Danmarks Tekniske Universitet

Special Course

Probabilistic Clustering

Manuel Montoya Catalá, Arturo Arranz

Last modified January 30, 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. In this document we will often reference it as the HMM algorithm.
- There are 3 distributions already implemented in the tool. The Gaussian (both full and diagonal covariance matrix), Watson and vonMissesFisher distributions.
- Clusters from different distributions can be combined in this implementation of the EM algorithm.
- All the characteristics of the distributions (initialization, estimation, singularity handling...) can be tuned using hyperparameters given using a dictionary.
- A user can easily modify any of the implemented distributions to include any desired constraint.
- A user can easily introduce a new distribution by just indicating a few functions about it. We will elaborate on this later.
- The output of the training algorithm includes the evolution of the cluster parameters through the iterations, as well as the incomplete data log likelihood and the model parameters.
- The code is optimized avoiding to use for loops whenever possible.
- Everything is implemented in logarithms in order to handle extreme values of the likelihoods.
- Several Code examples of use, creation of Figures and Cross-validation of hyper-parameters are provided.
- The implementation can handle the two main crashing reasons of the EM algorithm:
  - Singularities, also called degenerated clusters. This situation happens when a not enough samples are associated to a cluster and we cannot estimate its parameters.
  - Extreme normalization constant. If the parameters of the distribution are too odd, the computing of the normalization constant can be intractable (or take too much time).
- The modular structure of the implementation makes it easy to include priors on the parameters and a Variational Posterior on the posterior probability of the clusters.
- The CEM class comes with methods to compute the incomplete data log-likelihood, responsibility, and decoding of a given set of data provided as input.
- The algorithm includes the possibility of different levels of verbose and timing analysis.

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.

1.1 Interface of the tool

In this Section we will describe the process of initializing and using the tool, including what how to call the different methods and what is the expected format of the input, and the format of the output.

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 below 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 access the functions of the library. During the initialization we provide the object with the configuration of the EM algorithm, but not the data. 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, but we do not provide the input data. The output of the initializer is of course an object of the class set will all the parameters given in the input. As remarks:

- The "clusters_relation" variable is used to switch transparently between the EM ("independent") and the BW ("MarkovChain1") algorithms, sharing both algorithm the same interface for all configuration parameters, inputs 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)
```

1.2 Interface of a distribution class

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 of a distribution, it is only used internally by the algorithm so it will not be described. Its purpose is making easier the managing of the logic for executing the functions of each active cluster.

Each distribution is defined by a set of functions and a set of hyperparameters provided 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 set of basic functions that are needed to be specified for each distribution are described in the following. A come complete description can be found inside the code, here we will only describe their functionality.

- *pdf_log_K(X, theta, Cs_log = None)*: This function is the probability density function (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) simultaneously. For optimization purposes, if the normalization constants are precomputed, they can be given as input. The output of the function is a NxK matrix with the likelihood of all the samples to all clusters.

- *get_Cs_log(theta_k, parameters = None)*:This function will compute the normalization constant of a cluster. Usually, the normalization constant is a computational bottleneck since it could take quite some time to compute. In an iteration of the EM algorithm, 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 parameters have 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 responsibility 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 reinitialize it using other hyperparameters.

- *degenerated_params_handler(X, rk , prev_theta_k , parameters = None)*: If 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, it is 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 hyperparameters.

- Then we **specify the distribution** functions. We can do this by either providing the ones implemented by the user, or choose one of the already implemented 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 of 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 of 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 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 of 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 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 an artificially generated dataset composed by 3 Gaussian clusters.

Since the library allows for different distributions to be combined, this Example Code makes use of this feature, being able to combine for example Watson and von Misses Fishers clusters.

In the **first example** we will show how the algorithm estimates 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 data log likelihood through the iterations.

In order to visualize the clusters better, they are represented in the Figures in the following manner:

- For Gaussian distributions we plot the 95% confidence interval ellipse of the 2D Gaussian.
- For the Watson and vonMisesFisher distributions, we plot a circumference of radius 1 plus the likelihood associated to each angle of the cluster. Which means that at any given angle ϕ, in the circumference defined but the directional distributions in 2D, we will plot the pdf of the distribution for that angle on top of the circumference.

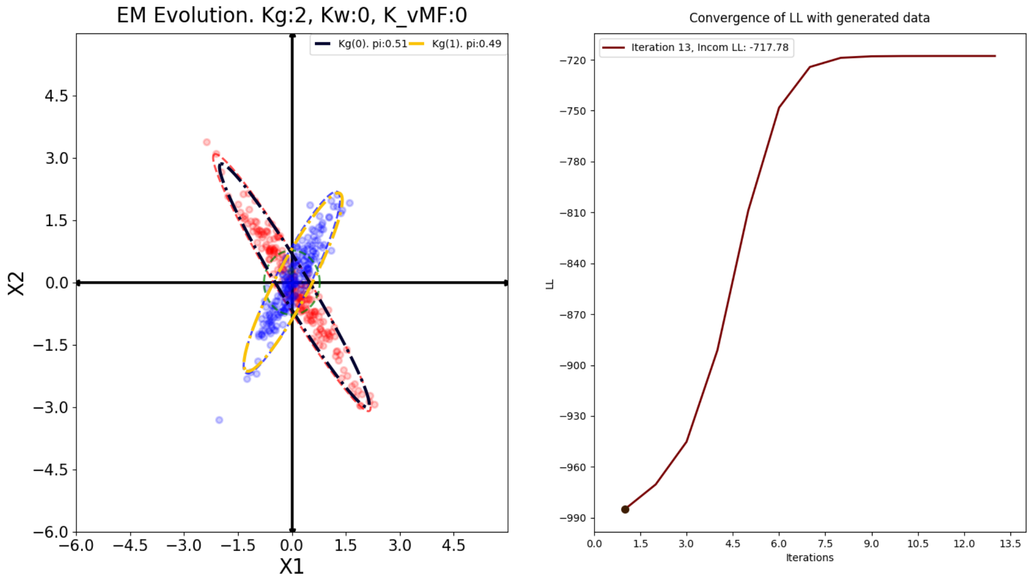

Figure 1: EM for 2 Gaussian clusters on the toy example data

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.

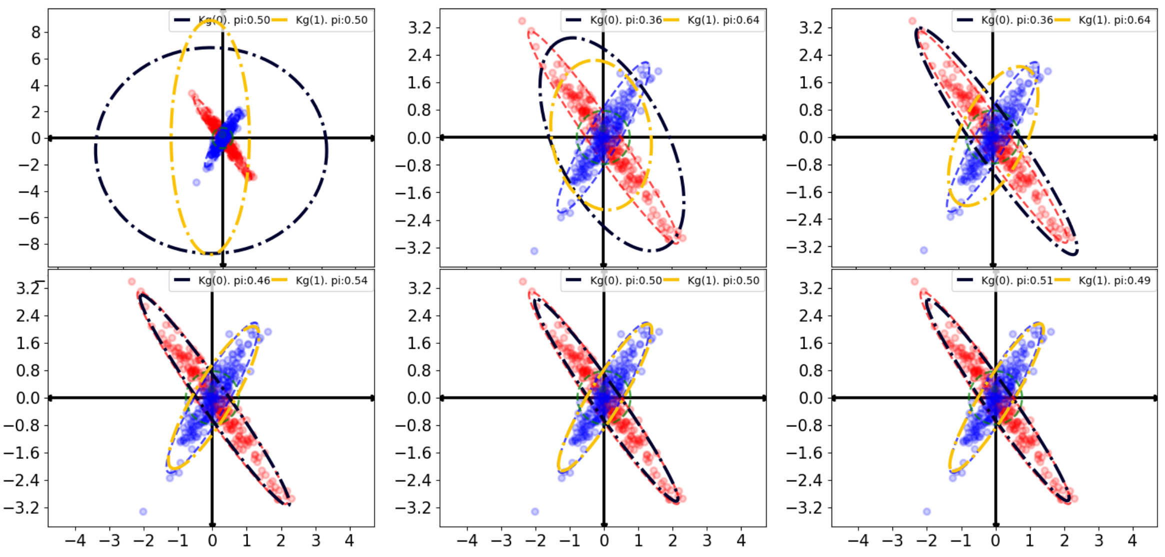

Figure 2: Iterations of the EM for 2 Gaussian clusters on the toy example data

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.

2 Theoretical background

In this section the implement algorithms are explained in detail. For doing so, mixture of Watson distributions is used. The reader might find easy to generalize to other distributions once this case is understood.

2.1 Expectation-Maximization algorithm

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_T \in \mathbb{S}^{p-1}\}$ be a sequence of i.i.d samples. If we want to model the data with K Watson distributions, let us denote $W_p(\boldsymbol{x}|\boldsymbol{\mu}_i, \kappa_i)$ as the probability density distribution of the i-th cluster and π_i the prior probability of the cluster. Then the observation $\boldsymbol{x}_i$, which is a column vector, has the probability density

$$f(\boldsymbol{x}_t|\boldsymbol{\mu}_1, \kappa_1, \pi_1..., \boldsymbol{\mu}_K, \kappa_K, \pi_K) = \sum_{i=1}^{K} \pi_i W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \kappa_i)$$

Then the log-likelihood for the entire dataset X is given by

$$\ell(\boldsymbol{X}|\boldsymbol{\mu}_1, \kappa_1, \pi_1..., \boldsymbol{\mu}_K, \kappa_K, \pi_K) = \sum_{t=1}^{T} log\Big(\sum_{i=1}^{K} \pi_i W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \kappa_i)\Big) \tag{2.1}$$

In order to calculate the parameters which maximize (2.1) the iterative method EM ([1]) can be used, which makes the assumption of independence between samples. In every iteration two steps are solved, the *Expectation-step* where the *responsibilities* of each sample is calculated i.e the posterior probability for each sample t to belong to the cluster i and the *Maximization-step* where the parameters are recalculated.
E-step:

$$r_t(i) = \frac{\pi_i W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \kappa_i)}{\sum_l \pi_l W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \kappa_l)} \tag{2.2}$$

M-step:

$$\begin{aligned} &\mu_i = s_1^j \quad if \quad \kappa_i > 0, \qquad \mu_i = s_p^i \quad if \quad \kappa_i < 0 \\ &\kappa_i = g^{-1}(1/2, p/2, r_i), \quad where \quad r_i = \mu_i^T S^i \mu_i \\ &\pi_i = \frac{1}{T}\sum_i r_t(i), \end{aligned} \tag{2.3}$$

where s_e^j represents the e-th eigenvector of the *weighted scatter matrix*:

$$S^i = \frac{1}{\sum_t r_t(i)} \sum_t r_t(i) \boldsymbol{x}_t \boldsymbol{x}_t^T \tag{2.4}$$

The algorithm iterate (2.2) and (2.3) until the chosen convergence criteria is fulfilled.

2.2 Expectation-Maximization algorithm for HMM

In this version of moW, we assume that the data sequence follows a Markov Chain of order 1. In this case, the probability of a sample being generated from a given cluster, depends on the cluster that the previous sample was generated from. In this section we will denote clusters also as "states" since these clusters now contain time information.
Therefore in this model we have an initial probability vector π, and a transition probability matrix, A being a_{ij} the element in the i-th row and the j-th column meaning the probability of jumping to state j when being at state i.

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1I} \\ a_{21} & a_{22} & ... & a_{2I} \\ ... & ... & a_{ij} & ... \\ a_{I1} & a_{I2} & ... & a_{II} \end{bmatrix} \qquad \pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ ... \\ \pi_I \end{bmatrix} \tag{2.5}$$

Adding the parameters of the clusters, the whole model is characterized by the set of parameters $\theta = \{\pi, A, B\}$ being B the parameters of the Watson distribution for the different K clusters, $B = \{\mu_k, \kappa_k\}_{k=1}^K$
The data consists of a set of N independent time sequences $Y = \{y^{(1)}, ..., y^{(N)}\}$ each of which has T_n samples , $y^{(n)} = \{x_1^{(n)}, x_2^{(n)}, ..., x_{T_n}^{(n)}\}$. We assume that each of the samples was generated from one of the I Watson distributions, but we do not have this information. We model the cluster from which each sample was generated by means of a latent variable s_t^n that can take discrete values from 1 to I corresponding to the cluster that generated the sample. This means that if we had complete information about the data, we would have data pairs $\{x_t^n, s_t^n\}$. The EM algorithm will use the incomplete data to estimate the model parameters θ in an iterative fashion, in order to do so it will compute the discrete distribution of each latent variable s_t^n given all the information available.
So, in order to compute the equations of the EM algorithm we need to compute the probability that the sample x_t was generated from the i-th cluster, that is the same as saying that the chain sequence is in the state i at the time t given all available information $P(s_t = i|Y, \theta)$. This is a nontrivial computation and it is done using the Forward-Backward Algorithm described in the next subsection.

2.2.1 Forward Backward Algorithm

The forward-backward algorithm is an algorithm that calculates the probability of being in the state i at the time t for a given sequence given all the observations and the model, $P(s_t = i|Y, \theta)$ using Dynamic Programming (reuses earlier computations stored in memory).
From now on, we will simplify the notation omitting the conditioning on the parameters θ, it just means that the model is already set, so just remember that all the parameters are given. We also denote the subvector $y_{a:b}$ as the vector of observations from time a to time b, being $y_{a:b} = \{x_a, x_{a+1}, ..., x_b\}$
Using Bayes Theorem we have:

$$P(s_t = i|Y) = \frac{P(s_t = i, Y)}{P(Y)} \propto P(s_t = i, Y) \tag{2.6}$$

We do not need to compute the normalization constant $P(Y|\theta)$ directly because we know that the distribution has to add up to 1 and its value does not depend on i, so we compute the constant later as:

$$P(Y) = \sum_{i=1}^{I} P(s_t = i|Y) \tag{2.7}$$

We can split this probability into 2 factors:

$$P(s_t = i|Y) = P(s_t = i, y_{1:t})P(y_{t+1:T}|s_t = i) \tag{2.8}$$

Where $P(s_t = i, y_{1:t})$ is the forward component, the probability of being in the state i at time t and having observed the current and past data points. And $P(y_{t+1:T}|s_t = i)$ is the backward component, the probability of observing the future sequence when we are in the state i at time t. We will use these probabilities to compute any arbitrary inference, parameters estimation and sampling of the posterior.
We will not got too much into the detail of the derivation of these probabilities. We will explain the process and recursion that is obtained.
For the forward component, we obtain the probability by means of recurrent marginalization since we have a Makov dependency of order 1. So, for the first step, we obtain $P(s_t = i, y_{1:t})$ as marginalization of $P(s_t = i, s_{t-1}, y_{1:t})$ computed as:

$$\begin{aligned} P(s_t = i, y_{1:t}) &= \sum_{j=1}^{I} P(s_t = i, s_{t-1} = j, y_{1:t}) \\ &= P(x_t|s_t = i) \sum_{j=1}^{I} P(s_t = i|s_{t-1} = j)P(s_{t-1} = j, y_{1:t-1}) \end{aligned} \tag{2.9}$$

Where the factor components of the probability are:

- The probability that the sample x_t is generated from the state i, $P(x_t|s_t = i) = p_i(x_t|B_i) = W(x_t|\mu_i, \kappa_i)$
- The probability of going from state j to state i, $P(s_t = i|s_{t-1} = j) = a_{ji}$
- The probability of being in the state j at time $t - 1$ and having observed the current and past data points $P(s_{t-1} = j, y_{1:t-1})$. As we can observe this has the same form as the probability we are computing. From this fact we obtain the recurrence used to compute the forward step.

Without loss of generalization, we denote the forward component for the n-th chain at time t for the state i as $\alpha_t^n(i)$ and so we obtain the equations:

$$\begin{aligned} \alpha_1^n(i) &= \pi_i \cdot p_i(x_1^n|B_i) = \pi_i \cdot W_i(x_1^n|\mu_i, \kappa_i) \\ \alpha_t^n(i) &= p_i(x_t^n|B_i) \sum_{j=1}^{I} a_{ji}\alpha_{t-1}^n(j) \end{aligned} \tag{2.10}$$

Note that when we follow the recursion to the beggining of the chain at time $t = 1$, the do not need to marginalize over the previous state, since the probability of the states is given by the initial probability vector π.

For the Backward component, the process is similar, we obtain the probability by means of recurrent marginalization over s_{t+1} instead of s_{t-1} So, for the first step, we obtain $P(y_{t+1:T}|s_t = i)$ as marginalization of $P(y_{t+1:T}s_{t+1}|s_t = i)$ computed as:

$$\begin{aligned} P(y_{t+1:T}|s_t = i) &= \sum_{j=1}^{I} P(y_{t+1:T}, s_{t+1}|s_t = i) \\ &= \sum_{j=1}^{I} P(s_t = i|s_{t+1} = j)P(x_{t+1}|s_{t+1} = j)P(y_{t+2:T}|s_{t+1} = j) \end{aligned} \tag{2.11}$$

Where the factor components of the probability are:

- The probability that the sample x_{t+1} is generated from the state j, $P(x_{t+1}|s_{t+1} = j) = p_j(x_{t+1}|B_j) = W(x_{t+1}|\mu_j, \kappa_j)$
- The probability of going from state i to state j, $P(x_{t+1}|s_{t+1} = j) = a_{ij}$
- The probability of observing the future sequence after $t + 1$ when we are in the state j at time $t + 1$ $P(y_{t+2:T}|s_{t+1} = j)$. As we can observe this has the same form as the probability we are computing. From this fact we obtain the recurrence used to compute the backward step.

Without loss of generalization, we denote the backward component for the n-th chain at time t for the state i as $\beta^n_t(i)$ and so we obtain the equations:

$$\begin{aligned} &\beta^n_T(i) = 1 \\ &\beta^n_t(i) = \sum_{j=1}^{I} a_{ij}p_j(x^n_t|B_j) \cdot \beta^n_{t+1}(i) \end{aligned} \tag{2.12}$$

Note that when we follow the recursion to the end of the chain at time $t = T$, we do not have a next step to marginalize and the probability is 1.

Finally, from the computation of $\alpha^n_t(i)$ and $\beta^n_t(i)$ for a given chain in a recurrent way storing the previous values (Dynamic programming) we can finally obtain the probability of being in the state i at time t as:

$$P(s_t = i|Y) = \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(Y)} \propto \alpha_t(i)\beta_t(i) \tag{2.13}$$

Where we denote $\gamma^n_t(i)$ as the probability of being in the state i at time t for the chain n. As we will see later, this probability will be used for further inference. Notice that we obtain the normalization constant $P(Y)$ by means of (2.2.2). Notice the analogy of this parameter to the responsibility parameter $r_t(i)$, they both have the same meaning.

2.2.2 Likelihood of a sequence

Given a sequence $y(n)$ of T_n samples, if we knew the state from which each sample x_t was generated, s_t we could compute the complete likelihood L_c of the data as the likelihood of the individual data points being generated from their state multiplied by the probability of observing that transition sequence.

$$L_c(y^{(n)}, s^{(n)}|\theta) = P(s^{(n)}|\theta)P(y^{(n)}|s^{(n)}, \theta) = \left(p(s_1|\pi)\prod_{t=2}^{T_n} p(s_t|s_{t-1}, A)\right) \cdot \left(\prod_{t=1}^{T_n} p(x_t|s_t, B))\right) \tag{2.14}$$

Since the state from which each sample was drawn is unknown to us, it is latent variable, we cannot compute this likelihood, we compute the incomplete likelihood instead, the likelihood of observing the chain, given the model, this likelihood can be obtained marginalizing the complete likelihood over all the possible states:

$$L(y^{(n)}|\theta) = \sum_{s_1=1}^{I}\sum_{s_2=1}^{I}\dots\sum_{s_{T_n}=1}^{I} P(y^{(n)}, s^{(n)} = \{s_1, s_2, \dots s_{T_n}\}) \tag{2.15}$$

This would require the sum of I^{T_n} elements. Another way to compute such likelihood is using by means of equation , which is computed from the Forward Backward algorithm. Due to dynamic programming nature of the algorithm we can compute this probability with the sum of only $I^2 T_n$ elements, making the algorithm feasible in practice. We can compute this likelihood as:

$$L(y^{(n)}|\theta) = \sum_{i=1}^{I} P(s_t = i|Y) = \sum_{i=1}^{I} \alpha_T^n(i) = \sum_{i=1}^{I} \beta_1^n(i) \cdot \pi_i \cdot p_i(x_1|B_1) \tag{2.16}$$

2.2.3 Transition probabilities

For the computation of the transitions probabilities we also need to compute the probability that we in the state j at time t and in the state i at time $t-1$. We denote this probability as:

$$\xi_t^n(i,j) = P(s_t^n = j, s_{t-1}^n = i|Y^{(n)}, \theta) \tag{2.17}$$

We can compute this probability from the Forward-Backward algorithm as:

$$\xi_t^n(i,j) \propto \alpha_t^n(i) a_{ij} p_j(x_{t+1}^n|B_j) \beta_{t+1}^n(j) \tag{2.18}$$

The normalization constant is obtained by marginalization over i and j.

2.2.4 E - step

In the E-step we derived the previous equations referring to probabilities that will be used later to weight the contribution of each cluster to each sample and the transition probabilities. In this step we basically compute the data structures:

$$\begin{array}{llll}
\alpha_t^n(i) & n = 1, \dots, N & t = 1, \dots, T^{(n)} & i = 1, \dots, I \\
\beta_t^n(i) & n = 1, \dots, N & t = 1, \dots, T^{(n)} & i = 1, \dots, I \\
\gamma_t^n(i) & n = 1, \dots, N & t = 1, \dots, T^{(n)} & i = 1, \dots, I \\
\xi_t^n(i,j) & n = 1, \dots, N & t = 1, \dots, T^{(n)} & i, j = 1, \dots, I
\end{array} \tag{2.19}$$

2.2.5 M - step

In the maximization step we will estimate the new model parameters for the iteration $r+1$ of the EM algorithm, namely θ^{r+1} using the past parameters θ^r and the updated parameters values $\{\gamma_t^n(i), \xi_t^n(i,j)\}$ from the E-step
The initial state probabilities π_i are obtained as:

$$\pi_i = \frac{N_i}{N} = \frac{1}{N}\sum_{n=1}^{N} \gamma_1^n(i) \qquad N = \sum_{i=1}^{I}\sum_{n=1}^{N} \gamma_1^n(i) \tag{2.20}$$

The transition probability elements a_{ij} from matrix A are obtained as:

$$a_{ij} = \frac{E_{ij}}{E_i} = \frac{1}{E_i}\sum_{n=1}^{N}\sum_{t=2}^{T_n} \xi_t^n(i,j) \qquad E_i = \sum_{i=1}^{I}\sum_{n=1}^{N}\sum_{t=2}^{T_n} \xi_t^n(i,j) \tag{2.21}$$

Since the distributions of the mixture $p_i(x|B_i)$ belong to the exponential family, we can obtain its parameters B_i from moment matching with the formula:

$$E\{\phi(Y)\} = \frac{1}{\Gamma_i}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_t^n(i)\phi(x_t^n) \qquad \Gamma_i = \sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_t^n(i) \tag{2.22}$$

And therefore the equations for the i-th Watson distribution can be computed from their weighted scatter matrix obtained as:

$$S^i = \frac{1}{\Gamma_i}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\gamma_t^n(i)x_t^{(n)}x_t^{(n)^T} \tag{2.23}$$

This is weighted scatter matrix computed from all samples of all chains. The μ_i and κ_i parameters are obtained in the same way as explained in the previous sections.

3 Experiments on EEG data

3.1 Microstates degmentation state of the art

One method for measuring the brain activity is the so called Electroencephalography (EEG) which consist of temporal recordings of the scalp surface electric fields. Evoked potentials (EPs) are a derivative of EEG techniques which involves averaging EEG measurements associated to some kind of stimuli at fix time. An important issue which neuroscience try to solve is relating this temporal sequences to functional states of the mind. A logical hypothesis is to assume that mental states remain for a short time of period which D. Lehmann proposed with his concept of mental *microstetes* [2]. Instead of looking at the wave-forms as a sequence of uncorrelated time samples they are seen as a sequence of map distributions. This concept is illustrated in the figure 3. The waveforms on the top of the image represent the electric activity of 42 electrodes during 4 seconds. Then a clustering method was applied to estimate the mental microstates and each time point was associated to the closer of them. It is important to observe how microstates persist for a period of time instead of changing abruptly.

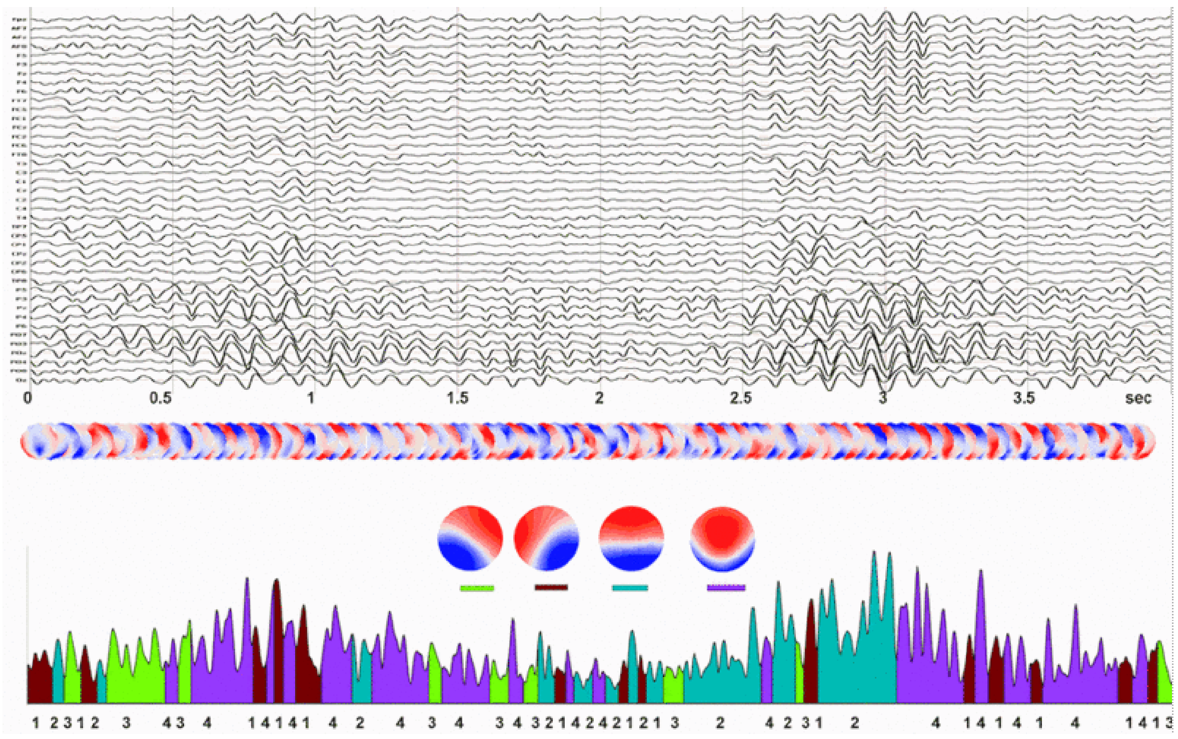

Figure 3: Microstate clustering of 4 seconds of EEG samples

The microstate main clustering algorithms for estimation used in the field are a modified version of the classical k-means [3] and the Atomize and Agglomerate Hierarchical Clustering (AAHC) [4]. The main modification of the former is a previous normalization of the data points resulting on a estimation of the cluster direction since it is believed that the information is not encoded signals

intensity. The latter is a modified version of the agglomerative hierarchical clustering. In this types of algorithms it is created as many cluster as data points and then following a ”bottom-up” approach the worst cluster is ”freed” and agglomerated to the closer one. The approach leads to progressively bigger clusters which is a major drawback when trying to describe short-duration periods of stable topography. The modified version address this problem by taking as measurement of fitness the Global explained variance (GEV) which does not penalize small clusters.
This methods are efficient algorithms for estimating clusters but both of them share a common drawback: they lack a proper method for calculating the optimal number of cluster that characterize the ERPs. This can be targeted with a probabilistic framework making possible to apply cross-validation (CV). This is precisely the the proposed improvement in the present project by modelling microstates with mixture of directional distributions such as von Mises and Watson distributions. Furthermore to strength the temporal correlation we have explore the assumption of Markov chain of order 1, which correlate each team point with the previous state.

3.2 The data and motivation of distributions

The working dataset consist of ERPs collected from 16 different subjects who were expose to three different of visual stimuli: watching a ”famous”, ”unfamiliar” or ”scrambled” face. A total of 70 sensors were used to acquire the measurements and the stimuli was released always at the same time (second 0). In the Figure 28 it is shown a whole ERPs chain from a single subject and one condition. The same overall behavior is found for the rest of the subjects. For ease of data description the following intervals are defined.

- R[1] Range -0.2 to 0.05: The sensors simply seem stable, a little random
- R[2] Range 0.05 to 0.15: There is a big disturbance
- R[3] Range 0.15 to 0.45: The big disturbance attenuate reversing the polarity 3 times.
- R[4] Range 0.45 to 0.8: The sensor values seem to converge to 0.

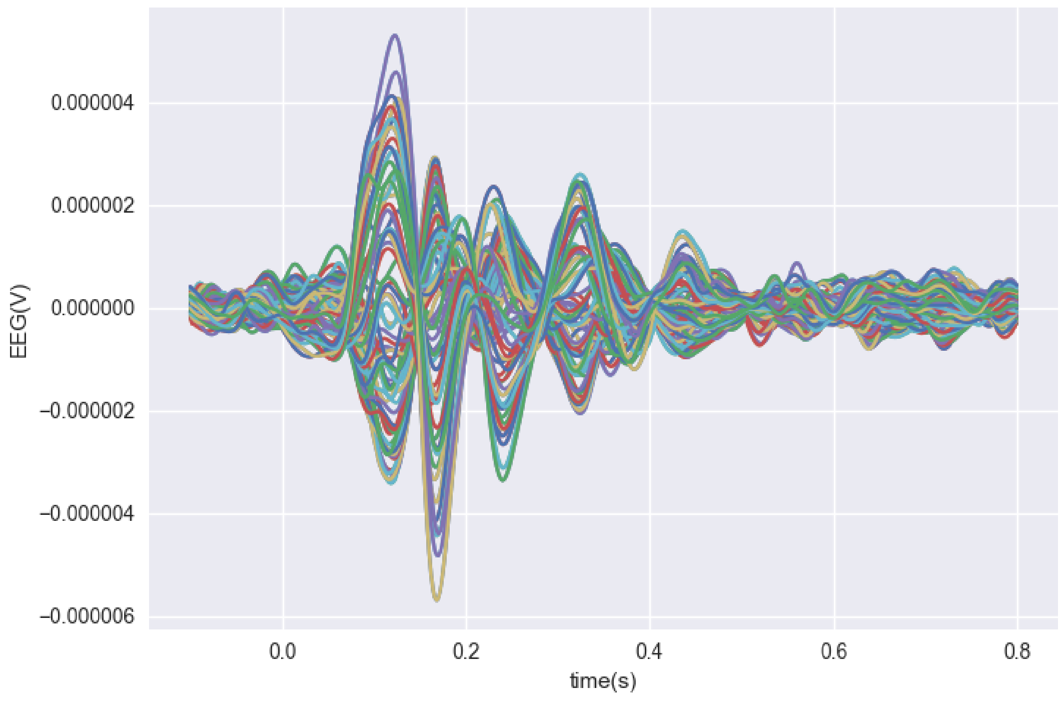

Figure 4: ERP signal from a single subject

As already mentioned, it is not believed that the signal information is encoded in the intensity but rather in the direction, and the motivation for the already aforementioned modified K-means. An analogous probabilistic model would be using a mixture of vonMises distributions(moVM) [5]. This distribution model data which is projected over the unit hypershpere and is parametrized by a mean direction μ and a concentration parameter, κ, which control how disperse the data is. This is one of models that implemented in the tool and will be used for analyzing the ERPs. However, in the signal we can find another interesting feature: the sensors continually reverse their polarity after the stimulus. This can be explained by the physical behaviour of neurons, which change their polarity constantly. This peculiar characteristic can be exploit with the Watson distribution. The only difference with the von Mises distribution is it models not only the data in the direction of μ but also the axially symmetric data on the opposite direction [6]. Thus a mixture of Watson distributions (moW) is being implemented and tested over the ERPs.

A first result that we will mention is the fact that we cannot combine the different distributions implemented in the EM algorithm. This is due to the fact that the likelihood of the samples in the dataset for each distribution implemented is very different. The likelihood values associated to a Gaussian clusters with full covariance matrix are much bigger than the ones associated to a Gaussian cluster with Diagonal covariance matrix, this situation causes that no samples are associated to the cluster of lower likelihood values, which eventually falls into a singularity. This is also true while combining Watson and von Misses Fisher clusters. This big difference can be observed 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.

3.2.1 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: 360.000
- Von Mises Fisher distribution: 32.000
- Watson 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.

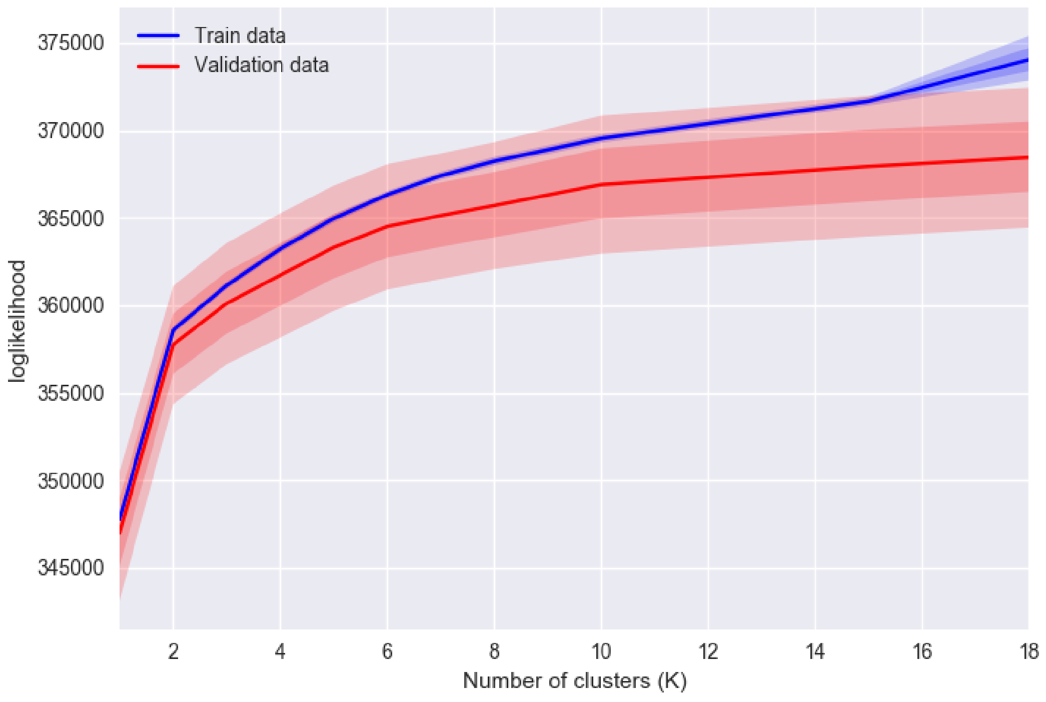

Figure 5: LOO CV of the number of clusters for the Diagonal Gaussian Distribution

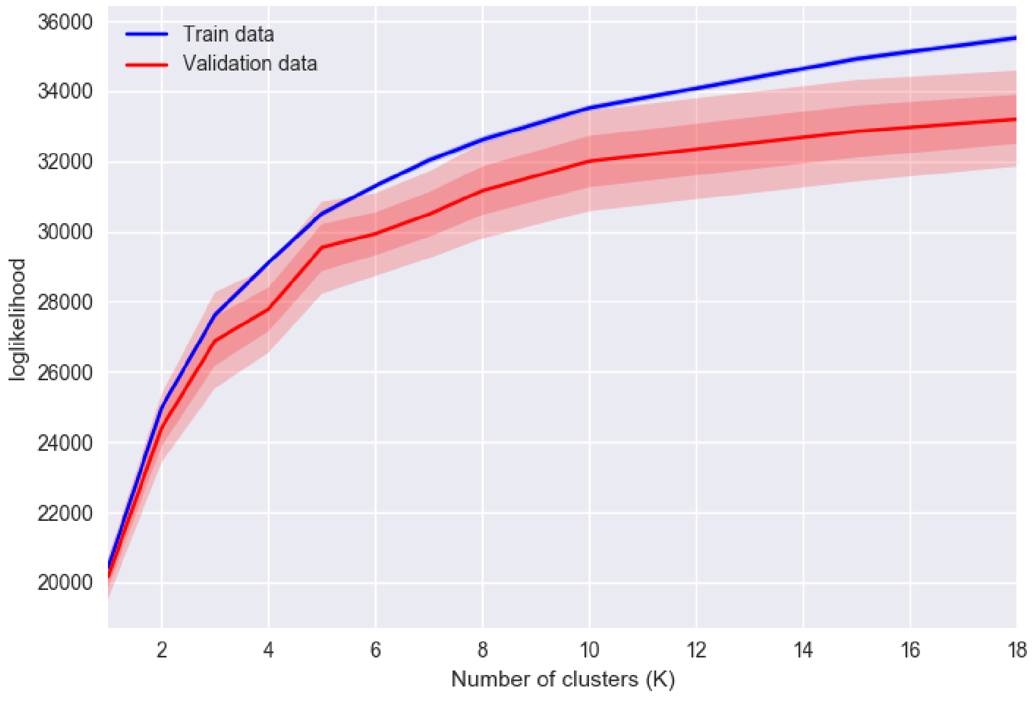

Figure 6: LOO CV of the number of clusters for the von Mises Fisher Distribution

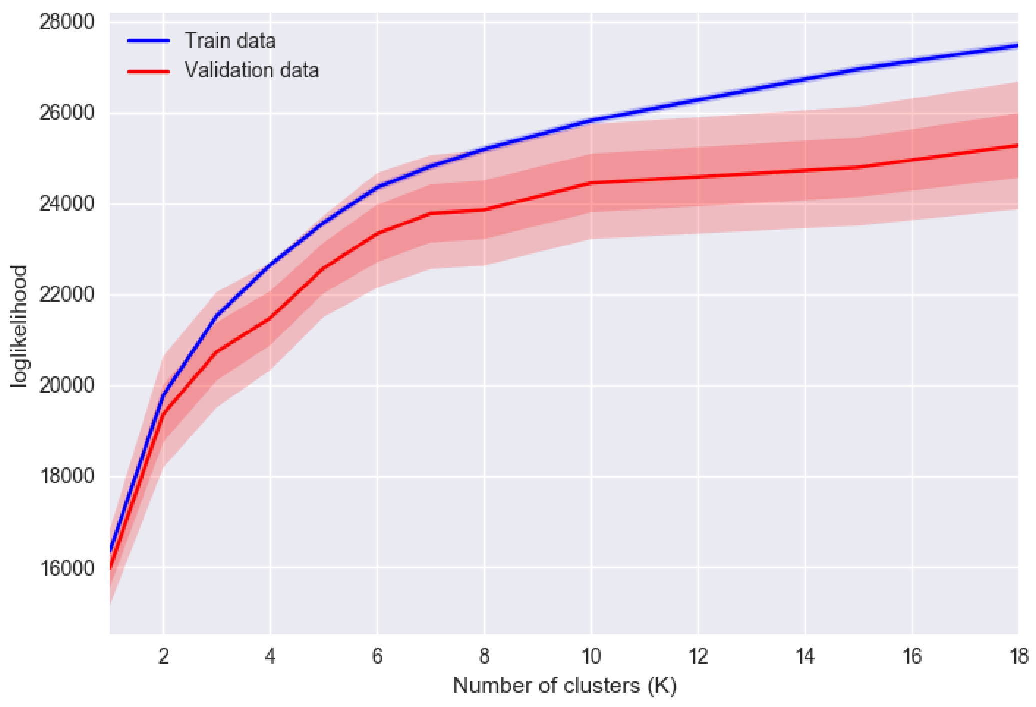

Figure 7: LOO CV of the number of clusters for the Watson Distribution

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.

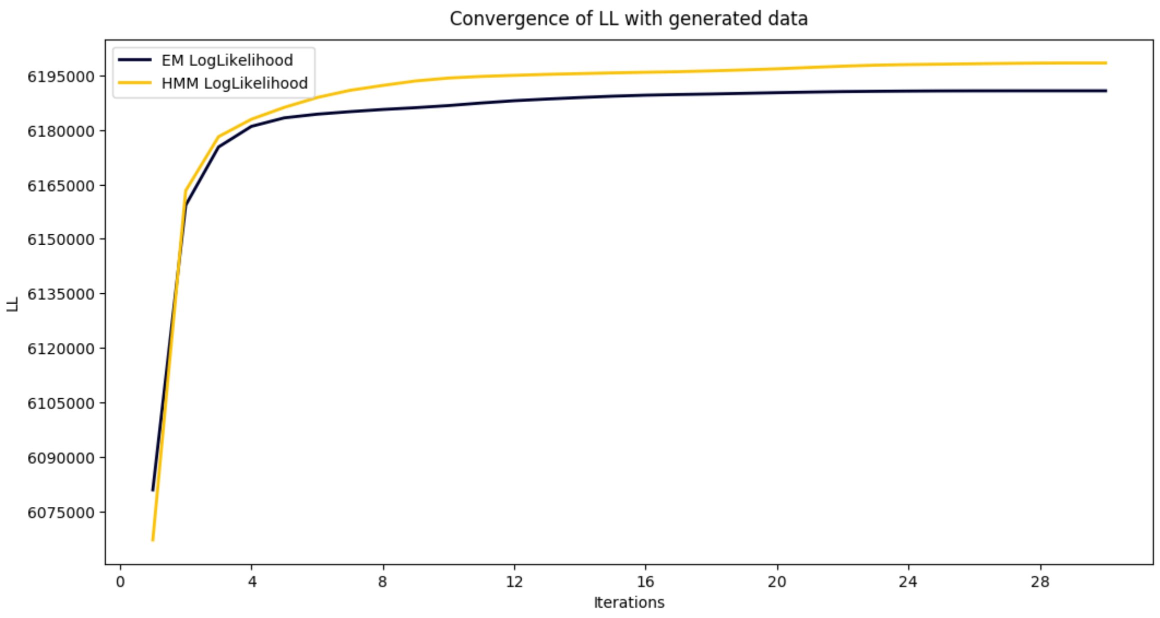

Figure 8: Convergence of the incomplete log likelihood for 4 Diagonal Gaussian clusters

3.3 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.

3.4 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.

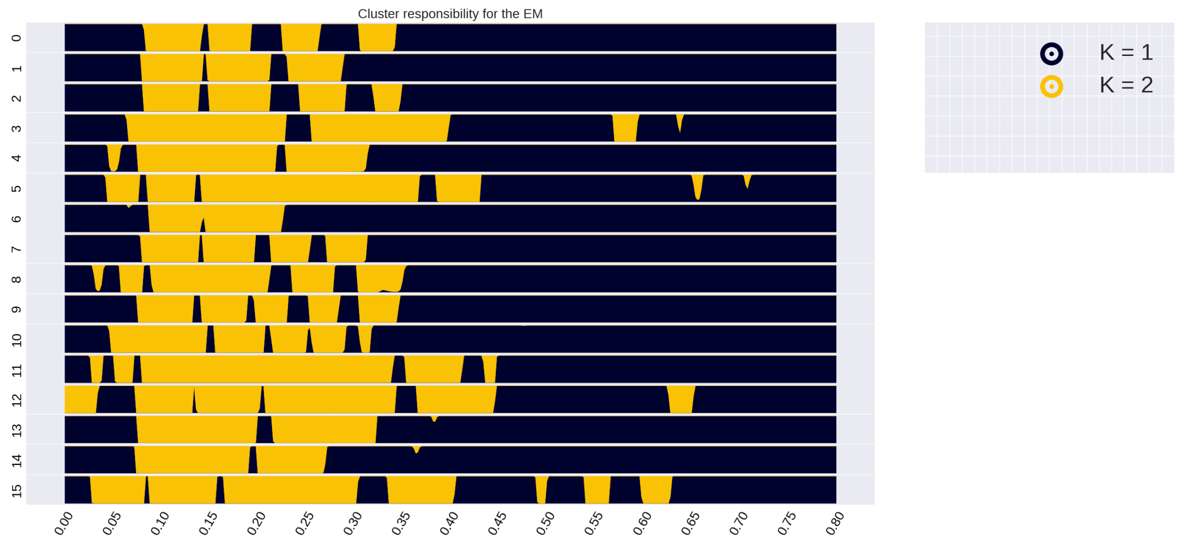

Figure 9: Time analysis for 2 Diagonal Gaussian clusters

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.

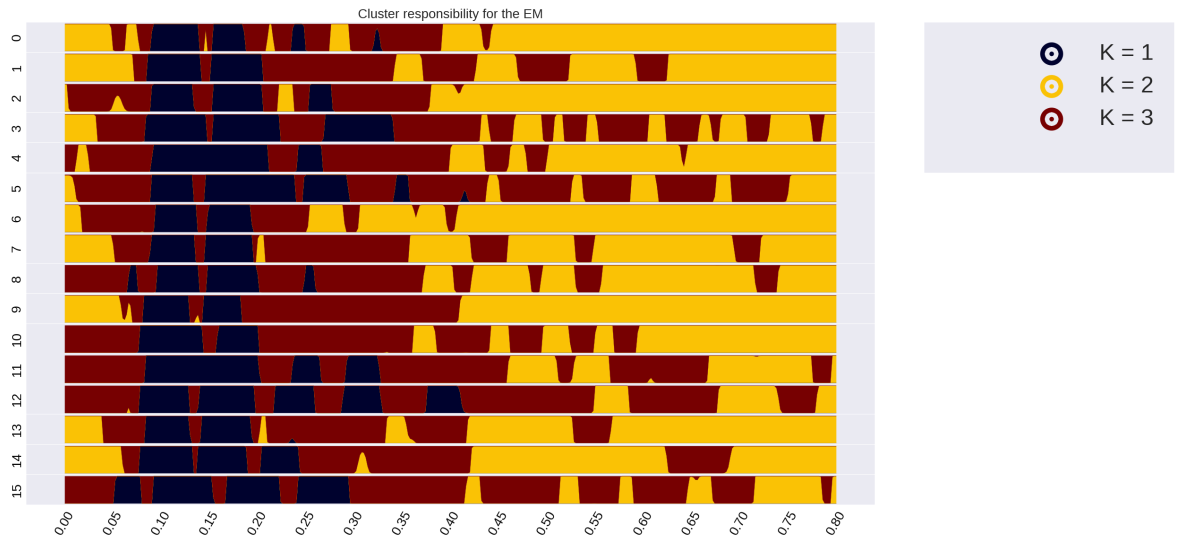

Figure 10: Time analysis for 3 Diagonal Gaussian clusters

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].

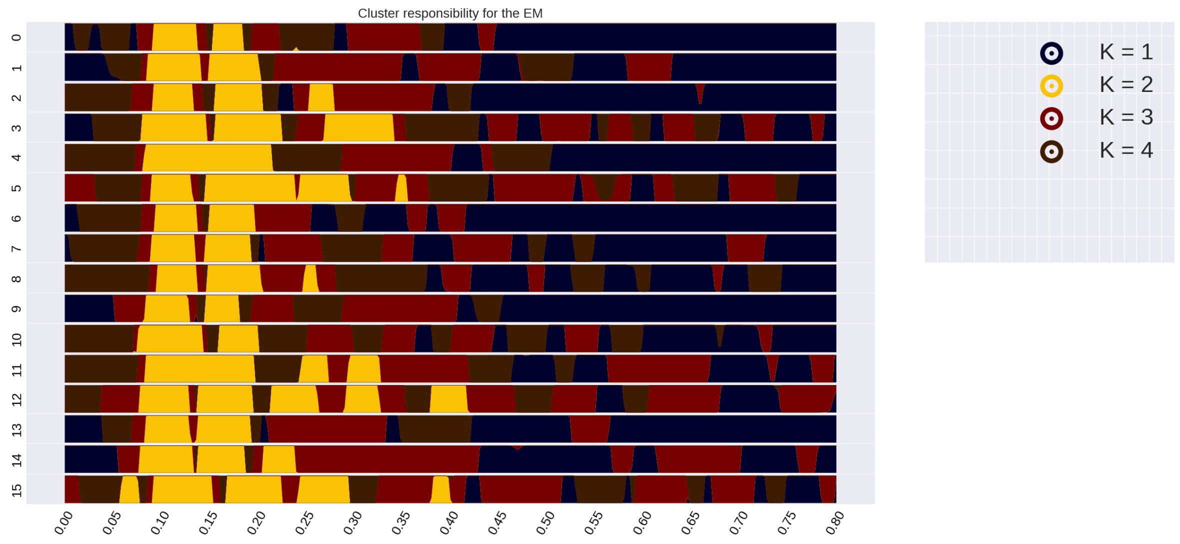

Figure 11: Time analysis for 4 Diagonal Gaussian clusters

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.

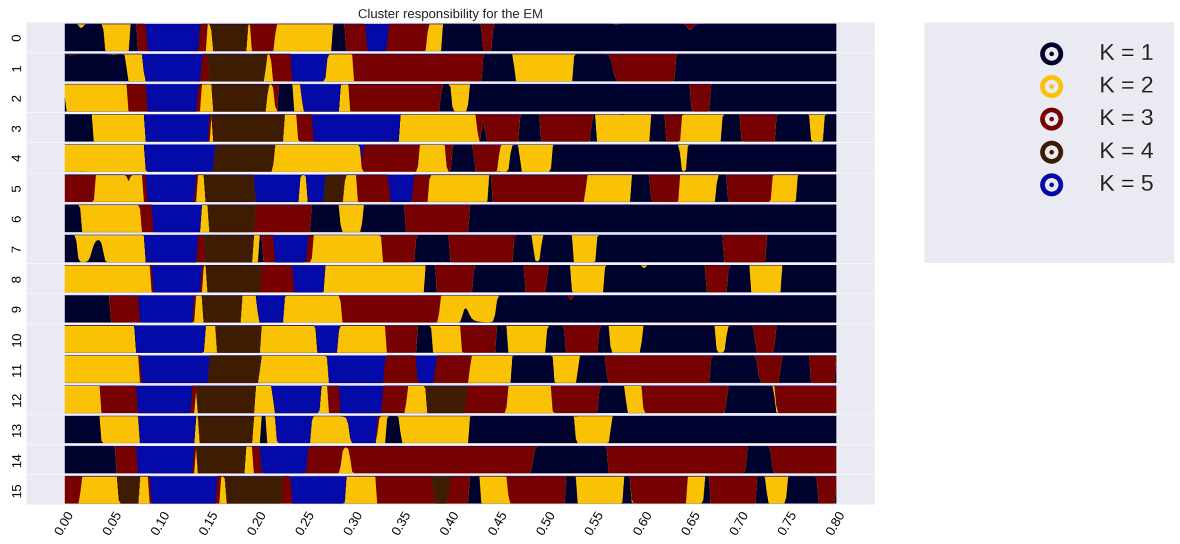

Figure 12: Time analysis for 5 Diagonal Gaussian clusters

3.4.1 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.

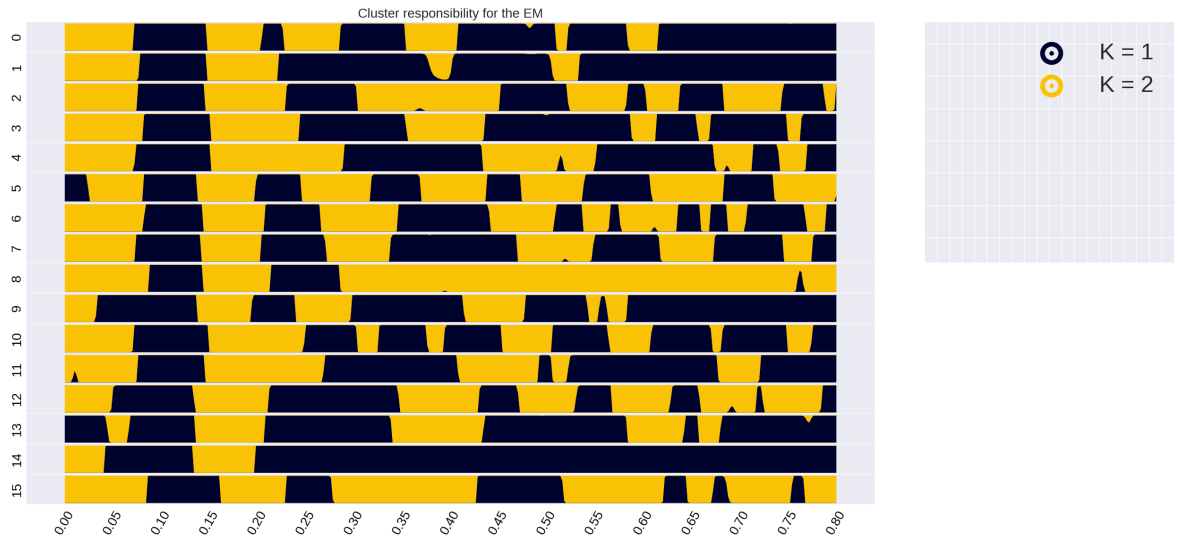

Figure 13: Time analysis for 2 von Mises Fisher Clusters

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.

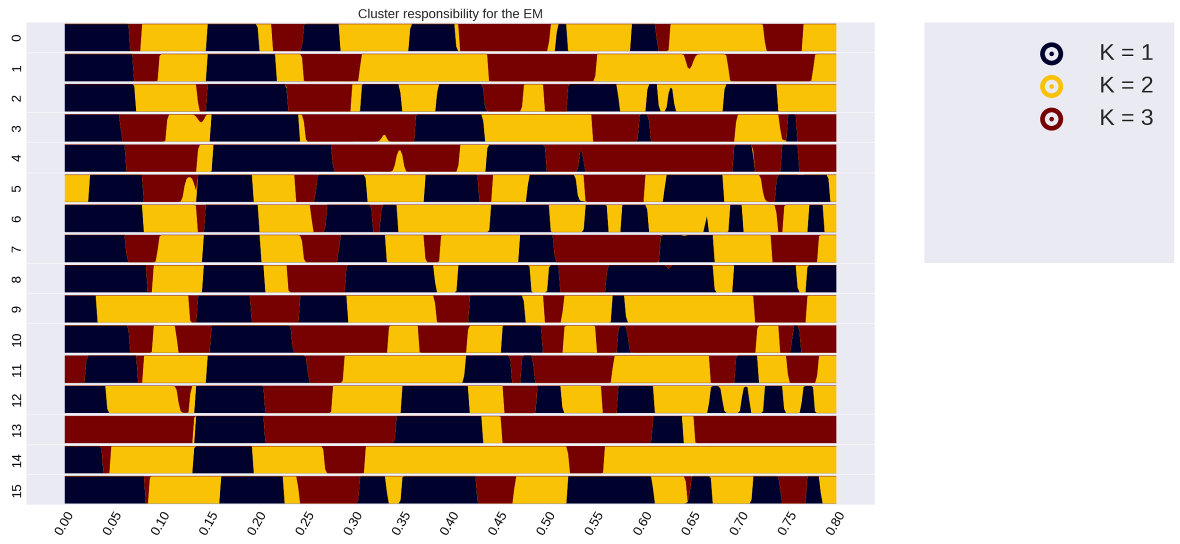

Figure 14: Time analysis for 3 von Mises Fisher Clusters

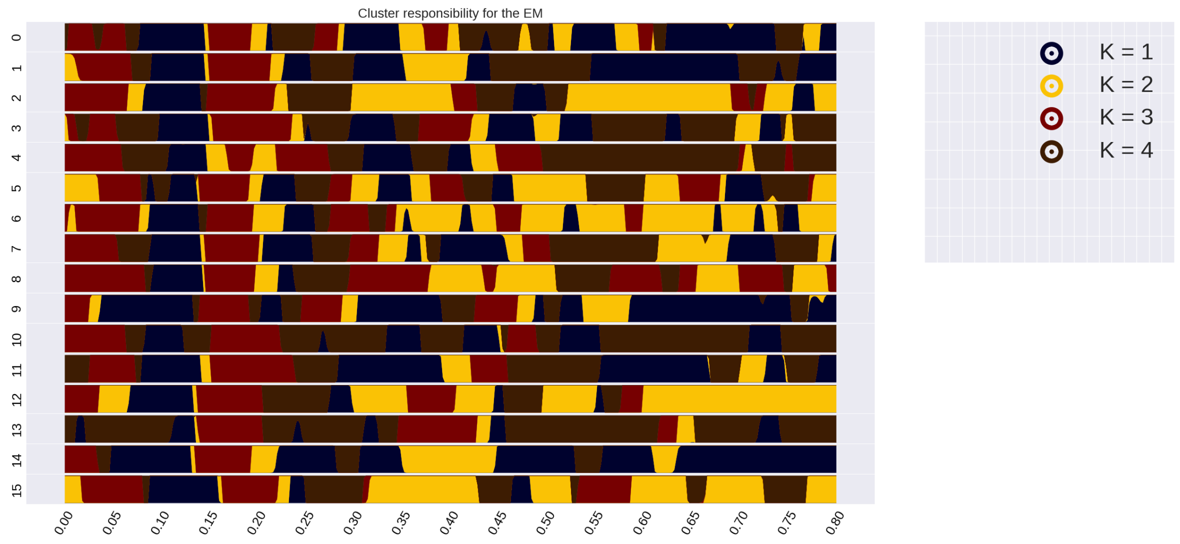

Figure 15: Time analysis for 4 von Mises Fisher Clusters

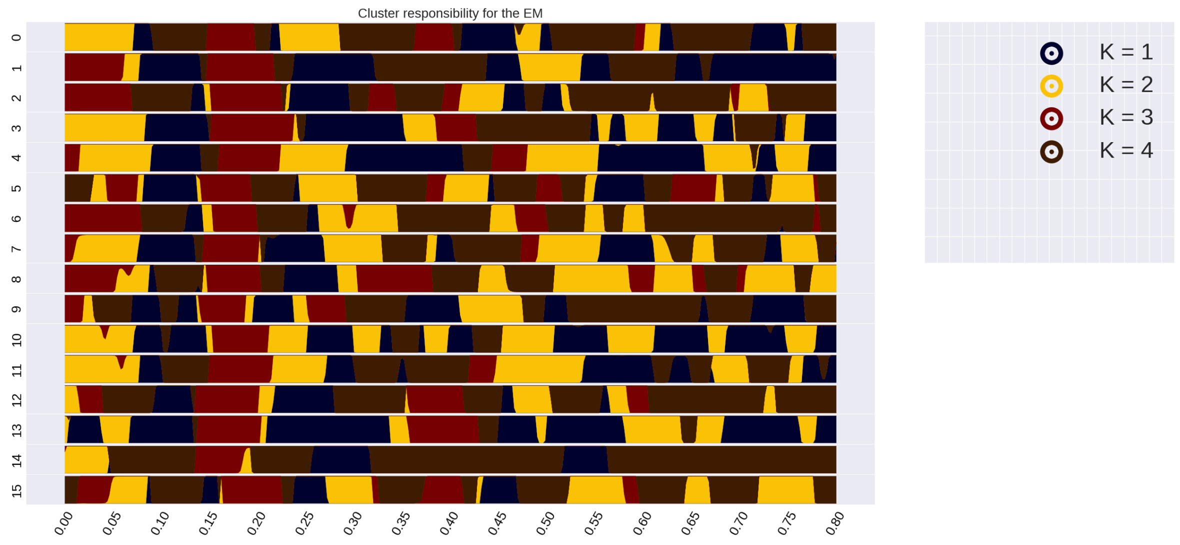

Figure 16: Time analysis for 5 von Mises Fisher Clusters

3.4.2 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.

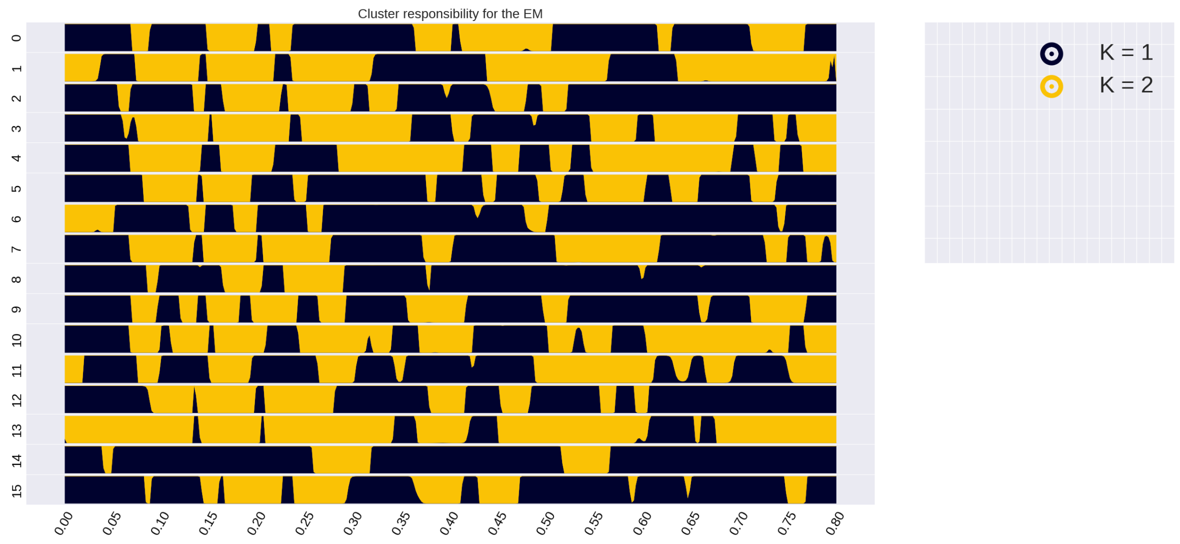

Figure 17: Time analysis for 2 Watson Clusters

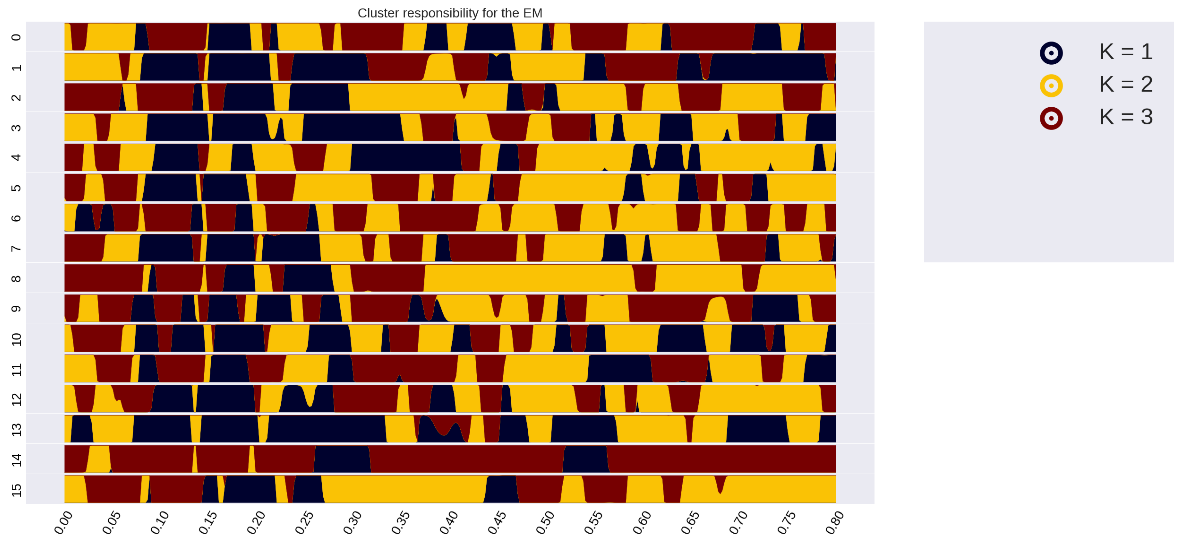

Figure 18: Time analysis for 3 Watson Clusters

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.

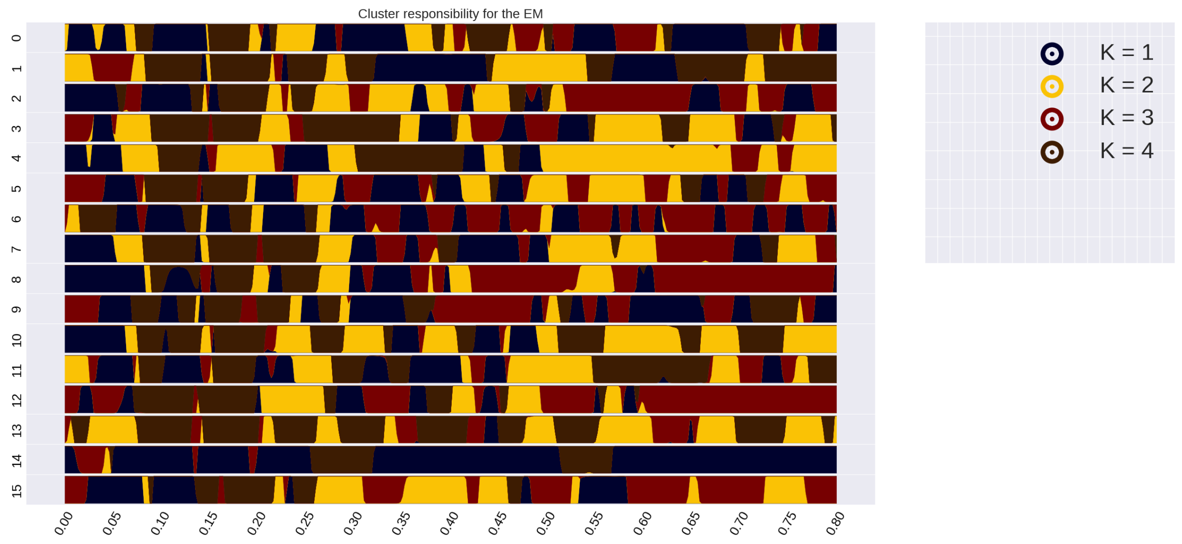

Figure 19: Time analysis for 4 Watson Clusters

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

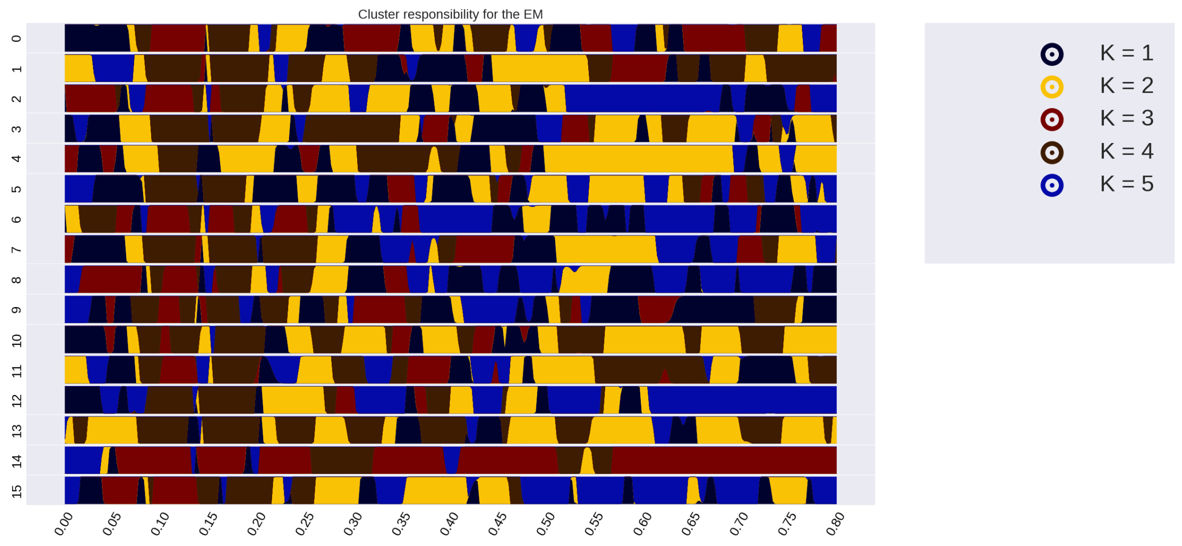

Figure 20: Time analysis for 5 Watson Clusters

3.4.3 Modified K-means

For 4 clusters it seems that the big disturbance is explained by a mix of 3 clusters, the black, red and blue. All the probabilistic models seems to characterize the region R[2] more homogeneously.

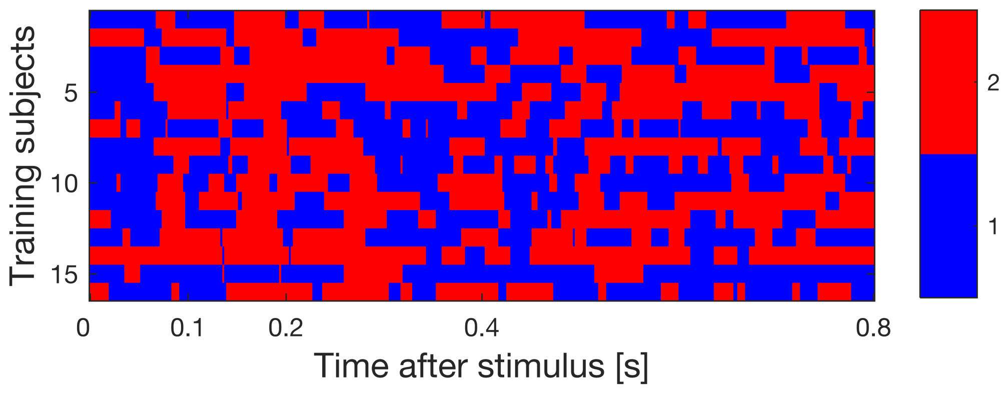

Figure 21: Time analysis for 2 k-means clusters

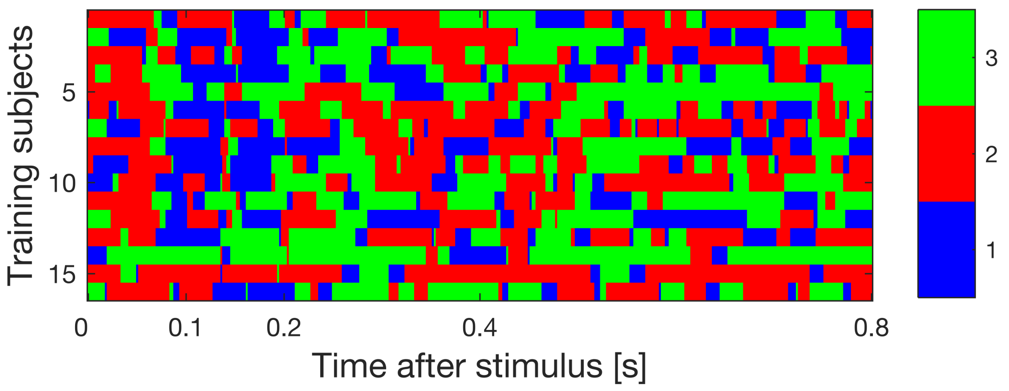

Figure 22: Time analysis for 3 k-means clusters

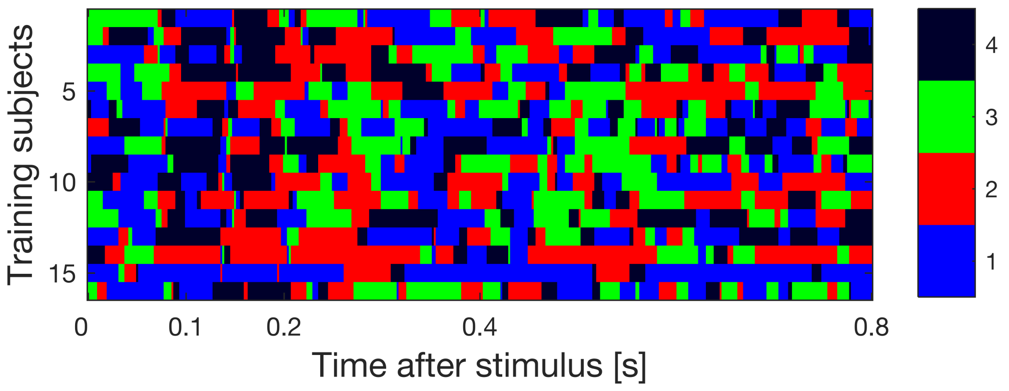

Figure 23: Time analysis for 4 k-means clusters

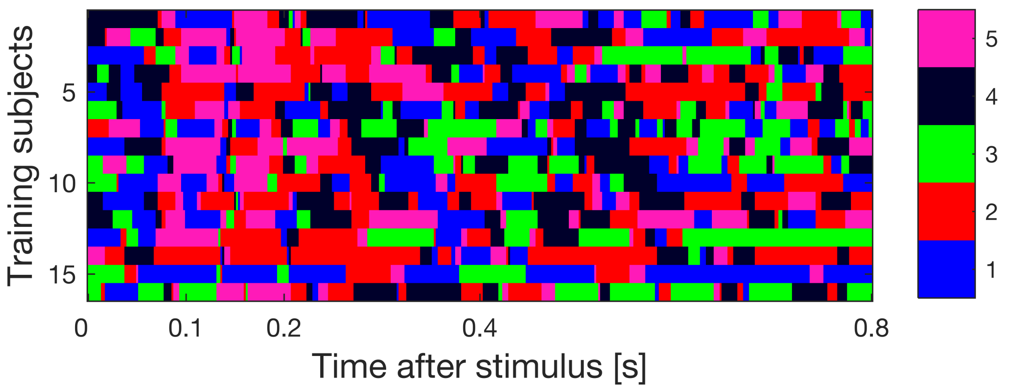

Figure 24: Time analysis for 4 k-means clusters

3.5 Scalp maps

In the following section we compare the clusters obtained with each of implemented methods and the modified K-means.

The region of the big disturbance associated with the stimuli release is characterize by the cluster 2 and followed by the cluster 3 from the Gaussian mixture. As we can see the in the scalp maps from Figure 25 the visual cortex seems to be active as we expected. We can say the same for the rest of distributions: von Mises(State 1), Watson (State 4) and K-means(State 1 and State 4). Some other interesting observations are that most active clusters during the big disturbance for Watson and and von Mises are almost the same, if we take in account the change of polarity.

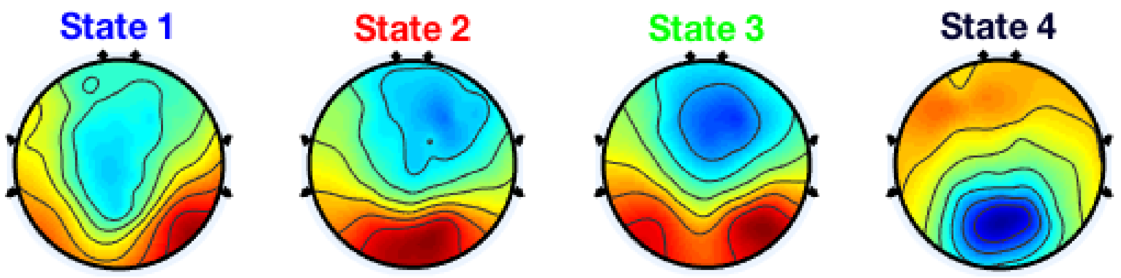

Figure 25: 4 Gaussian clusters

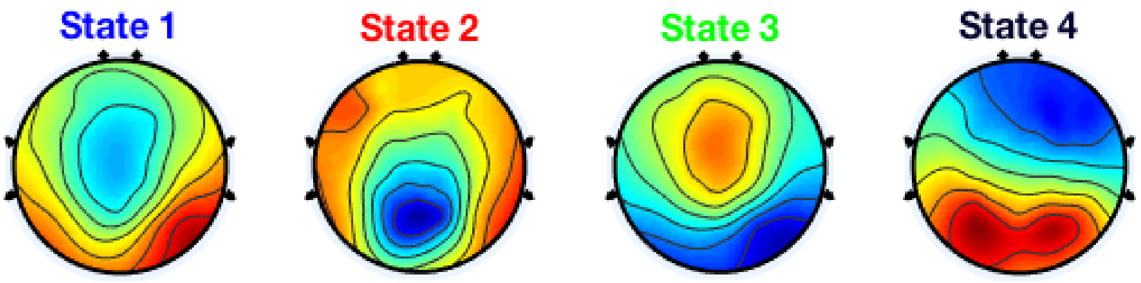

Figure 26: 4 von Mises clusters

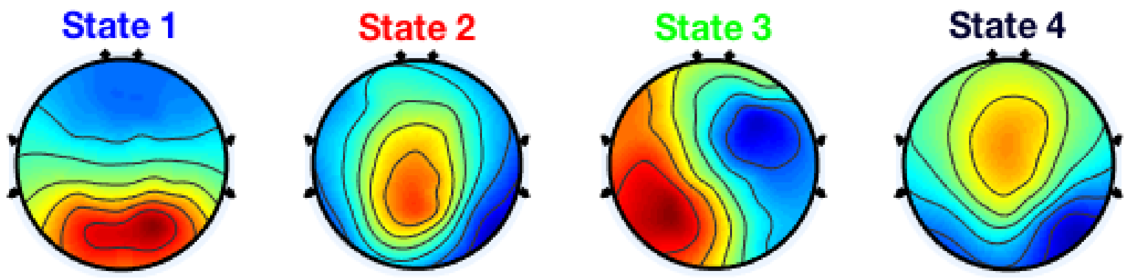

Figure 27: 4 Watson clusters

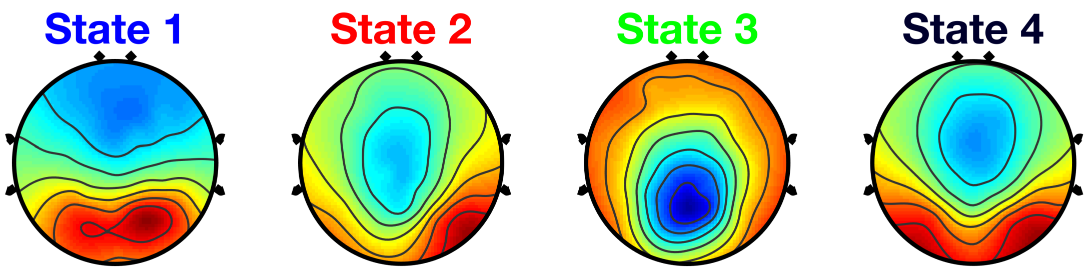

Figure 28: 4 K-means clusters

3.6 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.

References

[1] Christopher Bishop. *Pattern Recognition and Machine Learning.* 1st edition, 2006.

[2] D.Lehmann. Principles of spatial analysis. *Electroencephalography and Clinical Neurophysiology*, 1(4):309–354, 1987.

[3] R. D. Pascual-Marqui, C. M. Michel, and Dieter Lehmann. Segmentation of brain electrical activity into microstates: model estimation and validation. *IEEE Transactions on Biomedical Engineering*, 42:658–665, 1995.

[4] Micah M. Murray, Denis Brunet, and Christoph M. Michel. Topographic erp analyses: A step-by-step tutorial review. *Brain Topography*, 20:249–264, 2008.

[5] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von mises-fisher distributions. *Journal of Machine Learning Research*, 6:1345–1382, 2005.

[6] Suvrit Sra and Dmitrii Karp. The multivariate watson distribution: Maximum-likelihood estimation and other aspects. *J. Multivariate Analysis*, 114:256–269, 2013.