Danmarks Tekniske Universitet

Special Course

Probabilistic Clustering

Manuel Montoya Catalá, Arturo Arranz

Last modified January 3, 2018

Contents

1 Tool description

We implemented a tool to perform the Expectation Maximization algorithm in Python. In the following we will present the features of the tool and some code examples on how to use it.

The **features** of the library include:

- Object Oriented Programming implementation with an interface similar to the scikit-learn library.
- The CEM class performs both the EM algorithm for independent clusters and the EM algorithm for clusters following a Markov Chain of order one, the Baum and Welch algorithm.
- There are 3 distributions already implemented in the tool. The Gaussian (both full or diagonal covariance matrix), Watson and vonMissesFisher distribution.
- Clusters from different distributions can be combined in EM algorithm.
- All the characteristics of the distributions (initialization, estimation, singularity handling...) can be tuned using hyperparameters given in a dictionary.
- An user can easily modify any of the implemented distributions to include any desired constraint.
- An used can easily introduce a new distribution by just indicating a few functions about it. We will elaborate on this later.
- The CEM output includes the evolution of the clusters thorugh the iterations, as well as the incomplete log likelihood and the model parameters.
- The code is optimized avoiding to use for loops whenever possible.
- Everything is implemented in logarithms to handle extreme values of the likelihoods.
- Several examples of use, creation of graphics and Crossvalidation of parameters are provided.
- The system can handle the two main crashing points:
  - Singularities, also called degenerated clusters. This situation happens when a cluster is not associated enough samples to estimate its parameters
  - Extreme normalization constant. If the parameters of the distribution are too odd, the computing of the normalization constant can be intractable.
- The modular structure of the system makes it easy to include priors on the parameters and a variational posterior on the posterior probability of the clusters.
- The EM object comes with functionalities to compute the likelihood, responsibility, and decoding of a given set of data provided.
- The algorithm includes the possibility of different levels of verbose and timing study.

In the Example code files there is also code to:

- Use the library including all of its implemented distributions over toy examples and EEG data.
- Perform Cross Validation of the number of clusters for toy examples and the EEG data.
- Use the outputs of the tool to generate graphs with:
  - The evolution of the likelihood through the iterations.
  - The Cross Validation likelihood for the different number of clusters.
  - The shape of the clusters in the 2D toy examples.
  - The time evolution of the EEG data.

Since it is easier to learn through a example, we will use it initially. In the following code, we will:

- Set the parameters of the EM algorithms
- Set the distributions and clusters to use.

- Initialize an object of the tool.
- Perform the EM algorithm over a dataset

Last things to implement: - obtain responsibility and loglikelihood from the HMM. - implement the decoder inside. - document the Variational Inference.
- Do the thing of performing HMM after EM. Seeing the covariance matrix.

1.1 Interface of the tool

In this Section we will describe the process of initializing and using the tool, include what its inputs should be.

1.1.1 Loading the library

First of all, in order to use the tool we need to import it using the following two lines which will load the tool and the the set of distributions implemented.

```
import CEM as CEM
import CDistribution as Cdist
```

In the Example code files we included first of all the following statements. The import folders file will be in charged of loading the directories that contain all the code files of the library. In the future this could be done automatically if converted into a real library. For now, we need to execute these lines so that all the code structure is loaded.

```
import os,sys
sys.path.append(os.path.abspath('../'))
os.chdir("../")
import import_folders
```

1.1.2 Initialization of the CEM object

In the initialization step we create an object of the CEM class which we will further use to use the library. During the initialization we provide the object with the configuration of the EM algorithm. The interface and the description of the inputs is provided below.

```
class CEM ():

    def __init__(self, distribution = None, clusters_relation = "independent",
                 T = 30, Ninit = 20, delta_ll = 0.01,
                 verbose = 0, time_profiling = "no"):
    """
     Inputs:
         - distribution: Distribution manager with the different distributions of the
         mixture and their associated clusters.
         - clusters_relation: Dependency between the clusters:
               - For "independent": The clusters are assumed independent
               - For "MarkovChain1": The clusters are assumed to follow discrete
                  MC of order 1
         - delta_ll: Minimum increase in likelihood for consideing that the algorithm
         has not converged. If in an iteration the incomplete loglikelihood does not
         increase by this minimum value, the iterations stop.
         - T: Maximum number of iterations.
         - model_theta_init: Initial parameters of the model. If not given they will
         be initilized uniformly. This is [pi] or [pi, A] for example.
         - theta_init: Initial parameters of the clusters. If not provided, the
         initializer provided in the
           distribution objects will initialize them.
         - Ninit: Number of random random re-initilizations.
         - verbose: The higher the value, the more partial outputs will be printed out.
             - 0: No verbose
             - 1: Indication of each initilization of the EM and its final ll.
```

```
            - 2: Indication of each iteration of the EM and its final ll.
        - time_profiling: If we want to see the times it takes for operations.

    Output:
        - logl: List of incomplete loglikelihoods asociated to each iteration.
          Notice it will have always the added initial loglikelihood asociated
          to the initialization. So maximum it will be "T+1" elements.
        - theta_list: List of the cluster parameters for each of iterations.
        - model_theta_init: List of the model parameters for each of the iterations
    """
```

As we can see, in the initialization we provide with all the configuration parameters of the algorithm. As remarks:

- The "clusters_relation" variable is used to switch transparently between the EM ("independent")and BW ("MarkovChain1") algorithms, sharing both algorithm the same interface for all configuration parameters and outputs of the interface.
- The distribution manager "distribution" contains all the information needed about the clusters to be used in the algorithm (mainly their estimation functions and number of clusters to be used)

The following code shows an example on how to initialize an object of the tool. In this example we also create a Distribution Manager with 3 Gaussian clusters which will be explained later.

```
######## Create the Distribution object ##################################################
## Create the distribution object and set it a Gassian.
Gaussian_d = Cdist.CDistribution(name = "Gaussian");
Gaussian_d.set_distribution("Gaussian")
### Add them together in the Manager
myDManager = Cdist.CDistributionManager()
K_G = 3      # Number of clusters for the Gaussian Distribution
myDManager.add_distribution(Gaussian_d, Kd_list = range(0,K_G))

############# SET THE CONFIGURATION PARAMETERS ###################################
Ninit = 10
delta_ll = 0.02
T = 60
verbose = 1;
clusters_relation = "independent"   # MarkovChain1  independent
time_profiling = "no"
############# Create the EM object #############
myEM = CEM.CEM( distribution = myDManager, clusters_relation = clusters_relation,
               T = T, Ninit = Ninit,  delta_ll = delta_ll,
               verbose = verbose, time_profiling = time_profiling)
```

1.1.3 Running the EM algorithm on some data

Once we have created the CEM object, we can use it to fit some data. For this purpose we use the .fit() function. Which will run the EM algorithm though the given data, we can optionally also indicate the initial values of the parameters of the cluster and the model. If they are not provided the initializers will be used.

1.2 Interface of a distribution

Probably the most complicated part of the library is understanding the distribution manager. Since the library is able to combine different distributions in the EM algorithm, this object contains:

- The required information about the different distributions (such as the functions to compute the loglikelihood of a sample or the estimation of the parameters)

- An interface to interact with the distributions.

Regarding the interface, it is used internally by the algorithm so it will not be described. It is basically encharged of managing the logic of executing the functions of each active cluster.

Each distribution is defined by a set of functions and a set of hyperparameters included in a dictionary. The dictionary of parameters is internally given to each of the functions so that they can obtain the parameters easily from them.

The basic functions that need to be set for each distribution are described in the following. The exact description of the interface of this functions is described in the code, here we will only describe their functionality.

- *pdf_log_K(X, theta, Cs_log = None)*: This function returns the pdf in logarithmic units of the distribution. It accepts any number of samples N and any number of set of parameters K (number of clusters) simultaneuously. For optimizaiton purposes, if the normalization constants are precomputed, they can be given as input.
- *get_Cs_log(theta_k, parameters = None)*:This function will compute the normalization constant of a cluster. Usually, the normalization constant is a computaitonal bottelneck since it could take quite some time to compute. In an iteration of the EM algoeithm, it should only be computed once per cluster. If given, the computation of such constants can be minimized.
- *init_params(X,K, theta_init = None, parameters = None)*: This function initializes the parameters of the K clusters if no inital theta has been provided. If a initial theta "theta_init" is specified when calling the "fit()" function then that initialization will be used instead. The minimum parameters to be provided are the number of cluster K and the dimensionality of the data D. In order to add expressivity we can use parameters of the dictionary
- *theta_estimator(X, rk = None, parameters = None)*: This function estimates the parameters of a given cluster k, k =1,...,K given the datapoint X and the responsability vector rk of the cluster.
- *degenerated_estimation_handler(X, rk , prev_theta_k , parameters = None)*: If during the estimation of the parameters there was a numerical error and the computation is not possible, this function will try to solve the situation. Some common solutions are to use the previous parameters theta_k of the cluster or reinitilizite it using other hyperperameters.
- *degenerated_params_handler(X, rk , prev_theta_k , parameters = None)*: It at some point the obtained parameters of the cluster makes it non-feasible to compute the pdf of the data, for example because the normalization constant is too big or too small, its inaccurate, or it takes too much time to compute then this function will attempt to recompute another set of parameters.

1.3 Implemented Distributions and their parameters

As we described, 3 distributions are implemented, here we will comment on how to use them and the different parameters that can be set.

- The first thing that we need to do is to **create a distribution object**, assigning a name to it that will be used to identify the distribution object inside the algorithm. With this name differentiation we could for example create clusters from the same distribution but different parameters.
- Then we can either **specify the distribution** function and parameters by hand or set one of these implemetented distributions using the method *.set_distribution(distribution_name)*
- After that we usually want to **set the hyperparameters of the distribution** (such as initialization properties, handling of singularities, constraints of the distributions).
- Finally we **add the distribution to the Distribution Manager, indicating the number of clusters that we want**. The way we indicate the clusters is by means with a list of integers that identify the clusters. A cluster will be initialized for each integer in the list. Each integer must be different since they are used to reference them.

1.3.1 Gaussian

The Gaussian distribution implemented uses unbiased estimators for the mean and covariance matrix. Its distribution name inside the tool is *"Gaussian"* Some features to be aware are:

- The clusters are initialized with diagonal covariance matrices with variance initialized at random between the parameters *"Sigma_min_init"* and *"Sigma_max_init"*. The mean of the clusters are initialized at random from a Gaussian distribution with 0 mean and variance given in *"mu_variance"*
- The algorithm supports the estimation of the full covariance matrix of the clusters, or a constrained estimation where the covariance matrix is diagonal through the parameter *"Sigma"*.
- When a cluster is degenerated due to a singularity or unable to compute the normalization constant, then the variance of the elements in the diagonal are set to a minimum of *"Sigma_min_singularity"* and *"Sigma_min_pdf"* respectively.

When implementing this distribution, the parameters will have default values but it is advised to change them depending on the properties of the data. The next code shows an example of creation of a set of 3 clusters from a Gaussian distribution:

```
import CDistribution as Cdist

## Create the distribution object and set it as a Gassian.
Gaussian_d = Cdist.CDistribution(name = "myGaussianX");
Gaussian_d.set_distribution("Gaussian")

Gaussian_d.parameters["mu_variance"] = 1
Gaussian_d.parameters["Sigma_min_init"] = 1
Gaussian_d.parameters["Sigma_max_init"] = 15
Gaussian_d.parameters["Sigma"] = "diagonal" #  "diagonal"    "full"
Gaussian_d.parameters["Sigma_min_estimation"] = 0.1
Gaussian_d.parameters["Sigma_min_distribution"] = 0.1

## Create a Distribution manager
myDManager = Cdist.CDistributionManager()
K_G = 3      # Number of clusters for the Gaussian Distribution
myDManager.add_distribution(Gaussian_d, Kd_list = range(0,K_G))
```

1.3.2 Von Misses Fisher

The von Mises Fisher distribution implemented uses the Newton Method in order to compute the value of its κ (kappa) parameter. Its distribution's name inside the tool is *"vonMisesFisher"*. Some features to be aware are:

- The clusters are initialized with a random initial direction and an initial kappa value at random between the parameters *"Kappa_min_init"* and *"Kappa_max_init"*.
- The number of iteration of the Newton Method can be specified in with the parameter *"Num_Newton_iterations"*.
- When a cluster is degenerated due to a singularity or unable to compute the normalization constant, then the kappa is saturated to the maximum given by *"Kappa_max_singularity"* and *"Kappa_max_pdf"* respectively.

When implementing this distribution, the parameters will have default values but it is advised to change them depending on the properties of the data. The next code shows an example of creation of a set of 3 clusters from a von Mises Fisher distribution:

```
import CDistribution as Cdist

## Create the distribution object and set it as a vonMisesFisher.
vonMisesFisher_d = Cdist.CDistribution(name = "myFisherX");
vonMisesFisher_d.set_distribution("vonMisesFisher")
```

```
vonMisesFisher_d.parameters["Num_Newton_iterations"] = 5
vonMisesFisher_d.parameters["Kappa_min_init"] = 0
vonMisesFisher_d.parameters["Kappa_max_init"] = 100
vonMisesFisher_d.parameters["Kappa_max_singularity"] =1000
vonMisesFisher_d.parameters["Kappa_max_pdf"] = 1000
## Create a Distribution manager
myDManager = Cdist.CDistributionManager()
K_vMF = 3       # Number of clusters for the Gaussian Distribution
myDManager.add_distribution(vonMisesFisher_d, Kd_list = range(0,K_vMF))
```

1.3.3 Watson

The Watson distribution implemented uses the Newton Method in order to compute the value of its κ (kappa) parameter. Its distribution's name inside the tool is *"Watson"*. Some features to be aware are:

- The clusters are initialized with a random initial direction and an initial kappa value at random between the parameters *"Kappa_min_init"* and *"Kappa_max_init"*.
- The number of iteration of the Newton Method can be specified in with the parameter *"Num_Newton_iterations"*.
- We can allow or not the computation of clusters with negative kappa using the parameter *"Allow_negative_kappa"*.
- When a cluster is degenerated due to a singularity or unable to compute the normalization constant, then the kappa is saturated to the maximum given by *"Kappa_max_singularity"* and *"Kappa_max_pdf"* respectively.

When implementing this distribution, the parameters will have default values but it is advised to change them depending on the properties of the data. The next code shows an example of creation of a set of 3 clusters from a Watson distribution:

```
import CDistribution as Cdist

## Create the distribution object and set it as a vonMisesFisher.
vonMisesFisher_d = Cdist.CDistribution(name = "myFisherX");
vonMisesFisher_d.set_distribution("vonMisesFisher")

vonMisesFisher_d.parameters["Num_Newton_iterations"] = 5
vonMisesFisher_d.parameters["Kappa_min_init"] = 0
vonMisesFisher_d.parameters["Kappa_max_init"] = 100
vonMisesFisher_d.parameters["Kappa_max_singularity"] =1000
vonMisesFisher_d.parameters["Kappa_max_pdf"] = 1000
## Create a Distribution manager
myDManager = Cdist.CDistributionManager()
K_vMF = 3       # Number of clusters for the Gaussian Distribution
myDManager.add_distribution(vonMisesFisher_d, Kd_list = range(0,K_vMF))
```

1.4 Toy examples

In order to test the capabilities of the library, we have created the Example code file: *3.main_EM_Gaussian_Watson_vonMisesFicher.py*. In this file we can use the library on a generated dataset composed by 3 Gaussian clusters.

Since the library allows for different distributions to combined, this file makes use of this feature. In order to combine directional and non-directional distributions, the normalization of the data (setting the module = 1) is performed only when computing the clusters of the directional distributions. This way the Gaussian clusters will be computed from the orinal data, and the directional clusters will be computed using the normalized data.

In the **first example** we will show how the algorithm finds the distribution of 2 Gaussian clusters using **2 Gaussian distributions** with full covariance matrix. In the following figure we can see

the final fitting of the clusters and the original data, along with the evolution of the incomplete log likelihood though the iterations.

The way the different clusters are represented in the image is:

- For Gaussian distributions we plot the 95% confidence interval ellipse.
- For the Watson and vonMisesFisher we plot a circumference of radius 1 plus the likelihood associated to each angle of the cluster.

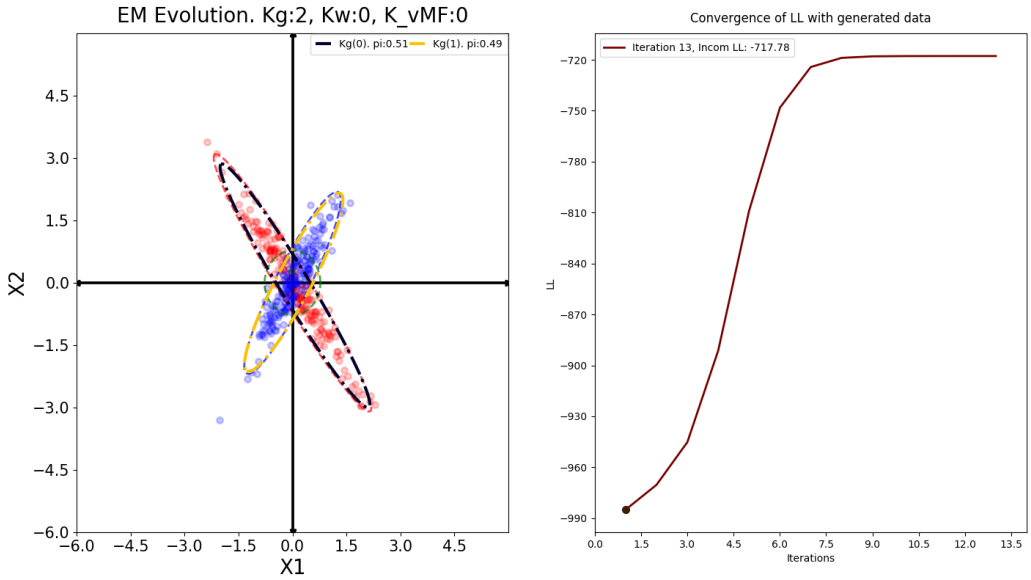

Figure 1: EM for 2 Gaussian clusters on the toy example data

The example codes also plot the evolution of the clusters in 6 different iterations of the algorithm. We can see how the algorithm starts with random diagonal gaussian clusters and then converges to fit the data.

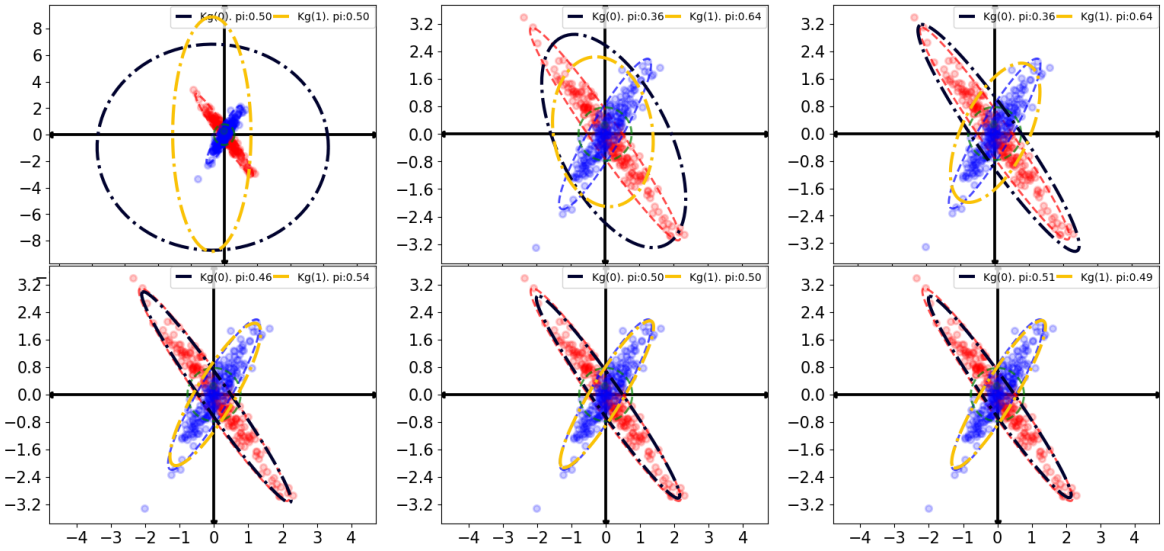

Figure 2: Iterations of the EM for 2 Gaussian clusters on the toy example data

The example codes also implement functionalities to create videos .avi and images .gif in which we can see the clusters through the iterations. In the appended folder there are videos that illustrate the learning process.

In the following **second example** we will use the 3 implemented distributions to fit the same data previously seen. We will use 1 diagonal Gaussian distribution, 1 Watson distribution and 2 von Mises Fisher distributions.

- The Gaussian distribution is intended to model the central samples which, when normalizing the data for the directional distributions would be projected into random directions. In this context they would be considered noise.
- The Watson distribution is intended to model one of the original Gaussian distributions.
- The 2 von Mises Fisher is intended to model the other Gaussian distribution.

The following image shows the final clusters and the evolution of the incomplete log likelihood through the iterations. As we can see, the Gaussian cluster captures the central samples and the directional clusters capture the directional component of the original Gaussian distributions.

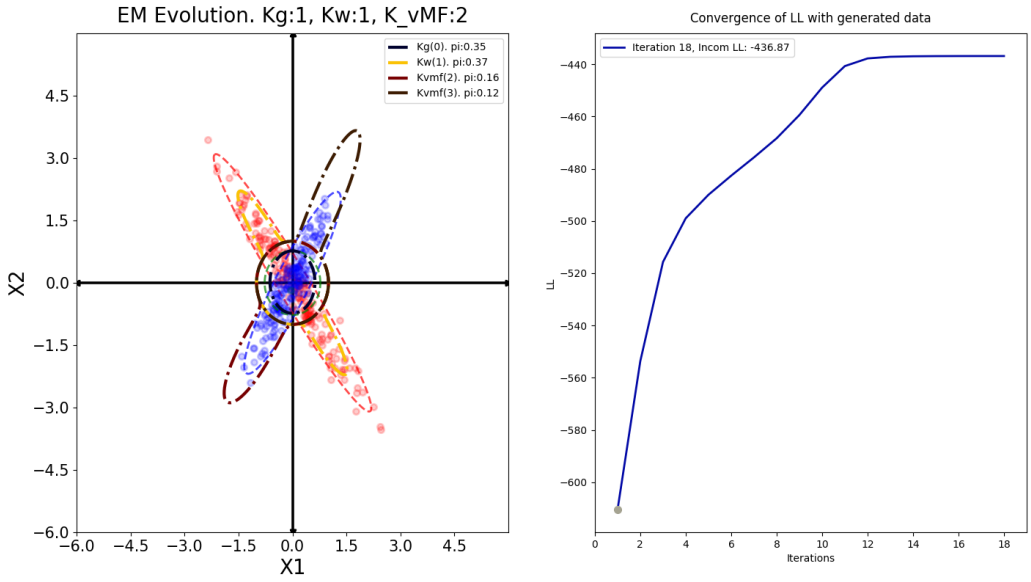

Figure 3: EM for 1 Gaussian, 1 Watson and 2 von Mises Fisher clusters on the toy example data

The next Figure plots the evolution of the clusters in 6 different iterations of the algorithm. We can see how the algorithm starts with clusters with random directions and it converges to fit the data. A video with the evolution is also included.

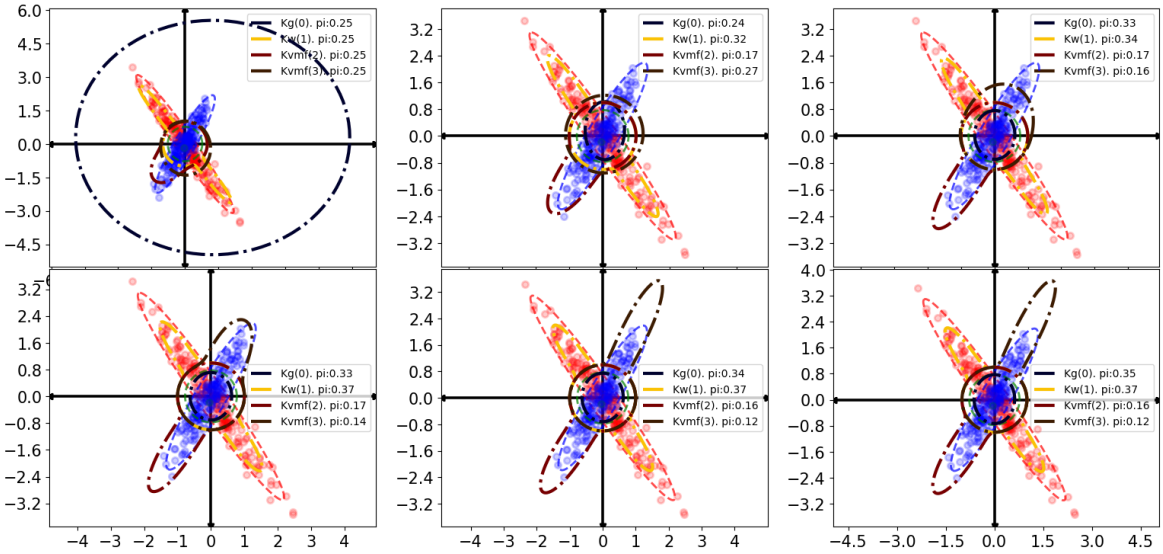

Figure 4: Iteration of the EM for 1 Gaussian, 1 Watson and 2 von Mises Fisher clusters on the toy example data

We can also use for example 2 Watson distributions and 1 Gaussian for the same purpose and show the responsibility of the clusters by color. The following image shows the responsibility for the Gaussian cluster in green, and the responsibility of the Watson components in red and blue.

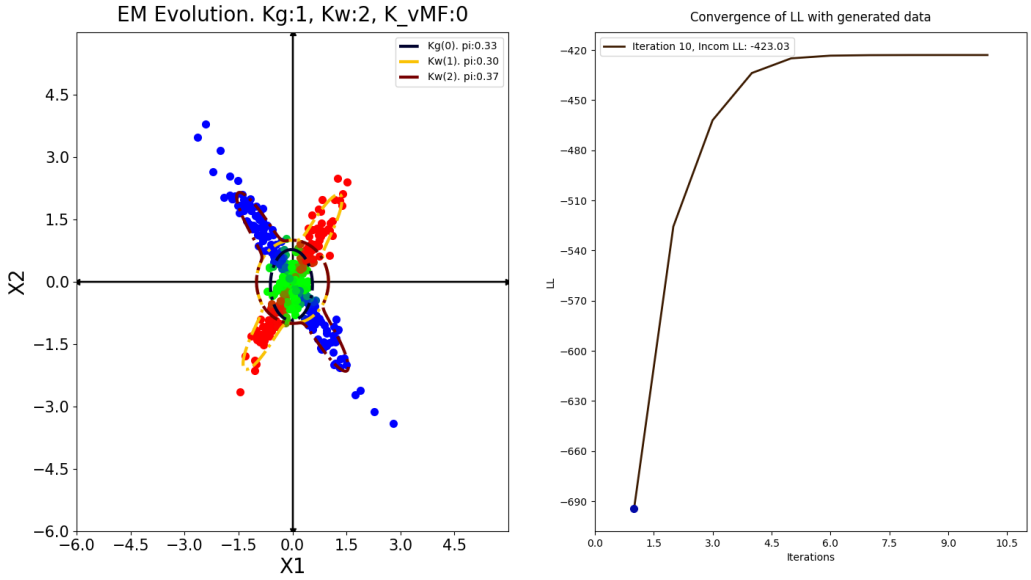

Figure 5: EM for 1 Gaussian and 2 Watson clusters on the toy example data, showing the responsibility by color

The next Figure plots the evolution of the clusters in 6 different iterations of the algorithm. We can see how the algorithm starts with clusters with random directions and it converges to fit the data.

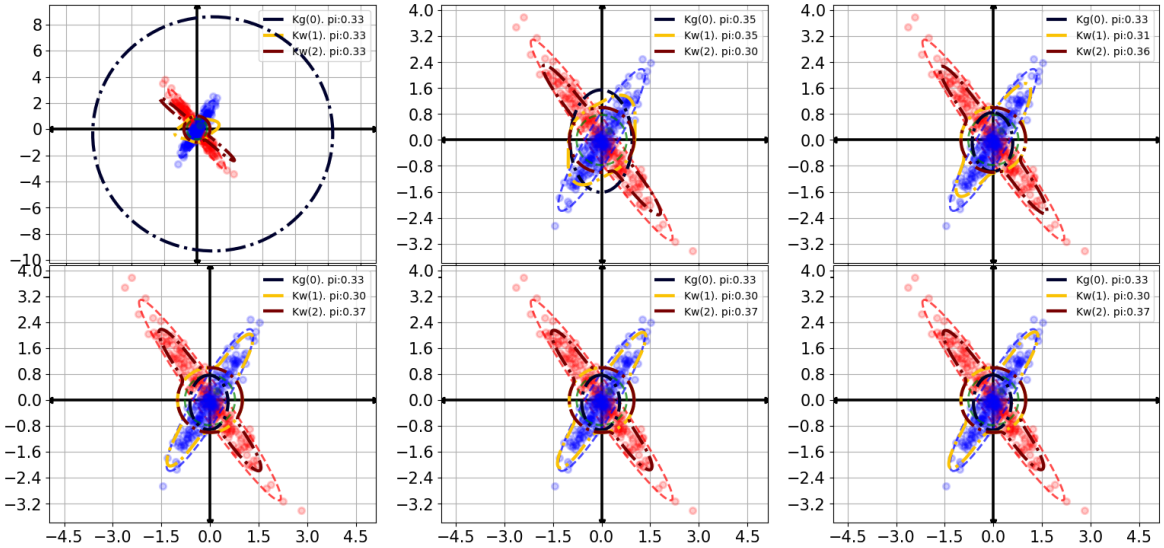

Figure 6: Iterations of the EM for 1 Gaussian and 2 Watson clusters on the toy example data

2 Experiments on EEG data

We used the tool over the EEG data, finding that the distributions cannot be combined since the likelihood of the dataset for each distribution is very different. The likelihood associated to Gaussian clusters is much bigger than the one associated to von Misses Fisher clusters, which at the same time is much bigger than the one associated to Watson clusters. This difference can be seen in the cross validation Figures for the different distributions.

This fact then causes that if for example Gaussian and von Mises Fisher clusters are combined, the responsibility of the von Mises Fisher clusters will be near 0, with virtually no samples associated to them, and therefore they will run into singularities.

In the following we will show some cross validation analysis for the different distributions and temporal analysis of the clusters.

2.1 Cross Validation of the number of clusters

A first thought was to find the optimal number of clusters for the 3 distributions by means of Leave One Out cross validation on the 16 subjects, each represented by a 70 dimensional chain of 451 time samples. The following 3 Figures show these results.

From the charts we can see that the likelihood of a chain for 4-6 clusters for the different distributions is around:

- Diagonal Gaussian distribution: 416.000
- Von Mises Fisher distribution: 32.000
- Von Mises Fisher distribution: 25.000

This would indicate that the Gaussian distribution is more suitable for expressing the data. Maybe a directional distribution in a 70-dimensional signal is too ill-conditioned by noise.

Regarding the optimal number of clusters. The diagonal Gaussian distribution does seem to converge around 6 clusters, but the directional distributions do not seem converge before 15 clusters. This again might happen because when we project the data into the hypersphere of 70 dimensions, a small noise could scatter the samples in all directions.

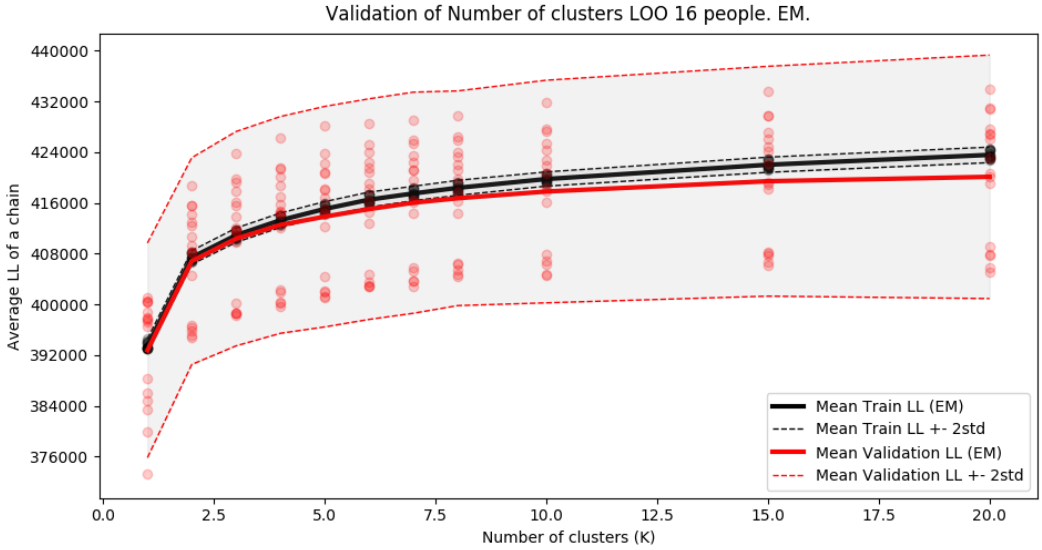

Figure 7: LOO CV of the number of clusters for the Diagonal Gaussian Distribution

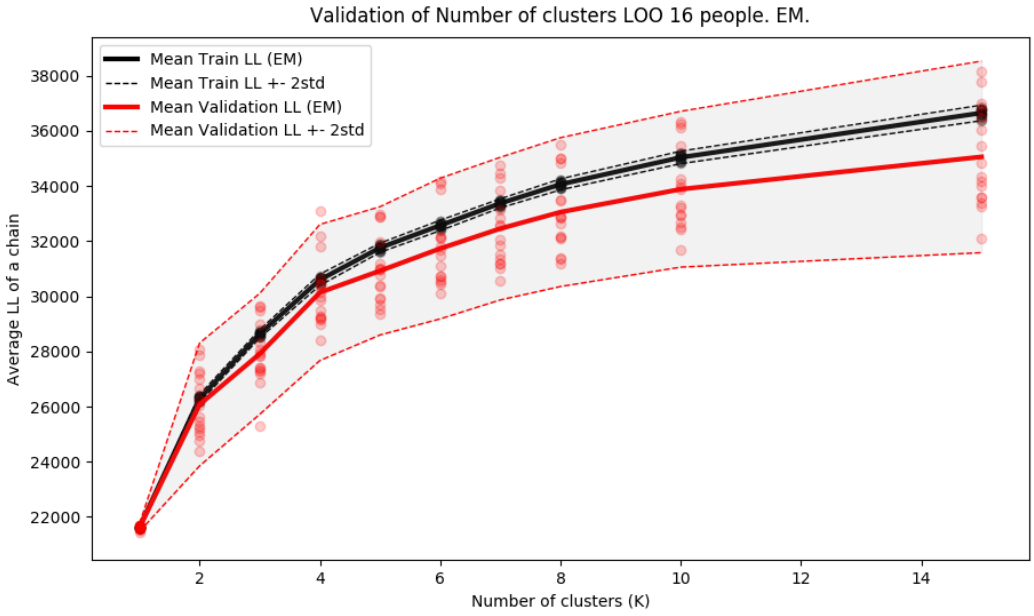

Figure 8: LOO CV of the number of clusters for the von Mises Fisher Distribution

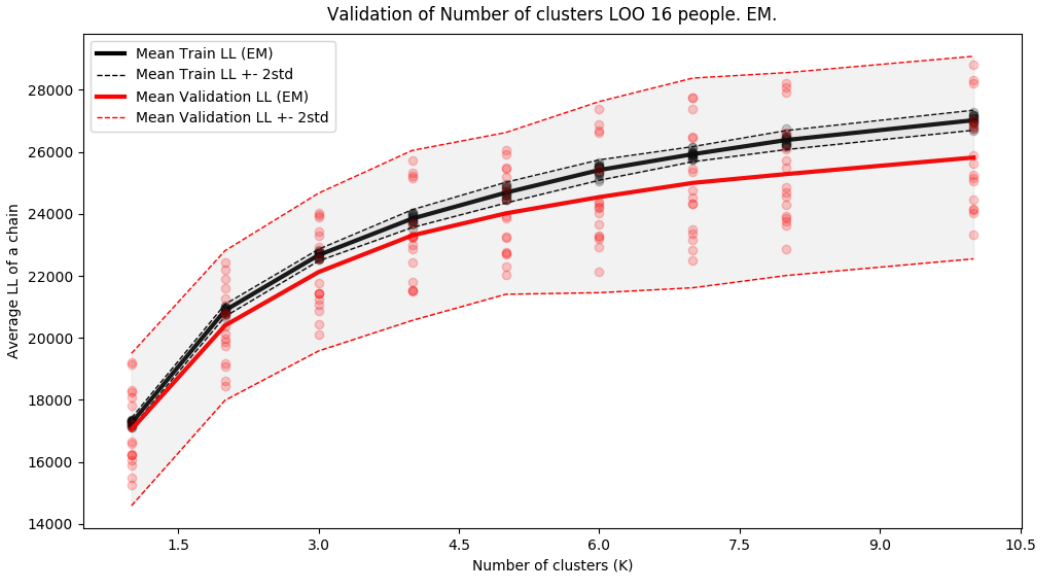

Figure 9: LOO CV of the number of clusters for the Watson Distribution

The results for the 3 different classes of images shown to the subjects are almost identical.

Also, the results for the HMM algorithm are basically the same since the likelihood of the samples is so big compared to the contribution of the transition matrix of the Discrete Markov Chain, that the later one does not significantly change the results.

The next Figure is representative of what a normal evolution of the incomplete data log likelihood through the iterations looks like for the EM and HMM algorithms. As we can see there an initial big jump and then a convergence. The likelihood of the HMM is bigger than that of the EM.

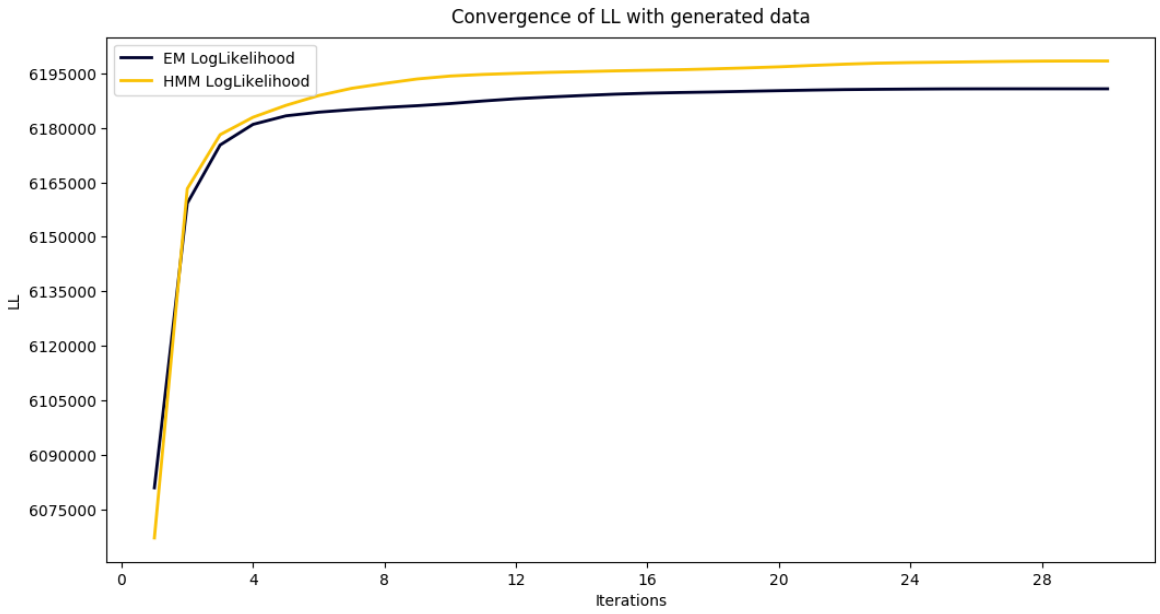

Figure 10: Convergence of the incomplete log likelihood for 4 Diagonal Gaussian clusters

2.2 Temporal analysis of the clusters

In the following we will show a study that aims to find a relation between the clusters found and their happening in time for all of the subjects.

In order to perform this analysis we carry out the following procedure:

- We run the EM algorithm using all of the subjects' data for training.
- We plot the responsibility of each sample to each of the obtained clusters. This is plotted in parallel for the 16 subjects, each having a 451 sample long chain.

Before showing the results it might be convenient to visualize the value of the 70 sensors for a given subject. The same overall behavior is found for the rest of the subjects. As we can appretiate, we can divide the signal in 5 intervals.

- R[1] Range 1-80: The sensors simply seem stable, a little random
- R[2] Range 81-120: There is a big disturbance
- R[3] Range 121-160: The big disturbance reverses polarity.
- R[4] Range 161-325: There is some seemingly smooth random oscillations of the sensors values,
- R[5] Range 326-521: The sensor values seem to converge to 0.

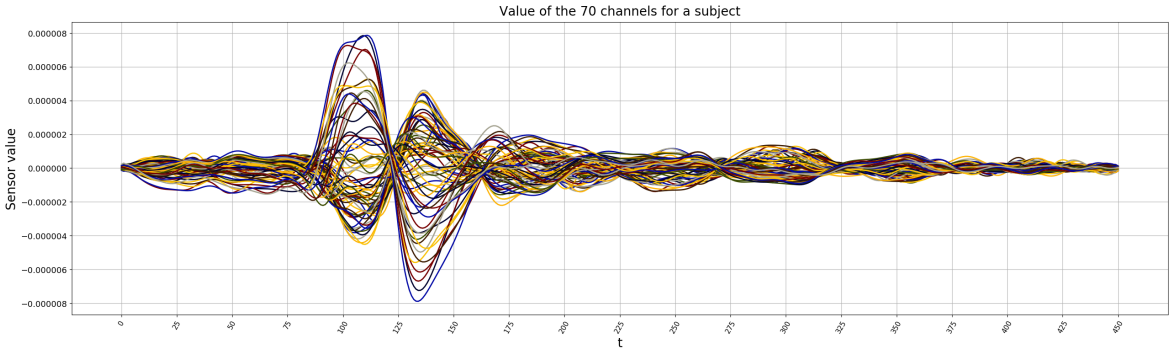

Figure 11: ECG signal of the 70 sensors

2.3 Diagonal Gaussian distribution

If we generate 2 Diagonal Gaussian clusters:

- The yellow cluster seems to represent Ranges R[2], R[3] and partially Range R[4]
- The dark blue cluster seems to represent the rest of the signal.

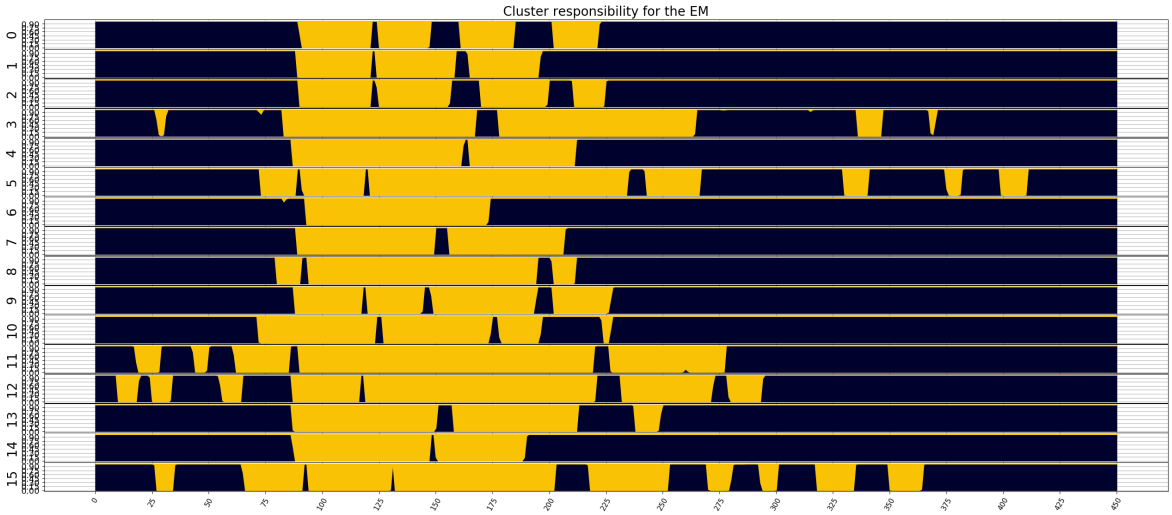

Figure 12: Time analysis for 2 Diagonal Gaussian clusters

If we generate 3 Diagonal Gaussian clusters:

- The dark blue cluster seems to represent Ranges R[2], R[3] and partially Range R[4]. But it is much more specific than the homologous cluster in the previous image.
- The red cluster seems to represent the surrounding of R[2] and R[3].
- The yellow cluster seems to represent the rest of the signal, at the beggining and at the end of the chains.

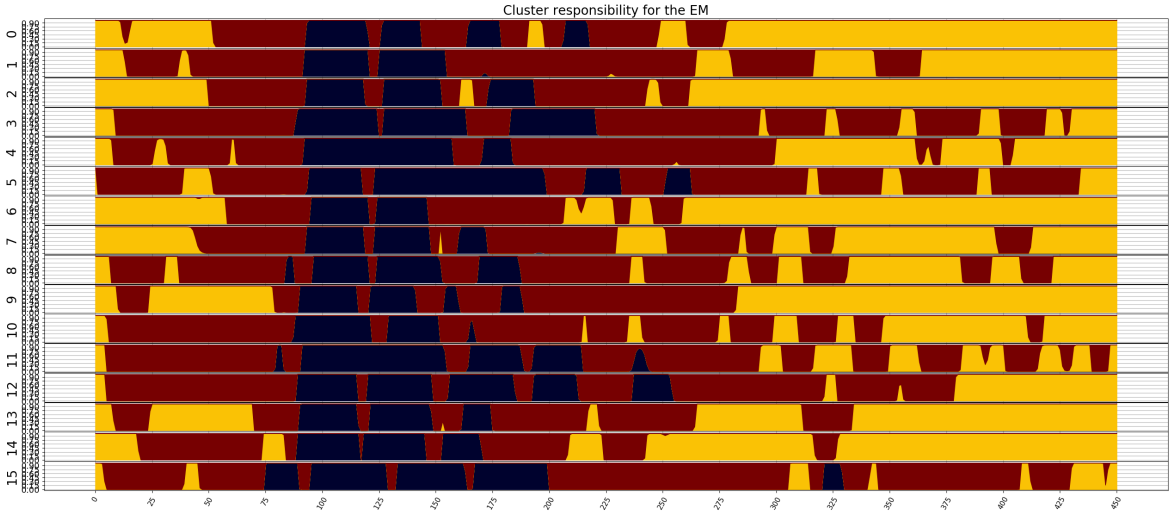

Figure 13: Time analysis for 3 Diagonal Gaussian clusters

For 4 Diagonal Gaussian clusters is harder to make conclusions, although the central cluster for R[2] and R[3] still seems valid. We can see how the red cluster mainly happens in R[1], the yellow cluster mainly happens in R[4] and the brown cluster mainly happens in R[5].

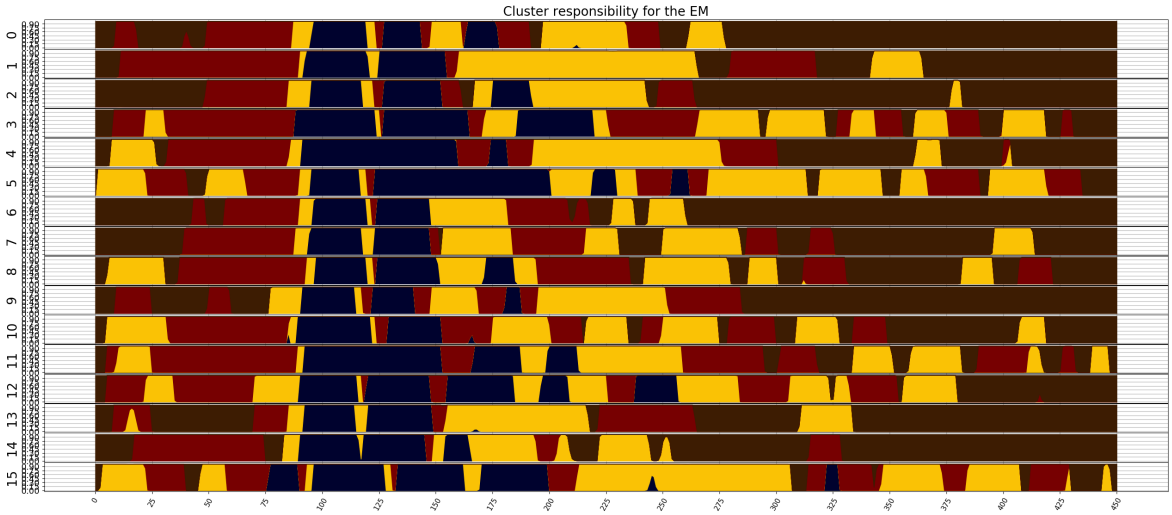

Figure 14: Time analysis for 4 Diagonal Gaussian clusters

For 5 Diagonal Gaussian clusters we see the same patters as before. In addition, we can also see how now the regions R[2] and R[3] are in different clusters instead of being in the same one, as it happened before. This behavior continues for a bigger number of clusters.

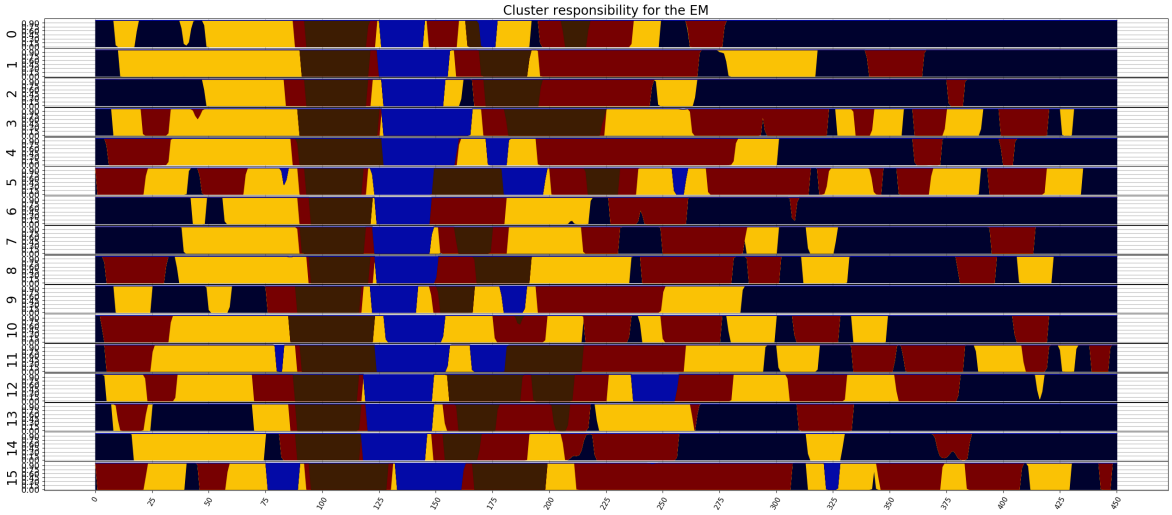

Figure 15: Time analysis for 5 Diagonal Gaussian clusters

2.3.1 von Mises Fisher distribution

For 2 von Mises Fisher Clusters, it seems like one cluster models the upper response R[2], and the other the lower response R[3]. The rest of the samples are just associated almost at random to them.

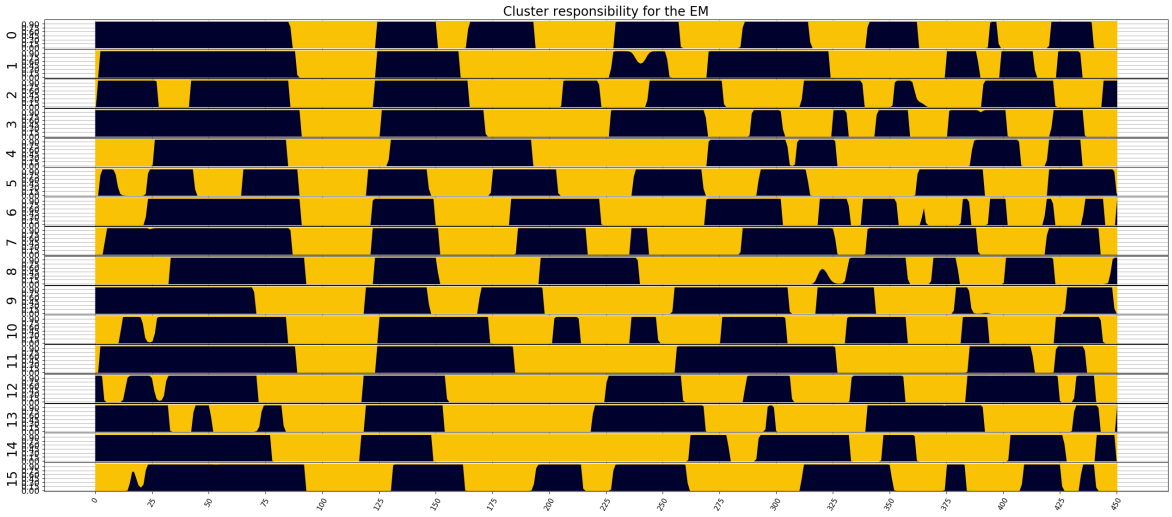

Figure 16: Time analysis for 2 von Mises Fisher Clusters

In the following 3 images we can see the responsibility for 3,4 and 5 clusters. Except for the fact that there is still a pattern in the regions R[2] and R[3] is hard to derive conclusions.

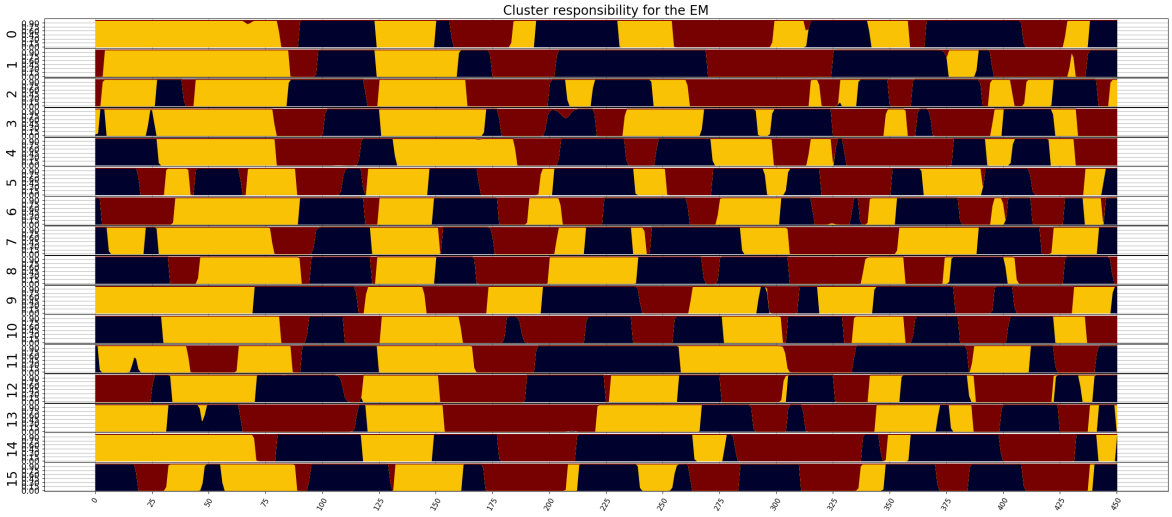

Figure 17: Time analysis for 3 von Mises Fisher Clusters

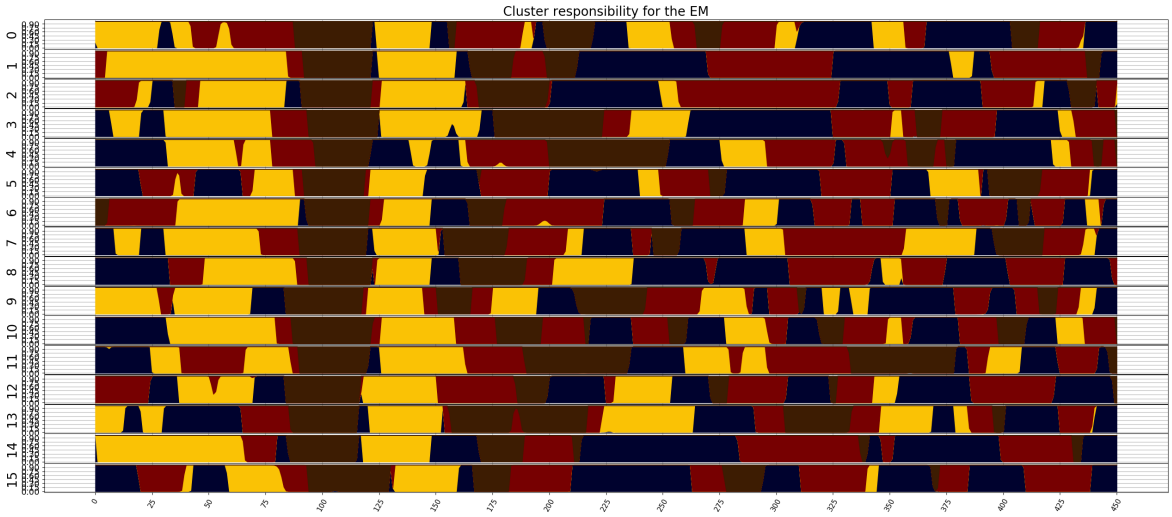

Figure 18: Time analysis for 4 von Mises Fisher Clusters

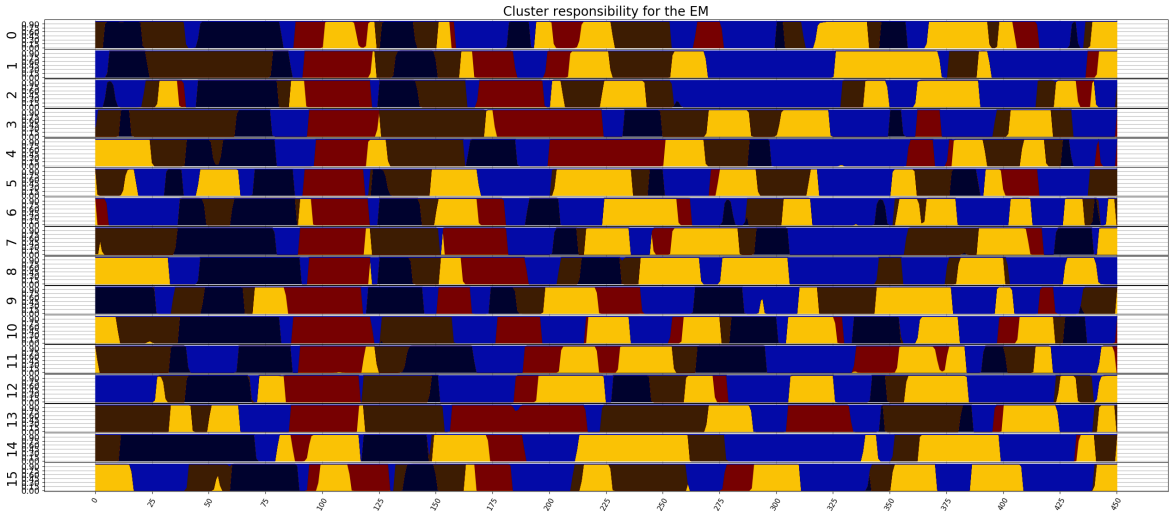

Figure 19: Time analysis for 5 von Mises Fisher Clusters

2.3.2 Watson distribution

For the Watson distribution and 2 clusters we can see a patter in the region R[2] and R[3], which appear to be associated do the dark blue cluster.

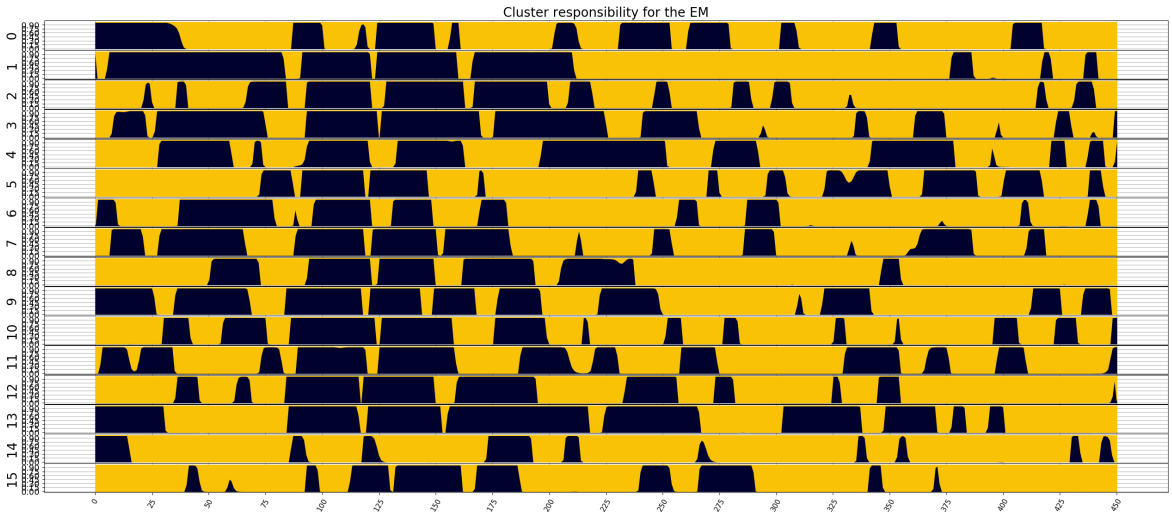

Figure 20: Time analysis for 2 Watson Clusters

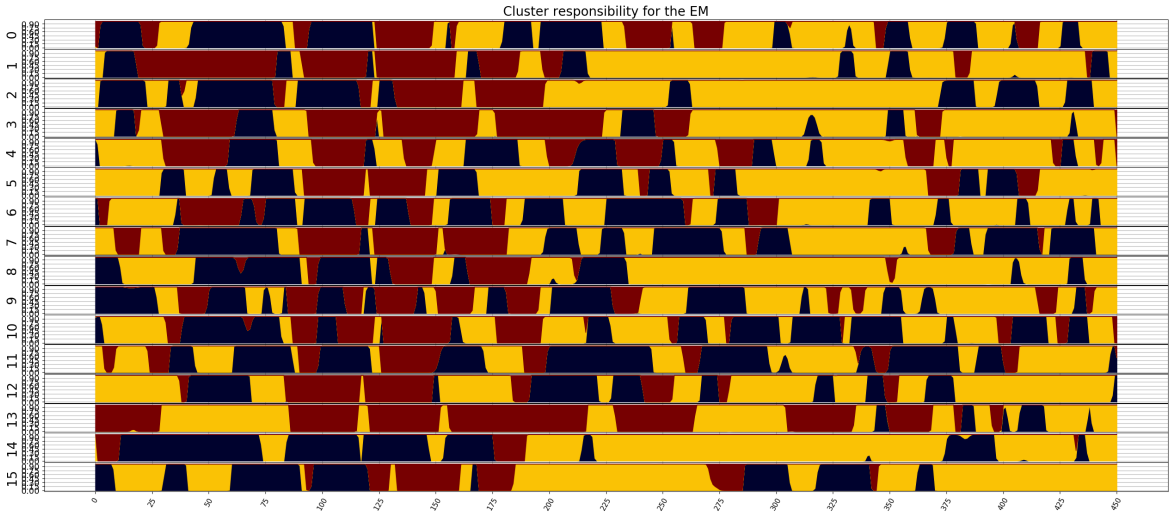

Figure 21: Time analysis for 3 Watson Clusters

For 3 clusters, it seems like:

- The brown cluster represents the high perturbances in R[2], R[3] and part of R[4]
- The yellow cluster represents more calmed periods
- The blue cluster does not really show a pattern.

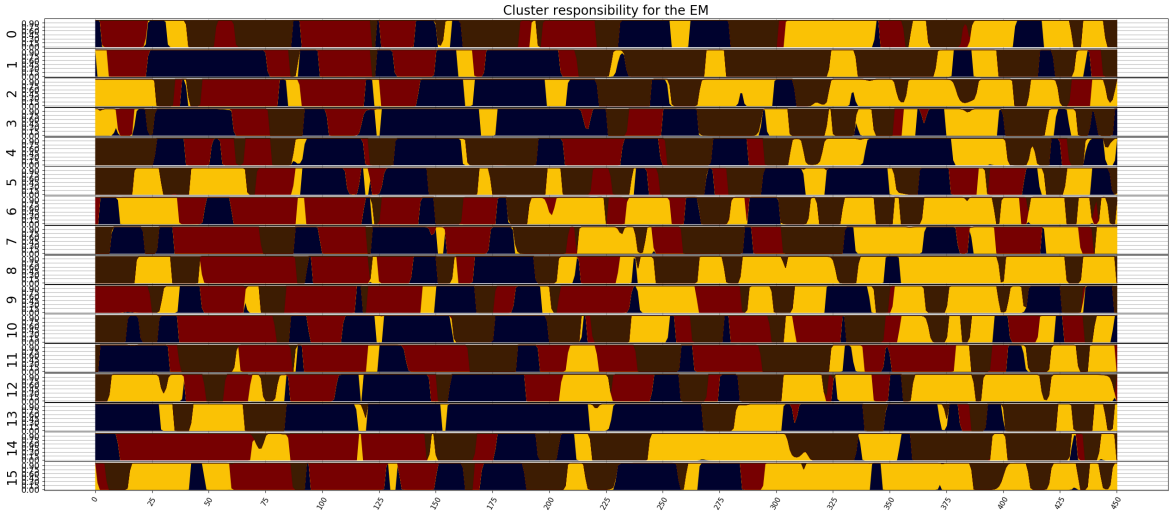

Figure 22: Time analysis for 4 Watson Clusters

For a bigger number of clusters it is hard to draw conclusions.

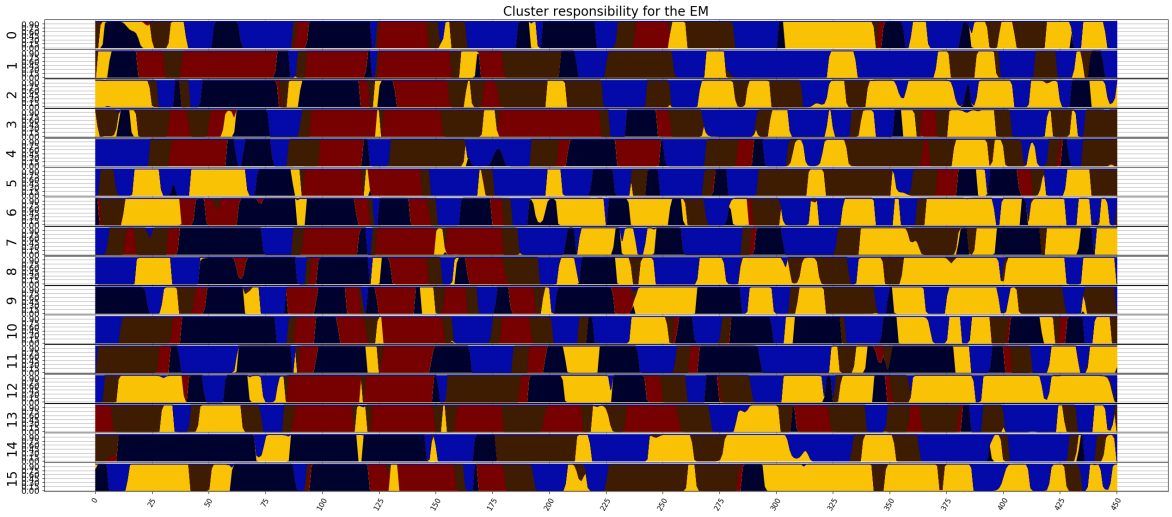

Figure 23: Time analysis for 5 Watson Clusters

2.4 Conclusions

The conclusions after running the tool over the EEG data are:

- The Gaussian distribution is the one that best expresses the data. Both in terms of likelihood and in the temporal interpretation of the clusters.
- The Directional distributions seem to capture the regions R[2] and R[3], which have a more intense pattern but they seem to fail to model the rest of the signal. A hypothesis is that since we have 70 dimensions, a small amount of noise could project the data point in random directions over the hypersphere.
- The optimal number of Gaussian clusters by LOO CV appears to be 4-6. It seems possible to draw relations between clusters and the time intervals of the signals.
- Using full Covariance matrix Gaussian clusters yields similar qualitative results to the ones using the Diagonal Covariance matrix, but it is more prone to overfitting.
- Since the likelihood of the samples is so high (and different among the clusters), the contribution of the Hidden Markov Model parameters is very low and it does not play a significant role.
- We cannot combine clusters of different distributions on the EEG data since the difference in likelihood is too high, therefore the clusters with lower likelihood will not be assigned any sample, falling into singularities. This holds true even combining Gaussian clusters with full covariance matrix with Gaussian clusters with Diagonal covariance matrix. This might be prevented with a good initialization but it is only a guess.
- Variational Inference could be used to solve the problem between unbalanced likelihoods of different distributions. Maybe also to give more importance to the transition matrix in the HMM implementation.

3 Variational Inference for the EM

In the following we will formulate the EM algorithm from Variational Inference perspective. In a clustering problem we want to maximize the likelihood of the data $\mathcal{D}$ given our model $\mathcal{M}$, that is, ideally we want to maximize $p(\mathcal{D}|\theta, \mathcal{M})$.

If our dataset contains N samples that we assume independent, we express it as $\mathcal{D} = [X_1, X_2, ..., X_N]$ where the samples X_i could be from point values, to vectors to chains. Then we can express the dataset as

In the usual EM algorithm we assume that each of the samples was drawn from a specific cluster k from the K possible ones, being each cluster a different distribution. We can model the cluster asociated to each sample as a hidden random variable Z the same way that it is done in Mixture

Models. The random variable Z is discretev with support $|Z| = 1, ..., K$, having an element per each of the clusters.

In order to compute the likelihood of the observed data $\mathcal{D}$ given the model, in the present of hidden random variables, we marginalize them away, obtaining what is known as the incomplete likelihood of the data. In a general scenario, we can express this marginalization as:

$$\begin{aligned} p(\mathcal{D}|\theta, \mathcal{M}) &= \int_Z p(\mathcal{D}, Z|\theta, \mathcal{M})dz = \int_Z p(\mathcal{D}|Z, \theta, \mathcal{M})p(Z|\theta, \mathcal{M})dz \\ &= \prod_{i=1}^{N} \int_Z p(X_i|Z, \theta, \mathcal{M})p(Z|\theta, \mathcal{M})dz \\ &= \prod_{i=1}^{N} E_{p(Z|\theta,\mathcal{M})}[p(X_i|Z, \theta, \mathcal{M})] \end{aligned} \tag{3.1}$$

Where we have that:

- $p(X_i|Z, \theta, \mathcal{M})$ is the probability (likelihood) of the sample i-th sample, given the model $\mathcal{M}$ that we are using, the parameters of the model θ and the value of the hidden variables Z.
- $p(Z|\theta, \mathcal{M})$ is the likelihood of the observed hidden variables Z given that we know the parameters θ and we are using the model $\mathcal{M}$.

The hidden variable Z represents the cluster from which each sample X_i was drawn. The variable is discrete, and therefore the marginalization over Z is a sum instead of an integral.

$$p(\mathcal{D}|\theta) = \prod_{i=1}^{N} \left(\sum_{k=1}^{K} p(X_i|Z = k, \theta)p(Z = k|\theta) \right) \tag{3.2}$$

We can identify the following terms in the equation:

- The first term is the likelihood that the sample X_i was generated from the distribution of the k-th cluster, which only depend on the parameters of the cluster so:

$$p(X_i|Z = k, \theta) = p(X_i|\theta_k)$$

- The second term is the probability of observing a sample from the k-th cluster. This only depends on the parameters π so we have:

$$p(Z = k|\theta) = p(Z = k|\pi_k) = \pi_k$$

  Notice that this term can be seen as a prior over the hidden variables Z. In the EM algorithm this is updated in each iteration.

In the original Expectation Maximization algorithm, we do not maximize the incomplete data log likelihood $(\mathcal{L})(\theta)$ but we split it into:

$$\begin{aligned} \mathcal{L}(\theta) = log(p(\mathcal{D}|\theta)) &= \sum_{i=1}^{N} log\left(\int_Z p(X_i|Z, \theta)p(Z|\theta)dz \right) \\ &= \sum_{i=1}^{N} log\Bigg(\underbrace{\sum_{k=1}^{K} p(X_i|Z = k, \theta)p(Z = k|\theta)}_{p(X_i|\theta)=g(\theta)} \Bigg) \end{aligned} \tag{3.3}$$

Going back our goal is to find θ that maximizes the incomplete likelihood of the data for each step. In the E-step we find the parameters that maximize the Expected complete data log likelihood with the current parameters. That is XXXXX

$$\begin{aligned}\mathcal{L}(\theta) = log(p(\mathcal{D}|\theta)) &= \sum_{i=1}^{N} log\Big(\int_Z p(X_i|Z,\theta)p(Z|\theta)dz\Big) \\ &= \sum_{i=1}^{N} log\Big(\underbrace{\sum_{k=1}^{K} p(X_i|Z=k,\theta)p(Z=k|\theta)}_{p(X_i|\theta)=g(\theta)}\Big)\end{aligned} \tag{3.4}$$

In order to compute it, we play the following trick. Since the incomplete data loglikelihood is computed as the expectation oe
. In this case, this is the same as maximizing its Log, which turns the product of the independent samples into a sum

$$\begin{aligned}\mathcal{L}(\theta) = log(p(\mathcal{D}|\theta)) &= \sum_{i=1}^{N} log\Big(\int_Z p(X_i|Z,\theta)p(Z|\theta)dz\Big) \\ &= \sum_{i=1}^{N} log\Big(\underbrace{\sum_{k=1}^{K} p(X_i|Z=k,\theta)p(Z=k|\theta)}_{p(X_i|\theta)=g(\theta)}\Big)\end{aligned} \tag{3.5}$$

For computing the best parameters we can derive and equal to 0. But in every case we will need to use the posterior probability of the hidden variable. The derivate with respect to the $j-th$ parameter belonging to the cluster k is:

$$\frac{\partial \mathcal{L}(\theta)}{\partial(\theta_{jk})} = \sum_{i=1}^{N} \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial(\theta_{jk})} = \sum_{i=1}^{N} \frac{1}{p(X_i|\theta)} \frac{\partial}{\partial(\theta_{jk})} \big(p(X_i, Z=k|\theta)\big) \tag{3.6}$$

Using Lagrange multipliers we obtain the update rules as a function of the responsibilities, that is, the posterior probability of the clusters:

$$Equation updates.$$

In the EM algorithm we can compute the exact posterior $p(Z|X,\theta)$ since we can compute the individual likelihoods $p(X_i|\theta)$ by means of marginalizing over Z which is discrete and we can compute everything. If instead of the In order to formulate the Variational Inference scheme we can multiply and divide the marginalization of the hidden variables by another probability function $q(Z|X_i,\theta)$ since they would cancel out. This probability distribution could be any, but as we will see later it will be replacing the posterior probability of the hidden variables.

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} log\Big(\int_Z \frac{p(X_i,Z|\theta)}{q(Z|X_i,\theta)} q(Z|X_i,\theta)dz\Big) \tag{3.7}$$

Using **Jensens Inequality**, we can introuduce the logarithm only in the fraction yielding a lower bound of the incomplete data log likelihood:

$$\mathcal{L}(\theta) \leq \sum_{i=1}^{N} \int_Z log\Big(\frac{p(X_i,Z|\theta)}{q(Z|X_i,\theta)}\Big) q(Z|X_i,\theta)dz \tag{3.8}$$

This $q(Z|X_i,\theta)$ is considered our approximation of the posterior distribution of the hidden variables. And it can be different for each sample X_i. For the EM algorithm this means that we can compute the responsibilities according to $q(Z|X_i,\theta)$ instead of the actual $p(Z|X_i,\theta)$, as long as $q(Z|X_i,\theta)$ follows the laws of probability and depends only on the sample X_i.

We can further express this lower bound of the incomplete log likelihood in terms of the expectation over the variational posterior. Using the chain rule we obtain:

$$\begin{aligned}\mathcal{L}(\theta) &\leq \sum_{i=1}^{N} E_{q(Z|X_i,\theta)}\left[\left(\frac{p(X_i|Z,\theta)p(Z|\theta)}{q(Z|X_i,\theta)}\right)\right] \\ &= \sum_{i=1}^{N} E_{q(Z|X_i,\theta)}\left[\underbrace{log(p(X_i|Z,\theta))}_{\text{LogLike}} + \Big(log(p(Z|\theta)) - log(q(Z|X_i,\theta))\Big)\right]\end{aligned} \tag{3.9}$$

This lower Bound for the incomplete log likelihood is called the Free energy $\mathcal{F}(\theta)$ and it has 2 terms:

- The expected log likelihood of the data over the approximate distribution $q(Z|X_i,\theta)$, commonly called variational posterior. Notice this
- The KL distance between the prior and variational posteior of the hidden variables.

3.1 Usefulness of the Variational Inference

In the basic implementation of the EM algorithm, we can compute the the exact posterior distribution $p(Z|X_i,\theta)$ so we actually do not need to approximate it using a variational posterior $q(Z|X_i,\theta)$. Nevertheless it gives us a framework to be able to modify the basic algorithm in order to fit it to our needs.

In particular there are 3 things we would like to improve upon:

- The high dimensionality makes the likelihoods to extreme to be able to have smooth transitions between clusters.
- The difference of likelihood between clusters of different distributions is too extreme to be able to combine them.

We could obtain a variational posterior by simply modifying the probabilities of the clusters individually as in the following equation:

$$q(Z=z|X_i,\theta) = \frac{g_z(p(X_i|Z=z,\theta))p(Z=z|\theta)}{\sum_{k=1}^{K} g_k(p(X_i|Z=k,\theta))p(Z=k|\theta)} \tag{3.10}$$

Where $g_1(), g_2()..., g_K()$ non-negative monotonically increasing functions. This way, the variational posterior obeys the laws of probability and maximization $g_k(p(X_i|Z=k,\theta)$ is also a maximization of the original cluster probability $p(X_i|Z=k,\theta)$. Examples of such functions could be multiplication, exponentiation, logarithms,,,

3.2 Toy Example

Using the toy example we can for example modify the likelihood of the Gaussian cluster, therefore varying the number of samples associated to it. The following 3 Figures show the clusters and responsibilities found for the toy example data where we have multiplied the likelihood function of the Gaussian cluster by a scalar value.

From the Figures we can see that when we multiply divide the likelihood by a 3, less samples are associated to the Gaussian cluster and they are assigned more intensely (since the Gaussian likelihood function decreases at the speed of a negative squared exponential). The opposite happens when we multiply it by 2, where more samples are associated, but less intensely. The incomplete log likelihood shown uses the modified likelihood functions, that is why it is lower when we divide by 3, and higher when we multiply by 2. This is not a proper likelihood, we probably should obtain the one using the real probabilities, since the modified probability functions do not add up to 1, only the variational posterior. The goal of this variational posterior is to find better clusters using the same original distributions, which differs from the common uses of it. It should be considered a modification in the responsibility space, not in the original cluster likelihoods space.

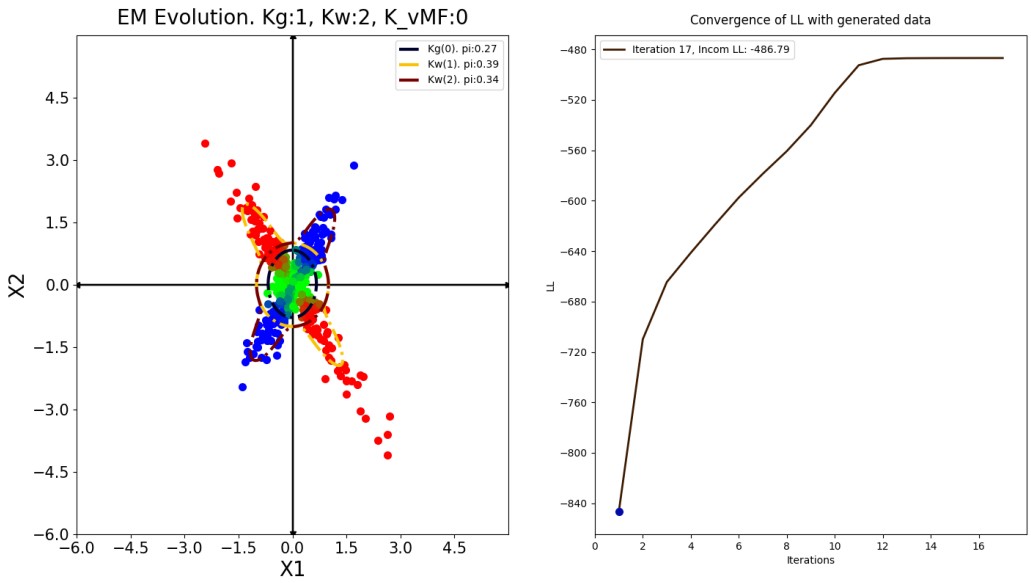

Figure 24: EM for 1 Gaussian and 2 Watson clusters without modifying the Gaussian cluster

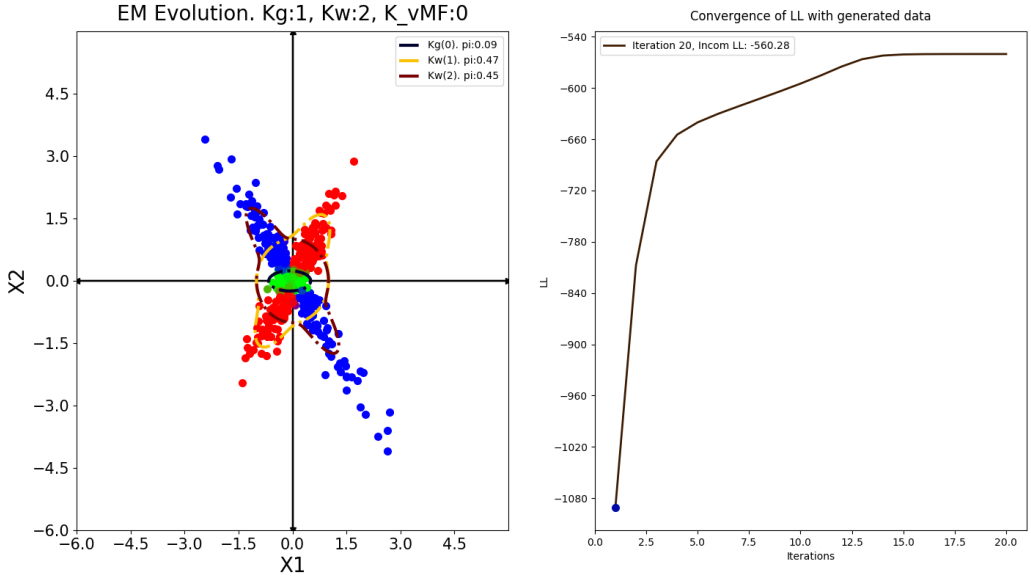

Figure 25: EM for 1 Gaussian and 2 Watson clusters dividing the Gaussian cluster likelihood by 3

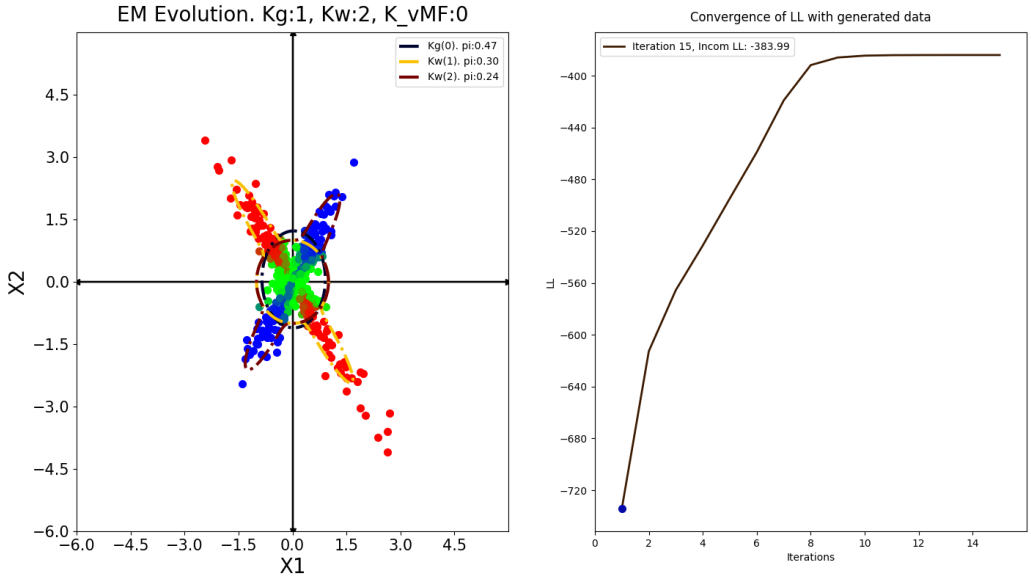

Figure 26: EM for 1 Gaussian and 2 Watson clusters multiplying the Gaussian cluster likelihood by 2

3.3 On the EEG data

A possible use over the EEG data to reduce the extreme values of likelihood given by the high dimensions is to take the n-root of the individual likelihoods. This will bring them together, creating smoother transitions between clusters.

For example, since we have 70-dimensional data, if we take the 70-root of the likelihood of the clusters and we apply the EM algorithm for 2 Diagonal Gaussian clusters we obtain the following responsibility time-analysis Figure. We can see now how the transitions are smoother.

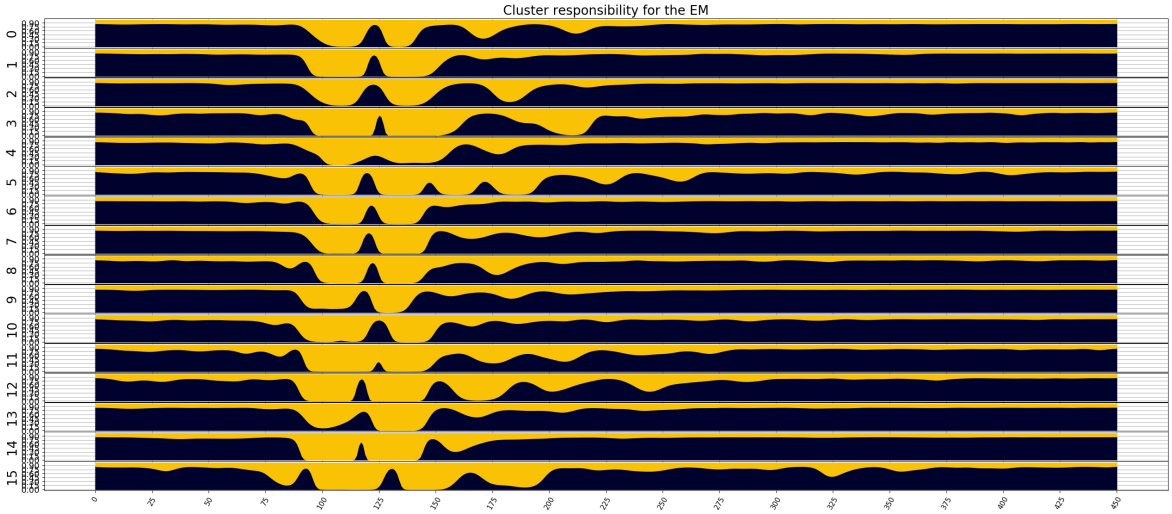

Figure 27: EM for 2 Diagonal Gaussian clusters

If we use 3 Diagonal Gaussian clusters and only take the 14-th root then we obtain the following chart.

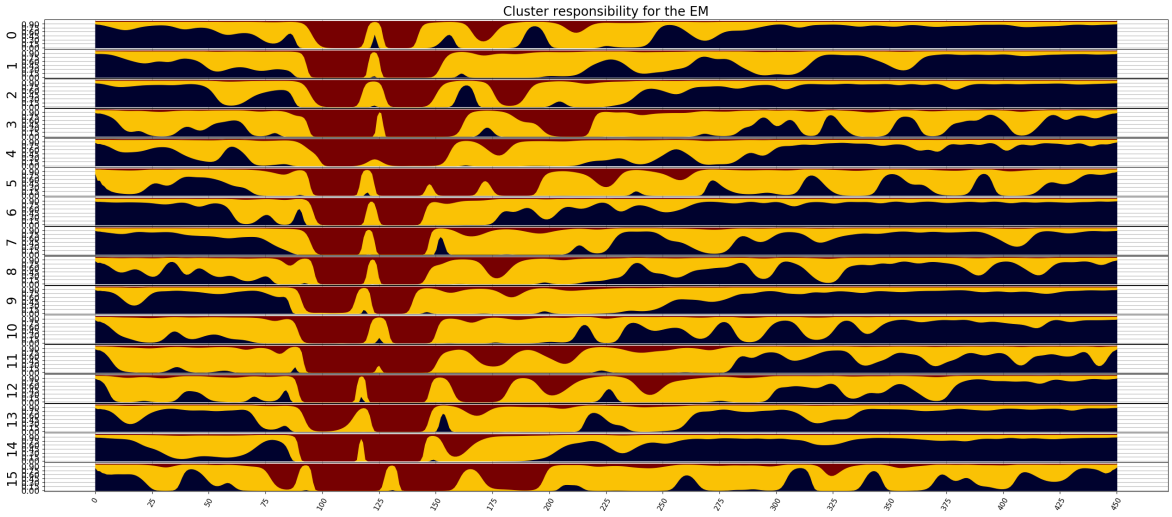

Figure 28: EM for 3 Diagonal Gaussian clusters

The overall analysis of the clusters is the same, we do not obtain different clusters, just smoother transitions in the responsibility space and smoother evolution of the incomplete data log likelihood in the iterations of the algorithm.