Andrew Yang  
Tutorial of the fm models

## Installation

* To install the package on a windows computer, put tutorial.R in a directory with the tar.gz and zip file. Right now, the package does not work on non-Windows computers. Run an R terminal and set the working directory to the one with the files. Then run the following commands. The tutorial.R file contains all the R commands used here.

### setwd('<INSERT DIRECTORY>')

### install.packages('fadermodels\_1.0.zip',contriburl=NULL)

### library('fadermodels')

* After doing this, it should say, package 'fadermodels' successfully unpacked and MD5 sums checked.
* Many of the functions, particularly the integrated models, require you to also have gsl installed. Thus, you should also run the following.

### install.packages('gsl')

### library('gsl')

## Introduction

* We can start with something simple. A simple count model. data.count contains a sample dataset. We first load it
  + data(data.count,package='fadermodels')
* The first column is occurrences and the second column is the number of people who had that number of occurrences. We can model this data using a 1 segment Poisson model. To do this, we call the function fm.

### mod=fm(data.count,'pois')

### mod

* The result is a fm object. We can ignore the warnings for now[[1]](#footnote-1). We can now run analysis on this model. Below are some statistics we can perform:

### mean(mod) #Mean of the Poisson fit

### vcov(mod) #Variance of the Poisson fit

* Next, we can look at the fit of the model. Below are some commands we can use.

### predict(mod) #Predicts the number of people at each occurrence

### resid(mod) #finds the residuals of the prediction

### sse(mod) #finds the sse of the prediction

### rmse(mod) #finds the rmse of the prediction

* A visual plot of the prediction vs. the actual can be seen with a barplot call.

### barplot(mod)

* We can perform a chi squared goodness of fit test on the data with the command:

### chitest(mod)

* The warning here comes from the fact that some bins have low count values, which technically violates the condition of a chi squared goodness of fit test. From our tests, we can see that this probably isn't the best model to use.

## Latent Class Models

* Let's fit latent class models on the data.

### mod2=fm(data.count,'pois',nseg=2)

### mod3=fm(data.count,'pois',nseg=3)

### mod4=fm(data.count,'pois',nseg=4,tries=5)

### mod5=fm(data.count,'pois',nseg=5,tries=5)

### mod6=fm(data.count,'pois',nseg=6,tries=5)

* The tries parameter in the fm function will allow you to start and multiple different starting points in an attempt to find the best fit. It will then pick the model with the highest likelihood. For models with multiple parameters, it is probably better to try from many different starting points.
* Now that we have many different models to choose from, which one should we choose? We can run a likelihood ratio test.

### lltest(mod,mod2)

### lltest(mod2,mod3)

### lltest(mod3,mod4)

### lltest(mod4,mod5)

### lltest(mod5,mod6)

* This test will look at the ll of each model and compare it to a chi squared distribution. The null hypothesis is that the models are identical and a low p value means that the alternative model is probably an improvement which didn't come about from a chance distribution of data. The first argument must be the null model and it must be nested within the second argument. As an alternative to the ll tests, we can look at the Bayesian information criterion to pick the best model.

### best=bictest(mod,mod2,mod3,mod4,mod5,mod6)

* We can look at how well this model fits the data:

### barplot(best)

### chitest(best)

### rmse(best)

* We see that it does a much better job.

## Spikes and Posterior Probabilities

* Now, let's add a spike to the 2 segment model.

### mod2s=fm(data.count,'pois',nseg=2,spike=TRUE)

* We can run the same tests as before.

### lltest(mod2,mod2s)

### barplot(mod2s)

* The spike does improve the model. If you give me a person, what is the probability that they are in a certain segment? We can find out with a Bayesian posterior update. This is done with the command post.

### post(mod2s)

## Mixing Distributions

* Now, we can put a gamma mixing distribution in and the model turns into an nbd model. So we call it and run the same diagnostics.

### mixmod=fm(data.count,'nbd')

### mixmod

### barplot(mixmod)

### mean(mixmod)

### chitest(mixmod)

## Analyzing models

* Of course, we can try multiple segments as well.

### mixmod2=fm(data.count,'nbd',nseg=2,spike=T)

### mixmod2

* We pick the best model out of the ones we have seen so far.

### best=bictest(mod,mod2,mod3,mod4,mod5,mod6,mod2s,mixmod,mixmod2)

### best

* Now that we have a mixture model, it may be beneficial to look at the underlying mixing distribution. We can see that with the following plot:

### paramplot(mixmod)

* Once again, this works with any model we have trained so far.

### paramplot(mod2s)

## Changing parameters

* Changing parameters can be done with a simple call to the update function:

### newmod=update(mixmod,r=1,alpha=0.1)

### newmod

### barplot(newmod)

### paramplot(newmod)

## Other features

* Count models have an additional added feature. We can change the total amount of time that has elapsed to see how many occurrences a person will have in a specific amount of time. When we train the data, if we don't specify a time t, 1 is used by default. Thus, if we would like to predict the number of occurrences made in 2 or 1/2 of these time periods, we can with the following commands.

### predict(best,t=2)

### predict(best,t=1/2)

* You can also change the number of occurrences for the prediction

### predict(best,t=3,x=0:50)

## Dirichlet Model

* The following shows an example with the dirichlet model.
  + data(data.dir,package='fadermodels')

### dir=fm(data.dir,'dir',tries=10)

### barplot(dir)

### mean(dir)

* All the above discussed functions work on the dir model as well. Furthermore, there are additional functions written specifically for the dirichlet.
  + pen.dir(dir) #Penetration level
  + loyal.dir(dir) #Probability that a customer will be 100% loyal in your segment
  + once.dir(dir) #Probability that a customer will be a one time customer in your segment
  + mkt.dir(dir) #Market share of each segment
  + scr.dir(dir) #Share of category requirements
* Additional methods such as no.ind.dir, freq.ind.dir, pen.ind.dir, loyal.ind.dir, once.ind.dir, and exp.ind.dir give individual level statistics for the dirichlet.
* Unfortunately, I noticed that these models have some problems converging at times. Thus, we can update it with the best parameters I found with much trial.

### dir=update(dir,x1=.195,x2=.054,x3=0.06,x4=0.116,x5=.362,x6=.139,x7=.151,x8=.175)

* The dirichlet model does allow for multiple segments too. Each function shown above can be called for two segment dirichlet models, as well as the post function.

## Integrated Models

* The following shows an example for integrated models. As a warning, these models have not been optimized and can be quite slow. The 2 segment models are especially slow and it may be wise to skip them.
  + data(data.conint,package='fadermodels')
  + data(data.disint,package='fadermodels')

### a=fm(data.conint,'pexp',1)

### b=fm(data.conint,'pexp',2,tries=5)

### c=fm(data.conint,'bgnbd',1)

### d=fm(data.conint,'bgnbd',2,tries=5)

### e=fm(data.conint,'pnbd',1)

### f=fm(data.conint,'pnbd',2,tries=5)

### g=fm(data.disint,'bgbb',1)

### h=fm(data.disint,'bgbb',2,tries=5)

### par(mfrow=c(2,4))

### c1=cumtracking(a)

### c2=cumtracking(b)

### c3=cumtracking(c)

### c4=cumtracking(d)

### c5=cumtracking(e)

### c6=cumtracking(f)

### c7=cumtracking(g)

### c8=cumtracking(h)

### #cumulative tracking plots

### plot(c1)

### plot(c2)

### plot(c3)

### plot(c4)

### plot(c5)

### plot(c6)

### plot(c7)

### plot(c8)

### #weekly tracking plots

### plot(diff(c1))

### plot(diff(c2))

### plot(diff(c3))

### plot(diff(c4))

### plot(diff(c5))

### plot(diff(c6))

### plot(diff(c7))

### plot(diff(c8))

### #Model and actual purchases

### barplot(a)

### barplot(b)

### barplot(c)

### barplot(d)

### barplot(e)

### barplot(f)

### barplot(g)

### barplot(h)

* Warnings from the cumtracking command are expected and is discussed in the notes.
* If you see an error such as "Problem finding a good starting point," then the model tried 5 different random starting points and each one was NaN. You can try it again. It is possible that by chance, each of the 5 starting points were very poor.
* Warning messages such as NaNs produced during the optimization could be problematic. However, most of the time, they are not. This means that the optimization algorithm happened on a set of parameters which generated NaNs while looking for an optimal point. The algorithm will then likely move away from this point, which means that the NaNs will not be a problem.

## Discrete Time Models

* The following shows an example of a discrete time model. It is very similar to a counting model. However, it requires the submission of a pop parameter for the total population. It also requires the marginal churn rate and not the cumulative values.
  + data(data.distime,package='fadermodels')

### geommod=fm(data.distime,'geom',pop=1000)

### geommod2=fm(data.distime,'geom',nseg=2,pop=1000)

### bgmod=fm(data.distime,'bg',pop=1000)

### dwmod=fm(data.distime,'dw',pop=1000)

### bdwmod=fm(data.distime,'bdw',pop=1000)

### post(geommod2)

### best=bictest(geommod,geommod2,bgmod,dwmod,bdwmod)

### best

### chitest(best)

### barplot(best)

### rmse(best)

### paramplot(best)

* There is one twist to a discrete time model. The last element of a predict call will always be the probability of being greater than the final value.
  + predict(geommod,x=3:15)
* We see that this has 14 values. The last one is the probability that a person will churn after 15

## Continuous Time Models

* The following shows an example of using continuous time models. They are very similar to the discrete time model discussed above.
  + data(data.contime,package='fadermodels')

### expmod=fm(data.contime,'exp',pop=1499)

### egmod=fm(data.contime,'eg',pop=1499)

### wgmod=fm(data.contime,'wg',pop=1499)

### ggmod=fm(data.contime,'gg',pop=1499)

### best=bictest(expmod,egmod,wgmod,ggmod)

### best

### data.frame(data.contime,mod=predict(best)[-length(predict(best))])

### paramplot(best)

### paramplot(best,bounds=c(0,1),main='Distribution of Mixing')

* The paramplot accepts a parameter called bounds to plot it on different values of x. It also accepts any parameter which you can submit to a default plot call.
* There is also one difference between a continuous time model and a discrete time model, which can be seen in a predict call.
  + predict(egmod,x=c(0,3:15))
* The first element here is the number who churn between time 0 and 3. The second is between 4 and 5. The penultimate is between 14 and 15 and the last is after time 15. In a discrete time model, the first element is just the number who churn after time 0.

## Choice Models

* The following shows an example of a choice model. We see that the best model here is a one segment model with no spike.
  + data(data.choice,package='fadermodels')
  + data(data.choice2,package='fadermodels')

### bbmod=fm(data.choice,'bb')

### bbmods=fm(data.choice,'bb',spike=T)

### bbmod2=fm(data.choice,'bb',nseg=2)

### lltest(bbmod,bbmods)

### lltest(bbmod,bbmod2)

### best=bictest(bbmod,bbmods,bbmod2)

### paramplot(bbmod2)

* Also, calling paramplot on a model with multiple segments will automatically weight the two mixtures together by their probabilities.
* There is another test dataset one can play around with: data.choice2.

## Notes:

* Not all models allow for spikes yet.
* I currently have to calculate likelihood values when I find the best parameter set for latent class models. This means that there could be underflow or overflow errors caused by floating number precision.
* The cumtracking plot for the integrated models pexp and pnbd dont seem to connect at the intersection between the holdout and the end of the trial period. I cannot figure out why. I have a warning in place for now.
* The mean and variance for several models haven't been coded up and I haven't tested them for others. Some of them are negative for some reason.
* The bb model currently doesn't allow for different sizes for m, which I would like to change.
* I still have to write a tutorial on how to program these models. I tried my best to make these models as easily extendable as possible so others could build off of it.
* I would eventually like to get this package on CRAN and help other students in your class use it for their projects.
* Let me know if you find any other bugs or have any other questions or suggestions.

1. I currently use a Nelder-Mead optimization, a call to optim, for each model. This doesn't work well for 1 parameter models, which is what the warning is telling you. [↑](#footnote-ref-1)