Latent Variable Models, part 1

OxWaSP module 3: applied statistics

Xavier Didelot

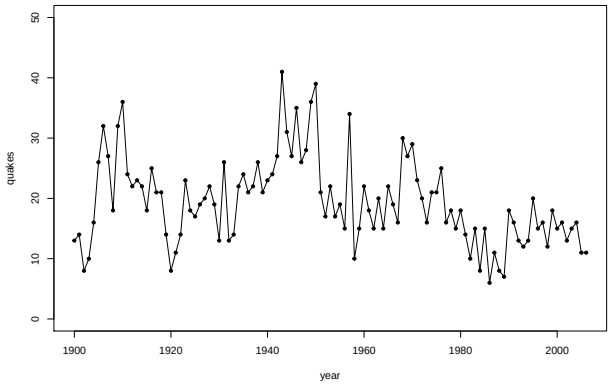
November 2018

Question 1

- Download the file `data.csv` at http://bit.ly/2EXIScZ
- This contains the number of major earthquakes (magnitude 7 or greater) in the world, from 1900 until 2006
- Load this data using `read.csv` and plot it

Answer 1

```
data=read.csv('data.csv')
plot(data,type='o',ylim=c(0,50),pch=20)
```



Question 2

- We initially assume that the number of quakes is iid for each year.
- Which common probability distribution would you suggest as a first attempt to model the number of quakes?
- Does this fit well or not?

Answer 2

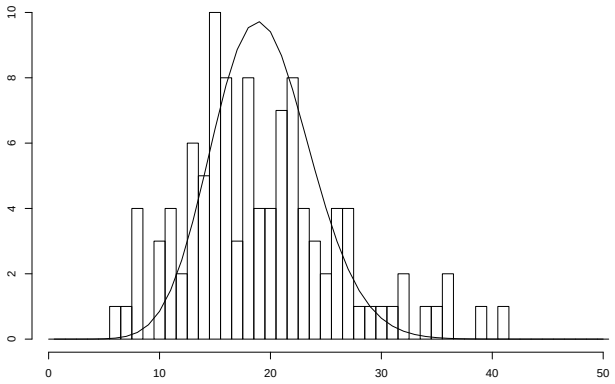
- A natural model would be a Poisson distribution with mean λ:

$$p(X_t = x_t) = e^{-\lambda}\lambda^{x_t}/x_t!$$

- The variance of a Poisson distribution is equal to its mean λ. But here we have a mean of 19.364486 and a variance of 51.5734438. So the Poisson model does not fit due to overdispersion of the data.

Answer 2

```
hist(data$quakes,breaks = 0.5:50.5,xlab='',ylab='',main='')
lines(0:50,length(data$quakes)*dpois(0:50,mean(data$quakes)))
```



Question 3

- Given that the Poisson model does not fit well, what would you suggest?

Answer 3

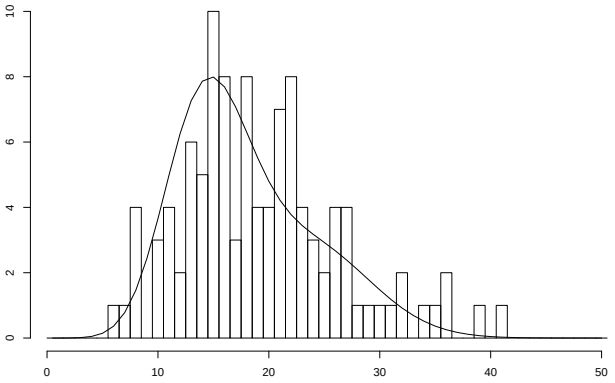
- We could consider a model with over-dispersion, for example a Negative Binomial distribution
- However, the distribution seems to have multiple modes which would not be captured
- Instead, let us consider an independent mixture model of (at least) two Poisson distributions
- For example, consider that 70% of the observations are from Poisson($\lambda_1 = 15$) and 30% are from Poisson($\lambda_2 = 25$):

$$p(X_t = x_t) = 0.7e^{-\lambda_1}\lambda_1^{x_t}/x_t! + 0.3e^{-\lambda_2}\lambda_2^{x_t}/x_t!$$

- In other words, the t^{th} observation X_t is generated by sampling C_t from Bernouilli(0.3) and then sampling X_t from Poisson(λ_{1+c_t})
- The fit is improved (or is it?), and could be improved further by fitting the parameters and/or considering mixtures of more components

Answer 3

```
hist(data$quakes,breaks = 0.5:50.5,xlab='',ylab='',main='')
dens=0.7*dpois(0:50,15)+0.3*dpois(0:50,25)
lines(0:50,length(data$quakes)*dens)
```



Question 4

- How can we fit the model which is a mixture of two Poisson?
- Three parameters
- Write the likelihood function, and optimise using `optim`

Answer 4

```
ll=function(p) -sum(log(p[1]*dpois(data$quakes,p[2])+
                        (1-p[1])*dpois(data$quakes,p[3])))
o=optim(c(0.5,10,30),ll,method="L-BFGS-B",lower=c(0,0,0),upper=c(1,Inf,Inf))
o$par
```

```
## [1]  0.6757244 15.7771012 26.8398792
```

```
-o$value
```

```
## [1] -360.369
```

Question 5

- Instead let's use the package `depmixS4`
- Use the function `mix` to form the model
- Use the function `fit` to fit the model
- Use the function `getpars` to print the parameters
- Use the function `logLik` to print the likelihood

Answer 5

```
library(depmixS4)
m=mix(quakes~1,nstates=2,data=data,family=poisson())
m=fit(m)
```

```
getpars(m)
```

```
##         pr1         pr2 (Intercept) (Intercept)
##   0.3248868   0.6751132   3.2894976   2.7582445
```

```
logLik(m)
```

```
## 'log Lik.' -360.3691 (df=3)
```

Question 6

- Is this model really better than the Poisson model?
- Calculate the likelihood under both models
- Compare
- Is this comparison fair?

Answer 6

```
sum(dpois(data$quakes,mean(data$quakes),log = T))
```

```
## [1] -391.9189
```

```
logLik(m)
```

```
## 'log Lik.' -360.3691 (df=3)
```

Answer 6

- Comparing the likelihoods is not fair, since the Poisson model has only one parameter and the mixture model has three
- The Poisson model can never win, since the mixture model can reduce to the Poisson model if we take $\lambda_1 = \lambda_2$ or $p = 0$ or $p = 1$
- We need to correct for the complexity of the models when comparing them

Model selection

- Two criteria are in common use to account for the complexity of models when comparing them
- The Akaike information criterion:

$$\mathrm{AIC} = -2\log(L) + 2p$$

- The Bayesian information criterion:

$$\mathrm{BIC} = -2\log(L) + p\log(T)$$

- p is the number of parameters of the model, and T the number of observations
- The model with the smallest AIC or BIC is selected

Question 7

- How many states should we use in the mixture model?

Answer 7

```
res=matrix(NA,10,3)
for (i in 1:10) {
  m=mix(quakes~1,nstates=i,data=data,family=poisson())
  m=fit(m)
  res[i,1]=logLik(m)
  res[i,2]=AIC(m)
  res[i,3]=BIC(m)
}
```

Answer 7

logLik	AIC	BIC
-391.9189	785.8379	788.5107
-360.3691	726.7381	734.7566
-356.8490	723.6980	737.0622
-356.7489	727.4978	746.2076
-356.7343	731.4686	755.5241
-356.7431	735.4862	764.8873
-356.7342	739.4684	774.2152
-356.7383	743.4765	783.5689
-356.7356	747.4711	792.9092
-356.7374	751.4747	802.2585

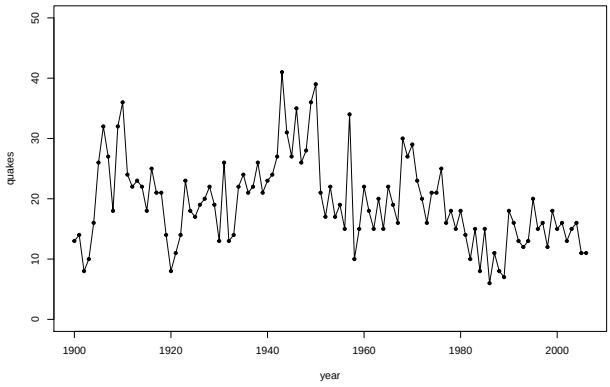
Question 8

- So far we assumed that the number of quakes is independent for each year.
- Do you think this is acceptable?
- What are we going to do about it?

Answer 8

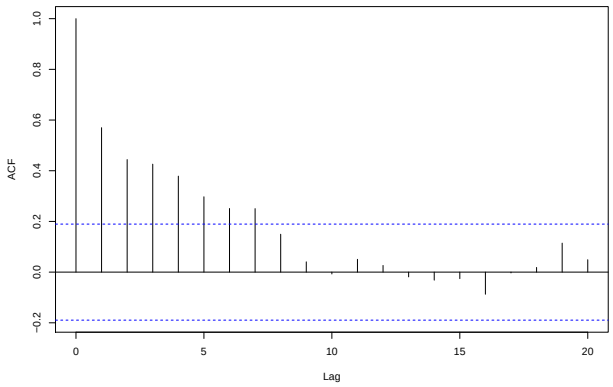
- It is clear by visual inspection that there is significant serial dependence in the data
- This can be made even clearer and tested by plotting the autocorrelation function (ACF) which is the Pearson's correlation at different lag values (lag=length of interval between observations)
- There is significant autocorrelation, with the first 8 ACF values being significantly greater than zero

Answer 8



Answer 8

```
acf(data$quakes,main='')
```



Latent Variable Models, part 2

OxWaSP module 3: applied statistics

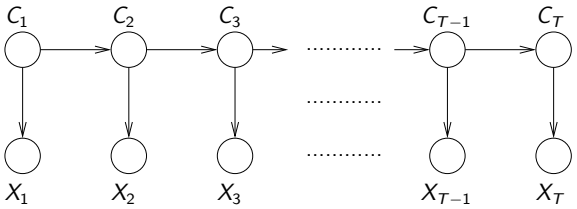
Xavier Didelot

November 2018

A first hidden Markov model

- The independent mixture model we used does not capture serial dependence
- In this independent mixture model, the C_t variables representing which Poisson to sample X_t from were independently identically sampled from Bernouilli(0.3)
- Let us now instead consider that C_t is a Markov chain (homogenous, with order 1)
- The C_t variables are unobserved, but the X_t variables are observed
- The resulting process is therefore called a hidden Markov model
- It can be thought of as a natural extension to the independent mixture model

Dependency graph of a hidden Markov model



$$p(X_1,...,X_T,C_1,...,C_T) = p(C_1)\prod_{k=2}^{T} p(C_k|C_{k-1})\prod_{k=1}^{T} p(X_k|C_k)$$

Refresher on Markov chains

- The Markov property states that:

$$p(C_t|C_{t-1},C_{t-2},...,C_1) = p(C_t|C_{t-1})$$

- A Markov chain is defined by the transition probabilities:

$$\gamma_{ij} = p(C_t = j|C_{t-1} = i)$$

- The matrix of transition probabilities is denoted $\mathbf{\Gamma}$
- If the Markov chain has m states, $\mathbf{\Gamma}$ is of size $m \times m$ with rows adding up to 1, so that it contains $m \times (m-1)$ free parameters. Such a matrix is sometimes called a Markov matrix.
- We can calculate the probability of a sequence of length L:

$$p(c_1,...,c_L) = p(c_1)\prod_{t=2}^{T}\gamma_{c_{t-1}c_t}$$

- The first term $p(c_1)$ is given by the initial distribution of the Markov chain

Stationary distribution

- The unconditional distribution of the Markov chain at t is:

$$\boldsymbol{u}(t) = (p(C_t = 1), p(C_t = 2), ..., p(C_t = m))$$

- The initial distribution is therefore $\boldsymbol{u}(1)$
- A Markov chain with transition probability matrix $\mathbf{\Gamma}$ has stationary distribution $\boldsymbol{\delta}$ if:

$$\boldsymbol{\delta\Gamma} = \boldsymbol{\delta} \text{ and } \boldsymbol{\delta 1'} = 1$$

- This can be solved as a system of equations, or by finding the eigenvector with eigenvalue equal to 1.
- If the initial distribution is the stationary distribution, the chain has the same unconditional distribution at all points, eg $\boldsymbol{u}(2) = \boldsymbol{u}(1)\mathbf{\Gamma} = \boldsymbol{\delta\Gamma} = \boldsymbol{\delta}$. This is a stationary Markov chain.

HMM definition

- A Hidden Markov Model (HMM) is a Markov chain in which the sequence of states is not observed but hidden
- Instead of observing the sequence of states, we observe a stochastic function of them called emissions or observations
- Let $X_1,...,X_T$ denote the (observed) sequence of T emissions and $C_1,...,C_T$ denote the (hidden) sequence of states
- A HMM is defined by two quantities:
  - The transition matrix $\mathbf{\Gamma}$ of elements γ_{ij} where i and j are states:

$$\gamma_{ij} = p(C_t = j|C_{t-1} = i)$$

  - The emission probabilities $p_i(x)$ where i is a state and x is an emission:

$$p_i(x) = p(X_t = x|C_t = i)$$

- Note that C_t is discrete, but X_t can be discrete, continuous, multivariate, etc.

A first hidden Markov model for the earthquake dataset

- HMM with two states
- Transition matrix:

$$\boldsymbol{\Gamma} = \begin{pmatrix} 0.94 & 0.06 \\ 0.14 & 0.86 \end{pmatrix}$$

- This gives the stationary distribution:

$$\boldsymbol{\delta} = (0.7, 0.3)$$

- Emission probabilities:

$$p_1(x) = e^{-15}15^x/x! \text{ and } p_2(x) = e^{-25}25^x/x!$$

Link between HMM and mixture model

- This HMM model has the same marginal distributions as our previous independent mixture model
- But the independent mixture model considered that the X_t are independent, whereas in the HMM the autocorrelation of X_t is explicitly modelled
- The HMM model would reduce to the mixture model if we defined:

$$\boldsymbol{\Gamma} = \begin{pmatrix} 0.7 & 0.3 \\ 0.7 & 0.3 \end{pmatrix}$$

- The HMM is therefore an extension of the mixture model

A bit of history... (1/2)

- Andrey Markov (Russian Empire) studied Markov chains in the early 20$^{\text{th}}$ century
- Fundamental HMM results were first described by Ruslan Stratonovich (USSR) in Russian publications in the late 1950s and translated to English in 1960
- Thorough analysis of HMM by Leonard Baum (USA) in the second half of the 1960s

A bit of history... (2/2)

- In the 1970s, HMMs became popular for application to speech recognition

  - A speech signal is recorded and divided each small pieces (frames) of ~10 milliseconds
  - Each frame is classified into 256 categories
  - Aim is to recognise the sequence of words being spoken
- Also attracted interest for military applications (eg target tracking)
- In the second half of the 1980s, HMMs began to be applied in biostatistics
- Became hugely popular for DNA analysis in the 1990s and remains ubiquitous since
- Many extensions still under active research (eg SMC)

Univariate marginal distribution

- What is $p(X_t = x)$?
- Decompose over states at t:

$$p(X_t = x) = \sum_{i=1}^{m} p(C_t = i)p(X_t = x|C_t = i)$$

- It is convenient to rewrite in matrix notation:

$$p(X_t = x) = \boldsymbol{u}(t)\boldsymbol{P}(x)\mathbf{1}'$$

  where $\boldsymbol{P}(x)$ is a diagonal matrix with i^{th} diagonal element equal to $p_i(x)$
- Since $\boldsymbol{u}(t) = \boldsymbol{u}(t-1)\boldsymbol{\Gamma}$ we have $\boldsymbol{u}(t) = \boldsymbol{u}(1)\boldsymbol{\Gamma}^{t-1}$ and therefore:

$$p(X_t = x) = \boldsymbol{u}(1)\boldsymbol{\Gamma}^{t-1}\boldsymbol{P}(x)\mathbf{1}'$$

- If the chain has initial distribution equal to the stationary distribution $\boldsymbol{\delta}$, then:

$$p(X_t = x) = \boldsymbol{\delta}\boldsymbol{P}(x)\mathbf{1}'$$

Bivariate marginal distribution

- What is $p(X_t = v, X_{t+k} = w)$?
- Decompose over states at both t and $t + k$:

$$p(X_t = v, X_{t+k} = w) = \sum_{i=1}^{m}\sum_{j=1}^{m} p(X_t = v, X_{t+k} = w, C_t = i, C_{t+k} = j)$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{m} p(C_t = i)p_i(v)\gamma_{ij}(k)p_j(w)$$

  where $\gamma_{ij}(k)$ denotes the (i, j) element of $\boldsymbol{\Gamma}^k$
- In matrix format:

$$p(X_t = v, X_{t+k} = w) = \boldsymbol{u}(t)\boldsymbol{P}(v)\boldsymbol{\Gamma}^k\boldsymbol{P}(w)\mathbf{1}'$$

- If the Markov chain is stationary:

$$p(X_t = v, X_{t+k} = w) = \boldsymbol{\delta}\boldsymbol{P}(v)\boldsymbol{\Gamma}^k\boldsymbol{P}(w)\mathbf{1}'$$

Likelihood

The likelihood of a HMM is given by:

$$L_T = p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) = \boldsymbol{u}(1)\boldsymbol{P}(x_1)\boldsymbol{\Gamma}\boldsymbol{P}(x_2)\boldsymbol{\Gamma}\boldsymbol{P}(x_3)...\boldsymbol{\Gamma}\boldsymbol{P}(x_T)\mathbf{1}'$$

Likelihood

Define the vector $\boldsymbol{\alpha}_t$ such that

$$\alpha_t(j) = p(\boldsymbol{X}^{(t)} = \boldsymbol{x}^{(t)}, C_t = j)$$

In particular:

$$\alpha_1(j) = p(X_1 = x_1, C_1 = j) = u_1(j)p_j(x_1)$$

In matrix format: $\boldsymbol{\alpha}_1 = \boldsymbol{u}(1)\boldsymbol{P}(x_1)$

We have the recursion:

$$\alpha_t(j) = \sum_{k=1}^{m} p(\boldsymbol{X}^{(t)} = \boldsymbol{x}^{(t)}, C_t = j, C_{t-1} = k) = \sum_{k=1}^{m} \alpha_{t-1}(k)\gamma_{kj}p_j(x_t)$$

In matrix format: $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{P}(x_t)$

Finally, note that $L_T = \sum_{k=1}^{m} \alpha_T(k) = \boldsymbol{\alpha}_T\mathbf{1}'$ and the likelihood equation follows.

The forward algorithm

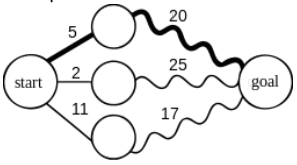
- An important consequence of the likelihood equation is that the likelihood can be calculated using the **forward algorithm**:
  - Set $\boldsymbol{\alpha}_1 = \boldsymbol{u}(1)\boldsymbol{P}(x_1)$
  - For t from 2 to T, calculate $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{P}(x_t)$
  - Return the likelihood $L_T = \boldsymbol{\alpha}_T\mathbf{1}'$
- This algorithm calculates the likelihood using a number of operations of order Tm^2
- This is much more efficient than the brute force approach of calculating the likelihood by summing over all m^T possible values for $\boldsymbol{C}^{(T)}$

Dynamic programming

- The forward algorithm, and other algorithms we will see later, is an example of **dynamic programming**
- In dynamic programming, a costly computation is replaced with a simpler one by exploiting a recursive form
- If we calculated all m^T combinations of $\boldsymbol{C}^{(T)}$, many subcalculations would be done over and over again
- By rearranging terms of the summation, or in other words reusing rather than recalculating certain terms, the algorithm becomes much more efficient

Dynamic programming

- Dynamic programming was developed by Richard Bellman in the 1950s
- Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner
- Divide and conquer
- Don't recalculate the same thing multiple times - memoisation



Filtering and smoothing

- The distribution $p(C_T|\boldsymbol{X}^{(T)})$ is often of interest
- This is the distribution of the last hidden state, at the end of the observed sequence
- The forward algorithm is a solution to this problem called **filtering**
- By definition $\alpha_T(j) = p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}, C_T = j)$ and therefore:

$$p(C_T = j|\boldsymbol{X}^{(T)}) = \frac{\alpha_T(j)}{\sum_{i=1}^{m}\alpha_T(i)} = \frac{\alpha_T(j)}{L_T}$$

- More generally, we might want to know the distribution $p(C_t|\boldsymbol{X}^{(T)})$ of hidden state at some point in the observed sequence
- This problem is called **smoothing** but can't be solved just using the forward algorithm...

Backward recursion

- Define the vector β_t such that

$$\begin{aligned}\beta_t(i) &= p(X_{t+1} = x_{t+1}, X_{t+2} = x_{t+2}, ..., X_T = x_T | C_t = i) \\ &= p(\boldsymbol{X}_{t+1}^T = \boldsymbol{x}_{t+1}^T | C_t = i)\end{aligned}$$

- In particular:

$$\beta_{T-1}(i) = p(X_T = x_T | C_{T-1} = i) = \sum_{k=1}^{m} \gamma_{ik} p_k(x_T)$$

- In matrix format: $\beta'_{T-1} = \boldsymbol{\Gamma P}(x_T)\mathbf{1}'$
- We have the recursion:

$$\begin{aligned}\beta_t(i) &= \sum_{k=1}^{m} p(X_{t+1} = x_{t+1}, \boldsymbol{X}_{t+2}^T = \boldsymbol{x}_{t+2}^T, C_{t+1} = k | C_t = i) \\ &= \sum_{k=1}^{m} \beta_{t+1}(k) \gamma_{ik} p_k(x_{t+1})\end{aligned}$$

- In matrix format: $\beta'_t = \boldsymbol{\Gamma P}(x_{t+1})\beta'_{t+1}$

Backward algorithm

- We deduce the **backward algorithm** which has similar properties to the forward algorithm:
  - Set $\beta'_{T-1} = \boldsymbol{\Gamma P}(x_T)\mathbf{1}'$
  - For t from $T-2$ down to 1, calculate $\beta'_t = \boldsymbol{\Gamma P}(x_{t+1})\beta'_{t+1}$
  - Return the likelihood $L_T = \boldsymbol{u}(1)\boldsymbol{P}(x_1)\beta'_1$
- This algorithm calculates the likelihood in an alternative manner to the forward algorithm, which is redundant, but it is important because of the next slide

Combining forward and backward values

- We have:

$$\begin{aligned}\alpha_t(i)\beta_t(i) &= p(\boldsymbol{X}^{(t)}, C_t = i) p(\boldsymbol{X}_{t+1}^T | C_t = i) \\ &= p(C_t = i) p(\boldsymbol{X}^{(t)} | C_t = i) p(\boldsymbol{X}_{t+1}^T | C_t = i) \\ &= p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}, C_t = i)\end{aligned}$$

- Therefore:

$$\boldsymbol{\alpha}_t \boldsymbol{\beta}'_t = p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) = L_T$$

- We now have T redundant ways of calculating L_T
- More importantly:

$$p(C_t = i | \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) = \frac{p(C_t = i, \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)})}{p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)})} = \alpha_t(i)\beta_t(i)/L_T$$

- We can therefore combine the forward and backward values to perform smoothing

Forward-backward algorithm

- The forward-backward algorithm can be used to perform smoothing
- It is simply the combination of the forward and backward algorithms
- Firstly we run the forward algorithm to calculate the values of $\boldsymbol{\alpha}_t$ and the likelihood L_T
- Secondly we run the backward algorithm to calculate the values of $\boldsymbol{\beta}_t$
- Finally we compute the values of $p(C_t = i | \boldsymbol{X}^{(T)})$ using:

$$p(C_t = i | \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) = \alpha_t(i)\beta_t(i)/L_T$$

Decoding terminology

- The smoothing problem is also called **local decoding**
- Decoding means to find the values of the hidden states. Local decoding means we find the value at a given point, as opposed to **global decoding** where we try and find the joint distribution of hidden states at all timepoints
- The path made from selecting the states with the highest marginal probability is called the **maximum accuracy path**
- But this path is not always best, and can even have a probability of zero
- A more principled solution is to try and find the path $\boldsymbol{c}^{(T)}$ with maximum probability, ie to maximise:

$$p(\boldsymbol{C}^{(T)} = \boldsymbol{c}^{(T)} | \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) \propto p(\boldsymbol{C}^{(T)} = \boldsymbol{c}^{(T)}, \boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)})$$

Global decoding

- We want to find the most probable state path π^*, ie the one with maximum probability:

$$\pi^* = \text{argmax}_{\boldsymbol{c}^{(T)}}(p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}, \boldsymbol{C}^{(T)} = \boldsymbol{c}^{(T)}))$$

- Could we consider all states paths? There are m^T possible paths, so even for the simplest of HMMs with $m = 2$ states this will not be doable for a short sequence of $T = 100$ observations...
- Luckily, this can be solved in a similar way as the likelihood calculation: using dynamic programming, ie using a recursive solution
- Let $v_k(i)$ denote the probability of the most probable path ending in state k for the sequence up to the i^{th} observation x_i
- Suppose $v_k(i)$ is known for all k. Then the probabilities can be calculated up to observation x_{i+1} using:

$$v_l(i+1) = p_l(x_{i+1}) \text{max}_k(v_k(i)\gamma_{kl})$$

The Viterbi algorithm (1/2)

- The probability that the chain starts in state k is $u_k(1)$ (for which we would often use the stationary distribution of the Monte Carlo with transition matrix $\mathbf{\Gamma}$)
- We first calculate recursively the $v_k(i)$ for i from 1 to T using the equation on the previous slide
- At the end of this recursion, we obtain the $v_k(T)$ for all k and we have that $\pi_T^* = \max_k(v_k(T))$ is the probability of the optimal path, and we know that this optimal path finishes in state $\text{argmax}_k(v_k(T))$
- We then trace our steps back from T down to 1 to reconstruct the chain of states that gave this optimal probability (to do this, we have to record pointers in the forward step)

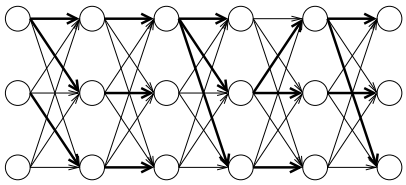
The Viterbi algorithm (2/2)

We deduce the famous Viterbi algorithm:

- Initialisation ($i = 1$):
  - $v_k(1) = u_k(1)p_k(x_1)$ for all $k = 1..m$
- Recursion ($i = 2..T$):
  - $v_l(i) = p_l(x_i)\max_k(v_k(i-1)\gamma_{kl})$
  - $\text{ptr}_i(l) = \text{argmax}_k(v_k(i-1)\gamma_{kl})$
- Termination:
  - $p(x, \pi^*) = \max_k(v_k(T))$
  - $\pi_T^* = \text{argmax}_k(v_k(T))$
- Traceback ($i = T..1$):
  - $\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$

Pointers

The pointer $\text{ptr}_i(l)$ stores from which state in step $i-1$ did we reach state l in step i. Recursively, this identifies a unique optimal path from the start (step 1) to state l in step i.



History of the Viterbi algorithm

- This algorithm is named after Andrew Viterbi, an Italian-born American electrical engineer
- He discovered the algorithm in 1967
- However, others discovered the algorithm independently soon before or after
- Discovered for military applications and kept secret?
- The Viterbi algorithm is another example of dynamic programming

Earthquake model

- Transition matrix:
$$\mathbf{\Gamma} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$
- We use as starting distribution the stationary distribution:
$$\boldsymbol{u}(1) = \boldsymbol{\delta} = (0.5, 0.5)$$
- Emission probabilities:
$$p_1(x) = e^{-15}15^x/x! \text{ and } p_2(x) = e^{-25}25^x/x!$$

Practical

- In `depmixS4` we can build a HMM model using the command `depmix` instead of `mix` which we used previously to build an independent mixture model
- Use `setpars` to set parameters, ie $\boldsymbol{u}(1) = (0.5, 0.5)$, $\mathbf{\Gamma} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$, $\lambda_1 = 15$ and $\lambda_2 = 25$
- Use the command `forwardback` and `viterbi` to run the algorithms we have described
- Compare local and global decoding for the quake data

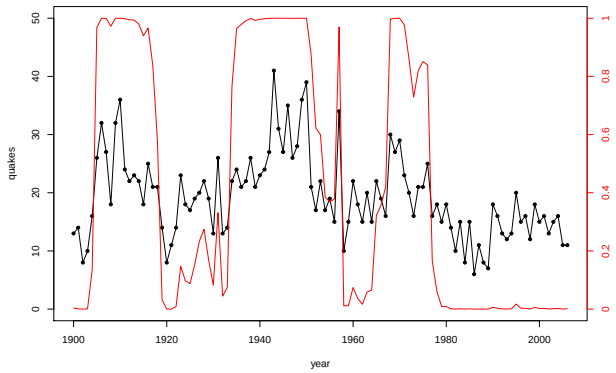
Earthquake example

```
library(depmixS4)
data=read.csv('data.csv')
m=depmix(quakes~1,data=data,nstates=2,family=poisson())
m=setpars(m,c(0.5,0.5,0.9,0.1,0.1,0.9,log(15),log(25)))
local=forwardbackward(m)$gamma #Local decoding
global=viterbi(m)$state        #Global decoding
summary(m)
```

```
## Initial state probabilties model
## pr1 pr2
## 0.5 0.5
##
## Transition matrix
##        toS1 toS2
## fromS1  0.9  0.1
## fromS2  0.1  0.9
##
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1           2.708
## St2           3.219
```

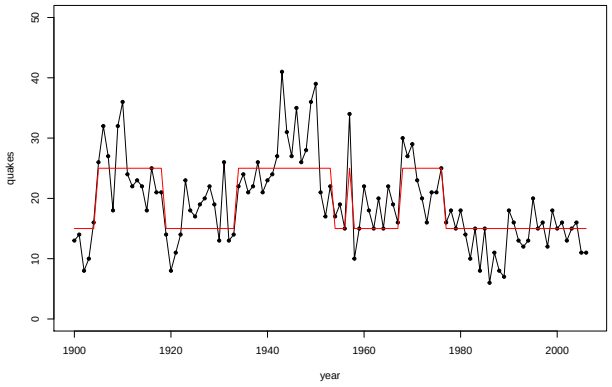
Earthquake example: local decoding

```
plot(data,type='o',ylim=c(0,50),pch=20)
lines(data$year,50*local[,2],col='red')
axis(4,at = seq(0,50,10),labels = seq(0,1,0.2),col='red',col.axis='red')
```



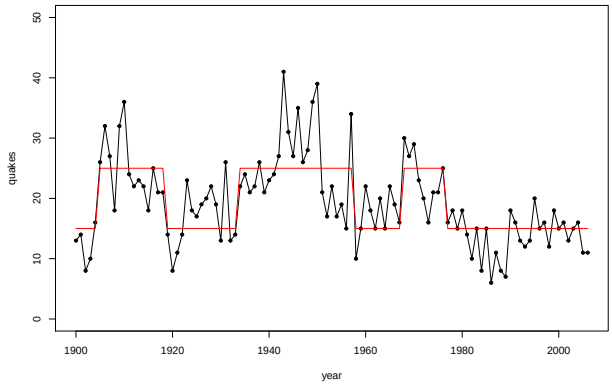
Earthquake example: local decoding

```
plot(data,type='o',ylim=c(0,50),pch=20)
lines(data$year,15+((local[,2]>0.5))*10,col='red')
```



Earthquake example: global decoding

```
plot(data,type='o',ylim=c(0,50),pch=20)
lines(data$year,15+(global-1)*10,col='red')
```



Latent Variable Models, part 3

OxWaSP module 3: applied statistics

Xavier Didelot

November 2018

Parameter estimation

- So far we discussed how to calculate the likelihood, and how to estimate the hidden states
- This assumed that we knew in advance the value of the transition matrix $\mathbf{\Gamma}$ and emission probabilities $p_i(x)$
- We are concerned with estimation of these parameters θ
- We consider that the structure of the HMM is known, especially the number of states, since this can be addressed using AIC/BIC as we saw for the independent case

Notation

- The following notations are going to be convenient:
- $u_j(t) = 1$ if and only if $c_t = j$ $(t = 1, 2, ..., T)$
- $v_{jk}(t) = 1$ if and only if $c_{t-1} = j$ and $c_t = k$ $(t = 2, 3, ..., T)$
- If we know the c_t then we can compute the $u_j(t)$ and $v_{jk}(t)$
- The likelihood can be rewritten using these notations

Using training data with known states

- Sometimes we have training data where the hidden states are known
- In machine learning terminology, this is supervised learning
- For example, in speech recognition, we may have a training dataset in which we know the spoken sentences
- In this case, we can count the number $f_{jk} = \sum_{t=2}^{T} v_{jk}(t)$ of transitions from state i to state j, and the transition probability can be estimated as:

$$\widehat{\gamma}_{jk} = \frac{f_{jk}}{\sum_{k=1}^{m} f_{jk}}$$

- This corresponds to the maximum likelihood estimate of γ_{jk} given $\boldsymbol{x}^{(T)}$ and $\boldsymbol{c}^{(T)}$

Using training data with known states

- The emission probabilities $p_k(x)$ can be estimated likewise via maximum likelihood for the positions where the state was k
- The exact form depends on the type of emission function $p_k(x)$
- For example if the emission distribution in state k is a Poisson distribution with mean λ_k (eg earthquake example), we have:

$$\widehat{\lambda}_k = \frac{\sum_{t=1}^{T} x_t u_k(t)}{\sum_{t=1}^{T} u_k(t)}$$

Maximising the likelihood

- Previously, we showed that the likelihood $L_T = p(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)})$ can be calculated using the forward algorithm
- We can therefore estimate the parameters using a standard numerical maximisation technique like the R command `optim`
- Risk of numerical underflow if the likelihood is calculated without directly without scaling or transformation
- There are constraints to satisfy, especially the fact that the rows of $\boldsymbol{\Gamma}$ must add up to one
- Risk of convergence to a local rather than global maximum

Baum-Welch Algorithm

- The Baum-Welch algorithm is a very popular alternative approach to estimate the parameters, first described by Leonard Baum and colleagues in 1970
- It is a special case of the Expectation-Maximisation (EM) algorithm
- The Baum-Welch algorithm was described before the general EM algorithm in 1977
- First we will describe the general EM algorithm

Expectation-maximisation

- The EM algorithm is a general algorithm for maximum-likelihood estimation with missing data
- We want to estimate the parameters θ given some data x
- The likelihood $p(x|\theta)$ is complicated due to missing data
- But if there was no missing data, the likelihood would be a relatively simple function $f(\theta)$
- We initiate the EM algorithm with a starting value for θ
- The E step and M step are repeated until convergence is reached
- E step: Compute the expectation of the (functions of the) missing data that appear in $f(\theta)$
- M step: Find the θ that maximises $f(\theta)$ in which the (functions of the) missing data are replaced by their expectations computed in the E step.

Application of EM to the mixture of two Poisson

- We want to estimate the parameters $\theta = \{\lambda_1, \lambda_2, q\}$ given some data $x = (x_1, ..., x_T)$ independently identically distributed from a mixture of two Poissons:

$$p(x|\theta) = \prod_{t=1}^{T} q d_{\text{Pois}}(x_t|\lambda_1) + (1-q) d_{\text{Pois}}(x_t|\lambda_2)$$

- This can be seen as a problem of missing data. Let $c = (c_1, ..., c_T)$ denote from which Poisson each x_t was sampled, then we have:

$$p(x, c|\theta) = \prod_{t=1}^{T} \left(q d_{\text{Pois}}(x_t|\lambda_1)\right)^{c_t} \left((1-q) d_{\text{Pois}}(x_t|\lambda_2)\right)^{1-c_t}$$

Application of EM to the mixture of two Poisson

- E step. Compute the expectation of c given x and θ.

$$\widehat{c}_t = \frac{q d_{\text{Pois}}(x_t|\lambda_1)}{q d_{\text{Pois}}(x_t|\lambda_1) + (1-q) d_{\text{Pois}}(x_t|\lambda_2)}$$

- M step. Find the θ that maximises the likelihood of c and x.

$$\widehat{\lambda}_1 = \frac{\sum_{i=1}^{T} x_t c_t}{\sum_{i=1}^{T} c_t}$$

$$\widehat{\lambda}_2 = \frac{\sum_{i=1}^{T} x_t (1-c_t)}{\sum_{i=1}^{T} 1 - c_t}$$

$$\widehat{q} = \frac{\sum_{i=1}^{T} c_t}{T}$$

Application of EM to the mixture of two Poisson

- For example $\widehat{q}$ is derived as follows:

$$\log p(x, c|\theta) = \sum_{t-=1}^{T} c_t \log(q) + (1-c_t)\log(1-q) + ...$$

$$\frac{\partial \log p(x, c|\theta)}{\partial q} = \sum_{t=1}^{T} \frac{c_t}{q} - \frac{1-c_t}{1-q}$$

$$0 = \sum_{t=1}^{T} c_t - \widehat{q}$$

$$\widehat{q} = \frac{\sum_{t-1}^{T} c_t}{T}$$

Baum-Welch Algorithm

- In the case of a HMM, the functions we need to estimate in the E step are the probability to be in a given state $u_j(t)$ and probability to transit from one step to another $v_{jk}(t)$
- Using local decoding and the forward-backward algorithm, we have already seen that:

$$\widehat{u}_j(t) = \alpha_t(j)\beta_t(j)/L_T$$

- Similarly we have:

$$\widehat{v}_{jk}(t) = \alpha_{t-1}(j)\gamma_{jk}p_k(x_t)\beta_t(k)/L_T$$

- In the E step, we compute $\widehat{u}_j(t)$ and $\widehat{v}_{jk}(t)$
- In the M step, we use the same equations to update θ that we described when the states were known (ie supervised learning), except that we replace $u_j(t)$ and $v_{jk}(t)$ with $\widehat{u}_j(t)$ and $\widehat{v}_{jk}(t)$, respectively.

Viterbi training

- Viterbi training algorithm:
  - Start with some parameter values θ
  - Find the hidden states using the Viterbi algorithm
  - Estimate new parameter values as if the states were known to be the output of the Viterbi algorithm (ie supervised learning)
  - Repeat until convergence
- This is less principled than the Baum-Welch algorithm
- It does not maximize the likelihood $p(\boldsymbol{x}^{(T)}|\theta)$
- Instead, it finds the value of θ maximising $p(\boldsymbol{x}^{(T)}|\theta, \pi^*)$ ie the contribution to the likelihood from the most probable path π^*
- The Viterbi algorithm is faster than the Baum-Welch algorithm
- If the final aim is to produce good global decoding with the Viterbi algorithm, maybe it makes sense to train using it

Practical

- In `depmixS4`, working with a HMM is almost as easy as working with an independent mixture model!
- We use `depmix` instead of `mix` to build the model
- Fit as before using `fit`, this uses the EM algorithm
- Use this to fit the 2-states model to the quakes data

Baum-Welch algorithm on quake data

```
m=depmix(quakes~1,nstates=2,data=data,family=poisson())
m=fit(m,verbose=F)

## converged at iteration 26 with logLik: -341.8787
logLik(m)

## 'log Lik.' -341.8787 (df=5)
summary(m)

## Initial state probabilties model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.928 0.072
## fromS2 0.119 0.881
##
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1           2.736
## St2           3.259
```

Practical

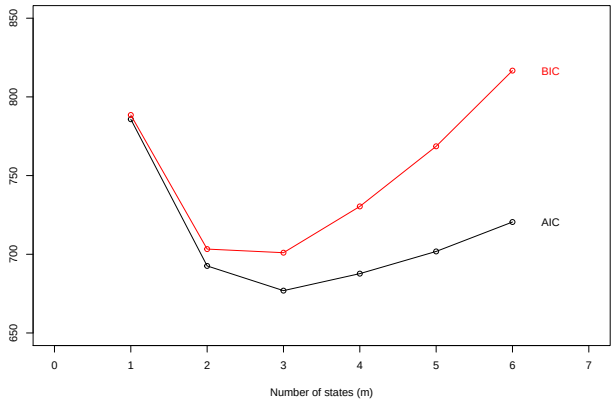
- Now it's time to combine together everything we've learnt today!
- Compare the HMM and IMM with different numbers of parameters on the quake data

Application to earthquakes data

- HMM = hidden Markov model ($p = m^2$)
- IMM = independent mixture ($p = 2m - 1$)

Model	m	p	logL	AIC	BIC
HMM	1	1	-391.9189	785.8	788.5
HMM	2	4	-342.3183	692.6	703.3
HMM	3	9	-329.4603	676.9	701.0
HMM	4	16	-327.8316	687.7	730.4
HMM	5	25	-325.9000	701.8	768.6
HMM	6	36	-324.2270	720.5	816.7
IMM	1	1	-391.9189	785.8	788.5
IMM	2	3	-360.3690	726.7	734.8
IMM	3	5	-356.8489	723.7	737.1
IMM	4	7	-356.7337	727.5	746.2

Application to earthquakes data



Application to earthquakes data

- Both AIC and BIC select the HMM with $m = 3$ states
- More generally, BIC and AIC do not always agree
- When $T > e^2$ (as is usually the case) the BIC penalizes larger models more than the AIC
- Independent mixture models are not as good as HMM for this dataset, despite the higher number of parameters in HMM

Model checking

- Even after selection of the best model, there remains the question of how good the model is in absolute terms
- Need to assess the goodness of fit of the model
- This is something commonly done for simpler model and that we need to adapt for HMM
- For example, in a simple linear regression model we have:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

  with $\epsilon_i \sim \mathrm{Norm}(0, \sigma^2)$
- The residuals $y_i - \beta_0 - \beta_1 x_i$ are therefore expected to be independently and identically distributed as $\mathrm{Norm}(0, \sigma^2)$ and this can be used to check the model
- For a HMM, what is the residual for each observation X_t?

HMM pseudo-residuals

- We consider that X_t is continuous (similar results can be derived for the discrete case)
- If X_t is from a distribution with cumulative density F_{X_t} then we can consider the pseudo-residual:

$$z_t = \Phi^{-1}(F_{X_t}(x_t))$$

where Φ is the cumulative density of a Normal(0,1)

- Since X_t has cumulative density F_{X_t} we have that $F_{X_t}(x_t)$ should be distributed as Unif(0,1) and therefore z_t should be distributed as Normal(0,1)
- What is the distribution of X_t?

HMM pseudo-residuals

- What is the distribution of X_t?
- One approach is to use the conditional distribution given all other data:

$$f_{X_t}(x) = p(X_t = x | \boldsymbol{X}^{(-t)} = \boldsymbol{x}^{(-t)})$$

- Using our previous calculation of the likelihood, we find:

$$f_{X_t}(x) = \frac{\boldsymbol{u}(1)\boldsymbol{P}(x_1)\boldsymbol{\Gamma}...\boldsymbol{P}(x_{t-1})\boldsymbol{\Gamma}\boldsymbol{P}(x)\boldsymbol{\Gamma}\boldsymbol{P}(x_{t+1})...\boldsymbol{\Gamma}\boldsymbol{P}(x_T)\boldsymbol{1}'}{\boldsymbol{u}(1)\boldsymbol{P}(x_1)\boldsymbol{\Gamma}...\boldsymbol{P}(x_{t-1})\boldsymbol{\Gamma}\boldsymbol{\Gamma}\boldsymbol{P}(x_{t+1})...\boldsymbol{\Gamma}\boldsymbol{P}(x_T)\boldsymbol{1}'}$$

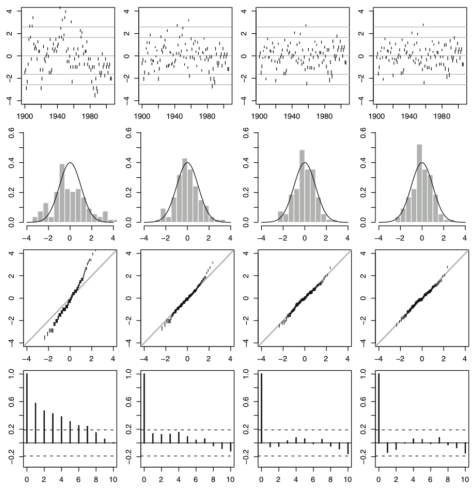
- Using the definitions of the forward and backward vectors:

$$f_{X_t}(x) \propto \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{P}(x)\beta_t'$$

HMM pseudo-residuals

- Use forward-backward algorithm to calculate vectors $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$
- Compute distribution of X_t given all other observations
- Compute pseudo-residual z_t for each x_t
- This can be used to detect outliers
- Can also be used to check validity of a model
- Plot distribution of pseudo-residuals vs Normal distribution
- Q-Q plot of observed (y-axis) vs expected (x-axis)
- ACF of pseudo-residuals

Model checking for earthquakes dataset



Observed vs expected ACF

- The observed ACF in the data can be compared to the ACF expected under the HMM
- $\text{Corr}(X_t, X_{t+k})$ can be computed analytically or simulated
- In the earthquake example, ACF of real data (bold) vs HMM with $m = 1, 2, 3$ states: