An Investigation into Stochastic Processes for Modelling Human-Generated Data

A 20-cp 3rd year project

Author:
Tim Jones

Supervisor:
Dr. Gordon Ross

Rendered on Wednesday 17th April, 2013

Contents

Chapter 1

Introduction

Modelling human-generated “random” data is notoriously difficult. In this project, we attempted to model the behavior of one user of twitter using various random processes. The user was observed over the course of around 6 months posting (emitting) a little over 3,300 tweets. The goal is to find some kind of model to fit these data, without using a hideously large number of parameters.

Chapter 2

A Zoo of Stochastic Processes

2.1 Markov Chains

2.1.1 A Formal Definition

A stochastic process [1, p590] is a collection $\{X_t | t \in T\}$ (sometimes $X(t)$ for continuous T) of random variables. These collections may be indexed arbitrarily, but tend to be used to describe the evolution of some random series of events, using T to be some representation of either discrete or continuous time, for instance X_t may be the number of observed emissions from a radioactive source after t minutes or the lisence plate of the t^{th} car to go past a speed camera.

A Markov Chain is a particular form of discrete-time stochastic process. We usually set $T = \{0, 1, 2, ...\} = \mathbb{N}$ and we say that $(X_t)_{t \in \mathbb{N}}$ obeys the Markov Property iff

$$\forall t \in T, s \in S \quad \mathbb{P}(X_{t+1} = s | X_0, X_1, ...X_t) = \mathbb{P}(X_{t+1} = s | X_t)[2]$$

If a stochastic process obeys the Markov Property, then we say that it is a Markov Chain (MC). We refer to S as the state-space, each $s \in S$ is a state, and X_t represents the state at time t. We usually write the Markov Property as "given the present, the future is conditionally independent of the past" [2].

There is also a continuous analogue to the Markov Chain - the Continuous Time Markov Process (CTMP). We set $T = \mathbb{R}^+$,and define a similar Markov Property;

$$\forall \delta > 0, s \in S \quad \mathbb{P}(X(t + \delta) = s | \{X(\tau), \tau < t\}) = \mathbb{P}(X(t + \delta) = s | X_t)$$

We'll be dealing exclusively with the discrete-space case in this project, ie a Markov Chain where S is discrete, though we'll need both MCs and CTMPs. It is popular for many authors to set $S \subseteq \mathbb{Z}$, though our states can be integers, real numbers, popes, or any other completely arbitrary non-empty set. Where S is discrete, we define $\delta = (\delta_i)_{i \in S}$, the initial probability vector, and $\Pi = (\pi_{ij})_{(i,j) \in S^2}$, the matrix of transition probabilities, such that.

$$\begin{aligned} \delta_i &= \mathbb{P}(X_0 = i) \\ \pi_{ij} &= \mathbb{P}(X_{t+1} = j | X_t = i) \forall t \in \mathbb{N} \end{aligned}$$

Every discrete-state Markov Chain can be uniquley defined by the triple (S, δ, Π). Naturally, the row sums of Π should be 1, ie

$$\forall i \in S, \sum_{j \in S} \pi_{ij} = 1$$

For CTMPs, we usually define the transition rate matrix Q such that

$$\forall i, j \in S, \delta > 0 \quad \mathbb{P}(X(t + \delta) = j | X(t) = i) = q_{ij}\delta + o(\delta)$$

Where $o(\delta)$ is some function such that $o(\delta) \to 0$ as $\delta \to 0$. This gives us that

$$\forall i,j \in S, \delta > 0 \quad \mathbb{P}(X(t+\delta) = j | X(t) = i) = (e^{\delta Q})_{ij}$$

That is, the i, j^{th} element of the matrix exponential of δQ. With the initial probability vector δ defined as before, we can uniquely define any CTMP with the triple (S, δ, Q). Q's row sums are always 0, with all off-diagonal elements being positive, and all diagonal elements being negative, ie

$$\forall i,j \in S, i \neq j \Rightarrow q_{ij} > 0$$
$$\forall i \in S, q_{ii} = -\sum_{j \neq i} q_{ij}$$

2.1.2 An Intuitive Interpretation

Whilst the above defines Markovian Processes, it fails to describe them in any intuitive way. Before attempting to use them, it is important to be able to deal with both the continuous and discrete-time Markov Chains in an intuitive way. My personal preferred method is to use edge-weighted directed graphs[3]. Each state in S is given a node in the graph, and for each $i, j \in S$ the edge (i, j) is given weight π_{ij} in the discrete case and q_{ij} in the continuous case.

As an example, let's use the following definitions, with δ and Π indexed in the order the elements of S are written;

$$S = \{\text{John Paul II}, \text{Adrian I}\}$$
$$\delta = (1, 0)$$
$$\Pi = \begin{array}{c} \\ JP2 \\ A1 \end{array} \begin{array}{cc} JP2 & A1 \\ \left(\begin{array}{c} 0.3 \\ 0.9 \end{array} \right. & \left. \begin{array}{c} 0.7 \\ 0.1 \end{array} \right) \end{array}$$

This defines a Discrete Time Markov Chain with the following graph:

$$\begin{tikzcd} \text{JP2} \arrow[loop left, "0.3"] \arrow[r, shift left, "0.7"] & \text{A1} \arrow[loop right, "0.1"] \arrow[l, shift left, "0.9"] \end{tikzcd}$$

We can then say that a discrete Markov Chain will hop from node to node at each time step with probabilities defined by the weights of the edges between the current node and its neighbors, and can be simulated by algorithm 1.

```
input  : $(S, \delta, \Pi)$, a Markov Chain, and $T$, a maximum number of steps to simulate
output: s, a vector of states $s_i$, recording the sequence of states visited by the chain
begin
    $s_1 \leftarrow \sigma$   $wp$ $\delta_\sigma$ /* s0 takes on state sigma with probability delta sigma             */
    for $n \leftarrow 2$ to $T$ do
        $s_n \leftarrow \sigma$   $wp$ $\pi_{s_{n-1},\sigma}$ /* sn takes on state sigma with the relevant probability   */
    end
    return s
end
```

Algorithm 1: A Simulation Algorithm for the generic Markov Chain

A CTMP is slightly more complex; on arriving in a state i, we generate a random time $T \sim Exp(-q_{ii})$. We say that the CTMP will remain in state i for T units of time, then jump to state j with probability $\frac{-q_{ij}}{q_{ii}}$. We can simulate a CTMP with algorithm 2.3.

```
input  : $(S, \delta, Q)$, a CTMP, and $T$, a maximum time for which to simulate the process
output: $\mathbf{t}$, a vector of pairs $(t_i, s_i)$, recording the time and destination of the $i^{th}$ transition.
begin
    $t_0 \leftarrow 0$
    $s_0 \leftarrow \sigma \quad wp \ \delta_\sigma$ /* s0 takes on state sigma with probability delta sigma */
    $n \leftarrow 0$
    while $t_n < T$ do
        $n \leftarrow n + 1$
        $\tau \leftarrow Exp(-q_{ii})$ /* tau takes on an exponentially distributed random value */
        $t_n \leftarrow t_{n-1} + \tau$
        $s_n \leftarrow \sigma \quad wp \ \frac{-q_{i\sigma}}{q_{ii}}$ /* sn takes on state sigma with the given probability */
    end
    return $((t_0, s_0), ..., (t_{n-1}, s_{n-1}))$
end
```

Algorithm 2: A Simulation Algorithm for the generic CTMP

2.1.3 The Poisson Process

The simplest form of CTMP is the homogeneous Poisson Process, where $S = \mathbb{N}$ and $\forall i \in S, q_{ii} = -\lambda, q_{i,i+1} = \lambda$. We call λ the rate of this process. If $N(t)$ is a Poisson process, we then have that

$$\forall i \in \mathbb{N}, t, \varepsilon \in \mathbb{R}^+, \mathbb{P}(N(t+\varepsilon) = i+1 | N(t) = i) = \lambda\varepsilon + o(\varepsilon)$$

The graph for this process is similarly simple

Implicit from this, we see that the process can only increase - once leaving a state, we never return, so we can define the emission times t_n, as $t_n = \min\{t | N(t) \geqslant n\}$. We can also define $\tau_n = t_n - t_{n-1}$, the inter-arrival times for the n^{th} jump for $n\{1, 2, ...\}$. We refer to these as emissions and arrivals since a Poisson Process generally models a counting process, whereby we record the times at which we observe things happening, eg the times at which radioactive particles are detected from a radioactive material.

A modification of the generic CTMP simulation algorithm can be made for simulating a Poisson Process. Since it's only possible to jump from state i to state $i+1$, we need not record the destinations of each jump; jump i will always take us to state i. This is detailed in algorithm 3.

```
input  : $\lambda$, a poisson process rate and $T$, a maximum time for which to simulate
output: $\mathbf{t}$, the emission times of a Poisson process of rate $\lambda$ terminating before time $T$
begin
    $t_0 \leftarrow 0$
    $n \leftarrow 0$
    while $t_n < T$ do
        $n \leftarrow n + 1$
        $\tau \leftarrow Exp(\lambda)$ /* tau takes on an exponentially distributed random value */
        $t_n \leftarrow t_{n-1} + \tau$
    end
    return $(t_0, ..., t_{n-1})$
end
```

Algorithm 3: A Simulation Algorithm for the Poisson Process

We can also sacrifice the Markov Property to define an inhomogeneous poisson process. Everything remains from before, except rather than having a rate parameter $\lambda \in \mathbb{R}^+$, we have a rate function $\lambda : \mathbb{R}^+ \to \mathbb{R}^+$, where

$$\forall i \in \mathbb{N}, t, \delta \in \mathbb{R}^+, \mathbb{P}(N(t+\delta) = i+1 | N(t) = i) = \lambda(t)\delta + o(\delta)$$

Since this process is no longer Markovian, it makes little sense to draw its transition rate graph, however it remains a powerful tool. Simulating one of these is, however, less simple. The algorithm for doing so is based on Bernoulli thinning and is equivalent to algorithm 4

```
input  : $\lambda : [0,T] \to [0, \lambda_{max}]$, a desired rate function for the resulting Poisson Process, and $\mathbf{t}$, the
         emission times of a Poisson Process of rate no greater than $\lambda_{max}$, terminating before time $T$,
         indexed from 1 to $n$
output: $\mathbf{t}'$, the emission times an inhomogeneous Poisson Process with rate function $\lambda$
begin
    $j \leftarrow 0$ for $i \leftarrow 1$ to $n$ do
        $r \leftarrow U(0,1)$ /* r takes on a uniformly distributed random value in [0,1]        */
        if $r < \frac{\lambda(t_i)}{\lambda_{max}}$ then
            $t'_j \leftarrow t_i$
            $j \leftarrow j + 1$
        end
    end
    return $\mathbf{t}'$
end
```

Algorithm 4: A Thinning Algorithm for Poisson Processes

So if we can find an upper bound for our rate function, we can combine algorithms 3 and 4 to simulate arbitrary inhomogeneous poisson processes.

2.2 Hidden Markov Models

The Hidden Markov Model is an extention of the Markov Chain used for fitting various stochastic models.

With discrete time and state space, rather than having a directly observable Markov Chain, we assume that the process we observe has an underlying unobservable Markov Chain. We then define an observation space O, and a set of probability distrubtions, $\{p_s | s \in S\}$. Before jumping out of state s, the Markov Chain emits observation o with probability $p_s(o)$. Alternatively, we can define a continuous O with probability densities instead of a probability distribution.

The traditional example is that of the "unfair casino". Imagine the dealer at the casino has two six-sided dice. Die one is fair, but die two is weighted such that it never shows a 6, and shows the numbers 1 to 5 with equal probabilities. The dealer will clandestinely swap the dice with probability 0.1 before rolling. He selects his first die uniformly. We can represent this as follows;

$$
\begin{aligned}
S &= \{1, 2\} \\
\delta &= (1, 0) \\
\Pi &= \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \\
O &= \{1, 2, 3, 4, 5, 6\} \\
x \in O \Rightarrow p_1(x) &= \frac{1}{6} \\
x \in O \setminus \{6\} \Rightarrow p_2(x) &= \frac{1}{5}
\end{aligned}
$$

We can only observe the results of the dice rolls, but what we're interested in is the underlying model. Firstly, we want to know what die the dealer is likely to be using at any given time - if I've seen the dealer roll a 6, I know he was using the fair die then, so he will use the fair die again with probability 0.9, for example. Further to this, suppose we didn't know the above parameters in advance. We may also want to estimate them from observations in some way.

2.3 The Markov-Modulated Poisson Process

The Markov-Modulated Poisson Process (MMPP) is a particularly devious form of Hidden Markov Model. We assume that, underlying an emittor, there is a CTMP. Each state is tied to a fixed Poisson Process rate, so $S = \{\lambda_1, ..., \lambda_n\}$. The instantaneous rate of the poisson process is then defined by the state in which the underlying CTMP resides. We can define such a process by letting $\mathbf{t} = ((t_0, s_0), ..., (t_n, s_n))$ be a trace of the CTMP as generated by algorithm , and $\lambda : [0, T] \to S$ be the step function generated by simulating the underlying CTMP, ie;

$$\lambda(\tau) = \begin{cases} s_i & \text{for} \quad \tau \in [t_i, t_{i+1}), \quad i \in [n-1] \\ s_n & \text{for} \quad \tau \in [t_n, T] \end{cases}$$

We then have that an inhomogeneous poisson process of rate λ is a single trace of this MMPP. In order to generate multiple traces, it is necessary for us to produce a new rate function λ from the underlying CTMP for each trace.

Though this is correct, I personally find that it is simpler to imagine an MMPP as the concatenation of a series of poisson processes, each generated within a particular state of the underlying CTMP, eg the underlying CTMP enters state s_1 at time t_1. It generates emissions as a Poisson Process of rate s_1 for $t_2 - t_1$ time units. It then stops acting like this poisson process, and starts generating emissions as a poisson process of rate s_2 for $t_3 - t_2$ time units.

It's not immediately obvious that this intuition matches the definition, so it's worth arguing that this is the case.

Let $\lambda_1 : [0, T_1) \to \mathbb{R}^+$ and $\lambda_2 : [0, T_2) \to \mathbb{R}^+$ be two rate functions. Let $T = T_1 + T_2$, and define λ, the concatenation $\lambda = \lambda_1 || \lambda_2$, as;

$$\lambda(\tau) = \begin{cases} \lambda_1(\tau) & \text{for} \quad \tau \in [0, T_1) \\ \lambda_2(\tau - T_1) & \text{for} \quad \tau \in [T_1, T) \end{cases}$$

Let $\{N_1(t) | t \in [0, T1)\}$ be a poisson process of rate λ_1, similarly for N_2 and N, and let $\widetilde{N}$ be defined thus;

$$\widetilde{N}(\tau) = \begin{cases} N_1(\tau) & \text{for} \quad \tau \in [0, T_1) \\ N_1(T_1) + N_2(\tau - T_1) & \text{for} \quad \tau \in [T_1, T) \end{cases}$$

$\widetilde{N}$ represents my intuition that an MMPP can be formed by gluing together emissions from shorter fixed rate poisson processes, and N represents the actual definition given above. If we can show that $\widetilde{N}$ and N are ideentically distributed, then we can inductively apply this result to arbitrarily long concatenations, and hence to the generic MMPP.

Let $\tau \in [0, T_1)$ and $\varepsilon > 0$ be such that $\tau + \varepsilon < T_1$. We then have that

$$\begin{aligned} \mathbb{P}(\widetilde{N}(\tau + \varepsilon) - \widetilde{N}(\tau) = 1) &= \mathbb{P}(N_1(\tau + \varepsilon) - N_1(\tau) = 1) \\ &= \lambda_1(\tau)\varepsilon + o(\varepsilon) \\ &= \lambda(\tau)\varepsilon + o(\varepsilon) \end{aligned}$$

So $\widetilde{N}$ is similar to a poisson process of rate λ within the range $[0, T_1)$.

Now let $\tau \in [T_1, T)$ and $\varepsilon > 0$ be such that $\tau + \varepsilon < T$. We have that

$$\begin{aligned} \mathbb{P}(\widetilde{N}(\tau + \varepsilon) - \widetilde{N}(\tau) = 1) &= \mathbb{P}(N_2(\tau - T_1 + \varepsilon) + N_1(T_1) - N_2(\tau - T_1) - N_1(T_1) = 1) \\ &= \mathbb{P}(N_2(\tau - T_1 + \varepsilon) - N_2(\tau - T_1) = 1) \\ &= \mathbb{P}(N_2(\tau - T_1 + \varepsilon) - N_2(\tau - T_1) = 1) \\ &= \lambda_2(\tau - T_1)\varepsilon + o(\varepsilon) \\ &= \lambda(\tau)\varepsilon + o(\varepsilon) \end{aligned}$$

So $\widetilde{N}$ is similar to a poisson process of rate λ within the range $[T_1, T)$. We need not deal with the case where τ and $\tau + \varepsilon$ are either side of T_1, since we can always choose an ε small enough such that this is not the case - these probabilities always assume that ε is close to 0. So we have that $\widetilde{N}$ is distributed identically to N.

So concatenating two poisson processes of rates λ_1 and λ_2 produces a new poisson process of rate $\lambda_1||\lambda_2$, and it follows from simple induction that an MMPP can be thought of as the concatenation of multiple bounded-length homogeneous poisson processes of rates determined by the state of the underlying CTMP, and time bounds determined by the length of time spent in each state of the underlying CTMP.

Chapter 3

Potential Modelling Techniques

Now that we've defined a selection of random processes, we can grab our data and start modelling it. We'll try out a few potential models, simulate some traces from them, and see how well we can re-establish the original model, then go on to discuss their use for this particular problem.

3.1 Fitting a Non-Homogeneous Poisson Process

The simplest place to start is with a non-homogeneous poisson process. We assume that the tweeter emits tweets according to an inhomogeneous poisson process, and attempt to find its rate function. For convenience, we'll describe our time in hours, though naturally the units could be arbitrary. We start by simulating a poisson process of known rate, and seeing how well we can recover the original rate function. We start with a simple step function,

$$\lambda(t) = \begin{cases} 5 & \text{for} \quad 0 \leqslant t < 30 \\ 10 & \text{for} \quad 30 \leqslant t < 50 \\ 5 & \text{for} \quad 50 \leqslant t < 100 \end{cases}$$

And we simulate the process for 100 hours, producing a trace such as;

We can fit a step function by observing multiple traces of these poisson processes, taking the differences in emission times, then attempting to cluster them with the k-means algorithm. The average number of emissions per trace is the integral of the rate wrt time over the time interval for which we observe, ie if N is the number of observed emissions in a single trace,

$$\mathbb{E}(N) = \int^1 00_0 \lambda(t)\, dt = 600$$

So if we run out 6 traces we'll observe roughly $3,600$ emissions, a little more than our real data set. Fitting a function to these gives us the results from

Eyeballing this, we see that it's not a terrible fit in this first attempt, but if we try a more complex rate function, happens

We could instead attempt to fit a polynomial function, but a smooth function is subject to one very large problem with twitter data - humans don't behave in a smooth manner. Our usage of twitter throughout a day does not vary along some continuum - in the simplest sense, we can't be half on and half off twitter at some given moment, but there's a worse problem, visible in

We can see an important feature between and . We'll call this a “burst”. The user is tweeting very rapidly for a short period of time. These bursts occur at random times throughout the day, and last for random lengths of time, but they all take on the same form. This heavily restricts our function. We can't simply say that at some particular time there is always a burst, but we also cannot deny their existence by smoothing them out since these bursts, by their very nature, will account for the majority of the observed tweets.

Clearly, a different approach is needed.

3.2 Fitting a Markov-Modulated Poisson Process

An MMPP seems ideal then, capturing the simplicity of a step function, and letting us model the idea of randomly distributed bursts throughout a day. Indeed, several authors have postulated that such a model would be ideal for simulating human behavior but there have so far been no actual quantifiable studies of its relevance. Part of the reason for this is likely the lack of algorithms.

Fitting a Hiddin Markov Model of any kind relies on two main algorithms, Baum-Welch and Viterbi . The Baum-Welch Algorithm is an expectation-maximisation algorithm for finding the transition probabilities/rates and the emission probabilities, given a set of possible emissions, a number of states to fit and an observed sequence of emissions. Viterbi will take an observed sequence of emissions and the parameters of an HMM, and produce the most likely state in which each of these emissions happened. The HiddenMarkov package hosted on CRAN [4] is the only easily-accessible Hidden Markov Model package which supports the Markov Modulated Poisson Process, but does not contain any implementation of the Viterbi algorithm. It therefore became necessary to write one.

3.2.1 A derivation of the classical viterbi algorithm

The Viterbi algorithm for a standard Discrete Time Hidden Markov Model relies on a known, finite observation space O, a known distribution on O for each state s, p_s, and known transition probabilities π_i, j, with known initial probabilities for each state s, δ_s. We let $[k] = \{1, 2, ..., k\}$. The goal is, given a sequence of T observations $(y_n)_{n\in[T]}$, $y_i \in O$, to find the sequence of states $(x_n)_{n\in[T]}$ satisfying

$$\mathbf{x} = \arg\max_{\widehat{\mathbf{x}}\in S^T} \mathbb{P}(\widehat{\mathbf{x}}|\mathbf{y})$$

We call $\mathbf{x}$ the Viterbi Path.

We let $V_{t,s}$ be the probability of the most probable state sequence responsible for the first t observations which ends in state s . We have that $V_{1,s}$ is the probability of both being in state s at time 1 and seeing observation y_1 from state s. A simple application of Bayes' Rule gives us that

$$V_{1,s} = \mathbb{P}(y_1|s)\mathbb{P}(x_1 = s)$$

Recall that δ_s is the probability of being in state s at time 1, ie $\mathbb{P}(x_1 = s)$, and that $p_s(y_1)$ is the probability of observing y_1 from state s, ie $\mathbb{P}(y_1|s)$. Both of these parameters are estimated by the Baum-Welch algorithm, hence,

$$V_{1,s} = p_s(y_1)\delta_s$$

Given $V_{\tau,s}$ for $\tau < t$, we can find $V_{t,s}$ by noting that the Markov Property implies a form of memorylessness. Given the present, the future is conditionally independent of the past. As such, we need only consider $V_{t-1,s}$ for each s, as well as our known parameters.

The probability of the most likely path that leads us to state s at time t is given by the probability of the most likely path that led us to s' at time $t-1$, and then jumped to s at time t, and then emitted y_t from state s. The probability of jumping from s' to s is $\pi_{s',s}$. The probability of emitting y_t from state s is $p_s(y_t)$. The probability of the most likely path that leads us to s' at time $t-1$ is $V_{t-1,s'}$. Hence,

$$V_{t,s} = p_s(y_t) \max_{s'\in S}(\pi_{s',s}V_{t-1,s'})$$

Using this recurrence, we can find $V_{t,s}\forall t \in [T]$ by a standard dynamic programming algorithm.

From here, we can then work backwards to find the Viterbi path. $x_T = \arg\max_{s\in S} V_{t,s}$ - the most likely final state is the state in which the path of maximum probability ends.

Let

$$T_{t,s} = \arg\max_{s'\in S}(\pi_{s',s}V_{t-1,s'})$$

Ie, $T_{t,s}$ is the state from which we are most likely to have come at time $t-1$ given that we are in state s at time t, we can then see that $x_{t-1} = Tx_t, t$. Note the similarities in the definitions of V and T - both can be calculated simultaneously - V is the maximum, T is the argument that maximises. $T_{1,s}$ is never used, so need never be defined.

Since we have an expression for x_{t-1} in terms of x_t, and for x_T, we can then recover $\mathbf{x}$, the Viterbi Path. 5 defines this algorithm.

```
Data: $(S, \delta, \Pi, O, p)$, a DTHMM
input  : $\mathbf{y}$, an observed sequence of $T$ emissions, indexed from 1 to $T$
output: $\mathbf{x}$, the most likely sequence of states generating these emissions
begin
    for $s \in S$ do
        $V_{1,s} \leftarrow p_s(y_1)\delta_s$
    end
    for $t \leftarrow 2$ to $T$ do
        for $s \in S$ do
            $V_{t,s} \leftarrow p_s(y_t) \max_{s' \in S}(\pi_{s',s} V_{t-1,s'})$
            $T_{t,s} \leftarrow \arg\max_{s' \in S}(\pi_{s',s} V_{t-1,s'})$
        end
    end
    $x_T \leftarrow \arg\max_{s' \in S}(V_{s',T})$
    for $t \leftarrow T$ to 2 do
        $x_{t-1} \leftarrow T_{x_t,t}$
    end
    return $\mathbf{x}$
end
```

Algorithm 5: The Viterbi Algorithm for DTHMMs

3.2.2 A First Approximation of the Viterbi Algorithm for the Markov-Modulated Poisson Process

Recall the dependencies for the DTHMM Viterbi Algorithm. We require knowledge of a finite O, p_s for each s, and π_{ij} for each pair of states i, j.

In an MMPP, we observe a Poisson Process of rate randomly varying between various known rates - the rates being our states. Let $S = \{\lambda_1, ..., \lambda_m\}$. Our observations can be interpreted as exponential random variables of these rates. Let $\tau_0 = 0$, and τ_i be the time of the i^{th} poisson emission for $i \in [n]$. Let $y_i = \tau_i - \tau_{i-1}$ for $i \in [n]$. Given that the underlying CTMP was in state λ_s at time τ_i, we have that $y_i \sim Exp(\lambda_s)$. From the properties of the generic CTMP, the probability that the process is in state j at time τ_i given that it was in state i at time τ_{i-1} is given by $(e^{Qy_i})_{\lambda_i,\lambda_j}$. To ease notation, we'll write this as $(e^{Qy_i})_{i,j}$, omitting the λs.

The state space S and transition rates Q are estimated by the Baum Welch algorithm as before, so after running Baum Welch over an observed trace, we can start to find the most likey state at each emission. Note that this "Viterbi Path" is not the most likely sequence of state transitions, it is instead the most likely state in which the underlying CTMP resides at the time of each emission. This quirk is discussed in

Since our emissions are continuous, we don't have any notion of "most probable" - if we model height continuously, the probability that I meet someone exactly 1.8m tall is the same as the probability that I meet someone exactly 18m tall, they're both 0 - so instead we'll base our likelihood calculations off probability density - capturing the idea that, even though I don't know for certain that I'll meet one of the two, it's more likely for me to meet the 1.8m tall person.

We let $p_s(t) = \lambda_s e^{-t\lambda_s}$, the probability density of an exponential random varaible of rate λ_s evaluated at t. Let $V_{t,s}$ be the probability density of the most likely path that leads us to emitting y_t from state s. We have that

$$V_{1,s} = \delta_s p_s(y_1)$$

The probability density of the most likely path that leads us to waiting for time y_1 before making an emission is given by the probability of starting in state s, multiplied by the probability density of waiting

y_1 for an emission from state s. The memorylessness property of a CTMP allows us to only consider $V_{t-1,s}$ when calculating $V_{t,s}$. We have that

$$V_{t,s} = p_s(y_t) \max_{s' \in S}(V_{t-1,s'}(e^{Qy_t})_{s',s})$$

The probability density of the most likely path that leads us to waiting for time y_t between the $(t-1)^{th}$ and t^{th} emissions in state s is given by the probability density of the most likely path that takes us to state s' for the $(t-1)^{th}$ emission, followed by jumping (along any arbitrary path) into state s for emission t, multiplied by the probability density of emitting y_t in state s.

From here, we can proceed as normal. We define T as before to record our most likely states at each transition, and work backwards to find $\mathbf{x}$, producing the following algorithm

```
Data: $(S, \delta, Q, p)$, an MMPP
input  : $\mathbf{y}$, the absolute times of an observed sequence of $T$ emissions, indexed from 1 to $T$
output: $\mathbf{x}$, the most likely sequence states in which the underlying CTMP resides for each emission in
        $\mathbf{y}$
begin
    for $t \leftarrow 2$ to $T$ do
        $\tau_{t-1} \leftarrow y_t - y_{t-1}$
    end
    $T \leftarrow T - 1$ for $s \in S$ do
        $V_{1,s} \leftarrow p_s(t_1)\delta_s$
    end
    for $t \leftarrow 2$ to $T$ do
        A ¡- $e^{Q\tau_t}$ for $s \in S$ do
            $V_{t,s} \leftarrow p_s(\tau_t) \max_{s' \in S}(A_{s',s}V_{t-1,s'})$
            $T_{t,s} \leftarrow \arg\max_{s' \in S}(A_{s',s}V_{t-1,s'})$
        end
    end
    $x_T \leftarrow \arg\max_{s' \in S}(V_{s',T})$
    for $t \leftarrow T$ to 2 do
        $x_{t-1} \leftarrow T_{x_t,t}$
    end
    return $\mathbf{x}$
end
```

Algorithm 6: An Approximate Viterbi Algorithm for MMPPs

The reason that this is an approximation is the fact that the algorithm assumes that either a jump happens instantaneously, or not at all - we always evaluate $p_s(y_t)$, rather than $p_s(y_t - ⅌)$[1], where ⅌ represents the time we wait for all the relevant transitions to occur. The times between state transitions are exponentially distributed, and we are dealing with the most likely outcomes. The most likely outcome of any exponential distribution is 0 so, if the underlying CTMP jumps from one state to another, the most likely time for that jump to happen is immediately, so ⅌ $= 0$. If multiple jumps happen, their individual times are exponentially distributed, but their sum does not have a mode of 0.

This first approximation is in fact very powerful. Approximating in this way assumes that multiple transitions between emissions are rare, alternatively that emissions within each state are more frequent than transitions out of that state. If the converse is true - transitions out of the state are more frequent than emissions within the state - then we have an odd situation whereby some state is frequently entered and then left again with no emissions happening. Whilst such states are perfectly valid and can indeed exist in a theoretical MMPP, detecting one in reality with no prior knowledge is very difficult.

As a final note, we can further refine the fitted model based on the results of the Viterbi algorithm by changing the estimated rate of each state to the observed rate of the emissions estimated to occur in that

[1]⅌- tinco - represents the voiceless alveolar stop in the Tengwar alphabet, as devised by JRR Tolkein. When dealing with time so frequently, we eventually run out of ways to write the letter t

state. By the nature of these estimation algorithms, these results are likely to differ, and with a large sample size what we observe is usually closer to the truth than what we assert.

3.2.3 Applying the Algorithm

The algorithm was written in R and added to the pre-existing HiddenMarkov [4] package as acquired from The University of Bristol's CRAN Mirror, which was then recompiled to be loaded into an R environment. A Python script was written to call into this library from a more conveneint language using RPy2 , with mathematical, statistical and visualisation functions loaded from matplotlib[5], NumPy and SciPy.

As before, we first simulate a model, then see if we can recover it. The model we simulated had the following parameters;

$$
\begin{aligned}
(\lambda_1, \lambda_2, \lambda_3) &= (0.01, 0.5, 2) \\
S &= \{\lambda_1, \lambda_2, \lambda_3\} \\
Q &= \begin{array}{c|ccc} & \lambda_1 & \lambda_2 & \lambda_3 \\ \lambda_1 & \frac{-1}{20} & \frac{1}{60} & \frac{1}{30} \\ \lambda_2 & \frac{1}{10} & \frac{-2}{15} & \frac{1}{30} \\ \lambda_3 & \frac{1}{10} & \frac{1}{60} & \frac{-7}{60} \end{array} \\
\delta &= \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)
\end{aligned}
$$

Because the expected course of an MMPP is somewhat harder to calculate, this one was just simulated for 10,000 hours. The resulting trace can be seen in

Running the Baum-Welch algorithm over the trace with a known state size of 3 for 208 iterations, at which the model converged sufficiently, gives us the following predictions, rounded to 2 significant figures;

$$
\begin{aligned}
(\lambda_1, \lambda_2, \lambda_3) &= (0.0095, 0.50, 1.99) \\
S &= \{\lambda_1, \lambda_2, \lambda_3\} \\
Q &= \begin{array}{c|ccc} & \lambda_1 & \lambda_2 & \lambda_3 \\ \lambda_1 & -0.062 & 0.028 & 0.034 \\ \lambda_2 & 0.087 & -0.10 & 0.053 \\ \lambda_3 & 0.083 & 0.025 & -0.11 \end{array} \\
\delta &= (1.00, 0.00, 0.00)
\end{aligned}
$$

On inspection, most these parameters give a reasonable approximation of the originals, but δ seems to be well off the mark. The reason for this is fairly simple - the simulated trace had to start somewhere, and in this case it started in state 1. The algorithm was only given one trace, so to minimise the probability of error, it estimated that the underlying CTMP always starts in the state in which it was estimated to start. The initial distribution of the underlying CTMP isn't hugely relevant, however. We're interested in how the user tweets in the long term, and regardless of initial distribution this markov chain will reach an invariant distribution.

Running the modified viterbi algorithm over the process gives

Comparing the estimated path to the original here seems daunting, but we can do this in what I find a fairly beautiful way by simply aligning the two traces along the same scale, recolouring each state such that the parameters align, then letting them overlap at 75% transparency. The resulting colours can then show us exactly where errors are happening. Displaying the entire thing in this document is near pointless - there's simply not enough space for this kind of detail - the fairest method, which also prevents me from mining my data for a favourable result, is to take 100 time units from a random point onwards, as generated by a random number generator The predicted path matched the actual path of the time, so this fitting algorithm is very accurate for this process, and it's well worth us applying it to the observed data to see how it fares.

The data was preprocessed such that rather than recording exact times and dates of tweets, it instead recorded time since the beginning of the observation in hours at which each tweet occured. This was then fed into a HiddenMarkov MMPP, over which BaumWelch and Viterbi were then run based on the assumption of 3 states. The resulting model was as follows:

$$
\begin{aligned}
(\lambda_1, \lambda_2, \lambda_3) &= (0.0289, 1.28, 15.5) \\
S &= \{\lambda_1, \lambda_2, \lambda_3\} \\
Q &= \begin{array}{c} \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{array} \begin{array}{c} \begin{array}{ccc} \lambda_1 & \lambda_2 & \lambda_3 \end{array} \\ \begin{pmatrix} -0.123 & 0.120 & 0.004 \\ 0.441 & -1.09 & 0.648 \\ 0.00 & 5.25 & -5.25 \end{pmatrix} \end{array} \\
\delta &= (1.00, 0.00, 0.00)
\end{aligned}
$$

And, with the states shaded in in the colours represented above, the transitions were as follows:

From this distance, we already see at least some kind of sensible behavior - the user seems to have regular sleeping patterns, tweeting during the day and not tweeting at night-time. Zooming in on a single day, we see some more sensible results - the burst mentioned before has been noticed and highlighted, and the normal day-to-day tweets all remain in the same state. The three states correspond to intuitive states for a human to occupy - in state λ_1 the user is asleep or otherwise away from the computer, tweeting roughly once every 50 hours, but only staying there for about 10 hours. In state λ_2, the user is awake, online, and going about his day as usual - as an active twitter user he tweets about 1.3 times per hour. Roughly every hour, the user will then either go to bed with probability 0.4, or enter into a conversation with someone and produce a burst with probability 0.6 - bursts are tweets produced at a rate of 15 per hour, or one every 4 minutes, and last around 12 minutes on average. This all seems fairly reasonable, though the probabilities of of the transitions out of the awake state seem

We can evaluate this model by performing a Kolmogorov-Smirnov test on the data for each state. It should follow an exponential distribution. Running this test, we find, however, that it very much does not;

Whilst 3 states makes sense intuitively, there's no reason why this is definitely the case, so let's try different numbers of states. We can find the most optimal number based on the Bayesian Information Criterion Given an estimated model, we defined the Bayesian Information Criterion, BIC, as

$$BIC = -2 \ln L + k \ln n$$

Where $\ln L$ is the log-likelihood of the fitted model, ie the natural logarithm of the probability of observing the data given that the model is correct, k is the number of parameters in the model, and n is the number of data points used to fit the model.

An MMPP of s states has $k = s^2 + s - 1$ free parameters - Q contains s^2 elements, but each of the s diagonal elements can be determined by the row on which it resides, we fit s states, each of which is freely choosable, and δ has s elements, one of which can be determined from the other $s - 1$. The number of data points is the number of tweets we've observed, and the log likelihood is given by the Baun-Welch algorithm.

500 iterations of Baum-Welch were run over the data until a local optimum was reached. This optimum occured at 7 states, described as follows:

Fitted by the same methods, we see the following results

Unfortunately, the Kolmogorov-Smirnov tests still fail. Fitting more states would probably give a better fit, but due to the quadratic scaling it would also start sending the number of parameters up to absurd levels.

3.2.4 An Alternative Approach

From this point, it is starting to seem like the tweeter does not follow an MMPP, but there is one last thing we can attempt in this vein. A DTHMM can still have a continuous observation space and all the previous methods will work, just with probability density in place of probability. Rather than observing a series of times, we will instead observe a series of time differences, and look for any small number of states that lets us gather these differences into the same exponential distribution. The results from such a model will be similar, but lose some information about how long our tweeter actually spends in each state. The crucial difference between the two is that in an MMPP emissions and transitions take place along the same timeline - compare the Viterbi algorithms for both the MMPP and DTHMM, and note that the transition probabilities between a DTHMM's states do not depend on the observed emissions, whilst the transition probabilities in the MMPP version do.

So we go through the same procedure again - start by fitting the intuitive 3 states to find the following model;

Then use the Bayesian Information Criterion to note that the optimal number of states is, again, 7;

And finally perform some new KS tests to find that...

The data cannot be well-clustered into a small number of exponentially distributed subsets. We can then conclude that these data do not follow any kind of simple poisson process, in spite of appearances.

3.3 A Zoo of Problems

Whilst this gives a fairly strong negative result that this tweeter is not a poisson process, it doesn't give any real explanation as to why. For that, we need to start looking at the actual emissions given in each state. For starters, let's go back to the MMPP fitted with my modified Viterbi algorithm, and inspect the values assigned to each state.

So we see that, whilst an exponential distribution seems valid for the low end of the distribution, the tail seems somewhat heavier. The means and standard deviations are equal, as expected from an exponential distribution, but the density is off by a significant margin, which is why the KS tests were failing. We see something similar for a DTHMM with exponential emissions, and the problem persists;

Indeed, on inspection, the observed emissions within each state seem closer to a lognormal distribution. If X follows a lognormal distribution, then $\log(X)$ follows a normal distribution . Plotting a lognormal density curve alongside the observed curve, we see a somewhat closer fit, even though these data were fitted to an exponential distribution.

Summing lognormal distributions into some stochastic process in the same way that we sum exponential distributions into a poisson process carries all manner of problems, however. The lognormal distribution is not memoryless, so the resulting process will be non-markovian, meaning that simple fitting algorithms like Baum-Welch and Viterbi require much more sophisticated modifications to work correctly.

We can, however, fit a discrete model to the data using a DTHMM with lognormal emissions, so let's do that.

Appendix A

Stuff

Bibliography

[1] Joseph L. Doob, The Development of Rigor in Mathematical Probability (1900-1950)
The American Mathematical Monthly, Vol. 103, No. 7 (Aug. - Sep., 1996), pp. 586-595
`http://www.jstor.org/stable/2974673`

[2] Weisstein, Eric W. "Markov Chain." From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/MarkovChain.html`

[3] Weisstein, Eric W. "Graph." From MathWorld–A Wolfram Web Resource.
`http://mathworld.wolfram.com/Graph.html`

[4] David Harte. *HiddenMarkov* v1.7-0, a hidden markov model library written in R and Fortran, available at `http://cran.r-project.org/web/packages/HiddenMarkov/index.html`

[5] Hunter, J. D. *MatPlotLib*, a 2D graphics environment available at `http://matplotlib.org/`
IEEE Computer Soc, Vol. 9, No. 3 (2007), pp. 90-95