R / Bioconductor: A Short Course

James H. Bullard
Sandrine Dudoit
Division of Biostatistics, UC Berkeley
www.stat.berkeley.edu/~bullard
www.stat.berkeley.edu/~sandrine

Cuernevaca, Mexico
January 21-25, 2008

Hidden Markov Models and the Yeast Transcriptome

Introduction

In this lecture we are going to go through an extensive example. In this example we will build an R package from scratch which will group together all of the functionality which we will implement. We will implement an HMM to use on tiling array data. Along the way we will need to introduce/refresh various concepts.

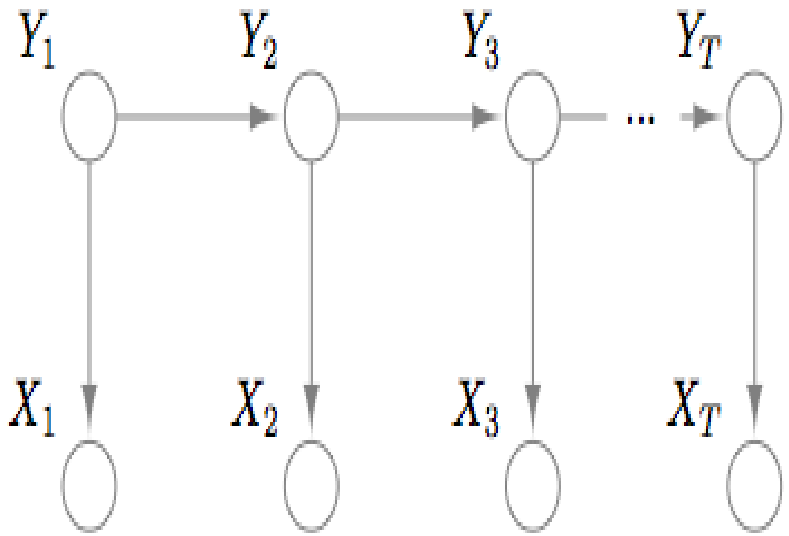
1. review Hidden Markov Models
2. overview of tiling microarray data
3. build data structures, simulate data
4. implement "forward" and "Viterbi" algorithms
5. build the functionality using an S4 class system
6. bundle things together into an R package

Hidden Markov Models

- A hidden Markov model is a "simple" probabilistic model for non-IID data.
- These models have been widely employed in Bioinformatics to detect regions of interest in sequence data. Examples include: gene finding, locating transcription factor binding sites, and detecting copy number variation.

The Model

Graphically we can depict an HMM as a backbone of connected nodes from which we connect a node which we observe.



- Here the Y_t are the hidden unobserved states and the X_t are the observed states.

The Model

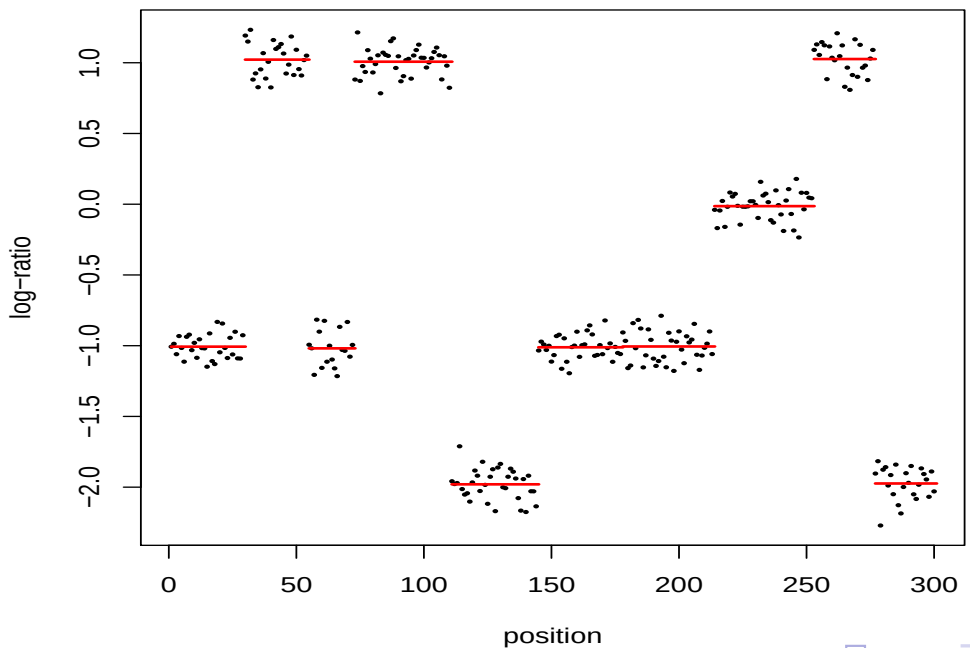
- For instance, the Y_t might be multinomially distributed with K possible states, and the X_t might be normally distributed or multinomially distributed as well - in gene finding applications typically all states are multinomially.

The Model

- In this setting there are three basic problems which we generally wish to solve (Rabiner 1989)
  1. How do we compute the probability of the data given the model (calculate the likelihood)?
  2. How do we choose a set of hidden states which is "optimal" for the observed sequence?
  3. How to we choose the model parameters to maximize the probability of the data?
- In bioinformatics we are very frequently interested in determining the most-likely hidden states given the observations.

The Data

In the figure below we display the simplest type of data which we might wish to use an HMM to model. This data clearly appears to be serially related. Furthermore, we might imagine that each chunk has a hidden underlying probability distribution which is known once we know which hidden state we are in.



The Parameterization

We need to be more precise about the particular components of our HMM. We are dealing with a discrete state homogenous Markov chain where we have K hidden states. We specify our transition matrix as A as below.

$$\mathbf{A} = \begin{pmatrix} a_{11} & \cdots & a_{1K} \\ \vdots & \ddots & \\ a_{K1} & & a_{KK} \end{pmatrix}$$

Our emission probability distribution can be any distribution we want[1], so we will specify it as $P(X_t|Y_t)$. Finally, we specify a prior distribution on the initial state as $\pi_1 \triangleq P(Y_1)$.

[1] This is not entirely true, we will concentrate exclusively on distributions in the exponential family.

Simulating a Dataset

Example

Simulating a Dataset

Now that we know the pieces of the puzzle it is time to start simulating data. For this exercise I want us to simulate a 3 state HMM with whatever transition matrix you like (although you should probably make all probabilities > 0), the emission distribution should be univariate normal (bivariate normal if you are feeling brave). From this exercise we want two functions: `makeModel` and `simulateData`. The function `makeModel` takes no arguments (an enhancement is to have it take K, or the number of states as an argument) and returns a model which for now we will implement as a list of lists. The function `simulateData` takes a model and a number N which returns a list containing two elements; the first element is the data, or the observed values from our HMM, the second element is the "true" hidden states which we know because we needed them to simulate the data. Finally, write a function which plots the data which you have generated.

Problem 1: Likelihood

If we imagine for a moment that we knew the hidden states of the chain as well as the observed states then we could easily calculate the likelihood (1).

$$P(x, y) = P(y_1) \prod_{t=1}^{T-1} P(y_{t+1}|y_t) \prod_{t=1}^{T} P(x_t|y_t) \quad (1)$$

The problem in calculating the likelihood however is that we do not observe the hidden states Y and thus we must sum over them as in equation (2).

$$P(x) = \sum_{y_1} \cdots \sum_{y_T} P(y_1) \prod_{t=1}^{T-1} P(y_{t+1}|y_t) \prod_{t=1}^{T} P(x_t|y_t) \quad (2)$$

Problem 1: Likelihood

This sum is order K^T which means we are not going to be able to evaluate this quantity directly. There is a lot of structure in the problem and we wish to exploit that structure to find a dynamic programming algorithm for computing the likelihood.[2]

[2] In what follows I have made heavy use of Rabiner (1989) and Michael I. Jordan's book in progress on Graphical Models.

The Forward Algorithm

- The forward algorithm is an efficient way to calculate the likelihood (2).
- The first thing we want to do is rewrite the probabilty so we condition on the observed states x (the likelihood).

$$P(y_t|x) = \frac{P(x|y_t)P(y_t)}{P(x)} \quad (3)$$

$$= \frac{P(x_1,\ldots,x_t|y_t)P(x_{t+1},\ldots,x_T|y_t)P(y_t)}{P(x)} \quad (4)$$

$$= \frac{P(x_1,\ldots,x_t,y_t)P(x_{t+1},\ldots,x_T|y_t)}{P(x)} \quad (5)$$

The Forward Algorithm

From this, we will define α and β which will define the forward and backward algorithms.

$$\alpha(y_t) = P(x_1, \ldots, x_t, y_t) \quad (6)$$

$$\beta(y_t) = P(x_{t+1}, \ldots, x_T | y_t) \quad (7)$$

Which leaves us with:

$$P(y_t|x) = \frac{\alpha(y_t)\beta(y_t)}{P(x)} \quad (8)$$

The αs are the probability of y_t being in a certain state and the probability of all the data up to that state. The βs are the probability of being in state y_t and then seeing all of the data after that. We see that if we sum over the possible values of y_T then we will have computed the likelihood which we wished

The Forward Algorithm

to compute. We have marginalized or summed out all of the hidden states!

$$\sum_{y_T} \alpha(y_T) = \sum_{Y_T} P(x_1, \ldots, x_T, y_T) \quad (9)$$

$$= P(x_1, \ldots, x_T) \quad (10)$$

The trick is to figure out a way to recursively calculate α. We simply start by writing down $\alpha(y_{t+1})$ and see if we can find a recurrence relation.

The Forward Algorithm

$$
\begin{aligned}
\alpha(y_{t+1}) &= P(x_1, \ldots, x_{t+1}, y_{t+1}) && (11)\\
&= P(x_1, \ldots, x_t | y_{t+1}) P(x_{t+1} | y_{t+1}) P(y_{t+1}) && (12)\\
&= \sum_{y_t} P(x_1, \ldots, x_t, y_t, y_{t+1}) P(x_{t+1} | y_{t+1}) && (13)\\
&= \sum_{y_t} P(x_1, \ldots, x_t | y_t) P(y_t) P(y_{t+1} | y_t) P(x_{t+1} | y_{t+1})\\
&= \sum_{y_t} P(x_1, \ldots, x_t, y_t) P(y_{t+1} | y_t) P(x_{t+1} | y_{t+1}) && (14)\\
&= \sum_{y_t} \alpha(y_t) P(y_{t+1} | y_t) P(x_{t+1} | y_{t+1}) && (15)
\end{aligned}
$$

The Forward Algorithm

We should see all the important pieces in the last line. We have α which will be computed recursively. We have the transition probabilities given by: $P(y_{t+1}|y_t)$. Finally, we have our emission probabilities with: $P(x_{t+1}|y_{t+1})$.

- The only thing left to determine is how to initialize the α recursion. It stands to reason that the prior probability term will have to play a role akin to the transition probability and the emission probability will remain unchanged.

$$\alpha(y_1) = \pi_1 P(x_1|y_1) \tag{16}$$

Problem 2: The Viterbi Algorithm

- Often the real reason we fit an HMM is to uncover the hidden states of the model. When we uncover the hidden states we have to realize that we are only doing so "probabilistically" - that is, at each data point we will have a probability distribution over the hidden states. At certain data points that could be something like: $(.99, 0, 0, .01)$ which would make our problem of choosing the maximizing state pretty easy, however it could also be the case that our probability distribution looks like: $(.25, .25, .25, .25)$.
- The Viterbi algorithm is an algorithm for finding the single best "path" or configuration of the hidden states for the data.
- The algorithm is essentially the forward algorithm with two major changes:

Problem 2: The Viterbi Algorithm

1. Summation is replaced by maximization.
2. We add a backtracking step to recover the maximizing configuration.

- Formally, the Viterbi algorithm seeks to solve equation (17).

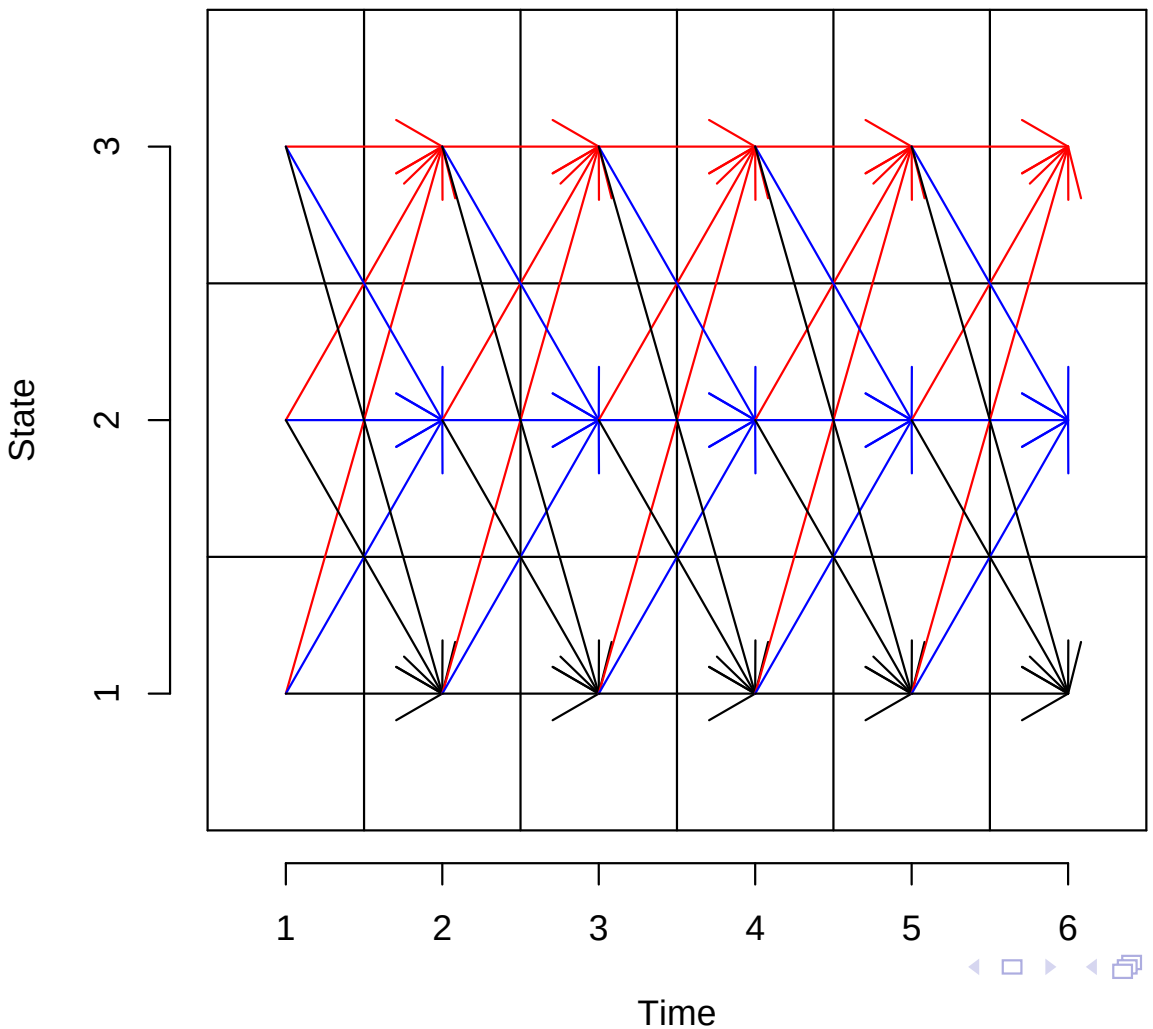
$$\arg\max_{y_1,\ldots,y_T} P(y_1,\ldots,y_T|x_1,\ldots,x_T) \tag{17}$$

x is fixed. We find the configuration of ys which produces the largest probability of seeing the data.

Visualizing the Algorithms

- It is often helpful to look at how the computations of both the forward and Viterbi algorithms proceed. These involve a dynamic programming matrix and a recursive algorithm.
- In the image below each cell represents the probabilty of being in a particular state at time t. We simply have to account for the possibility of coming from any of the other previous states - that is what makes the algoritm quadratic in the number of states.

Visualizing the Algorithms



Implementing The Forward and Viterbi Algorithms

Example

Before we get our hands dirty with real data we want to implement these two algorithms for use on our simulated data set. The first thing to do is start with the forward algorithm and then once you have that under control implement the Viterbi. The first questions we need to ask ourselves are: 1) what parameters does our `forward` function take? 2) what does it return. Along the way we can work out what need to go on inside. How will you test the forward algorithm? Do we have to deal with numerical issues?

Problem 3: EM

- Thus far we have been noticeably quite about estimating the parameters.
- The parameters of our HMM are the transition matrix: a, the prior: π_1, and the parameters of our emission distribution. For concreteness, lets say that our emission distribution is a normal distribution indexed by k; that is we have k distinct normal distributions with parameters: (μ_k, σ_k). If we have 3 hidden states then we have: $3 \times 3 + 3 \times 2 + 1 = 16$ parameters to estimate.
- The third problem and very much the most difficult of all three is to estimate these parameters given a sequence of observations.

Problem 3: EM

- We can see that properly determining the hidden states is dependent on choosing the "right" parameters. When we have data we can generally decide on the emission distribution's form, but determining the right parameter estimates is much more difficult. Again, if we knew the hidden states we could figure this out pretty easily - and once we have the hidden states then figuring out good parameter estimates is pretty easy. That is essentially what the Expectation Maximization algorithm does.
- The EM algorithm essentially does just that. First, we choose some estimates for the parameters. We then take the expected values of the hidden states: y. Then we repeat until convergence.

Problem 3: EM

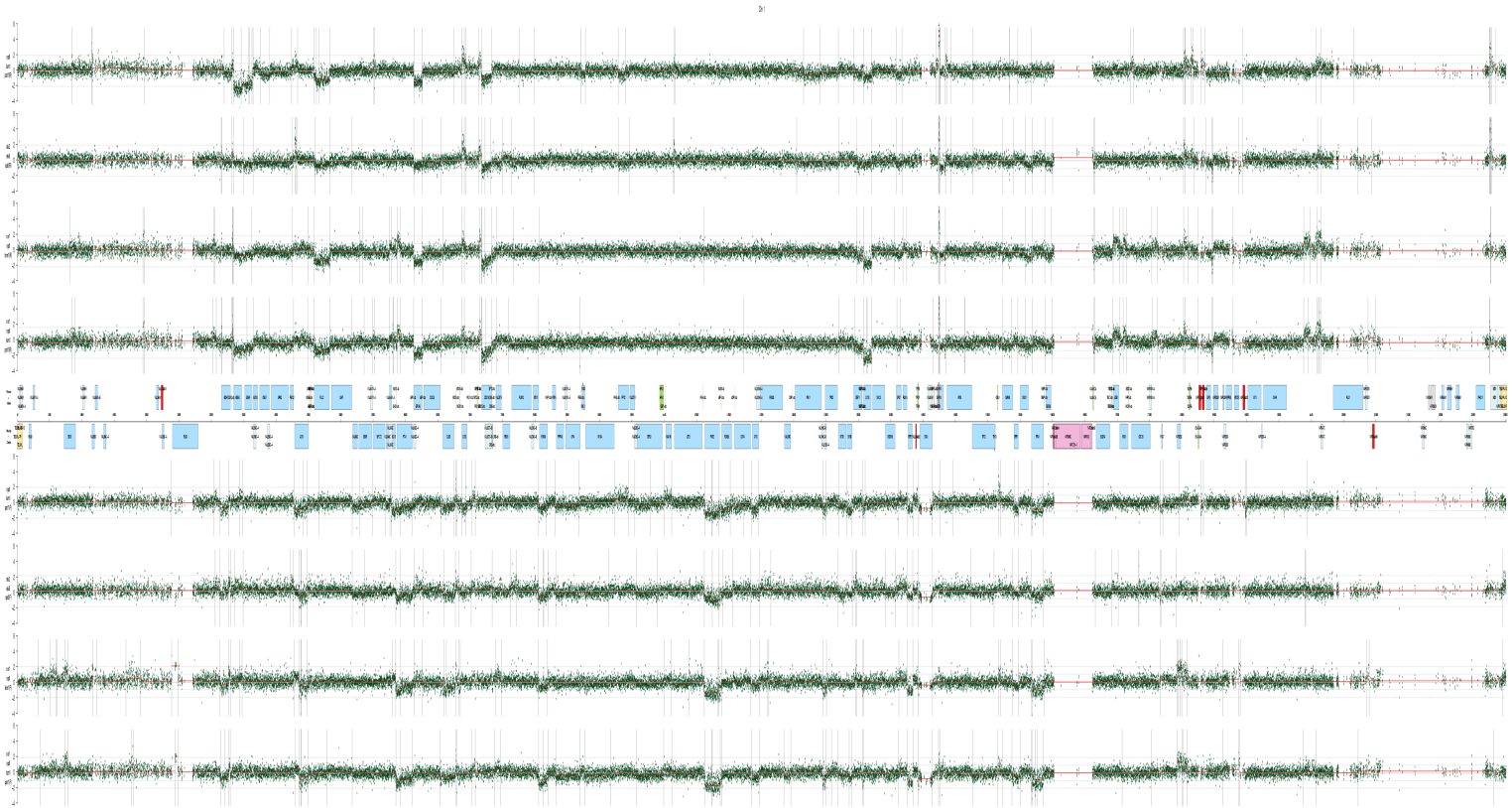
- The EM algorithm is more complicated than the previous two, it is detailed in Rabiner (1989). The issue is that we have to specify the emission distribution in order to complete the algorithm.

GeneChip S. cerevisiae Tiling 1.0R Array

The Affymetrix tiling array was designed to completely tile a genome. This tiling allows for precise determination of features of both genomic DNA as well as transcription activity.

- Comprehensive S. Cervisae Microarray, average spacing between probes 5bp, 5 μm features
- 25 base oligonucleotide probes PM/MM
- Forward and Reverse Strand arrays
- $2560 \times 2560 = 6553600$ features 1/2 perfect match.

The data will be partitioned into linear segments corresponding to the 17 chromosomes. The goal is to identify transcripts, ie. find regions of upregulation



Obtaining the Data

- The data has been prepared in the directory: data/hmm_tiling. The data has been divided into a separate file for each chromosome each file has four columns corresponding to a separate chip. These are the raw intensity values sorted in chromosomal location.
- The First thing we need to do is understand the data a little, for this we return to the exploratory data analysis.

EDA

Example

In order to familiarize ourselves with the data from this class of experiment.

1. Compute summary statistics on each of the chromosomes and chips
2. What kind of transformations can we make to the data?
3. Is the raw data normal? What distribution is the transformed data?
4. Are the chips more alike or are the chromosomes more alike? (boxplots)

An Example


```
> chr1 <- read.table("../../data/hmm_tiling/chr-1.dta
> chr1.l2 <- log2(chr1)
> mm <- makeModel(mus = c(10.5, 12),
+     sigmas = c(1, 1.5), A = matrix(c(0.99,
+         0.01, 0.15, 0.85), nrow = 2,
+         byrow = T))
> aa = Viterbi(mm, chr1.l2[4000:7000,
+     1])
> plot(chr1.l2[4000:7000, 1], pch = 16,
+     cex = 0.2)
> points(ifelse(aa == 1, 10.5, 14),
+     col = "red")
```

Packages

- R packages provide support for adding functionality into the core R system.
- Much of the low-level tools which we have seen are actually implemented as packages `sessionInfo()`
- Packages allow for an excellent distribution mechanism. Additionally, packages provide means to bundle both data sets and functions

Package Contents

A package consists of the following directories and files

1. *DESCRIPTION* : A file describing the contents and version of the package
2. *NAMESPACE* : A description of what R and C code will be available to the loaded package
3. *R* : `R` source code which will be available when the package is installed depending on the NAMESPACE file
4. *src* : A directory containing C or C++ source code which can be called directly from the R code in the package
5. *man* : A directory containing the help files for a given package, these can be easily created using `prompt()`
6. *data* A directory containing data sets available once the package is loaded

Package Contents

In addition to the directories listed above packages can also contain the following directories: *demo*, *exec*, *inst*, *po*, and *tests*.
In addition the following files can be in the top level directory: *INDEX*, *configure*, *cleanup*, *LICENSE*, *LICENCE*, and *COPYING*.
These directories and files are less often used but sometimes important, to make a simple package we need only *DESCRIPTION* and *R*

DESCRIPTION


```
Package: HMM
Title: An HMM for a K state model
Version: 1.0
Author: James Bullard
Depends: R (>= 2.6.1)
Description: A package for the HMM parsing
             for tiling array data
Maintainer: <bullard@stat.berkeley.edu>
License: LGPL
```

The exact rules for this file can be found at: DESCRIPTION

Making a Simple Package

Example

We now want to build and install a simple R package which will contain our R code for generating simulated data sets as well as our plotting and fitting functions (viterbi, forward) functions.

1. Construct a simple package
2. Install the package using R CMD INSTALL to install the package
3. Test that we can access our functions and data sets using require or library
4. use prompt to create a helpfile for one of the functions which we have created

Whats Next

Now that we have fit an HMM to our data. We have a package. There are various enhancements which we can make:

- Make better plots, including residual plots. Residual plots might help us uncover where our model is deficient.
- We can consider changing the emission distribution or the number of states.
- We can attempt to remove outliers to make our analysis more robust.
- We can compare what we have found to known areas of annotation using BiomaRt.
- Make an S4 class so that we can generalize some of the thing which we have done.