Project 2
Statistical Neuroscience

Erick Cobos
École Polytechnique Fédérale de Lausanne
erick.cobos@epfl.ch

Eleftherios Zisis
École Polytechnique Fédérale de Lausanne
eleftherios.zisis@epfl.ch

June 03, 2014

In this project we use information measures and Gaussian Mixture Models to analyze the response of neurons in the motor cortex of a monkey during reaching tasks. The data was kindly provided by Prof. J. M. Carmena, University of California, Berkeley.

1 Firing Rate Entropies and Information Rates

(a)
Entropies were calculated with the script provided. In Figure 1 the entropy for all neurons is plotted versus the neuron indices for three values of N. We can observe that as the number of N increases, the entropy estimate increases sub-linearly.

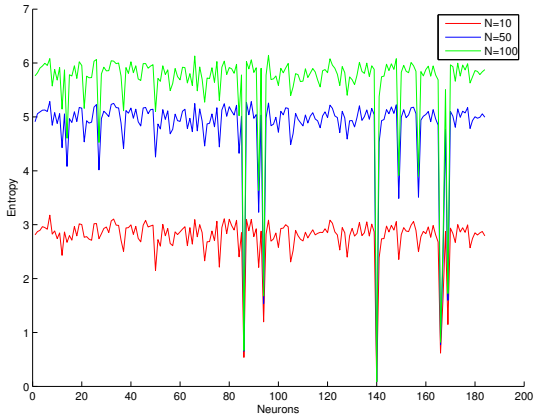


Figure 1: Plot of the entropy value for all neurons and for different bin numbers N

This can be further visualized in the following plots where a 3D plot of the neurons vs. entropy vs. N is created, along with the entropy averaged over the neurons. It is apparent that entropy is highly dependent on the number of bins, a fact which renders histograms not a good candidate for entropy estimation if the number of data is not fairly big.

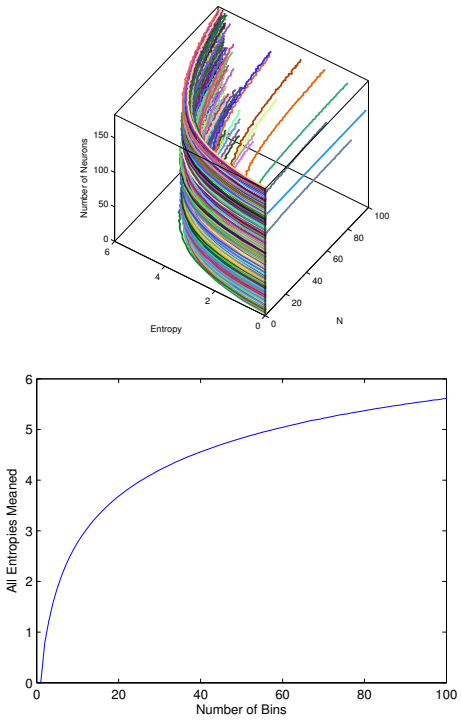


Figure 2: Left: Entropy of each neuron as a function of the number of bins. Right: Mean value of the entropy as a function of bin number N

(b)
The function for the calculation of the joint probability for a neuron pair is as follows:

```
% Calculates the 2D histogram of two vectors
function  H  = histogram2D(X,Y)

  % Round all the rates of neuron X and neuron Y
  X = round(X); Y = round(Y);

  numTrials = size(X,1);  % Number of trials. Pressumably 208

  % Store the max rates plus one of both neuron vectors
  max_N1 = max(X)+1; max_N2 = max(Y)+1;
  min_N1 = min(X); min_N2 = min(Y);

  % Create two arrays of rates with a step of 1 rate
  N1_range = min_N1:max_N1; N2_range = min_N2:max_N2;
```

```
  lenN1 = length(N1_range); lenN2 = length(N2_range);
  index_i = 1:lenN1; index_j = 1:lenN2;

  % Create all the combinations of rate pair indices
  [p,q] = meshgrid(index_i, index_j);
  inds = [p(:) q(:)];

  % Calculate their length
  Lpairs = size(inds,1);

  % Initialize the histogram
  H = zeros(lenN1,lenN2);

  for i = 1:Lpairs % For all the combination indices pairs
    % Parse all the X,Y pairs
    for j=1:numTrials
      % If the rates with the ith respective indices is found in X,Y
      % add one count to the histogram
      if (N1_range(inds(i,1)) == X(j) && N2_range(inds(i,2))==Y(j))
        H(inds(i,1),inds(i,2)) = H(inds(i,1),inds(i,2)) + 1;
      end
    end
  end
end
```

After the estimation of the histogram, in order to be a valid joint probability, it has to be normalized by the sum of the counts. Therefore $H_{norm} = H./sum(H(:))$. Three pair

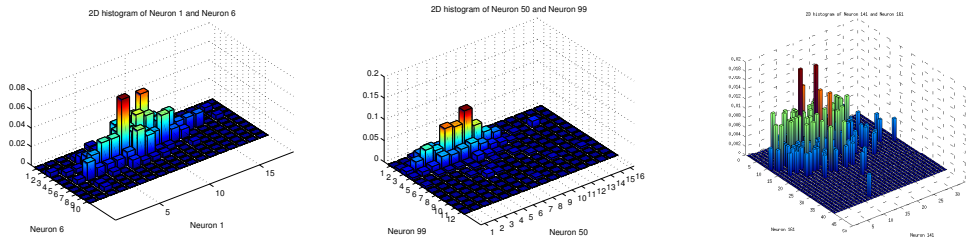


Figure 3: 2D Histograms of the joint probability of selected neurons

of neurons were selected to calculate the mutual information from their respective 2D histograms(shown in Table 1):
(c)
The mutual information for all neuron pairs is represented in the following connectivity-like plot where x and y axes have the neuron numbers and the (i,j) coordinate the color coded mutual information value. The entropy of each neuron, that is the mutual information with itself was not included into the diagonal because it is significantly larger than the pairwise ones and thus shifts the color map of the latter towards lower hues making it difficult for visualization. In addition, the mutual information of the pairs which included the neuron

Neuron	Neuron	Mutual Information
1	6	0.4204
50	99	0.2732
161	141	1.5831

Table 1: Mutual Information for selected pairs

140 were set to zero manually due to the almost zero firing rates of the latter (it crashed the information script).
A graph visualization was attempted, where each vertex (neuron) was placed clockwise and the edges with their respective weights(Mutual Information) where visualized as color coded lines. It is apparent that some of the neurons such as 161, 139 and 7 have a lot of strong connections(large mutual information) with each other and with others.

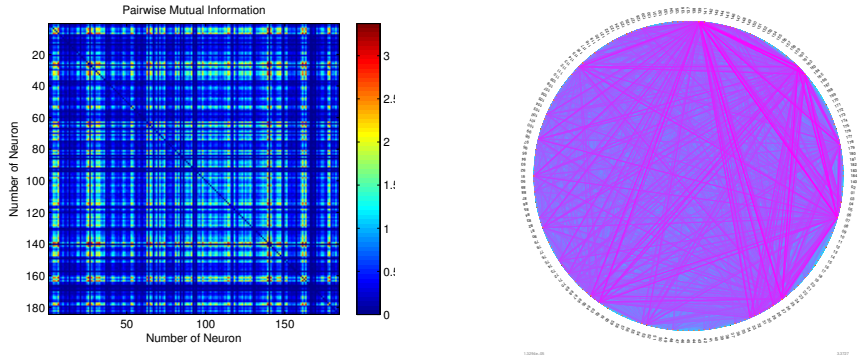


Figure 4: Mutual Information Matrix and Graph Representation

A clustering script was used in order to calculate the cliques(number of the complete subgraphs) of the adjacency matrix, which were computed by thresholding the information matrix on 3 bits. The number of connections left in the mutual information matrix after the thresholding were 22.

$$A = \text{InformationMatrix} > 3 \text{ bits}$$

Clique	Neuron	Neuron	Neuron
1	7	161	163
2	26	139	161
3	139	161	163

Table 2: Number of cliques with information thresholding to 3 bits

By thresholding the information in 2.5 bits more cliques are expected because the number of connections in the information matrix are 164.
(d)

Clique	Neuron	Neuron	Neuron	Neuron	Neuron	Neuron
1	4	7	26	139	161	163
2	7	139	141	161	163	178
3	26	139	141	161	163	178
4	7	114	139	161	163	
5	26	114	139	161	163	
6	28	139	161	163	178	
7	63	65	139	161	163	
8	5	7	139	161		
9	7	139	147	161		
10	26	65	161	163		
11	31	139	161	163		
12	33	139	161	163		
13	7	98	161			
14	26	65	139			
15	81	139	163			
16	91	139	161			
17	98	139	161			

Table 3: Number of cliques for information thresholding to 2.5 bits

The sample covariance was calculated between all neuron pairs and the formula for the mutual information was used. The new Mutual Information Matrix is: In this case the

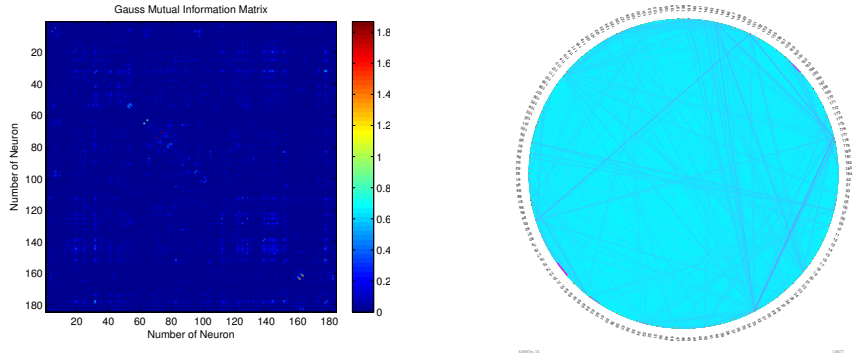


Figure 5: Gaussian Mutual Information Matrix and Graph Representation

mutual information is significantly lower than when using 2D histograms. The maximum information goes up to 1.8677 in contrast to 3.3727 bits with the previous approach. By thresholding the information to over 0.3 bits, 58 connections remain and the constituted cliques are contained in the following graph:

If the threshold is lowered by 0.1 bits, the number of connections become 206. The cliques that are formed are: Two approaches have radically different information scales and the number of cliques depend on the information thresholding. However, in both cases there are clusters corresponding to different directions, which are not close to the real one (eight). It is interesting that the formed cliques in both approximations share vertices between the

Clique	Neuron	Neuron	Neuron
1	32	177	178
2	53	54	55
3	161	163	182

Table 4: Number of Cliques for information thresholding to 0.3 bits

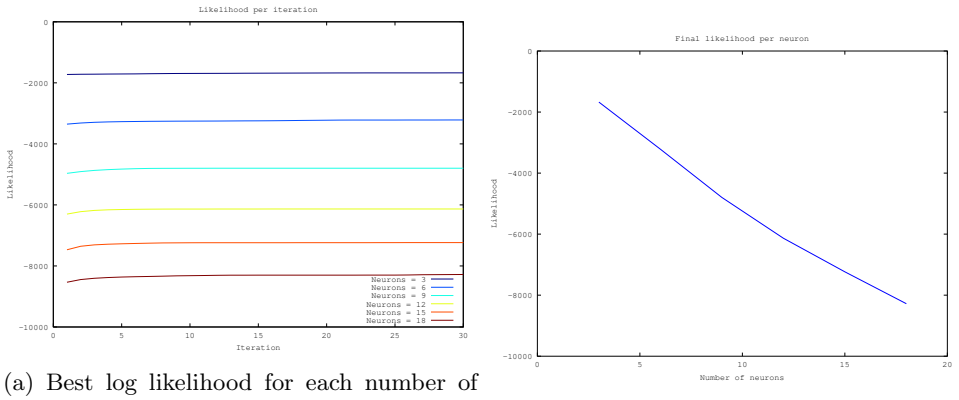
Clique	Neuron	Neuron	Neuron	Neuron	Neuron
1	20	138	144	151	177
2	32	138	144	151	177
3	20	112	144	177	
4	32	139	144	151	
5	112	125	128	144	
6	125	128	139	144	
7	138	139	144	15	
8	7	95	96		
9	19	20	151		
10	20	125	144		
11	32	177	178		
12	42	144	177		
13	53	54	55		
14	66	87	132		
15	83	177	178		
16	99	101	102		
17	122	125	144		
18	128	144	147		
19	144	145	147		
20	161	163	182		

Table 5: Number of Cliques for information thresholding to 0.2 bits

cliques. This can be interpreted as a causal link between the neuron groups.

2 Decoding reaching directions from firing rates

A subset of 15 neurons were selected using the entropy as a heuristic. Standard deviation and a combination of entropy and standard deviation were also tried as heuristics, but the entropy gave consistently better results (log likelihood) after training our model. We tried to use the log likelihood in a crossvalidation fashion to select the best number of neurons to use but results were somehow counterintuitive showing a decrease in likelihood as the number of neurons increased (see Figure 6). We chose by heart to use 15 neurons because given that we were going to use a diagonal covariance matrix we thought it was small enough so as to not overfit but big enough so as to actually pick up good features from our data.
(a) (b)
After the parameters were setted we run the EM algorithm for 20 random initializations and 40 iterations, compute the posterior for each data point($p(v|u)$) and assign a cluster to



(a) Best log likelihood for each number of neuron

(b) Log likelihood per neuron

Figure 6: Likelihoods for different number of neurons choosen using the entropy

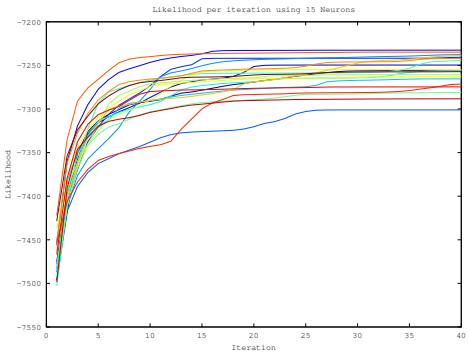
each data point. We present the different likelihood per iterations for each run, the cluster assignment for each data point and a histogram with the number of data points assigned to each cluster. Results are shown in Figure 7.

It is hard to see any structure in the results and as the data is multidimensional (15-d in our case) a simple projection in two dimensions where we could look to the scatterplot or to the gaussian mixture distributon is not enough and even though there appear to be some patterns in the assignments as we do not have a true assignment for each data point we have no means to grade (via an error function) our results other than using the log likelihood of the data. Considering that an equal number of data points are assigned to each cluster, we expect the second plot on figure 7 to look like a uniform distribution which is close to be true[1]. We know at least, as will be shown later, that we produce consistent results; we notice that the same neurons get clustered together when using similar configurations although it may be hard to see this only by looking at the barplots given that the cluster numbers change from one run to the other (clusters are ordered by α_k before being assigned to try and match the same cluster number in different runs but this is not always the case). Allowing the covariance matrix to take random initialization values instead of being equals to $diag(\sigma_1, \sigma_2, ..., \sigma_N)$ as was originally set produces similar results (although some random initializations resulted in singular covariance matrices). We notice that the likelihood converges to around -7250 as before meaning probably that in both cases we are getting to the same results. To avoid problems with singular covariance matrices we decided to stick with the normal initialization instead of completely random (means are still assigned at random). A random initialization of α_k was also tried and produced similar results.

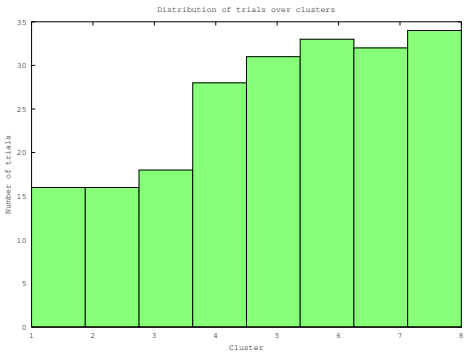
(c)

To restrain the covariance matrix to be $\sigma^2 * I$ we assigned $\sigma^2 = mean(\sigma_1^2, \sigma_2^2, ..., \sigma_N^2)$ (this is done at each M-step). In this case we are assuming each cluster has an spherical distribution (in N-dimensions) with variance σ_k^2 for each cluster, this could be advantageous if we know that the data is originally clustered in this way and we expect to obtain a better and simpler model. In general if we give less freedom to the covariance matrix (and by that to the gaussian distributions) we simplify our model and make computations easy, this could

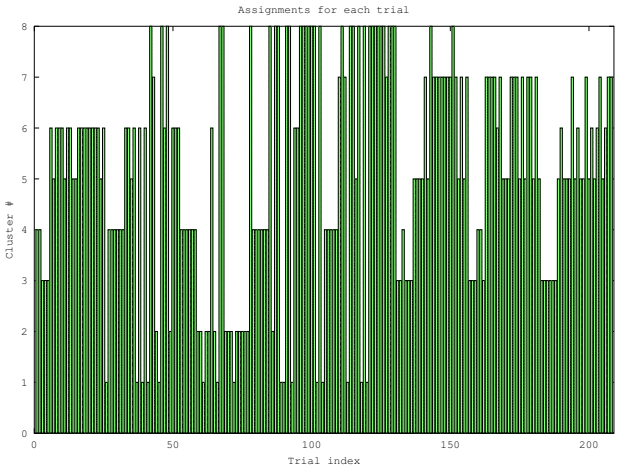
[1]It is worth noticing though that a completely random classification of data points will generate a uniform plot and so this is not a guarantee of good results



(a) Log likelihood for each run



(b) Number of trials per cluster



(c) Cluster assignment per trial

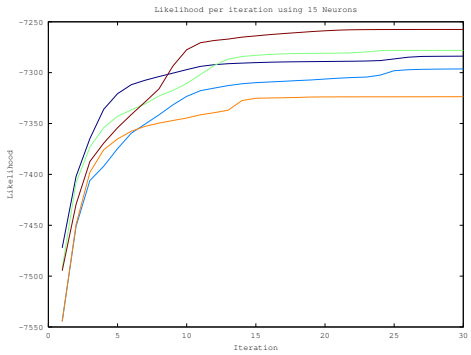
Figure 7: Results of GMM with 15 neurons

be useful when complexity is an issue or in our case when overfitting is an issue; nonetheless, when using a more simple model we may lose some important information embedded in our data and obtain generic oversimplifying results. Whenever possible a complete matrix with a good number of clusters(to avoid overfitting) should be a good option, a diagonal matrix also offers a good option though you lose some variation in the clusters but the number of parameters to fit gets greatly reduced [2].

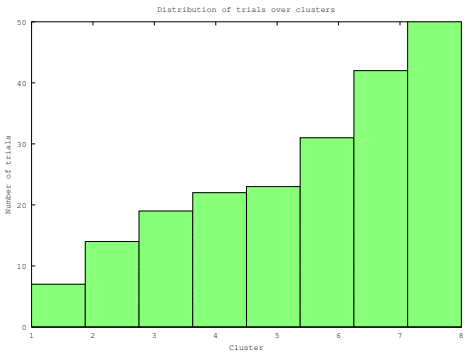
Results for this case are shown on figure 9. The results are similar to the ones obtained with the diagonal matrix, although as expected the log likelihood decline from -7200 to -7900 and also there is more local minima in this case.

To restrain the covariance matrix to be the same for all clusters we computed $\Sigma = \frac{1}{K}\sum_{k=1}^{K}\Sigma_k$ and assigned it to each cluster (at each M-step). In this case this restriction asserts each of our cluster will have the same shape (either $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_N)$ or $\Sigma = \sigma^2 * I$) and if that is what the application is looking for this could be a good option. However, to have the same shape for every cluster could be very restrictive and we do not earn much in terms

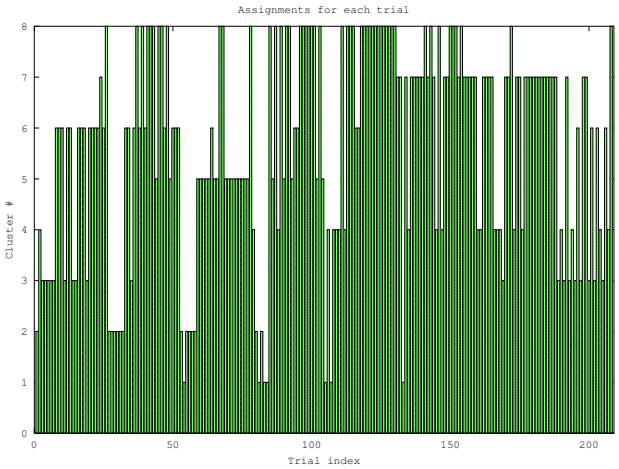
[2] A cool demo can be found on `bit.ly/1pQI7iE`.



(a) Log likelihood for each run



(b) Number of trials per cluster



(c) Cluster assignment per trial

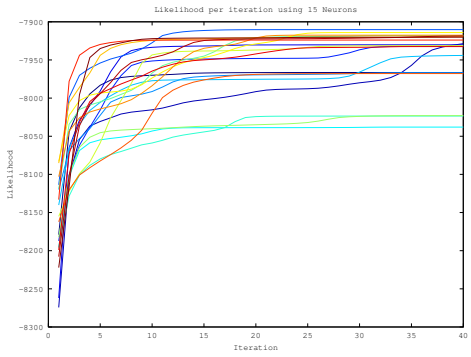
Figure 8: Results of GMM with random initialization of covariance matrix

of complexity because the number of operations needed are roughly the same to the ones needed to fit each cluster with a different covariance matrix. We present the results for the same diagonal covariance matrix for each cluster, the option to have the same simple covariance matrix for all clusters, effectively only fitting one parameter σ, seems very restrictive and was computed but is not presented.

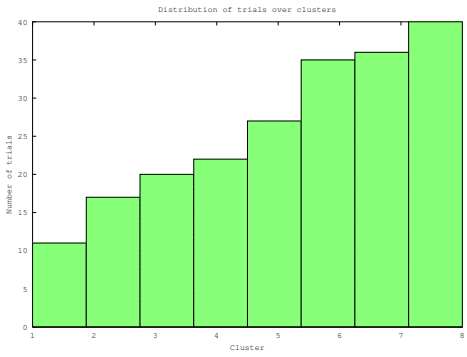
We notice that we do not lose much information in going to the same covariance for each cluster, the log likelihood only falls to -7350 and the plots seem similar to the ones without this restriction. Another thing to notice over all the results presented is that the distribution of data points in clusters remain fairly similar from one configuration to the other pointing to the fact that our results are probably consistent although some parameters are changed.

(d)

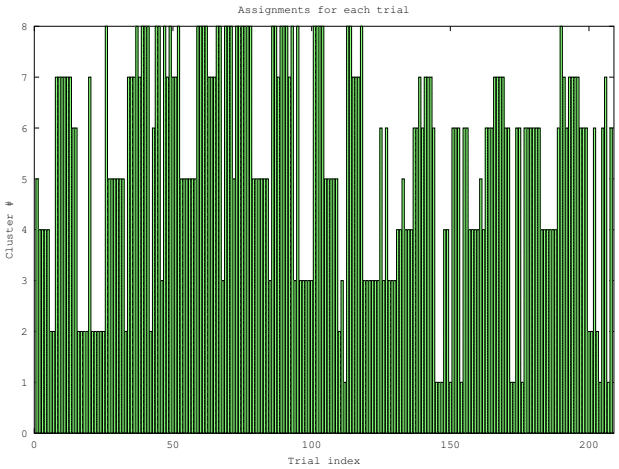
We used an script similar to the one used to fit the number of neurons. Results are shown on Figure 11. As can be seen the likelihood increases as the number of clusters increases, which could mean either that there is more 'underlying causes of the data' than expected and that the training should be done selecting a number of clusters greater than eight or



(a) Log likelihood for each run



(b) Number of trials per cluster



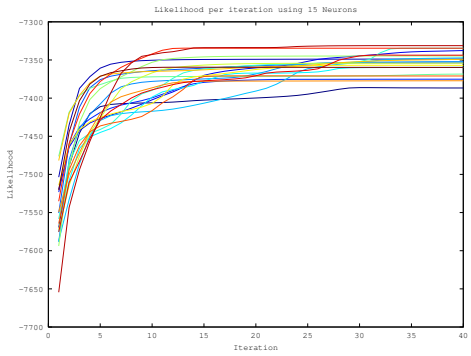
(c) Cluster assignment per trial

Figure 9: Results of GMM with constraint $\Sigma = \sigma^2 * I$

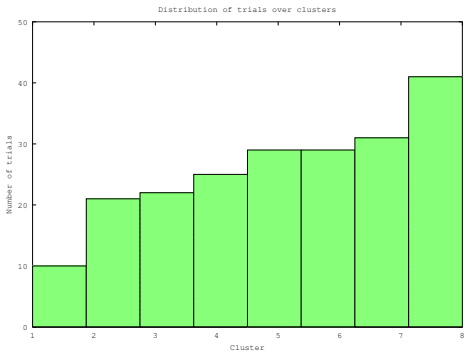
that given the very small and noisy sample (208 trials) the model is already overfitting even at a small number of clusters.

(e)

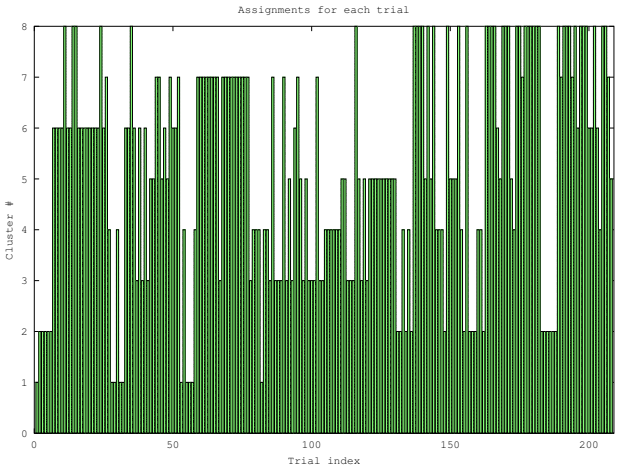
When we choose to use a Gaussian Mixture Model, we are assuming our data can be clustered in K simpler gaussian distributions. Although a Gaussian distribution with a full covariance matrix can be very malleable, there is some intrinsic limitations in trying to model data that follow other patterns that will not get picked up by a GMM. If we have some insights on what kind of form our data has we could choose to build a mixture model with this new kind of functions (assuming the EM algorithm has a closed form solution for this new function). If we do not have any insight into the shape of our date, we could try fitting to other well known multivariate probability distributions either by their own or in a mixture model.



(a) Log likelihood for each run

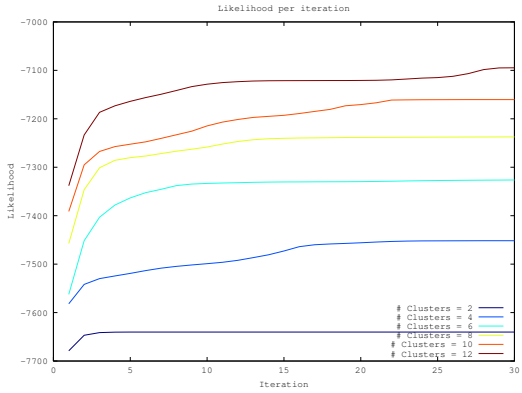


(b) Number of trials per cluster

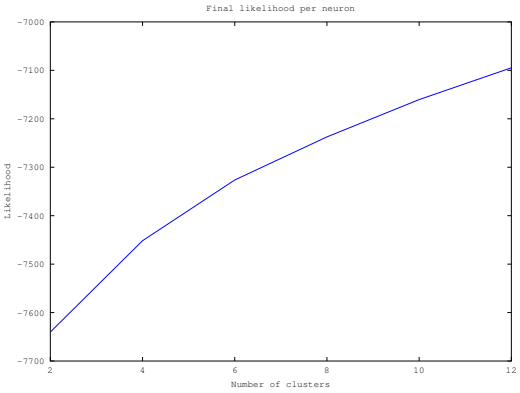


(c) Cluster assignment per trial

Figure 10: Results of GMM with same diagonal covariance matrix for each cluster



(a) Best log likelihood for each number of cluster



(b) Log likelihood per cluster

Figure 11: Likelihoods for different number of clusters