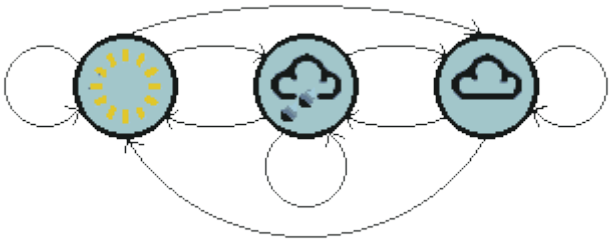
To make the weather example a little more realistic, introduce a third state - cloudy. Unlike the traffic light example, we cannot expect these three weather states to follow each other deterministically, but we might still hope to model the system that generates a weather pattern.

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;

		Today		
		sun	cloud	rain
Yesterday	sun	0.50	0.375	0.125
	cloud	0.25	0.125	0.625
	rain	0.25	0.375	0.375

- 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.

- 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.

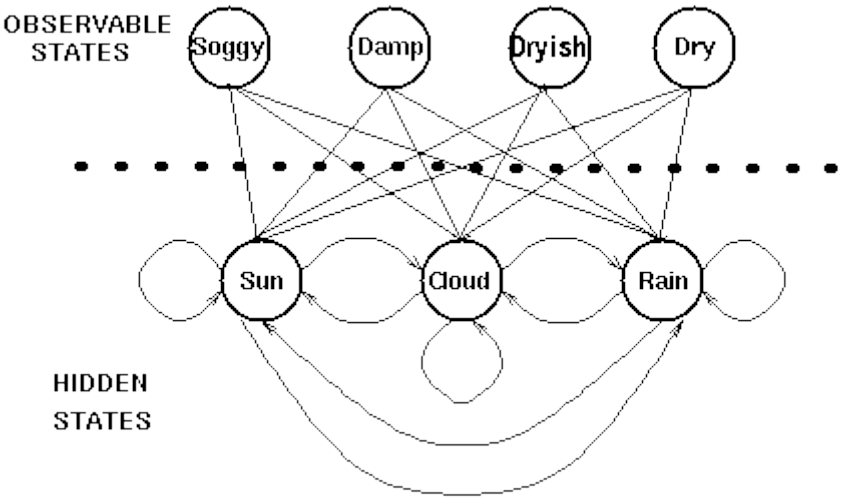
A more realistic problem is that of recognising speech; the sound that we hear is the product of the vocal chords, size of throat, position of tongue and several other things. Each of these factors interact to produce the sound of a word, and the sounds that a speech recognition system detects are the 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;

weather	Seaweed: Dry	Dryish	Damp	Soggy
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).
- π **vector** : contains the probability of the (hidden) model being in a particular hidden state at time t = 1.
- **state transition matrix** : holding the probability of a hidden state given the previous hidden state.
- **confusion matrix** : containing the probability of observing a particular observable state given that the hidden model is in a particular hidden state.

Thus a hidden Markov model is a standard Markov process augmented by a set of observable states, and some probabalistic relations between them and the hidden states.

Hidden Markov Models

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.

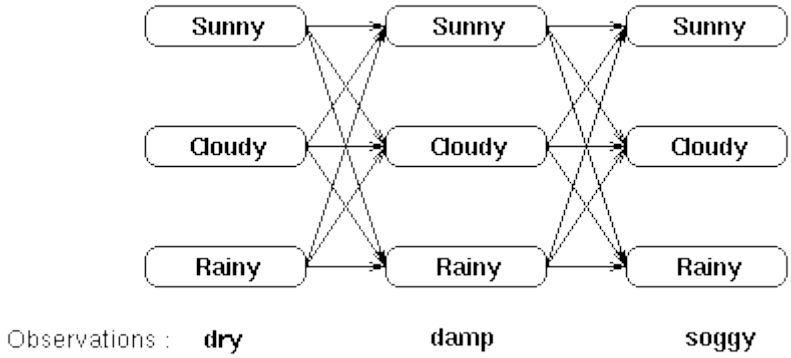
Forward Algorithm

Finding the probability of an observed sequence

1. Exhaustive search for solution

We want to find the probability of an observed sequence given an HMM - that is, the parameters (π,A,B) are known. Consider the weather example; we have a HMM describing the weather and its relation to the state of the seaweed, and we also have a sequence of seaweed observations. Suppose the observations for 3 consecutive days are (dry,damp,soggy) - on each of these

days, the weather may have been sunny, cloudy or rainy. We can picture the observations and the possible hidden states as a trellis.



Each column in the trellis shows the possible state of the weather and each state in one column is connected to each state in the adjacent columns. Each of these state transitions has a probability provided by the state transition matrix. Under each column is the observation at that time; the probability of this observation given any one of the above states is provided by the confusion matrix.

It can be seen that one method of calculating the probability of the observed sequence would be to find each possible sequence of the hidden states, and sum these probabilities. For the above example, there would be 3^3=27 possible different weather sequences, and so the probability is

Pr(dry,damp,soggy | HMM) = Pr(dry,damp,soggy | sunny,sunny,sunny) + Pr(dry,damp,soggy | sunny,sunny ,cloudy) + Pr(dry,damp,soggy | sunny,sunny ,rainy) + ... Pr(dry,damp,soggy | rainy,rainy ,rainy)

Calculating the probability in this manner is computationally expensive, particularly with large models or long sequences, and we find that we can use the time invariance of the probabilities to reduce the complexity of the problem.

2. Reduction of complexity using recursion

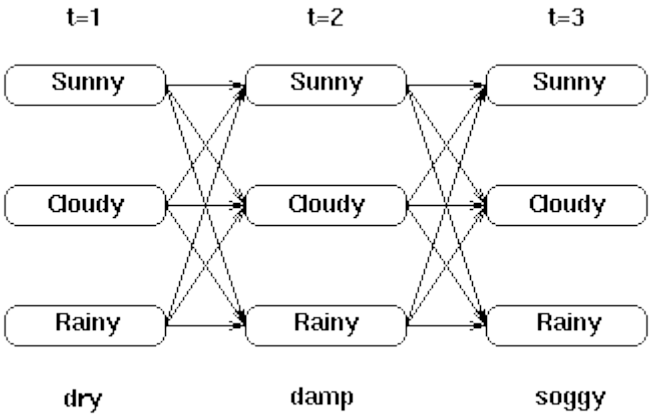
We will consider calculating the probability of observing a sequence recursively given a HMM. We will first define a partial probability, which is the probability of reaching an intermediate state in the trellis. We then show how these partial probabilities are calculated at times t=1 and t=n (> 1).

Suppose throughout that the T-long observed sequence is

$$(Y_{k_1} , Y_{k_2} , \ldots\ldots , Y_{k_T})$$

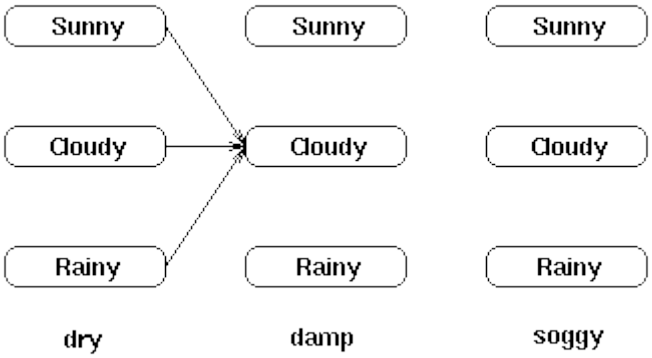
2a. Partial probabilities, (α's)

Consider the trellis below showing the states and first-order transitions for the observation sequence dry,damp,soggy;



We can calculate the probability of reaching an intermediate state in the trellis as the sum of all possible paths to that state.

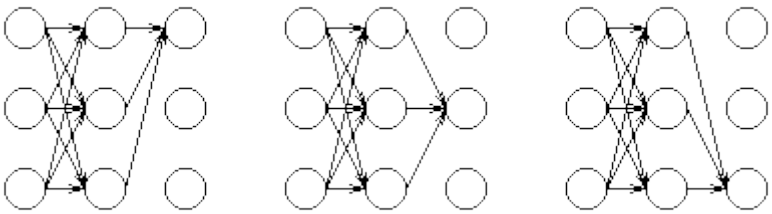
For example, the probability of it being cloudy at t = 2 is calculated from the paths;



We denote the partial probability of state j at time t as αt (j) - this partial probability is calculated as;

αt (j)= Pr(observation | hidden state is j) x Pr(all paths to state j at time t)

The partial probabilities for the final observation hold the probability of reaching those states going through all possible paths - e.g., for the above trellis, the final partial probabilities are calculated from the paths :



It follows that the sum of these final partial probabilities is the sum of all possible paths through the trellis, and hence is the probability of observing the sequence given the HMM.

Section 3 introduces an animated example of the calculation of the probabilities.

2b. Calculating α's at time t = 1

We calculate partial probabilities as :

α t (j)= Pr(observation | hidden state is j) x Pr(all paths to state j at time t)

In the special case where t = 1, there are no paths to the state. The probability of being in a state at t = 1 is therefore the initial probability, i.e. Pr(state | t = 1) = π(state), and we therefore calculate partial probabilities at t = 1 as this probability multiplied by the associated observation probability;

$$\alpha_1(j) = \pi(j).b_{jk_1}$$

Thus the probability of being in state j at intialisation is dependent on that state's probability together with the probability of observing what we see at that time.

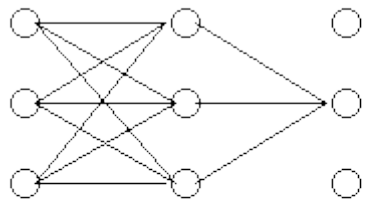
2c. Calculating α's at time, t (> 1)

We recall that a partial probability is calculated as :

αt (j)= Pr(observation | hidden state is j) x Pr(all paths to state j at time t)

We can assume (recursively) that the first term of the product is available, and now consider the term Pr(all paths to state j at time t).

To calculate the probability of getting to a state through all paths, we can calculate the probability of each path to that state and sum them - for example,



The number of paths needed to calculate α increases exponentially as the length of the observation sequence increases but the α's at time t-1 give the probability of reaching that state through all previous paths, and we can therefore define α's at time t in terms of those at time t-1 -i.e.,

$$\alpha_{t+1}(j) = b_{jk_{t+1}} \sum_{i=1}^{n} \alpha_t(i) a_{ij}$$

Thus we calculate the probabilities as the product of the appropriate observation probability (that is, that state j provoked what is actually seen at time t+1) with the sum of probabilities of reaching that state at that time - this latter comes from the transition probabilities together with a from the preceding stage.

Notice that we have an expression to calculate α at time t+1 using only the partial probabilities at time t.

We can now calculate the probability of an observation sequence given a HMM recursively - i.e. we use α's at t=1 to calculate α's at t=2; α's at t=2 to calculate α's at t=3; and so on until t = T. The probability of the sequence given the HMM is then the sum of the partial probabilities at time t = T

2d. Reduction of computational complexity

We can compare the computational complexity of calculating the probability of an observation sequence by exhaustive evaluation and by the recursive forward algorithm.

We have a sequence of T observations, O. We also have a Hidden Markov Model, l=(π,A,B), with n hidden states.

An exhaustive evaluation would involve computing for all possible execution sequences

$$X_i = (X_{i_1}, X_{i_2}, \ldots, X_{i_T})$$

the quantity

$$\sum_X \pi(i_1) b_{i_1 k_1} \prod_{j=2}^{T} \alpha_{i_{j-1} i_j} b_{i_j k_j}$$

which sums the probability of observing what we do - note that the load here is exponential in T. Conversely, using the forward algorithm we can exploit knowledge of the previous time step to compute information about a new one - accordingly, the load will only be linear in T.

3. Summary

Our aim is to find the probability of a sequence of observations given a HMM - (Pr (observations | λ).

We reduce the complexity of calculating this probability by first calculating partial probabilities (α's). These represent the probability of getting to a particular state, s, at time t.

We then see that at time t = 1, the partial probabilities are calculated using the initial probabilities (from the πvector) and Pr(observation | state) (from the confusion matrix); also, the partial probabilities at time t (> 1) can be calculated using the partial probabilities at time t-1.

This definition of the problem is recursive, and the probability of the observation sequence is found by calculating the partial probabilities at time t = 1, 2, ..., T, and adding all α's at t = T.

Notice that computing the probability in this way is far less expensive than calculating the probabilities for all sequences and adding them.

Forward algorithm definition

We use the forward algorithm to calculate the probability of a T long observation sequence;

$$Y^{(k)} = y_{k_1}, \ldots, y_{k_T}$$

where each of the y is one of the observable set. Intermediate probabilities (α's) are calculated recursively by first calculating α for all states at t=1.

$$\alpha_1(j) = \pi(j).b_{jk_1}$$

Then for each time step, t = 2, ..., T, the partial probability α is calculated for each state;

$$\alpha_{t+1}(j) = \sum_{i=1}^{n} \left(\alpha_t(i) a_{ij} \right) b_{jk_t}$$

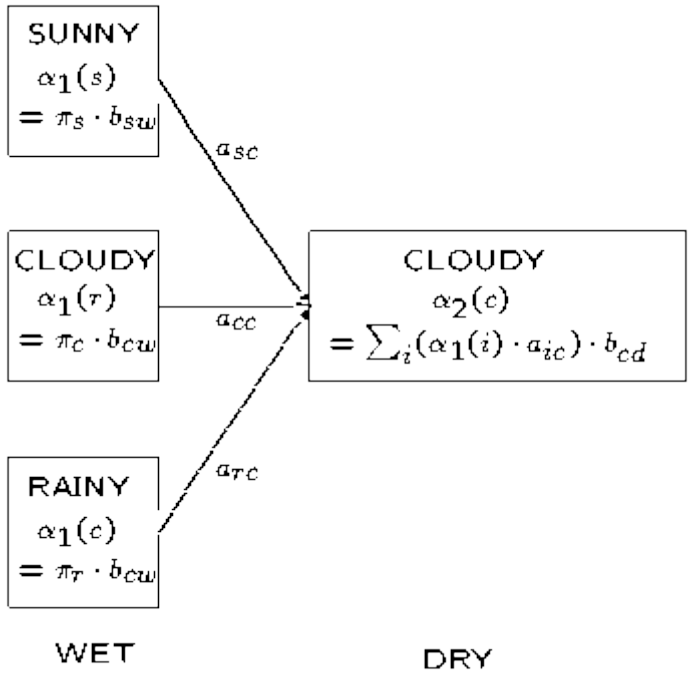
that is, the product of the appropriate observation probability and the sum over all possible routes to that state, exploiting recursion by knowing these values already for the previous time step.

Finally the sum of all partial probabilities gives the probability of the observation, given the HMM, λ.

$$Pr\left(Y^{(k)} \right) = \sum_{j=1}^{n} \alpha_T(j)$$

To recap, each partial probability (at time t > 2) is calculated from all the previous states.

Using the `weather' example, the diagram below shows the calculation for α at t = 2 for the cloudy state. This is the product of the appropriate observation probability b and the sum of the previous partial probabilities multiplied by the transition probabilities α.



Example

Page 3 of this section contains an interactive example of the forward algorithm.

To use the example follow these steps :

1. Enter a number of valid observed states in the input field.
2. Press 'Set' to initialise the matrix.
3. Use either 'Run' or 'Step' to make the calculations.
    - 'Run' will calculate the α's for each and every node and return the probability of the HMM.
    - 'Step' will calculate the α value for the next node only. Its value is displayed in the output window.

When you have finished with the current settings you may press 'Set' to reinitialise with the current settings, or you may enter a new set of observed states, followed by 'Set'.

States may be entered in either or a combination of the following :
Dry, Damp, Soggy
or
Dry Damp Soggy
i.e. valid separators are comma and space. If any invalid state or separator is used then the states remain unchanged from their previous settings

See next page for the example.

You may also run the example in a separate window

Please note
The next page may take a short while to display.
Your browser must be capable of displaying Java applets.

Valid States : **Dry Dryish Damp Soggy**
There is a animation showed in the website(http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html) that cannot be displayed here. The reader can open the website for more information.

A full description of the model used can be found here.

Description of model used in the example

Hidden States (weather)
Sunny
Cloudy
Rainy

Observed States (seaweed)
Dry
Dryish
Damp
Soggy

Initial State Probabilities (ΠΠ Vector)	
Sunny	0.63
Cloudy	0.17
Rainy	0.20

Description of model used in the example

State transition matrix ('A' matrix)

weather yesterday	weather today			
		Sunny	**Cloudy**	**Rainy**
	Sunny	0.500	0.375	0.125
	Cloudy	0.250	0.125	0.625
	Rainy	0.250	0.375	0.375

Confusion matrix ('B' matrix)

hidden states	observed states				
		Dry	**Dryish**	**Damp**	**Soggy**
	Sunny	0.60	0.20	0.15	0.05
	Cloudy	0.25	0.25	0.25	0.25
	Rainy	0.05	0.10	0.35	0.50

Summary

We use the forward algorithm to find the probability of an observed sequence given a HMM. It exploits recursion in the calculations to avoid the necessity for exhaustive calculation of all paths through the execution trellis.

Given this algorithm, it is straightforward to determine which of a number of HMMs best describes a given observation sequence - the forward algorithm is evaluated for each, and that giving the highest probability selected.

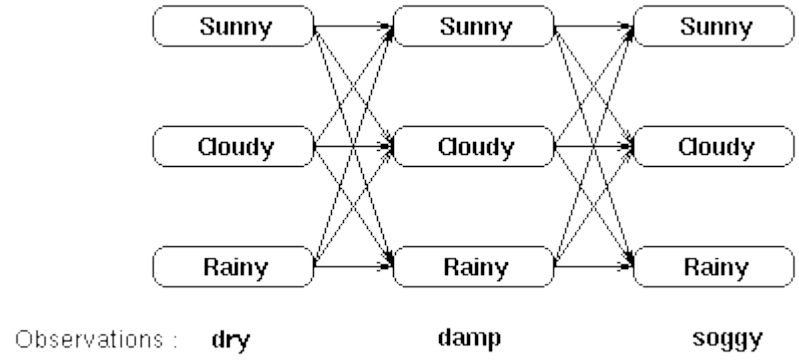
Viterbi Algorithm

Finding most probable sequence of hidden states

We often wish to take a particular HMM, and determine from an observation sequence the most likely sequence of underlying hidden states that might have generated it.

1. Exhaustive search for a solution

We can use a picture of the execution trellis to visualise the relationship between states and observations.



We can find the most probable sequence of hidden states by listing all possible sequences of hidden states and finding the probability of the observed sequence for each of the combinations. The most probable sequence of hidden states is that combination that maximises

Pr(observed sequence | hidden state combination).

For example, for the observation sequence in the trellis shown, the most probable sequence of hidden states is the sequence that maximises :

Pr(dry,damp,soggy | sunny,sunny,sunny), Pr(dry,damp,soggy | sunny,sunny,cloudy), Pr(dry,damp,soggy | sunny,sunny,rainy), ... Pr(dry,damp,soggy | rainy,rainy,rainy)

This approach is viable, but to find the most probable sequence by exhaustively calculating each combination is computationally expensive. As

with the forward algorithm, we can use the time invariance of the probabilities to reduce the complexity of the calculation.

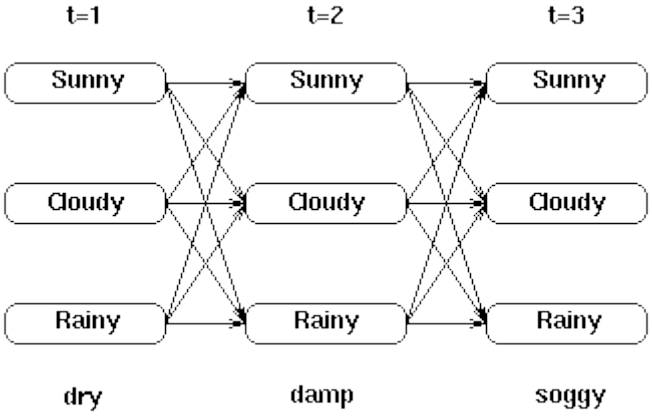
2. Reducing complexity using recursion

We will consider recursively finding the most probable sequence of hidden states given an observation sequence and a HMM. We will first define the partial probability δ, which is the probability of reaching a particular intermediate state in the trellis. We then show how these partial probabilities are calculated at t=1 and at t=n (> 1).

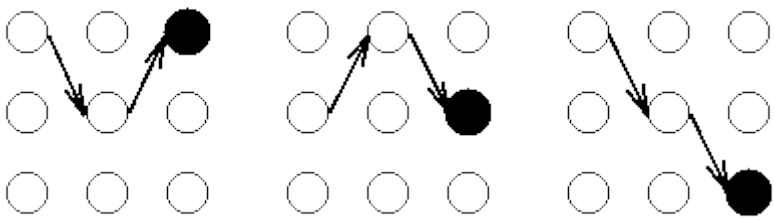
These partial probabilities differ from those calculated in the forward algorithm since they represent the probability of the most probable path to a state at time t, and not a total.

2a. Partial probabilities (δ's) and partial best paths

Consider the trellis below showing the states and first order transitions for the observation sequence dry,damp,soggy;



For each intermediate and terminating state in the trellis there is a most probable path to that state. So, for example, each of the three states at t = 3 will have a most probable path to it, perhaps like this;



We will call these paths partial best paths. Each of these partial best paths has an associated probability, the partial probability or δ. Unlike the partial probabilities in the forward algorithm, δ is the probablity of the one (most probable) path to the state.

Thus δ(i,t) is the maximum probability of all sequences ending at state i at time t, and the partial best path is the sequence which achieves this maximal probability. Such a probability (and partial path) exists for each possible value of i and t.

In particular, each state at time t = T will have a partial probability and a partial best path. We find the overall best path by choosing the state with the maximum partial probability and choosing its partial best path.

2b. Calculating δ 's at time t = 1

We calculate the δ partial probabilities as the most probable route to our current position (given particular knowledge such as observation and probabilities of the previous state). When t = 1 the most probable path to a state does not sensibly exist; however we use the probability of being in that state given t = 1 and the observable state k1 ; i.e.

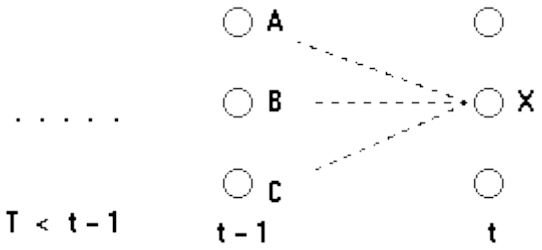
$$\delta_1(i) = \pi(i) b_{ik_1}$$

- as in the forward algorithm, this quantity is compounded by the appropriate observation probability.

2c. Calculating δ 's at time t (> 1)

We now show that the partial probabilities δ at time t can be calculated in terms of the δ's at time t-1.

Consider the trellis below :



We consider calculating the most probable path to the state X at time t; this path to X will have to pass through one of the states A, B or C at time (t-1) Therefore the most probable path to X will be one of

(sequence of states), . . ., A, X

(sequence of states), . . ., B, X

or (sequence of states), . . ., C, X

We want to find the path ending AX, BX or CX which has the maximum probability.

Recall that the Markov assumption says that the probability of a state occurring given a previous state sequence depends only on the previous n states. In particular, with a first order Markov assumption, the probability of X occurring after a sequence depends only on the previous state, i.e.

Pr (most probable path to A) . Pr (X | A) . Pr (observation | X)

Following this, the most probable path ending AX will be the most probable path to A followed by X. Similarly, the probability of this path will be

Pr (most probable path to A) . Pr (X | A) . Pr (observation | X)

So, the probability of the most probable path to X is :

$$Pr(X \ at \ time \ t) =$$
$$\max_{i=A,B,C} Pr(i \ at \ time \ (t-1)) \times$$
$$Pr(X|i) \times Pr(obs. \ at \ time \ t|X)$$

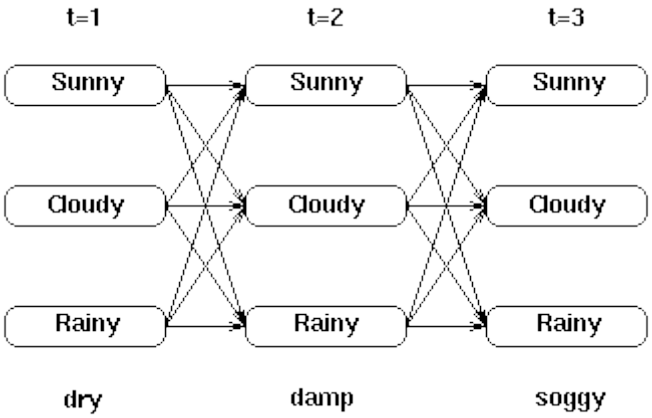
where the first term is given by δ at t-1, the second by the transition probabilities and the third by the observation probabilities.
Generalising the above expression, the probability of the partial best path to a state i at time t when the observation kt is seen, is :

$$\delta_t(i) = \max_j(\delta_{t-1}(j)a_{ji}b_{ik_t})$$

Here, we are assuming knowledge of the previous state, using the transition probabilites and multiplying by the appropriate observation probability. We then select the maximum such.

2d. Back pointers, ϕ's

Consider the trellis



At each intermediate and end state we know the partial probability, δ (i,t). However the aim is to find the most probable sequence of states through the

trellis given an observation sequence - therefore we need some way of remembering the partial best paths through the trellis.
Recall that to calculate the partial probability,δ at time t we only need the δ's for time t-1. Having calculated this partial probability, it is thus possible to record which preceding state was the one to generate δ(i,t) - that is, in what state the system must have been at time t-1 if it is to arrive optimally at state i at time t. This recording (remembering) is done by holding for each state a back pointer ϕ which points to the predecessor that optimally provokes the current state.

Formally, we can write

$$\phi_t(i) = argmax_j(\delta_{t-1}(j)a_{ji})$$

Here, the argmax operator selects the index j which maximises the bracketed expression.

Notice that this expression is calculated from the δ's of the preceding time step and the transition probabilites, and does not include the obervation probability (unlike the calculation of the δ's themselves). This is because we want these ϕ's to answer the question `If I am here, by what route is it most likely I arrived?' - this question relates to the hidden states, and therefore confusing factors due to the observations can be overlooked.

2e. Advantages of the approach

Using the Viterbi algorithm to decode an observation sequence carries two important advantages:

1. There is a reduction in computational complexity by using the recursion - this argument is exactly analogous to that used in justifying the forward algorithm.
2. The Viterbi algorithm has the very useful property of providing the best interpretation given the entire context of the observations. An alternative to it might be, for example, to decide on the execution sequence

$$\mathbf{X}_i = (X_{i_1}, X_{i_2}, \ldots, X_{i_T})$$

where

$$i_1 = argmax_j(\pi(j)b_{jk_1})$$
$$i_t = argmax_j(a_{i_{t-1}k_t}b_{jk_t})$$

Here, decisions are taken about a likely interpretation in a `left-to-right' manner, with an interpretaion being guessed given an interpretation of the preceding stage (with initialisation from the π vector).

This approach, in the event of a noise garble half way through the sequence, will wander away from the correct answer.

Conversely, the Viterbi algorithm will look at the whole sequence before deciding on the most likely final state, and then `backtracking' through the $\phi$ pointers to indicate how it might have arisen. This is very useful in `reading through' isolated noise garbles, which are very common in live data.

3. Section Summary

   The Viterbi algorithm provides a computationally efficient way of analysing observations of HMMs to recapture the most likely underlying state sequence. It exploits recursion to reduce comuputational load, and uses the context of the entire sequence to make judgements, thereby allowing good analysis of noise.

   In use, the algorithm proceeds through an execution trellis calculating a partial probability for each cell, together with a back-pointer indicating how that cell could most probably be reached. On completion, the most likely final state is taken as correct, and the path to it traced back to t=1 via the back pointers.

Viterbi algorithm definition

1. Formal definition of algorithm

The algorithm may be summarised formally as:

For each i,, i = 1, ... , n, let :

$$\mathbf{X}_i = (X_{i_1}, X_{i_2}, \ldots, X_{i_T})$$

- this intialises the probability calculations by taking the product of the intitial hidden state probabilities with the associated observation probabilities.

For t = 2, ..., T, and i = 1, ... , n let :

$$\delta_t(i) = \max_j(\delta_{t-1}(j)a_{ji}b_{ik_t})$$

$$\phi_t(i) = argmax_j(\delta_{t-1}(j)a_{ji})$$

- thus determining the most probable route to the next state, and remembering how to get there. This is done by considering all products of transition probabilities with the maximal probabilities already derived for the preceding step. The largest such is remembered, together with what provoked it.

Let :

$$i_t = argmax(\delta_T(i))$$

- thus determining which state at system completion (t=T) is the most probable.

For t = T - 1, ..., 1

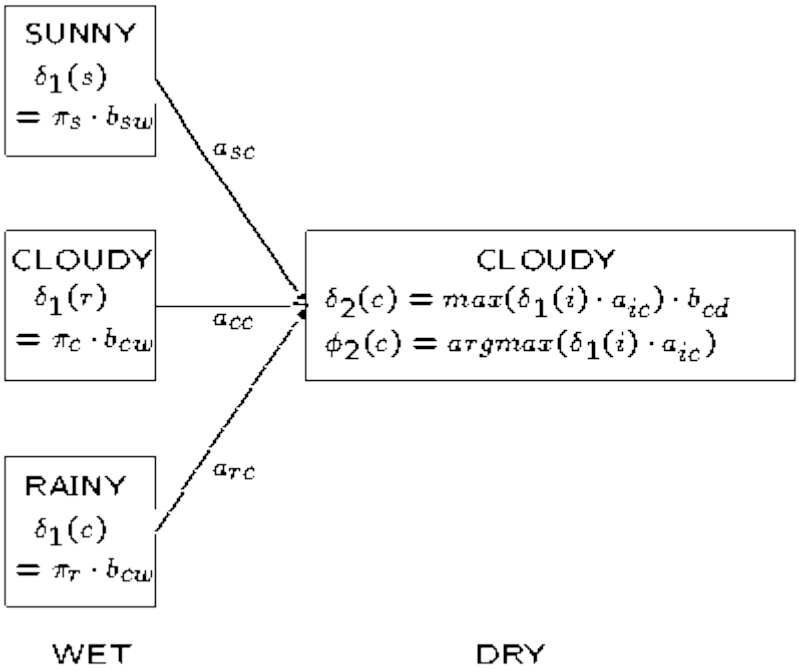
Let :

$$i_t = \phi_{t+1}(i_{t+1})$$

- thus backtracking through the trellis, following the most probable route. On completion, the sequence i_1 ... i_T will hold the most probable sequence of hidden states for the observation sequence in hand.

2. Calculating individual δ's and Φ's

The calculation of δ's is similar to the calculation of partial probability (α's) in the forward algorithm. Compare this diagram showing δ's and Φ's being calculated with the diagram at the end of section 2 under the forward algorithm.



The only difference is that the summation (Σ) in the forward algorithm is replaced with *max* to calculate the δ's - this important difference picks out the *most likely* route to the current position, rather than the total probability. We also, for the Viterbi algorithm remember the best route to the current position by maintaining a `back-pointer', via the *argmax* calculation of the ϕ 's.

Example

Page 3 of this section contains an interactive example of the Viterbi algorithm.

To use the example follow these steps :

1. Enter a number of valid observed states in the input field.
2. Press 'Set' to initialise the matrix.
3. Use either 'Run' or 'Step' to make the calculations.
    - 'Run' will calculate the δ's and ϕ's for each and every node and return the most probable path.
    - 'Step' will calculate the δ and ϕ value for the next node only. Its value is displayed in the output window.

When you have finished with the current settings you may press 'Set' to reinitialise with the current settings, or you may enter a new set of observed states, followed by 'Set'.

States may be entered in either or a combination of the following :
Dry, Damp, Soggy
or
Dry Damp Soggy
i.e. valid separators are comma and space. If any invalid state or separator is used then the states remain unchanged from their previous settings

See next page for the example.

You may also run the example in a separate window

There is a animation showed in the website(http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html) that cannot be displayed here. The reader can open the website for more information.

Please note

The next page may take a short while to display.
Your browser must be capable of displaying Java applets.

Valid States : **Dry Dryish Damp Soggy**

A full description of the model used can be found here.

Description of model used in the example

Hidden States (weather)
Sunny
Cloudy
Rainy

Observed States (seaweed)
Dry
Dryish
Damp
Soggy

Initial State Probabilities (π Vector)	
Sunny	0.63
Cloudy	0.17
Rainy	0.20

Description of model used in the example

State transition matrix ('A' matrix)

weather	weather today

yesterday		Sunny	Cloudy	Rainy
	Sunny	0.500	0.375	0.125
	Cloudy	0.250	0.125	0.625
	Rainy	0.250	0.375	0.375

Confusion matrix ('B' matrix)

hidden states	observed states				
		Dry	Dryish	Damp	Soggy
	Sunny	0.60	0.20	0.15	0.05
	Cloudy	0.25	0.25	0.25	0.25
	Rainy	0.05	0.10	0.35	0.50

Summary

We use the forward algorithm to find the probability of an observed sequence given a HMM. It exploits recursion in the calculations to avoid the necessity for exhaustive calculation of all paths through the execution trellis.

Given this algorithm, it is straightforward to determine which of a number of HMMs best describes a given observation sequence - the forward algorithm is evaluated for each, and that giving the highest probability selected.

Forward-Backward Algorithm

Forward-backward algorithm

The `useful' problems assosciated with HMMs are those of evaluation and decoding - they permit either a measurement of a model's relative applicability, or an estimate of what the underlying model is doing (what `really happened'). It can be seen that they both depend upon foreknowledge of the HMM parameters - the state transition matrix, the observation matrix, and the π vector.

There are, however, many circumstances in practical problems where these are not directly measurable, and have to be estimated - this is the learning problem. The forward-backward algorithm permits this estimate to be made

on the basis of a sequence of observations known to come from a given set, that represents a known hidden set following a Markov model.

An example may be a large speech processing database, where the underlying speech may be modelled by a Markov process based on known phonemes, and the obervations may be modelled as recognisable states (perhaps via some vector quantisation), but there will be no (straightforward) way of deriving empirically the HMM parameters.

The forward-backward algorithm is not unduly hard to comprehend, but is more complex in nature than the forward algorithm and the Viterbi algorithm. For this reason, it will not be presented here in full (any standard reference on HMMs will provide - see the Summary section).

In summary, the algorithm proceeds by making an initial guess of the parameters (which may well be entirely wrong) and then refining it by assessing its worth, and attempting to reduce the errors it provokes when fitted to the given data. In this sense, it is performing a form of gradient descent, looking for a minimum of an error measure.

It derives its name from the fact that, for each state in an execution trellis, it computes the `forward' probability of arriving at that state (given the current model approximation) and the `backward' probability of generating the final state of the model, again given the current approximation. Both of these may be computed advantageously by exploiting recursion, much as we have seen already. Adjustments may be made to the approximated HMM parameters to improve these intermediate probabilities, and these adjustments form the basis of the algorithm iterations.

The readers can refer to the Chinese version for more information including codes and animation.

Summary

Frequently, patterns do not appear in isolation but as part of a series in time - this progression can sometimes be used to assist in their recognition. Assumptions are usually made about the time based process - a common assumption is that the process's state is dependent only on the preceding N states - then we have an order N Markov model. The simplest case is N=1.

Various examples exists where the process states (patterns) are not directly observable, but are indirectly, and probabalistically, observable as another set of patterns - we can then define a hidden Markov model - these models

have proved to be of great value in many current areas of research, notably speech recognition.

Such models of real processes pose three problems that are amenable to immediate attack; these are :

- Evaluation : with what probability does a given model generate a given sequence of observations. The forward algorithm solves this problem efficiently.
- Decoding : what sequence of hidden (underlying) states most probably generated a given sequence of observations. The Viterbi algorithm solves this problem efficiently.
- Learning : what model most probably underlies a given sample of observation sequences - that is, what are the parameters of such a model. This problem may be solved by using the forward-backward algorithm.

HMMs have proved to be of great value in analysing real systems; their usual drawback is the over-simplification associated with the Markov assumption - that a state is dependent only on predecessors, and that this dependence is time independent.

A full exposition on HMMs may be found in:

L R Rabiner and B H Juang, `An introduction to HMMs', iEEE ASSP Magazine, 3, 4-16.