Analysing and Predicting Patient Arrival Times in Hospitals using Hidden Markov Models

Tiberiu Chis and Peter G. Harrison

Department of Computing, Imperial College London,
Huxley Building, 180 Queens Gate, London SW7 2RH, UK
{tc207,pgh}@doc.ic.ac.uk

Abstract. We fit a Hidden Markov Model (HMM) to patient arrivals data, input as a discrete data trace, collected over many months. The processing of the data trace makes use of a simple binning technique, followed by clustering, before it is input into the Baum-Welch algorithm, which estimates the parameters of the underlying Markov chain's state-transition matrix. Upon convergence, the HMM is used to predict its own synthetic traces of patient arrivals, behaving as a fluid input model. Utilizing the Viterbi algorithm, one can decode the meaning of the hidden states of the HMM, further understanding the varying rate of patient arrivals at different times of the hospital schedule. The HMM is validated by comparing means, standard deviations and autocorrelation functions of raw and synthetic traces. Finally, an efficient set up is explored to provide optimal parameter initialization for the HMM, including choosing the number of hidden states. We conclude with a summary of our findings, comparing results with other work in the field, and proposals for future work.

1 Introduction

Hidden Markov models (HMMs) have been used in various fields, ranging from Bioinformatics to Storage Workloads [5]. HMMs were first used in the late 1960s in statistical papers by Leonard E. Baum for statistical inference of Markov chains [1] and also for statistical estimation of Markov process probability functions [2]. Speech recognition became a field for training HMMs in the 1970s and 1980s [10], with many such speech models still used today [4]. In the late 1980s, HMMs acted as tools to analyse biological sequences and one result was the prediction of protein coding regions in genome sequences [12]. Following this, another use of HMMs has been to model common groups of protein sequences, an important topic in computational biology.

HMMs have therefore been an asset in locating genes given an uncharacterized DNA sequence. The GENSCAN HMM ([12]) has been used for Eukaryotic gene finding and we sketch the model in the following diagram:

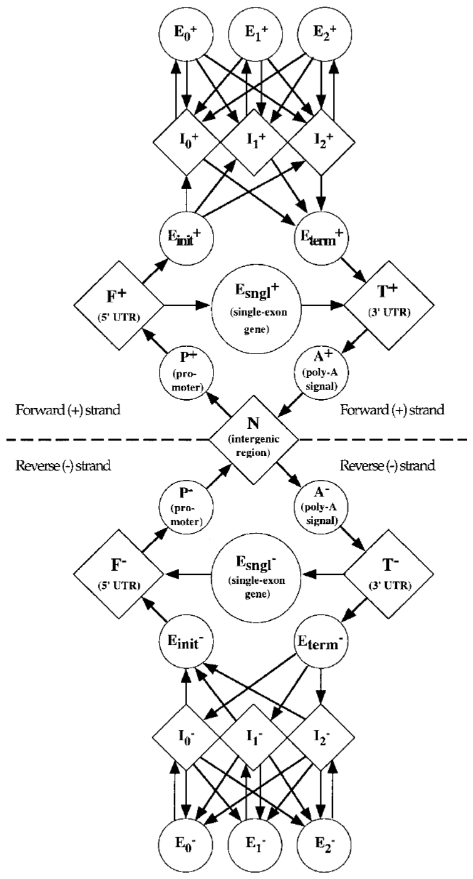

Fig. 1. The general model of the structure of genomic sequences [12].

The GENSCAN HMM models the length distribution and sequence composition for each sequence type. The most probable path through the model (for the particular sequence) is found using the Viterbi algorithm. Note the path returned by Viterbi will contain the coordinates of the predicted genes. The accuracy of the GENSCAN HMM has been tested using metrics such as SENSITIVITY and SPECIFICITY of the data examples used. These accuracy metrics are given below:

$$\text{SENSITIVITY} = \frac{TP}{TP+FN} \qquad \text{SPECIFICITY} = \frac{TN}{TN+FN}$$

where TP = True Positives, TN = True Negatives and FN = False Negatives.

These are just some possible HMM applications in Biology. Extensions include the classification of proteins given an amino-acid sequence, modelling multiple sequences with HMM pairs, etc. In this paper, however, we focus on modelling patient arrivals at a hospital. In training a HMM on patient workloads using the standard Baum-Welch algorithm, to produce synthetic traces of their arrivals, one can decode the hidden states with the Viterbi algorithm and provide meaning to the trends in the data. The HMM is simulated over 1000 runs to produce means and standard deviations (with 95% confidence intervals) for raw and synthetic traces. Autocorrelation functions are computed, based on original and lagged versions of the patient arrival traces, which act as another validation metric for our model. However, before considering these HMM-generated simulations, the HMM is defined more precisely and three associated problems underlying the estimation of its parameters and its interpretation are solved.

1.1 What is a hidden Markov model?

A hidden Markov model (HMM) is a probabilistic model (a bivariate Markov chain) which encodes information about the evolution of a time series. The HMM consists of a hidden Markov chain $\{C_t\}$ (where t is an integer) with states not directly observable and a discrete time stochastic process $\{O_t\}_{t\geq 0}$, which is observable. Combining the two, we get the bivariate Markov chain $\{(C_t, O_t)\}_{t\geq 0}$ where all the statistical inference is done on $\{O_t\}$, as $\{C_t\}$ is not observed. Worth noting is that C_t governs the distribution of the corresponding O_t, and thus we assume that C_t is the only variable of the Markov chain that affects the probability distribution of O_t. An illustration of the Markov chain and the interaction of hidden states with the possible observations is shown in Fig. 2, which shows a directed acyclic graph (DAG) specifying *conditional independence* relations for a HMM. The Markov chain (with its hidden states C_i) has each node conditionally independent from its non-descendants given its parents. For example, given $C_1, C_2, \ldots, C_{t-1}, C_t$, we have that C_{t+1} is independent of $C_1, C_2, \ldots, C_{t-1}$ (which is the first Markov property). The observations O_i are linked to the Markov chain $\{C_t\}$ through *probability emissions* (i.e. C_i produces O_i at time i with a specified probability), where only the O_i are revealed in the HMM.

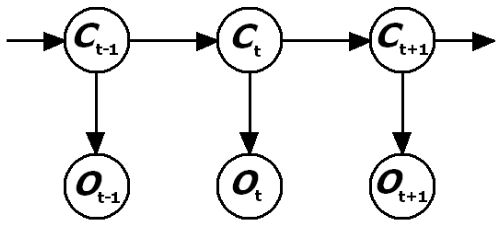
$$\begin{tikzcd}
{} \arrow[r] & C_{t-1} \arrow[r] \arrow[d] & C_t \arrow[r] \arrow[d] & C_{t+1} \arrow[r] \arrow[d] & {} \\
 & O_{t-1} & O_t & O_{t+1} &
\end{tikzcd}$$

Fig. 2. A directed acyclic graph (DAG) showing conditional independence relations for a HMM.[13]

Generally, when constructing a HMM, there are three main problems that need to be solved. First, given the model λ, find the probability of a particular sequence of observations O, which can be solved by the *Forward-Backward algorithm*. Second, given a sequence of observations O, find the most optimal set of model parameters A, B, π[1]. This may be solved by statistical inference through the *Baum-Welch Algorithm* [3], which uses the Forward-Backward algorithm iteratively. Third, find the path of hidden states most likely to generate a sequence of observations O. This is solved using a posteriori statistical inference in the *Viterbi Algorithm* [7]. These three problems are solved using their respective statistical algorithms, as explained in the next section.

2 Solution of the three main problems

In this section, solutions are provided to the main problems along with an implementation solution to underflow. Similar format of the solutions for the Forward-Backward and Baum-Welch algorithms has been used previously in prior work [6]. The Viterbi solution is inspired from work in [13].

2.1 Forward-Backward algorithm

The Forward-Backward algorithm, aims to find $P(O;\lambda)$, the probability of the given sequence of observations $O = (O_1, O_2, \dots, O_T)$ given the model $\lambda = (A, B, \pi)$, where T is the number of observations, A is the state transition matrix, B is the observation matrix and π is the initial state distribution. This is equivalent to determining the likelihood of the observed sequence O occuring. We use Rabiner's solution [9] to aid our calculations, focusing initially on the α-pass, which is the "forward" part of the Forward-Backward algorithm. Then, we solve the corresponding β-pass, aka. the "backward" part of the algorithm.

We define $\alpha_t(i)$ as the probability of obtaining the observation sequence up to time t together with the state q_i at time t, given our model λ. Using N as the number of states and T as the number of observations, the mathematical notation is

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, s_t = q_i; \lambda) \tag{1}$$

where $i = 1, 2, \dots, N$, $t = 1, 2, \dots, T$, and s_t is the state at time t.

Proceeding inductively, we write the solution for $\alpha_t(i)$ as follows:

1. For $i = 1, 2, \dots, N$,

$$\alpha_1(i) = \pi_i b_i(O_1).$$

2. For $i = 1, 2, \dots, N$ and $t = 1, 2, \dots, T-1$,

[1] A is the state transition matrix, B is the observation emissions matrix, and π is the initial distribution matrix.

$$\alpha_{t+1}(i) = [\sum_{j=1}^{N} \alpha_t(j)a_{ji}]b_i(O_{t+1})$$

where $\alpha_t(j)a_{ji}$ is the probability of the joint event that we observe $O_1, O_2, \ldots O_t$ and move from state q_j at time t to state q_i at time $t+1$.

3. It follows that,

$$P(O;\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

where $\alpha_T(i) = P(O_1, O_2, \ldots, O_T, s_T = q_i; \lambda)$

The backward variable, $\beta_t(i)$, is defined as the probability of obtaining the observation sequence from time $t+1$ to T, given state q_i at time t and the model λ. So we have,

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T; s_t = q_i, \lambda) \tag{2}$$

and the solution of $\beta_t(i)$ is given by

1. For $i = 1, 2, \ldots, N$,

$$\beta_T(i) = 1$$

2. For $i = 1, 2, \ldots, N$ and $t = T-1, T-2, \ldots, 1$,

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}b_j(O_{t+1})\beta_{t+1}(j)$$

where we note that O_{t+1} can be observed from any state q_j.

2.2 Baum-Welch algorithm

The Baum-Welch algorithm attempts to maximise $P(O;\lambda)$ by adjusting the parameters A, B, π given the model $\lambda = (A, B, \pi)$ and the observation sequence $O = (O_1, O_2, \ldots, O_T)$. This is done as an iterative process. We first define the probability of making a transition from state q_i at time t to state q_j at time $t+1$, given O and λ, as

$$\xi_t(i,j) = P(s_t = q_i, s_{t+1} = q_j; O, \lambda) \tag{3}$$

Computing $\xi_t(i,j)$ can be described as a three-step process. Firstly, the observations $O_1, O_2, \ldots, O_t$ finishing in state q_i at time t will be covered by $\alpha_t(i)$. Secondly, the transition from q_i to q_j, where O_{t+1} was observed at time $t+1$, is represented by the term $a_{ij}b_j(O_{t+1})$. Thirdly, the remaining observations $O_{t+2}, O_{t+3} \ldots O_T$ beginning in state q_j at time $t+1$ are covered by β_{t+1}. Putting those together, and dividing by a normalizing term $(P(O;\lambda))$ we have

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O;\lambda)} \tag{4}$$

We now sum the terms in (4) over j and notice that this gives the probability of being in state q_i at time t, given the observation sequence O and model λ. This probability is defined as

$$\gamma_t(i) = P(s_t = q_i; O, \lambda) = \sum_{j=1}^{N} \xi_t(i,j)$$

Summing $\gamma_t(i)$ over time t up to T, we get the number of times we expect to visit state q_i. Similarly, summing up to $T-1$ gives the expected number of transitions made from q_i. Thus:

$$\sum_{t=1}^{T} \gamma_t(i) = \text{Expected times state } q_i \text{ is visited.}$$

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Expected transitions from } q_i.$$

Similarly, we sum $\xi_t(i,j)$ over t as follows:

$$\sum_{t=1}^{T} \xi_t(i,j) = \text{Expected visits of } q_i \text{ then } q_j.$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{Expected transitions } q_i \text{ to } q_j.$$

Using these terms, the re-estimation formulas for our HMM parameters are:

$$\pi_i' = \gamma_1(i)$$

$$a_{ij}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_t(i,j)}$$

$$b_j(k)' = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Using these re-estimation formulas, we can update our model $\lambda' = (A', B', \pi')$, where $A' = \{a_{ij}'\}$, $B' = \{b_j(k)'\}$ and $\pi' = \{\pi_i'\}$. Our model will have fixed parameters once $P(O;\lambda') > P(O;\lambda)$, which means the iterative process to find the optimal model λ' ends.

2.3 Viterbi algorithm

The third problem assumes there is a sequence of observations $O = (O_1, O_2, \ldots, O_T)$ with model $\lambda = (A, B, \pi)$. The aim is to find an optimal state sequence for the underlying Markov chain and thus, reveal the hidden part of the HMM λ. The criterion is to find the best sequence of states (i.e. $S = (S_1, S_2, \ldots, S_T)$) such that

$$S^* = argmax_S P(S; O, \lambda)$$

As $P(O;\lambda)$ is independent of S, we have

$$= argmax_S P(S; O, \lambda) P(O; \lambda)$$

$$= argmax_S P(S, O; \lambda)$$

The Viterbi algorithm returns an optimal state sequence S^*. At each time step t, the Viterbi algorithm allows S^* to retain all optimal paths that finish at the N states. At $t+1$, the N optimal paths will be updated and S^* continues to grow in this manner.

Let $S_t^*(i)$ be the optimal path ending in state S_i for the observations $O_1, O_2, \ldots, O_t$. Then, we define $\delta_t(i) = P(O_1, O_2, \ldots, O_t, S_t^*(i); \lambda)$, which is the probability of generating observations $O_1, O_2, \ldots, O_t$ from path $S_t^*(i)$. The variable $\psi_t(i)$ keeps track of each t and i that has maximized the last $\delta_t(i)$. The complete steps of the Viterbi algorithm are as follows:

1. Initialise the following variables:

$$\delta_1(i) = \pi_i b_i(O_1) \text{ for } i = 1, 2, \ldots N$$

$$\psi_1(i) = 0$$

2. Recurse for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$ on the variables:

$$\delta_t(j) = \max_{1 \leq i \leq N}[\delta_{t-1}(i) a_{ij}] b_j(O_t)$$

$$\psi_t(j) = argmax_{1 \leq i \leq N}[\delta_{t-1}(i) a_{ij}]$$

3. Termination:

$$P^* = \max_{1 \leq i \leq N}[\delta_T(i)]$$

$$S_T = argmax_{1 \leq i \leq N}[\delta_T(i)]$$

4. Backtrack through the state sequence for $t = T-1, T-2, \ldots, 1$ as such:

$$S_t^* = \psi_{t+1}(i_{t+1}^*)$$

2.4 Underflow

The solutions to the three main problems may be implemented using any double point precision language (e.g. C) and produce convergent results for a small sequence of observations. However, with increasing sequences, the Baum-Welch algorithm can succumb to *underflow*. As the size of the sequences grow with more observations, the probability values in the algorithm get ever smaller and, after enough iterations, become very close to zero. We discuss a solution to this problem for the Baum-Welch algorithm in terms of normalizing terms. The Viterbi algorithm avoids underflow using simple logarithmic techniques.

2.5 Normalized Baum-Welch algorithm

We now show how to normalize the αs and the βs of the Forward-Backward algorithm to solve this issue of underflow. The normalizing procedure used in this section is adapted from [14]. Firstly, we normalize $\hat{\alpha}_t(j)$ so that all the terms (from 1 to N) sum to 1: $\sum_{i=1}^N \hat{\alpha}_t(i) = 1$. Therefore, we have

$$\hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{i=1}^N \alpha_t(i)}$$

$$= \frac{P(O_1,O_2,\ldots,O_t,s_t=q_i|\lambda)}{P(O_1,O_2,\ldots,O_t|\lambda)}$$

$$= \frac{P(O_1,O_2,\ldots,O_t,s_t=q_i,\lambda)/P(\lambda)}{P(O_1,O_2,\ldots,O_t,\lambda)/P(\lambda)}$$

Cancelling out the $P(\lambda)$ in the fraction gives us

$$= \frac{P(O_1,O_2,\ldots,O_t,s_t=q_i,\lambda)}{P(O_1,O_2,\ldots,O_t,\lambda)}$$

$$= P(s_t = q_i \mid O_1, O_2, \ldots, O_t, \lambda)$$

Hence, the solution of $\hat{\alpha}_t(i)$ is as follows:

1. To initiate the forward probabilities, for $i = 1, 2, \ldots, N$, we have

$$\hat{\alpha}_1(i) = \frac{\pi_i b_i(O_1)}{\sum_{j=1}^{N} \pi_j b_j(O_1)}.$$

2. Then, for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T-1$ we have

$$\hat{\alpha}_{t+1}(i) = \frac{[\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{ji}] b_i(O_{t+1})}{\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})}$$

For the $\hat{\beta}_t(i)$s, we use the same normalizers as those for the αs. However, unlike the αs, the βs do not sum to 1 at any time t. Thus, the solution of $\hat{\beta}_t(i)$ is given by:

1. Initially, for $i = 1, 2, \ldots, N$, we have

$$\hat{\beta}_{T-1}(i) = \beta_{T-1}(i) = 1.$$

2. Then, for $i = 1, 2, \ldots, N$ and $t = T-1, T-2, \ldots, 1$ we have

$$\hat{\beta}_t(i) = \frac{\sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})}.$$

Notice $\hat{\alpha}_t(i)\hat{\beta}_t(i)$ can be written as:

$$\hat{\alpha}_t(i)\hat{\beta}_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_{T-1}(i)} = \frac{\alpha_t(i)\beta_t(i)}{P(O;\lambda)}$$

Therefore, the $\gamma_t(i)$s have the same formula as one can divide by the normalizers without changing the fraction:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{j=1}^{N} \hat{\alpha}_t(j)\hat{\beta}_t(j)}$$

where throughout we have $t = 1, 2, \ldots, T$

However, the ξs are computed differently and are given by the following formula:

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{[\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})][\sum_{j=0}^{N-1} \hat{\alpha}_t(j)\hat{\beta}_t(j)]}$$

By definition of $\gamma_t(i)$, this gives us

$$= \frac{\gamma_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{[\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})] \hat{\beta}_t(i)}$$

The re-estimation formulas for our HMM parameters (π, A, B) now use the new γs and ξs, but otherwise remain the same.

2.6 Logarithmic Viterbi algorithm

With the Viterbi algorithm, underflow can be avoided simply by taking logarithms. As a consequence, we sum the logarithms of the terms instead of multiplying them using products. Note, that this could be used for the Baum-Welch algorithm. We could also take logarithms of the α and β values, but computing the γ values would require dealing with a sum of $\alpha_t(i)$, not conducive to logarithms.

Therefore, taking logarithms of our terms, the enhanced Viterbi algorithm becomes:

1. Initialise the following variables:

$$\delta_1(i) = log(\pi_i b_i(O_1)) \text{ for } i = 1, 2, \ldots N$$

$$\psi_1(i) = 0$$

2. Recurse for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T-1$ on the variables:

$$\delta_t(j) = \max_{1 \le i \le N}[\delta_{t-1}(i) + log(a_{ij})] + log(b_j(O_t))$$

$$\psi_t(j) = argmax_{1 \le i \le N}[\delta_{t-1}(i) + log(a_{ij})]$$

3. Termination:

$$P^* = \max_{1 \le i \le N}[\delta_T(i)]$$

$$S_T = argmax_{1 \le i \le N}[\delta_T(i)]$$

4. Backtrack through the state sequence for $t = T-1, T-2, \ldots, 1$ as such:

$$S_t^* = \psi_{t+1}(i_{t+1}^*)$$

3 Collecting the Hospital Arrival Trace

The data describing patient arrival times were collected from an internal PostgreSQL database (namely *aesop_artery*) from the Department of Computing at Imperial College London, the source data originating from the Emergency Department of a major North London hospital. Within this database, we accessed the *arrivals* table to extract the arrival times for a period of four weeks using a simple SQL query. The resulting “Hospital trace” was output into a csv file, which was read into a Java class. With both traces collected and stored in arrays, the next stage of the transformation process is assigning “bins” to these traces.

This hospital trace was binned by assigning the number of patients arriving every hour. After analysing the frequency of patient arrivals over four weeks, almost one third of cases witnessed no patients. On the other hand, two patients arrived in the same hour about 17% of the time. In fact, on very few occasions were there more than eight patients in one hour. Thus, choosing the one hour bin sizes resulted in an ideal range of values for forming clusters around our data points. The Hospital trace becomes a vector with arrivals and is input into a K-means clustering algorithm to obtain our observation traces.

The Hospital trace was limited to 5 or less clusters because after clustering with $K = 6$, there were two empty clusters present (i.e. with centroids 0.0). As we input the value of K manually, we decided to use a value of 3 clusters, as it gave closer means to the raw data when compared to HMM-generated data. The three clusters are listed here and are essentially our observation values:

$$\begin{pmatrix} 2.4 \\ 5.09 \\ 0.4 \end{pmatrix} \tag{5}$$

In (5), observation values (from top to bottom) represent: moderately frequent arrivals, frequent arrivals and very few arrivals. Having performed the K-means clustering on the Hospital trace, we obtain the observation trace ideal for input into the Baum-Welch algorithm.

4 Training the Baum-Welch algorithm

For the training, we observe patient arrivals for 5000 hours and therefore can input an observation trace of 5000 data points into the Baum-Welch algorithm. Note that we use the sequence of observation values as defined in 5 to populate this observation trace. Once the parameters (A, B, π) converge, the simulation continues with the Baum-Welch algorithm producing 1000 different synthetic observation traces. Means and standard deviations (with confidence intervals) are obtained from the 1000 traces and presented in tables. For now, we introduce our model and initialize it with meaningful parameters.

4.1 Initialization

The initialization of the model includes two hidden states with parameters set up as follows:

1. For the initial hidden state distribution, we have an equiprobable distribution:

$$\pi_0 = (0.5, 0.5)$$

2. For the transition probabilities, we shall assume the following distribution based on the patient arrival times seen in the raw trace. Most of the time, the transition will not move to a different state. A suitable initial transition probability matrix is therefore:

$$A_0 = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

3. For the initial choice of emission probabilities, we assume again an equiprobable distribution:

$$B_0 = \begin{pmatrix} 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \end{pmatrix}$$

4.2 Simulation Results

Using the initial observation trace of 5000 points as input, the Baum-Welch algorithm converged to produce the following parameters. First, the **initial distribution** converged to:

$$\pi = (1.0, 0.0)$$

The **state transition matrix** for the HMM (approximated to 4 decimal places) converged to:

$$A = \begin{pmatrix} 0.8631 & 0.1369 \\ 0.1143 & 0.8857 \end{pmatrix}$$

The **observation emissions matrix** (to 4 decimal places) was as follows:

$$B = \begin{pmatrix} 0.4896 & 0.3278 & 0.1826 \\ 0.0814 & 0.0 & 0.9186 \end{pmatrix}$$

From the results above, we can observe that initially, there is a certainty we will start in state 1. The probability we stay in this state is 0.8631 and therefore the probability that we move back to the other state is 0.1369 (as the rows in the transition probability matrix must sum up to 1). Once in state 2, the probability of going to state 1 is 0.1143 and the probability that we stay in state 2 is 0.8857. Overall, matrix A shows us that the model will most likely stay in the current state for some time. This is confirmed by the Viterbi-generated sequence of hidden states (as explained later).

The first row in the observation emissions matrix (B) reveals various levels of patient arrivals which are all likely to occur. It seems that this state (state 1) is more active in general. On average, there is approximately a 82% chance of seeing at least 2.4 patients per hour in this state. Therefore, we label state 2 as the *dense* state. On the other hand, state 1 observation 3 is highly likely to occur, and observation 2 never occurs. Therefore, from this state we expect to observe few patient arrivals in the one hour interval. In fact, one may label this state the *sparse* state.

Once we have obtained the initial distribution, state transition matrix and observation emissions matrix, the HMM will generate its own sequence of 5000 observations using these parameters. Then, means and standard deviations of the raw and HMM-generated traces are compared to validate our model. These results are presented in Table 1.

Trace	Mean	Std Dev
Raw	1.6294	1.6828
HMM	1.6293 ± 0.0029	1.6815 ± 0.0014

Table 1. Arrivals/bin statistics on the raw and HMM-generated Hospital traces after 1000 simulations

Analysing the arrivals/bin (i.e. the expected arrivals per hour) after 1000 simulations produces excellent results. Table 1 reveals that the bin-means match almost perfectly, and more pleasingly, the standard deviations are identical to two decimal places. The 95% confidence intervals are small, given the 1000 population size (i.e. number of simulations). These statistics suggest that the HMM faithfully reproduces meaningful representations of patient arrival times at the individual bin level.

5 Viterbi-generated Sequence of Hidden States

Another validation technique for the HMM is to train the Viterbi algorithm on observation sequences (i.e. sequences with values 1-3) to generate its corresponding hidden state sequence. Equivalently, it reveals which state produced which observation and more generally provide some explanation to the hidden states.

Analysing an observation sequence of 4800 states, produced by Viterbi after training on a sequence of 4800 observations, the initial state is the sparse state (state 1). After several observations in this state, we switch to state 2, the dense state. We continue to oscillate between these two states until the end of the sequence, almost in symmetric fashion. An explanation to this oscillating pattern can be given as follows: state 1 represents *day* and state 2 represents *night*. To test this claim, the Viterbi sequence of 4800 states was analysed in greater detail by counting the number of times each state occurred. The following results were obtained:

State	Frequency
Day	2533
Night	2267

Table 2. Distribution of Days and Nights from Viterbi state sequence.

From Table 2, the Viterbi algorithm has generated 52.77% day states and 47.23% night states, approximately fifty-fifty. This is expected because a single 24 hour day may divide into two main periods: 12 hours of day (e.g. 7am to 7pm) and 12 hours of night (e.g. 7pm to 7am). Therefore, the labelling of day and night as states has given meaning to the distribution of hidden states, and this theory is supported by the results in Table 2. We now move on to our last form of HMM validation, the autocorrelation function, which reflects the dynamics of the arrivals sequence rather than just static, per-bin statistics.

6 Autocorrelation

This section shows the results of comparing the autocorrelation functions (ACFs) of the raw, unclustered traces and then on the HMM-generated traces. The graphs below show the autocorrelation of patient arrival times at increasing lags for each trace. Fig. 3 and Fig. 4 match well as both show little autocorrelation. The HMM-generated ACF shows less variation than the raw ACF, due to the choice of clusters perhaps. In the next section, we provide some insight into the process of choosing the optimal hidden states for a HMM.

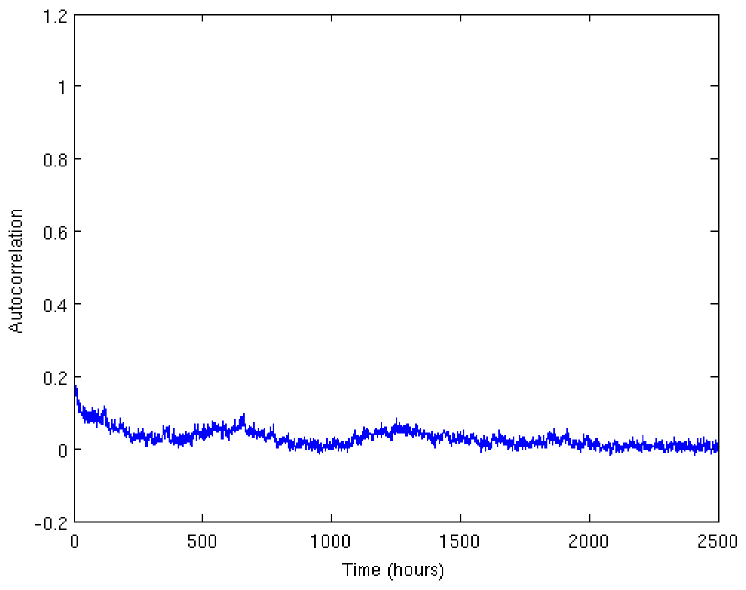

Fig. 3. ACF for raw patient arrivals.

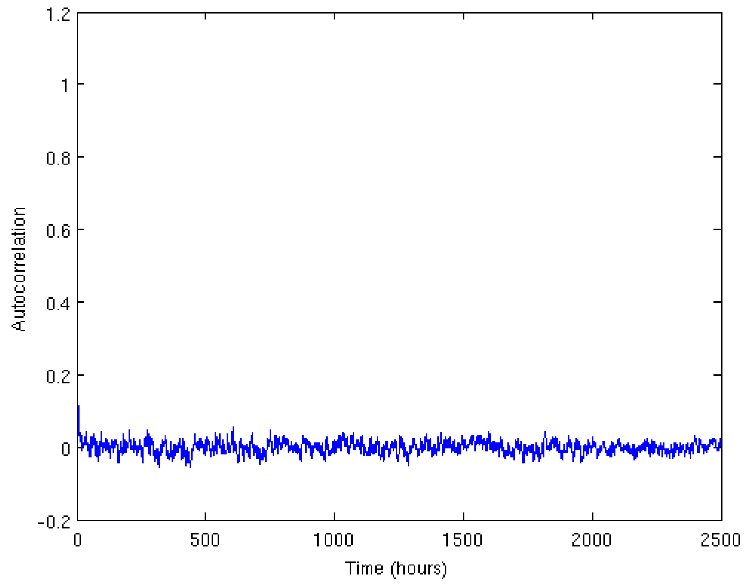

Fig. 4. ACF for HMM-generated patient arrivals.

7 Optimal Number of Hidden States

As discussed in [15] there are two methods for finding the optimal number of hidden states. The first is called *top-down* and is a binary split scheme, where initially we have few states and iteratively split each state into two new states, etc. We carry on in this fashion until no further improvement can be made. The second method is called *bottom-up* and can be seen in [11]. In this scheme, we initialize the number of states to a large value and during training, we merge states together. Eventually, we will end up with a small, optimal (with respect to this method) number of states.

We can choose between these two methods of finding the best number of states with which to initialize the Baum-Welch algorithm. There are ways of checking whether we have an optimal number of states if a HMM is already set up. One such way is to observe the emission probabilities that are output from the Baum-Welch algorithm. For example, if two rows in the emission matrix are very similar (i.e. almost identical set of entries) then we have (at least) one too many states for our HMM. Below, is an example of our emission matrix that was produced for the hospital patient arrival times which we gave to the Baum-Wlech algorithm. We used 3 d.p. for the entries of the matrix:

$$\begin{pmatrix} 0.003 & 0.422 & 0.575 \\ 0.437 & 0.157 & 0.406 \\ 0.02 & 0.941 & 0.039 \\ 0.0 & 0.996 & 0.004 \end{pmatrix}$$

Notice that the third and fourth rows are too similar to be a coincidence. Either our algorithm has not converged fully or, more likely, we have too many hidden states and thus our HMM had not been optimally set up. It is easy to deduce now that the Hospital HMM must take either two or three hidden states as input. Therefore, in this case, the use of the *top-down* or *bottom-up* methods are not necessary for such simple models.

After we apply our own method of analysing rows in the emission matrix, we can compare our two remaining cases to find which number of hidden states helps the HMM perform better. The comparison lies in the analysis of the entries in the emission matrix, of course. In [15], a graph of directed accuracies is produced for different hidden states, to identify the point of convergence precisely:

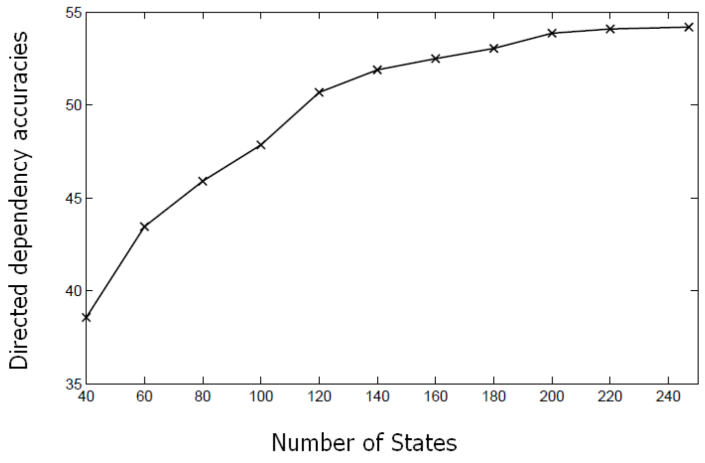

Fig. 5. Directed dependency accuracies given number of states [15].

For this problem the use of top-down or bottom-up techniques seems too complex for the simplicity of our HMM structure. For example, the small number of states and observation traces we used meant we could achieve the optimal set up of our HMM without the need to merge states. In the next section, we explore an attempt on a simple bottom-up method of merging hidden states.

7.1 Merging version of the Baum-Welch algorithm

An optimal way of choosing hidden states in a HMM is the *bottom up* technique. This approach starts off with many states and merges similar states together until the optimal number is achieved. Ideally, an extension to this paper would be to derive a systematic approach for this method which works in any given scenario (e.g. storage workloads, patient arrivals, etc.) and produces the optimal number of hidden states in that scenario. This would involve analysis of the emission probability matrix as a starting point to identify the effect of each state on the observation set. Then, a modification of the Baum-Welch algorithm needs to be executed to merge the chosen states whilst keeping any information about transition probabilities in the process. This would give us an approximation to the transition probability matrix for optimal states. Finally, we can complete the Baum-Welch algorithm by using the new transition matrix to construct the new emission proability matrix and calculate the initial probability distribution.

Although our merging technique, as described above, seems computationally expensive, it is advantageous for achieving an HMM with optimal states when contextual information of the time series is not available. The power of this technique lies in the identification of the "most efficient set up", just by using the original HMM parameters of a less efficient set up. Therefore, this technique is entirely self-contained in the sense that the computations are limited within the bounds of the Baum-Welch algorithm.

8 Conclusion and Extensions

The Hospital arrivals model was found to train successfully on patient arrivals, collected over months of analysis. The means and standard deviations matched well for raw and HMM-generated traces and both traces exhibited little autocorrelation. HMM parameters, fully converged after training, were used to predict the model's own synthetic traces of patient arrivals, therefore behaving as a fluid input model (with it's own rates). An enhancement could be to assume instead that the arrival process is Poisson, with corresponding rates, and produce a cumulative distribution function for the patient arrivals workload.

For our model, there was significant focus on the Viterbi algorithm. Given the correct model parameters, as validated by means and standard deviations of raw and synthetic traces as well as autocorrelation functions, we focused on the hidden trends highlighted by the logarithmic Viterbi algorithm. By simply counting the number of occurences of each state in the hidden state sequence and assigning each state to a category, the meaning of the hidden states became more apparent. The power, and straight-forward implementation, of this process can lead to decoding of time series from which we want answers. Other applications of the Viterbi algorithm seen in industry include speech recognition ([9]), biology([12]), etc. These papers have tried similar decoding techniques using the Viterbi algorithm to deduce the physical significance of traces or parts of traces. We aim to use Viterbi on different periods of the patient time series and understand how the hidden state distribution is updated. We also plan to consider more expressive traces that include type information for each arriving patient, so that more than two hidden states would be necessary.

References

1. Baum, L. E., Petrie, T.: Stastical Inference for Probabilistic Functions of Finite Markov Chains, In *The Annals of Mathematical Statistics*, **37**, pp. 1554-63 (1966)
2. Baum, L. E., Eagon, J. A.: An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology, In *Bulletin of the American Mathematical Society*, **73**, pp. 360-3, (1967)
3. Baum, L. E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, In *The Annals of Mathematical Statistics*, **41**, pp. 164-171 (1970)
4. Ashraf, J., Iqbal, N., Khattak, N. S., Zaidi, A. M.: Speaker Independent Urdu Speech Recognition Using HMM (2010)
5. Harrison, P. G., Harrison, S. K., Patel N. M., Zertal, S.: Storage Workload Modelling by Hidden Markov Models: Application to Flash Memory, In: *Performance Evaluation*, **69**, pp. 17-40 (2012)
6. Chis, T., Harrison, P. G.: Incremental HMM with an improved Baum-Welch Algorithm, Proceedings of Imperial College Computing Student Workshop (2012)
7. Viterbi, A. J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, In *IEEE Transactions on Information Theory*, **13**, pp. 260-269 (1967)
8. Krough, A., Brown, M., Mian, S., Sjolander, K., Haussler, D.: Hidden Markov Models in Computational Biology, In: *Journal of Molecular Biology*, pp. 1501-1531, Computer and Information Sciences, University of California, Santa Cruz, USA, (1994)
9. Rabiner, L. R., Juang, B. H.: An Introduction to Hidden Markov Models, In *IEEE ASSP Magazine*, **3**, pp. 4-16 (1986)
10. Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, In *IEEE*, **77**, pp. 257-286 (1989)
11. Brand, M.: An entropic estimator for structure discovery, In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, MIT Press, pp. 723-29, Cambridge, MA, USA (1999)
12. Burge, C., Karlin, S.: Prediction of complete gene structures in human genomic DNA, In *Journal of Molecular Biology*, pp. 78-94, Computer and Information Sciences, Stanford University, California, USA (1997)
13. Chis, T.: Hidden Markov Models: Applications to Flash Memory Data and Hospital Arrival Times, Department of Computing, Imperial College London (2011)
14. Zhai, C. X.: A Brief Note on the Hidden Markov Models (HMMs), Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA, (2003)
15. Zhang, L., Chan, K. P.: Bigram HMM with Context Distribution Clustering for Unsupervised Chinese Part-of-Speech tagging, Department of Computer Science, Hong Kong (2010)