Experimental Model

Eric Bridgeford

August 26, 2016

Basis for Model

The basis for the experimental model is Expectation-Maximization of the Gaussian Mixture Model (GMM). Here, we will detail several experiments that show that our model works through data-driven simulations, demonstrate the adjustments to GMM that we make, show that they work for simulation data, and finally investigate how the experimental data fits the model.

Basic Gaussian Mixture Model

The basic Gaussian Mixture Model hinges on Expectation-Maximization, a process for finding a locally maximizing fit for mixed distributions of the exponential family. GMM has 2 steps, the expectation (or responsibility) step, and the maximization step, and repeats until our log likelihood converges.

Experimental Model

For our experiment, we have task fMRI data in which test subjects have a task presented before them at known points in time. We theorize that the latent signals driving thinking patterns can be represented by a gaussian mixture, and that we can adjust EM to predict the covariance matrices of each experimental task condition. We define our density function of an observed x of some latent signal z:

$$p(x_i) = \sum_c \pi_{ic} N(x_i|\mu_c, \Sigma_c) \tag{1}$$

Using our predicted covariance matrices for each experimental task condition, we can then use the Wilcoxon signed-rank test to find rois that show differential activity based on task condition.

GMM Framework

E Step

Like traditional GMM, our expectation step consists of computing the responsibilities of each potential cluster on our individual data points.

$$r_{ic} = \frac{\pi_{ic} N(x|\mu_c, \Sigma_c)}{\sum_{c'} \pi_{ic'} N(x|\mu_{c'}, \Sigma_{c'})} \tag{2}$$

where r_ic represents the given responsibility of a cluster c on data point i. This is a relatively intuitive process, as we are essentially considering the probability that a given data point is a member of a cluster and normalizing by the probability that it is in any of the clusters.

M Step

Again, much of our process here will look very similar to traditional GMM. As we have information about how the task conditions evolve over time, however, we can skip prediction of the π vector entirely. For our π vector, we can convolve a spike train f of condition onsets with a response function that gives the expected proportion of signal at each successive timestep after an impulse, such that:

$$\pi_{ic} = f_c(i) * h(i) \tag{3}$$

We normalize our π vector to sum to 1 at each time point to coincide with the π vector predicted by the M-step of GMM.

The cumulative responsibility of a given cluster c

$$m_c = \sum_i r_{ic} \tag{4}$$

For updated means. The logic behind the mean maximization is that we consider how much a particular cluster "explains" our ith data point, so μ is more strongly impacted by data points that it explains more effectively.

$$\hat{\mu_c} = \frac{1}{m_c} \sum_i r_{ic} x_i \tag{5}$$

For updating the covariances. The logi here is again the same; if a point is explained well by a particular distribution, its covariance is more strongly considered than if it isn't explained by that distribution.

$$\hat{\Sigma_c} = \frac{1}{m_c} \sum_i r_{ic} (x_i - \hat{\mu_c})^T (x_i - \hat{\mu_c}) \tag{6}$$

We repeat the expectation and maximization steps until we reach convergence of the likelihood (and consequently the log likelihood) of our predicted clusters matching our data.

$$log(p(X)) = \sum_i log[\sum_c \pi_{ic} N(x_i | \mu_c, \Sigma_c)] \tag{7}$$

Wilcoxon Signed-Rank Test

Here, the data is clearly paired, as we would expect a relationship between correlation values taken at the within-subject level, and the within-scan level. We do not want to assume a Gaussian distribution for the correlation values being analyzed, so we proceed with the non-parametric wilcoxon signed-rank test.

Experiments

For our experiments, we want to show that when we linearly combine draws from a distribution for each condition, that we can predict the distributions using the known contributions from each condition at each time step. Our simple simulation can be summarized as follows:

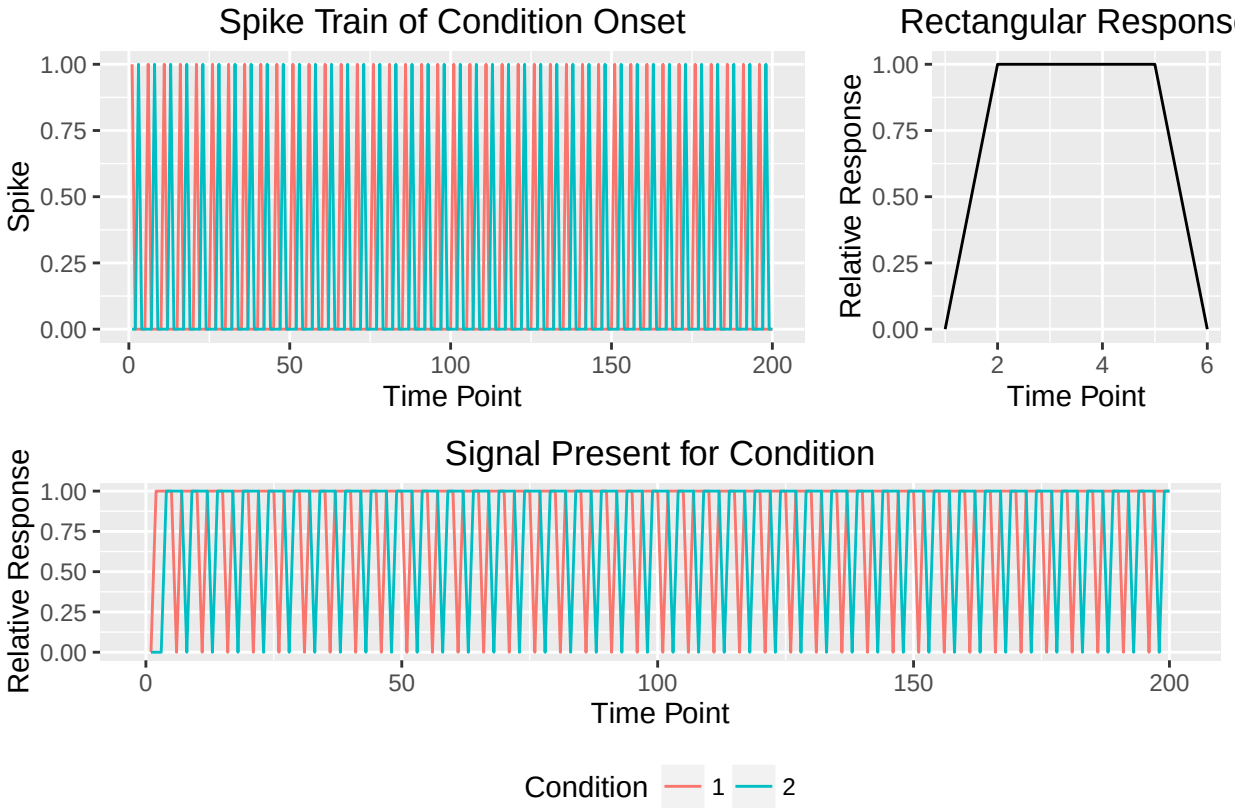
$$x_i = \sum_c \pi_{ic} * y_c \tag{8}$$

for some $y_c \in Y_c$ such that $Y_c \overset{iid}{\sim} N(\mu_c, \Sigma_c)$.

Simple Simulation 1

For our first experiment, we consider a simple $|n| = 15$ roi model, with $|c| = 2$ task conditions for this experiment, and $|i| = 200$ timesteps. We use a simple rectangular window for our expected signal lag vector.

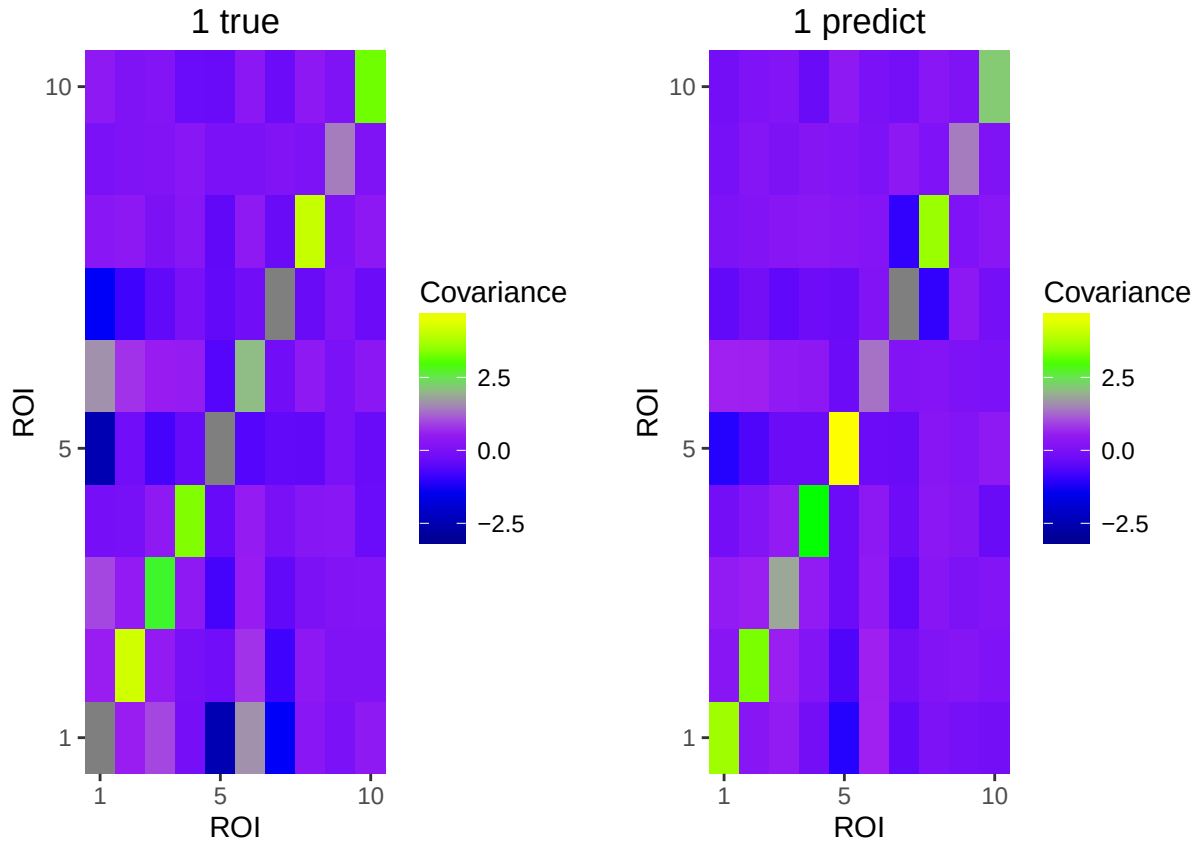
```
t=200
conditions <- array(0, dim=c(t, 2)) # initialize empty condition vector
conditions[seq(1, t, 5), 1] <- 1 # set the spikes for condition onsets for each condition
conditions[seq(3, t, 5), 2] <- 1
fx <- array(rev(c(0, 1, 1, 1, 1, 0))) # define a simple square window
source('../drivers/convolve_trial_onsets.R')
response <- convolve_conditions(conditions, fx)
```

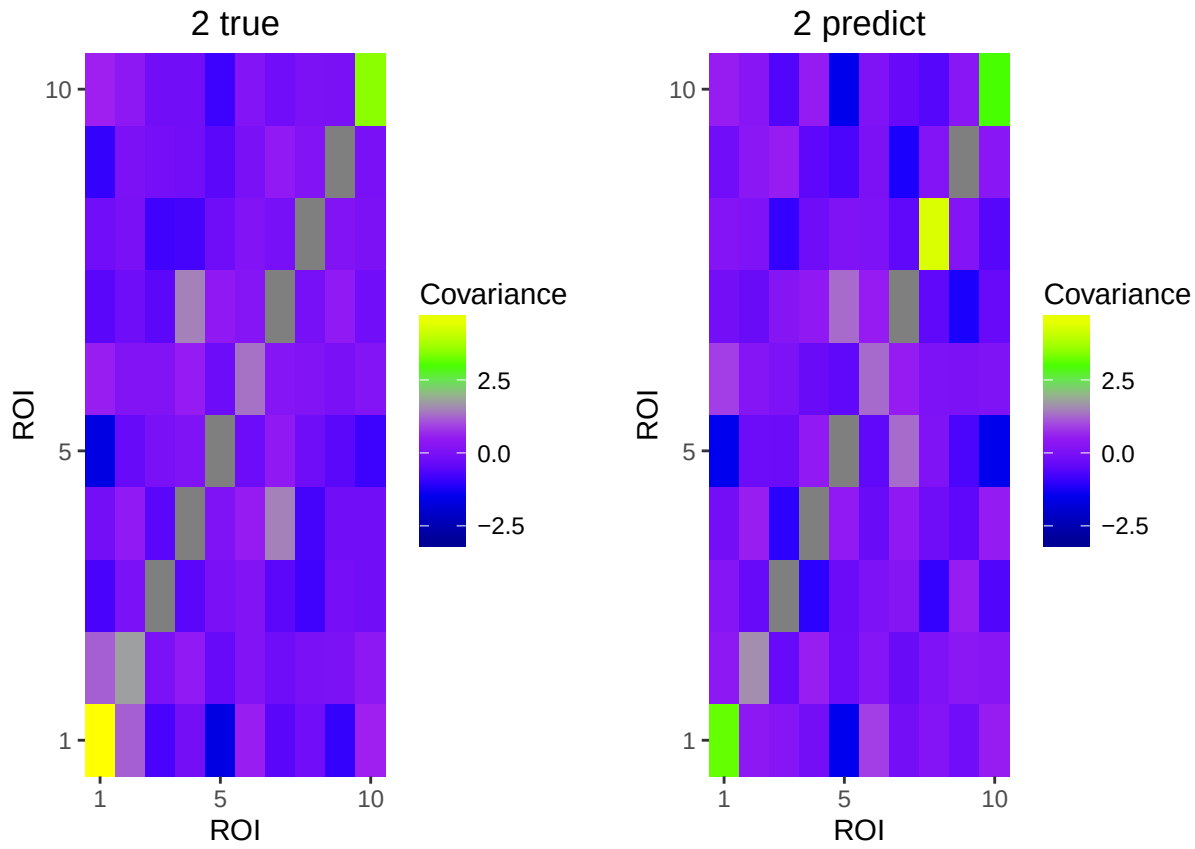


Below, we create 2 random distributions, one for each condition, and linearly combine draws from this distribution for each time step based upon our pi vector.

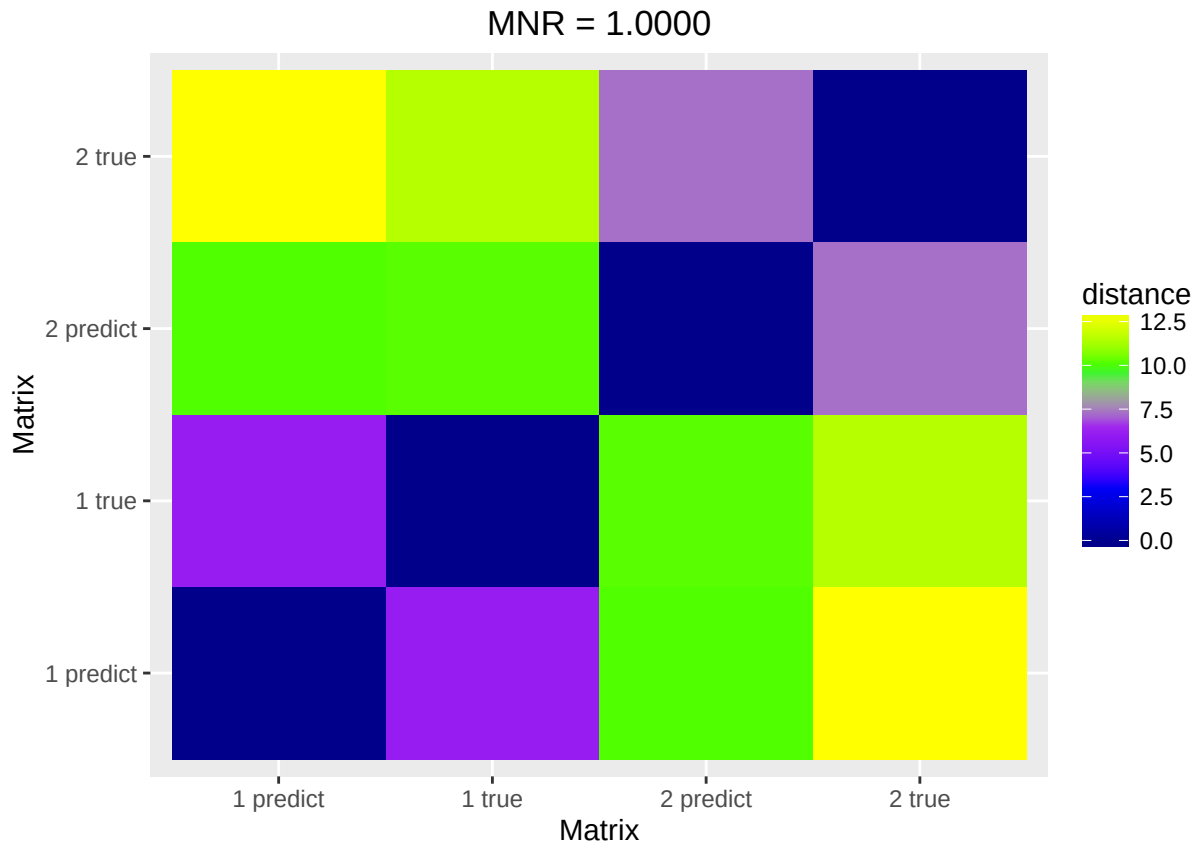
```
pi_vec <- t(apply(response, 1, function(x) x/sum(x))) # normalize the response
require('MASS')
require('clusterGeneration')
nroi=10
latent_signal <- array(0, dim=c(dim(conditions)[2], t, nroi))
for (i in 1:dim(conditions)[2]) {
  latent_signal[i,,] <- mvrnorm(n=t, mu=rnorm(nroi), Sigma=genPositiveDefMat(nroi)$Sigma) # use rando
}
# define the observed signal as a combination of the latent signals, where the contribution
# of each is the pi vector at that time step
observed_signal <- t(sapply(1:dim(latent_signal)[2], function(t) {
```

```
  apply(pi_vec[t,]*latent_signal[,t,], 2, sum)
  }))
pi_vec <- pi_vec[complete.cases(observed_signal),]
observed_signal <- observed_signal[complete.cases(observed_signal),]
true_means <- apply(latent_signal, c(1,3), mean)
true_cov <- sapply(1:dim(latent_signal)[1], function(x) cov(latent_signal[x,,]), simplify='array')
source('gmm_known_pi.R')
params <- gmm_known_pi(observed_signal, true_means, true_cov, pi_vec)
require(abind)
covs <- abind(true_cov, params$covs, along=3)
```






```
D <- distance(covs)
mnr <- mnr(rdf(D, trials))
```



As we can see, even though the predictions are not perfect, we could correctly distinguish which prediction covariance matrices fit best with their correpsonding truth covariance matrices.