MODELING EEG MICRO-STATES USING MIXTURES OF THE WATSON DISTRIBUTION

Arturo Arranz (S160412), Manuel Montoya (S162706) & Martin Simon (S163008)

Technical University of Denmark

ABSTRACT

Current state-of-the-art techniques for modeling electroencephalographs (EEG) data are based in a modified K-means clustering (Pascual-Marqui (1995)). Such techniques lack a statistical framework that allows to find the optimal number of clusters to express the distribution. The aim of this project is to model EEG using mixture of Watson distributions (moW), to compute the parameters of the mixture we use the Expectation-Maximization algorithm (EM) with different assumption on the data samples. First we assume that the data points are independent, resulting in regular moW and later we assume a Markov Chain dependency between the samples, resulting in a Hidden Markov Model. In EEG data, the key information is believed to be scale and polarity invariant so an axially symmetric directional distribution, such as Watson, is a good candidate for the data modeling.

Index Terms— Electroencephalography clustering, mixture models, Watson distribution, Expectation-Maximization algorithm, Hidden Markov Models

1. INTRODUCTION

Brain micro-states are physiological states, which correspond to specific neural computations. A non-invasive method to evaluate these states is by measuring the electric activity of neurons with electrodes distributed over the scalp. Micro-states, typically lasting 80-120 ms (Khanna (2014)) are analyzed by modelling as a time sequence of non-overlapping states by using temporal maps. In this analysis very narrow sequences are discarded. However, current clustering methods do not attribute a dependency between the former and prior state when classifying the micro-states.

Since EEG data is a time-series, the temporal information of the samples could contain information about the distribution. Due to this fact, 2 different models are used, EM with an independence assumption between the samples, and EM with a Markov Chain of order 1 assumption between the samples, thus modelling the data as an Hidden Markov Model.

We aim to learn the number of states needed to express the data by means of cross-validation and analyze and compare the clusters found for different EEG data.

The EEG data used to evaluate the algorithms comes from multi-subject, multi-modal human neuro-imaging recordings. The volunteers performed a simple perceptual task on pictures of famous and scrambled faces during two visits to the laboratory (Wakeman (2015)). The mean value of the time series for the 70 recorded channels can be observed in Figure 6.

2. STATE OF THE ART

Event related brain activity can be modeled by micro-states obtained from the spatial distributions of electric potential measured by EEG. The event-related potential (ERP) micro-states oscillates in polarity and is independent of the intensity of the signal (Lehmann (2009)). Most studies reveal the same 4 classes of micro-state topography (e.g Koenig (1999, 2002)):

- right-frontal to left-posterior activity
- left-frontal to right-posterior activity
- frontal to occipital activity
- frontal and medial and slightly less occipital activity

These four micro-states are believed to describe up to 80% of the variance and the mixture of duration and sequences give a rich temporal map (Lehmann (2010)). Although many studies come to this conclusion, there are exceptions to the rule, where, depending on the application, five (Michalopoulos (2013); Hatz (2015)) or even seven (Gschwind (2016)) micro-states converge as optimal.

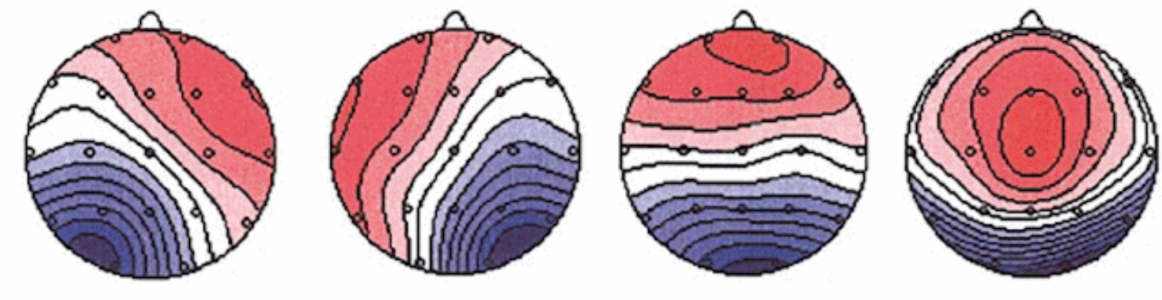


Fig. 1: The 4 main classes of micro-state topography (Koenig (2002)).

Current state-of-the-art methods use K-means clustering as it can efficiently classify a large number of continuous numerical data of high-dimensions. A modified version of K-means (Pascual-Marqui (1995)) is used in an EEGlab(Delorme) toolbox (Poulsen (2017)).

For this method the micro-state model for average reference data is expressed as

$$V_t = \sum_{k=1}^{N_\mu} \alpha_{kt} \Gamma_k \tag{2.1}$$

where N_μ is the number of electrodes of different micro-states, V_t is an $N_s * 1$ vector consisting of the scalp electric potential measurements at time instant $t(t = ...N_T)$, Γ_k is the normalized $N_s * 1$ vector representing the k-th micro-state, intensity at time instant t. All α_{kt} except for one must be kept zero at every time instant t to prevent overlapping of micro-states (Pascual-Marqui (1995)).

The micro-state parameters are estimated by minimizing

$$F = \frac{1}{N_T(N_s - 1)} \sum_{t=1}^{N_T} ||V_t - \sum_{k=1}^{N_\mu} \alpha_{kt} \Gamma_k||^2, \tag{2.2}$$

with the steps given in the basic N-micro-state algorithm (Algorithm 1).

Algorithm 1 The Basic N-Micro-state Algoerithm

1: Given N_μ, set $\sigma_0^2 = 0$, and set the convergence criterion parameter ε (e.g $\varepsilon = 10^{-6}$)
2: Select initialization 2a or 2b:
    2a: Given $\Gamma_k, k = 1...N_\mu$, satisfying $||\Gamma_k|| = 1$ and $(\Gamma'_{k1} * \Gamma_{k2}) < 1$ for $k_1 \neq k_2$, go to step 3
    2b: Given $L_t, t = 1...N_T$, taking integer values in the range $1...N_\mu$, go to step 3
3: For $t = 1..N_T$, compute $L_t = \underset{k}{\operatorname{argmax}}\{(V'_t * \Gamma_k)^2\}$
4: For $k = 1...N_\mu$,
    4a: Compute $S_k = \sum_{t\mathcal{L}_t=k} V_t * V'_t$
    4b: Compute
5:
6: **end**

The modified K-means generates directional clusters but it lacks a statistical framework. It can be considered a special case of the EM algorithm for independent moW if we impose a hard decision on the clusters and the same concentration parameter for all clusters. It does not guarantee to find the minimum and may need to initialize multiple times with randomly selected V_t vectors from the dataset (Pascual-Marqui (1995)).

The toolbox also includes a method to automate optimal micro-state segmentation, which is based on minimizing the modified predictive residual variance. The method was first published in the Journal of Neuroscience Methods (Hennings (2008)), where the authors described setting a maximum of 14 micro-states and the segmentation was repeated 10 times to find the one with lowest residual variance. Although the results of this study yielded four micro-states as optimal it is clear that researchers wish to refrain from limiting the number of clusters with a convention (Pascual-Marqui (1995)).

Another recent approach is to instead of calculating α_{kt} as a binary output to calculate the probability $p(\alpha_{kt})$ instead (Zdyb (2016)) as

$$p(\alpha_{kt}) = (\frac{1}{K})^{\alpha_{kt}}, p(A) = \sum_{t}^{T} \sum_{k}^{K} p(\alpha_{kt}), \tag{2.3}$$

where $p(A)$ is the probability matrix of all microstates over trial time.

The method outperfomed EEGlab K-means clustering at higher numbers of clusters (i.e K=7) and was competitive in lower number of clusters (i.e K=4).

3. THE MULTIVARIATE WATSON DISTRIBUTION

The Watson distribution models data which is axially symmetric ($\pm\boldsymbol{x}$ vectors are equivalent) and scale invariant i.e. the vectors are unitary. This are two interesting properties for micro-state modeling since we are not interested in the magnitude of the measurements nor in their polarity.

Let $\mathbb{S}^{p-1} = \{\boldsymbol{x}|\boldsymbol{x} \in \mathbb{R}^p, ||\boldsymbol{x}||_2 = 1\}$ be the (p-1)-dimensional hypersphere centered in the origin. Then the Watson probability density function is

$$W_p(\boldsymbol{x}|\boldsymbol{\mu}, \kappa) = c_p(\kappa)e^{\kappa(\boldsymbol{\mu}^T\boldsymbol{x})^2}, \qquad \boldsymbol{x} \in \mathbb{S}^{p-1} \tag{3.1}$$

where $\kappa \in \mathbb{R}$, is the *concentration parameter* and $\mu \in \mathbb{S}^{p-1}$ is the *mean direction*. The normalisation constant $c_p(\kappa)$ is given by

$$c_p(\kappa) = \frac{\Gamma(p/2)}{2\pi^{(p/2)}M(\frac{1}{2}, \frac{p}{2}, \kappa)}, \tag{3.2}$$

where M is Kummer's confluent hypergeometric function defined as

$$M(a, c, \kappa) = \sum_{j\geq 0} \frac{a^{\bar{j}}\kappa^j}{c^{\bar{j}}j!}, \qquad a, c, \kappa \in \mathbb{R} \tag{3.3}$$

and $a^{\bar{0}} = 1$, $a^{\bar{j}} = a(a+1)...(a+j-1), j \geq 1$, denotes the rising-factorial.

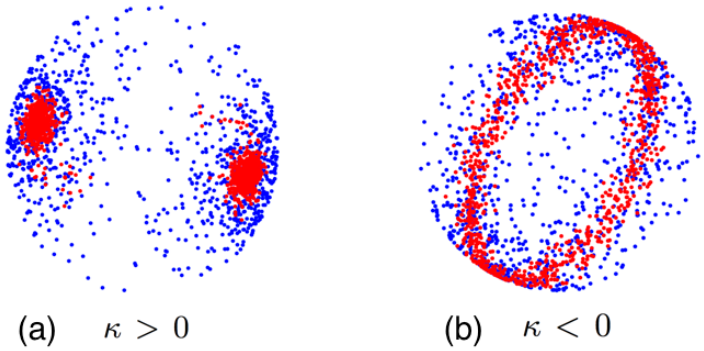


Fig. 2: concentration values: (a) positive and (b) negative

The Figure 2 shows a scatter plot from Watson distribution samples. Note that for $\kappa \to \infty$ the samples concentrate around the mean direction while for $\kappa \to -\infty$ the samples concentrate around a ring orthogonal to the mean direction. The uniform distributed case would correspond to $\kappa = 0$.

3.1. Maximum Likelihood Estimation

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2...\mathbf{x}_n|\mathbf{x}_i \in \mathbb{S}^{p-1}\}$ be a set of i.i.d samples from $W_p(x|\boldsymbol{\mu}, \kappa)$. The corresponding log-likelihood function is

$$l(\boldsymbol{\mu}, \kappa|\mathbf{X}) = n(\kappa\boldsymbol{\mu}^T S\boldsymbol{\mu} - logM(1/2, p/2, \kappa) + \gamma), \tag{3.4}$$

where $S = n^{-1}\sum_{i=1}^{n} x_i x_i^T$ is the *simple scatter matrix* and γ is a constant term that we can ignore. Maximizing (3.4) yiealds to

$$\hat{\boldsymbol{\mu}} = \mathbf{s}_1 \quad if \quad \hat{\kappa} > 0, \qquad \hat{\boldsymbol{\mu}} = \mathbf{s}_p \quad if \quad \hat{\kappa} < 0, \tag{3.5}$$

where $\boldsymbol{s}_1, \boldsymbol{s}_2...\boldsymbol{s}_p$ are the normalised eigenvectors of the *scatter matrix*. The *concentration parameter*, κ is estimated by solving

$$g\big(1/2, p/2|\hat{\kappa}\big) := \frac{M'(\frac{1}{2}, \frac{p}{2}, \hat{\kappa})}{M(\frac{1}{2}, \frac{p}{2}, \hat{\kappa})} \quad = \quad \hat{\boldsymbol{\mu}}^T S\hat{\boldsymbol{\mu}} := r \tag{3.6}$$

In order to estimate κ it is necessary to calculate first μ. However, the estimation of μ, depends on the sign of the concentration. This yield to a coupled system (3.5)-(3.6). To solve this problem both options are calculated and the solution with higher log-likelihood is chosen.

In order to get $\hat{\kappa}$, the implicit formula (3.6) can be calculated using the iterative Raphson-Newtons method fed with a initial guess. However, if the guess is not sufficiently good, the method may not converge. Fortunately, an analytical approximation (Sra Suvrit (2011)) can be calculated and used as initial guess.

$$\hat{\kappa} = \frac{cr - a}{r(1-r)} + \frac{r}{2c(1-r)} \tag{3.7}$$

where $a = 1/2$ and $c = p/2$

4. EXPECTATION-MAXIMIZATION ALGORITHM FOR INDEPENDENT MOW

In the previous section we derived the maximum likelihood estimator for a single Watson distribution. However, now we turn our attention to a mixture of several Watson distributions.

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

$$\ell(\boldsymbol{X}|\boldsymbol{\mu}_1, \kappa_1, \pi_1..., \boldsymbol{\mu}_K, \kappa_K, \pi_K) = \sum_{t=1}^{T} log\Big(\sum_{i=1}^{K} \pi_i W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \kappa_i)\Big) \tag{4.1}$$

In order to calculate the parameters which maximize (4.1) the iterative method EM (Bishop (2006)) 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:

$$r_t(i) = \frac{\pi_i W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \kappa_i)}{\sum_l \pi_l W_p(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \kappa_l)} \tag{4.2}$$

M-step:

$$\begin{aligned} &\mu_i = s_1^j \quad if \quad \kappa_i > 0, \qquad \mu_i = s_p^i \quad if \quad \kappa_i < 0 \\ &\kappa_i = g^{-1}(1/2, p/2, r_i), \quad where \quad r_i = \mu_i^T S^i \mu_i \\ &\pi_i = \frac{1}{T}\sum_i r_t(i), \end{aligned} \tag{4.3}$$

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

$$S^i = \frac{1}{\sum_t r_t(i)} \sum_t r_t(i) \boldsymbol{x}_t \boldsymbol{x}_t^T \tag{4.4}$$

The algorithm iterate (4.2) and (4.3) until the chosen convergence criteria is fulfilled.

5. EXPECTATION-MAXIMIZATION ALGORITHM FOR HMM MOW

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.

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1I} \\ a_{21} & a_{22} & ... & a_{2I} \\ ... & ... & a_{ij} & ... \\ a_{I1} & a_{I2} & ... & a_{II} \end{bmatrix} \qquad \pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ ... \\ \pi_I \end{bmatrix} \tag{5.1}$$

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.

5.1. 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:

$$P(s_t = i|Y) = \frac{P(s_t = i, Y)}{P(Y)} \propto P(s_t = i, Y) \tag{5.2}$$

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:

$$P(Y) = \sum_{i=1}^{I} P(s_t = i|Y) \tag{5.3}$$

We can split this probability into 2 factors:

$$P(s_t = i|Y) = P(s_t = i, y_{1:t})P(y_{t+1:T}|s_t = i) \tag{5.4}$$

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:

$$\begin{aligned} P(s_t = i, y_{1:t}) &= \sum_{j=1}^{I} P(s_t = i, s_{t-1} = j, y_{1:t}) \\ &= P(x_t|s_t = i) \sum_{j=1}^{I} P(s_t = i|s_{t-1} = j) P(s_{t-1} = j, y_{1:t-1}) \end{aligned} \tag{5.5}$$

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:

$$\begin{aligned}\alpha_1^n(i) =& \pi_i \cdot p_i(x_1^n|B_i) = \pi_i \cdot W_i(x_1^n|\mu_i, \kappa_i) \\ \alpha_t^n(i) =& p_i(x_t^n|B_i) \sum_{j=1}^{I} a_{ji}\alpha_{t-1}^n(j)\end{aligned} \tag{5.6}$$

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:

$$\begin{aligned}P(y_{t+1:T}|s_t = i) &= \sum_{j=1}^{I} P(y_{t+1:T}, s_{t+1}|s_t = i) \\ &= \sum_{j=1}^{I} P(s_t = i|s_{t+1} = j)P(x_{t+1}|s_{t+1} = j)P(y_{t+2:T}|s_{t+1} = j)\end{aligned} \tag{5.7}$$

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 $\beta_t^n(i)$ and so we obtain the equations:

$$\begin{aligned}\beta_T^n(i) &= 1 \\ \beta_t^n(i) &= \sum_{j=1}^{I} a_{ij}p_j(x_t^n|B_j) \cdot \beta_{t+1}^n(i)\end{aligned} \tag{5.8}$$

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 $\alpha_t^n(i)$ and $\beta_t^n(i)$ 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:

$$P(s_t = i|Y) = \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(Y)} \propto \alpha_t(i)\beta_t(i) \tag{5.9}$$

Where we denote $\gamma_t^n(i)$ 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 (5.2). Notice the analogy of this parameter to the responsibility parameter $r_t(i)$, they both have the same meaning.

5.2. 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.

$$L_c(y^{(n)}, s^{(n)}|\theta) = P(s^{(n)}|\theta)P(y^{(n)}|s^{(n)}, \theta) = \left(p(s_1|\pi)\prod_{t=2}^{T_n} p(s_t|s_{t-1}, A)\right) \cdot \left(\prod_{t=1}^{T_n} p(x_t|s_t, B))\right) \quad (5.10)$$

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:

$$L(y^{(n)}|\theta) = \sum_{s_1=1}^{I}\sum_{s_2=1}^{I}...\sum_{s_{T_n}=1}^{I} P(y^{(n)}, s^{(n)} = \{s_1, s_2, ...s_{T_n}\}) \quad (5.11)$$

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^2T_n elements, making the algorithm feasible in practice. We can compute this likelihood as:

$$L(y^{(n)}|\theta) = \sum_{i=1}^{I} P(s_t = i|Y) = \sum_{i=1}^{I} \alpha_T^n(i) = \sum_{i=1}^{I} \beta_1^n(i) \cdot \pi_i \cdot p_i(x_1|B_1) \quad (5.12)$$

5.3. 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:

$$\xi_t^n(i,j) = P(s_t^n = j, s_{t-1}^n = i|Y^{(n)}, \theta) \quad (5.13)$$

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

$$\xi_t^n(i,j) \propto \alpha_t^n(i)a_{ij}p_j(x_{t+1}^n|B_j)\beta_{t+1}^n(j) \quad (5.14)$$

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

5.4. 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:

$$\begin{array}{llll} \alpha_t^n(i) & n = 1, ..., N & t = 1, ..., T^{(n)} & i = 1, ..., I \\ \beta_t^n(i) & n = 1, ..., N & t = 1, ..., T^{(n)} & i = 1, ..., I \\ \gamma_t^n(i) & n = 1, ..., N & t = 1, ..., T^{(n)} & i = 1, ..., I \\ \xi_t^n(i,j) & n = 1, ..., N & t = 1, ..., T^{(n)} & i, j = 1, ..., I \end{array} \quad (5.15)$$

5.5. 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:

$$\pi_i = \frac{N_i}{N} = \frac{1}{N}\sum_{n=1}^{N}\gamma_1^n(i) \qquad N = \sum_{i=1}^{I}\sum_{n=1}^{N}\gamma_1^n(i) \quad (5.16)$$

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

$$a_{ij} = \frac{E_{ij}}{E_i} = \frac{1}{E_i} \sum_{n=1}^{N} \sum_{t=2}^{T_n} \xi_t^n(i,j) \qquad E_i = \sum_{i=1}^{I} \sum_{n=1}^{N} \sum_{t=2}^{T_n} \xi_t^n(i,j) \tag{5.17}$$

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:

$$E\{\phi(Y)\} = \frac{1}{\Gamma_i} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \gamma_t^n(i)\phi(x_t^n) \qquad \Gamma_i = \sum_{n=1}^{N} \sum_{t=1}^{T_n} \gamma_t^n(i) \tag{5.18}$$

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

$$S^i = \frac{1}{\Gamma_i} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \gamma_t^n(i) x_t^{(n)} x_t^{(n)^T} \tag{5.19}$$

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.

6. IMPLEMENTATIONS

In this section we will explain some of the particularities, challenges, decisions and properties of the Python implementation of the system.

6.1. Implementation of the Watson pdf and sampler

We coded the Watson distribution pdf in Python from scratch in a logarithmic form, to be able to express very high and low likelihoods. The function is optimized in a way that it accepts multiple samples and distribution parameters at a single time to increase the speed. It can also accept the precomputed normalizing constant $c_p(\kappa)$, thus significantly reducing the computation costs. For the Kummer function we first use a built-in implementation from the library "scipy", if the value is 0 or infinity then we use our own implementation in logarithms, and if it does not converge, then we generate an error to be handled later.

We also coded a Watson distribution sampler in Python from scratch based on a Matlab code implementation. We performed some improvements, i.e., in their implementation they first perform a grid of the univariate Watson distribution for obtaining the maximum of the pdf, this is not optimal since the maximum can be easily computed as $W_{max} = c(\kappa) \cdot e^{\kappa}$ if the kappa is positive and $W_{max} = c(\kappa)$ if the kappa is negative.

There is however some small problem of their implementation for 2 Dimensions due to the way they consider a sample is valid, samples in the most probable area are not generated. But the ML estimating of the parameters is still the proper one so it is not a problem. The next 2 Figures show the probability density function (pdf) of the Watson distribution in Spherical and Cartesian coordinates and some samples of the distribution, for the 2D and 3D cases.

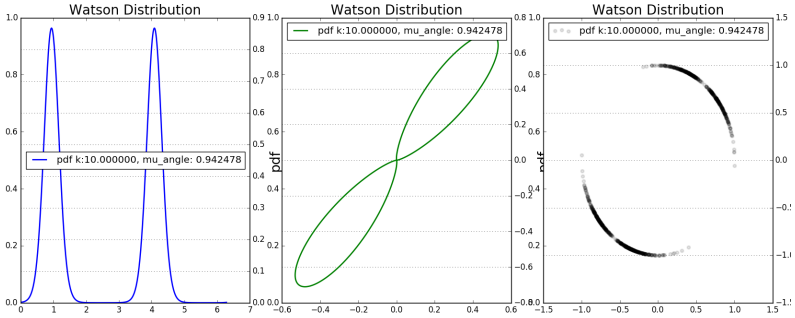


Fig. 3: Example Watson Distribution in 2D. (a) Shows the pdf in polar coordinates, (b) shows the distribution in Cartesian coordinates and (c) shows samples drawn from the distribution.

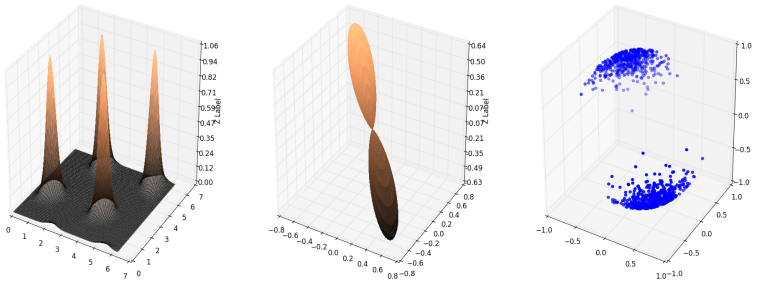


Fig. 4: Example Watson Distribution in 3D. (a) Shows the pdf in polar coordinates, (b) shows the distribution in Cartesian coordinates and (c) shows samples drawn from the distribution.

6.2. Implementation of the Watson weighted Maximum Likelihood estimator

We implemented the Watson distribution Maximum Likelihood estimator of the distribution. Our first guess uses the equation (3.7). Then we implemented the Newton's algorithm using the optimized equation that only needs to compute the Kummer funciton once.

$$\kappa_{r+1} = \kappa_r - \frac{g(a,c,\kappa_r) - r}{(1 - c/\kappa)g(a,c,\kappa_r) + a/\kappa - g(a,c,\kappa_r)^2)} \tag{6.1}$$

. The algorithm estimates the parameters (κ, μ) for both the positive and negative kappa. It then chooses the kappa that maximizes weighted likelihood of the data.

Due to complications when computing the Kummer function, its value can be too high is kappa is very negative or too low if kappa is very positive, prior to compute them, we check that their estimated value will not be too high and the computation will converge. In the case that the computation of the Kummer function does not converge then we create an Exception that generated an error if it is not handled. In the next section we will explain how we handle such exception.

We tested this estimation function with the artificially generated data and it successfully found the value of the parameters in about a 2% margin of error for the kappa value.

For testing the further clustering algorithms, we have generated samples from 3 different Watson distributions. These 3 distributions can be seen in the next Figure. We produced a hard enough problem where the clusters overlap and have positive and negative kappas.

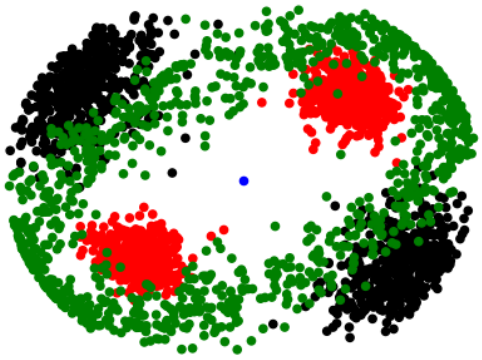


Fig. 5: moW with 3 clusters

Also, using the 3 clusters (Watson distributions) we generated Hidden Markov Chains of order 1 (HMM), defining the parameters initial probability distribution and the transition matrix (π, A). This code is also implemented from scratch. The same samples will be used to train the EM and the HMM algorithms (when we say EM we mean EM with assumption of independent samples and HMM is EM with assumption of Markov Chain of order 1 between the samples). If we use the same samples for both algorithms we can compare the likelihoods and see that the one for the HMM is bigger without over fitting.

6.3. Implementation of the EM algorithm

We have implemented the EM algorithm from scratch in Python, since the Watson distribution belongs to the exponential family, the EM equations are obtained directly from moment matching. The implemented algorithm takes as input the data X, the number of clusters K, the maximum number of iterations T, and the convergence tolerance for the incomplete loglikelihood $delta$, if the increase in loglikelihood is lower than this value, we stop the iterations. It can also take as input the initialization values of the parameters, if not provided, it will initialize them in the following manner. The π is a uniform vector, the cluster centroids μ_k are obtained from a normalized Gaussian N-dimensional vector, and the κ_k are obtained from a uniform distribution from -20 to 20.

It is also implemented to run several random initializations of the EM and pick the only with the highest final incomplete loglikelihood. The function outputs for each iteration, the loglikelihood, the model parameters π and the cluster parameters (μ_k, κ_k for all the clusters $k = 1, ..., K$.

Everything is implemented in logarithms, to avoid the problem of probability densities that are too high or too low (they would be taken as 0 or infinity since Python's precision for a float number is about $10^{\pm 200}$, when we obtain the responsibility matrix, then we convert the probability densities back to natural units and compute the parameters.

In order to overcome the problem of "degenerated clusters", that is, a cluster that falls in a region with a small number of samples (usually just one), in which case, the estimation of its parameters is ill-defined, its parameters are computed virtually only from one sample, which will lead to infinity kappa, infinity Kummer function, non-defined correlation matrix, complex mus, non-converging Newton Method for computing kappa... among others. Each of these problems will make the software to crash.

The policy developed to deal with this situations is as follows. First we have an initial pre-detection where we check if the normalized explained variance r for the positive kappa is too high $r- > 1$ or the normalized explained variance r for the negative kappa is too low $r- > 0$. In this situation we do not compute them for that kappa. If we cannot compute any of the kappas, we generate an error. In the case that we try to compute the kappas and the Kummer function still does not converge, then we generate the same error.

Whenever we try to estimate the cluster parameters, we are listening to this error, if it happens, we update the mu as the new mu and set the kappa to a saturated maximum value of ± 1000. Of course, in the next iteration, if the responsibility vector of that cluster is still degenerated, the error will happen again until the the iterations converge or the responsability vector changes in a way that the cluster is responsible for more samples.

Other cluster managing policies that we have implemented is just removing the cluster if it degenerated, but for the purposes of this project, in the end we chose to saturate the kappa value.

The decoder of a sequence is easily implemented by computing the responsibility of each cluster to each sample, choosing as generating cluster of a sample, the cluster that maximizes the responsibility of the sample.

6.4. Implementation of the HMM algorithm

Everything is implemented in logarithms otherwise the alphas and betas would be too high or too low and the system breaks

It accepts any number or random chains with different length. It suffers from the same "degenerated clusters" problem as the EM. The computing of alphas and betas is not parallelizable due to its dynamic programming nature. The computation of the parameters of the A matrix, $\xi_t^n(i, j)$, require a memory of $N \cdot T \cdot I^2$, so it is constrained in memory if the number of states is high. It also takes more time to compute than the EM algorithm. We can always do a first study with the EM algorithm and then use the final EM as initialization of the HMM when we have already haunted down the number of clusters.

We implemented 3 decoders, the step by step MAP decoder, Viterbi ML and Viterbi MAP. The Viterbi works for the first 50 samples or so of the chains and then it converges to a cluster, we don't know if it is because ML is sub-optimal or there is something wrong with the implementation, the other 2 work properly.

7. RESULTS WITH GENERATED DATA

In the next section, the results of the algorithm for the artificially generated clusters in Figure 5 will be exposed. In this section we will compute the clusters using the implementations for the independence assumption (EM) and the Markov Chain of order 1 assumption (HMM). Then we will discuss the evolution of the likelihoods, the difference in their final value and finally we will obtain the optimal number of clusters needed to express the data by means of cross-validation.

In order to assess the correctness and properties of both algorithms, the same samples are used to train and validate the EM and the HMM clustering algorithms, the difference is that the EM algorithm ignores

the sequential information of the data. We implemented Python code to create the HMM sequences with an invented transition matrix A and initial probabilities π.

First of all we will see the evolution of the centroids with the iterations of the EM algorithm. This insightfull graph is shown in the next Figure. We can observe the centroids converge to the centre of the clusters with positive kappa (i.e. the blue line). For the negative kappa case, the centroid is actually perpendicular to the data as can appreciate in the green line. Technical note: Since the distribution is axially symmetric, sometimes the centroids are estimated in one direction and other times in the other, to avoid the resulting dumping, we constrained the solution to one side of the hyper-plane.

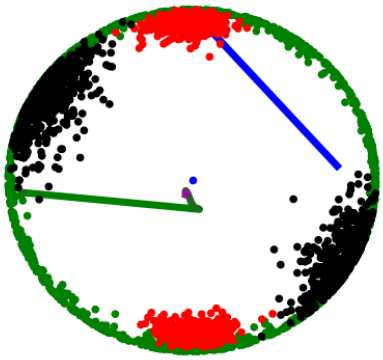


Fig. 6: Evolution of the centroids with the iterations.

As we can observe in Figure 7, the log-likelihood is monotonically increasing with the iterations until it converges. We can also notice that the HMM likelihood is higher. The likelihood value of the random initialization is omitted because it is too low compared to the others and it makes the figures less visible.

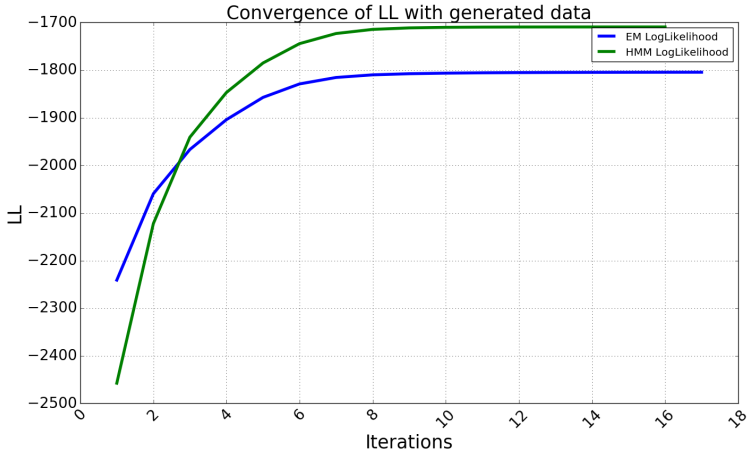


Fig. 7: LL convergence for HMM and EM.

The centroids obtained from both algorithms are almost the same, so the difference in likelihood comes from the difference in the model parameter between the π from EM and the $\{\pi, A\}$) of the HMM.

For the EM algorithm, since the samples are considered independent, the likelihood of the sequence is computed as the product of the likelihood of each individual data point x_i. Being the likelihood of each sample $LL(x_i)$ computed by means of marginalization over the possible clusters, modeled using the latent discrete variable Z, according to the cluster probabilities.

$$L_{EM}(x_t) = \sum_{k=1}^{K} P(s_t^n = k|\theta, Y) = \sum_{k=1}^{K} \pi_k p_k(x_t^n|B_k) = \sum_{k=1}^{K} \pi_k W(x_i|\mu_k, \kappa_k)$$

The incomplete likelihood of a given sequence with T samples $y^{(n)} = [x_1^{(n)}, x_2^{(n)}, ..., x_T^{(n)}]$ is therefore:

$$L_{EM}(y^{(n)}) = \prod_{i=t}^{T} LL_{EM}(x_t) = \prod_{t=1}^{T} \sum_{k=1}^{K} \pi_k W(x_t|\mu_k, \kappa_k)$$

In the case of the HMM, the samples are not independent so this marginalization is more complex and $P(s_t^n = i|\theta, Y)$ is obtained in a recursive fashion. Using the Forward algorithm to compute the likelihood we obtain the equation.

$$\begin{aligned} L_{HMM}(y^{(n)}) &= \sum_{i=1}^{I} \alpha_T^n(i) \\ &= \sum_{i=1}^{I} p_i(x_T^n|B_i)\left[\sum_{j=1}^{I} a_{ji}\alpha_{T-1}^n(j)\right] \\ &= \sum_{i=1}^{I} p_i(x_T^n|B_i)\left[\sum_{j=1}^{I} a_{ji}p_j(x_{T-1}^n|B_j)\left[\sum_{k=1}^{I} a_{kj}\alpha_{T-2}^n(k)\right]\right] \\ &= \sum_{i=1}^{I} p_i(x_T^n|B_i)\left[\sum_{j=1}^{I} a_{ji}p_j(x_{T-1}^n|B_j)\left[\sum_{k=1}^{I} a_{kj}...\left[\sum_{l=1} \pi_l \cdot p_l(x_1^n|B_l)\right]...\right]\right] \end{aligned} \tag{7.1}$$

As we can appreciate we can draw analogies between the incomplete likelihood of the EM and the HMM algorithms. The EM likelihood is a product of the marginalization of the data samples for each of the independent clusters, this marginalization is given by the π_{EM} parameters. For the HMM likelihood, we also have this marginalization, but the marginalization coefficients for the likelihood of the samples given a cluster, $p_l(x_1^n|B_l)$, instead of being π_{EM}, they are the marginalization of the previous forward step using the coefficients of the A matrix, $\sum_{j=1}^{I} a_{ji}\alpha_{T-1}^n(j)$.

Notice that if the coefficientes a_{ji} were independent of the state j, and therefore $a_{ji} = \pi_i$ then this likelihood would be the same as for the EM. So if the cluster parameters B were the same, the increase in likelihood of the HMM comes mainly from this difference of a_{ji} for different states j, where a_{ji} would be higher for the highest $p_j(x_{T-1}^n|B_j)$.

Notice also that, if the likelihood values $p_j(x_{T-1}^n|B_j)$ are very high or low, then the contribution of a_{ji} to the likelihood is relatively smaller. Probably its contribution should be analyzed independently of the total likelihood of the model.

For the generated data, the estimated (π, A) of the HMM are close to the ones they were generated from and the estimated π from EM is close to the asymptotic probability of the cluster of the HMM model. $\pi_{EM} = \pi_{HMM} \cdot A^{\inf}$, being the parameters of the clusters almost identical. In this case, we have 1208 samples, and the difference in likelihood of the 2 models is 100.

Now we will compute the number of clusters needed to express the data by means of cross validation (CV). We perform a 2-fold CV for different number of clusters K and visualize the results in Figure 26. As we can see, at $K = 3$ the training and validation accuracy saturates and therefore 3 is the number clusters needed to express the data.

In a normal machine learning problem we would have noisy data and so the likelihood of train will keep increasing (more) and the validation likelihood would decrease at some point due to over fitting, but since in this case the train and validation data was coming form the actual distribution we obtain this saturation.

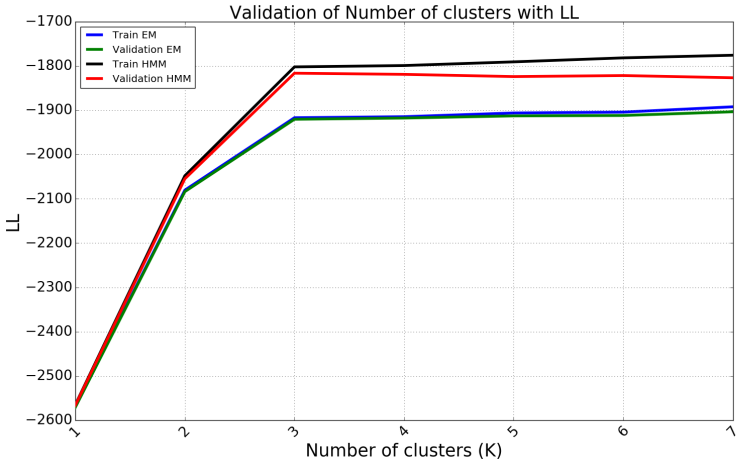


Fig. 8: Clusters validation for HMM and EM.

8. DATA DESCRIPTION AND PREPROCESSING

The objective of the project is analyze the mind-state of subjects which are presented with different stimulus, specifically, visual stimulus associated to see pictures of familiar faces from famous people and scramble pictures of faces. From now we will refer to each condition as ”Famous face” and ”Scramble face”.

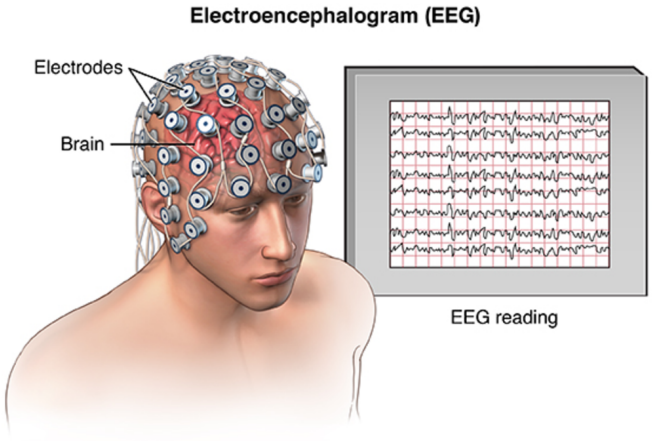


Fig. 9: EEG sampling

Then, EEG allows us to extract mind information by measuring the electric potential that the brain generate in the surface of the head. This technique is represented in Figure 9.

Our dataset contain data from 16 different subjects exposed to the aforementioned stimulus using 70 electrodes/channels. Each subject generated ∼295 trials from each of the conditions and each trial is a time series of 451 EEG samples in the span of 1 second. Every instant can also be represented as the scalp-maps in Figure **??** to have a spacial understanding of the brain activity. Figure 10 shows the average of the trials for both classes. As we can see the data is very noisy, and it is even more for single trials, which makes the task of classification very difficult.

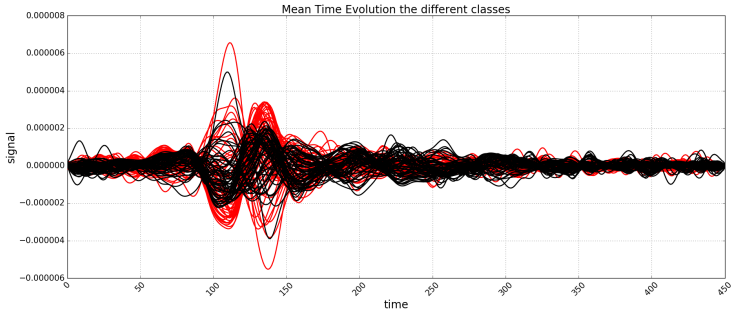


Fig. 10: Average of trials for "Famous face"(red) and "Scramble face"(black)

8.1. Preprocessing

The data has already being treated to reduce the noise, for example the electric spikes produced by eye movement have being filtered, but further prepossessing is needed. The 3 applied treatments are *average several trials* from same condition, *mean removal* of channels and *projection over the unit hypersphere*.

The aim of *averaging trials* is reducing the noise and it consist on getting one single trial by averaging all of the same condition for each subject. Let N denote the number of trials for the same condition, $\boldsymbol{x}_{n,i}$ the i-th sample from the trial *j* and $\boldsymbol{X}_{ave} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n\}$ the average trial where

$$\boldsymbol{x}_i = \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{x}_{i,j}$$

Mean removal consist on subtracting the mean of all the channels in every instant. This makes all the channels have the same importance since the the information is not in its mean value. If we denote D as the number of dimensions for each sample and $\boldsymbol{x}_{i,processed} = (x_{1,proc}, x_{2,proc}, ...x_{D,proc})$ the i-th sample from the mean trial after the mean removal, then

$$x_{i,proc} = x_i - \frac{1}{D}\sum_{j=1}^{D} x_j$$

Lastly, all the samples are divided by its L2-norm to *project over the unit hypersphere* which is the constraint imposed by the Watson distribution.

$$\boldsymbol{x}_{i,projected} = \frac{\boldsymbol{x}_{i,processed}}{||\boldsymbol{x}_{i,processed}||_2}$$

Another possible preprocessing would be using a subset of the channels or a linear combination of them using techniques such as PCA. Also we could have removed the mean of each of the 70-channels across all trials, as it is usually done in a standard machine learning problem. We implemented this preprocessing steps but did not continue to perform further analysis.

8.2. Effect of the preprocessing of the real data in the Watson distribution estimation

In the estimation of the cluster parameters (μ, κ), we compute both positive and negative κ posibilities and then keep the kappa with the highest weighted log likelihood obtained from the weighted samples that the cluster is responsible for. This creates the following problem.

For the negative kappa, κ_{neg}, one must pick as μ, the eigenvector corresponding to the lowest eigenvalue, μ_{neg}. This is the linear projection that has the lowest variance r, being:

$$r = \mu S \mu^T \qquad S = XX^T \tag{8.1}$$

In a problem with a high number dimensions, this eigenvalue is very low, around $1e-10$, which a very low explained variance r as well. This leads to the estimation of a highly negative κ_{neg} of the order of thousands and hundreds of thousands. It can be appreciated from the equation of the first guess of κ. The Kummer function would not converge in such a case and the estimation would crash (notice we implemented security measures for this case).

In our particular problem, we remove the mean of each sample vector, thus reducing the dimensionality of the each sample and so the last eigenvector might have no variance (leading to a negative infinite kappa). This effect can be appreciated in the next Figure where we removed the mean of the a set of 3D sample vectors.

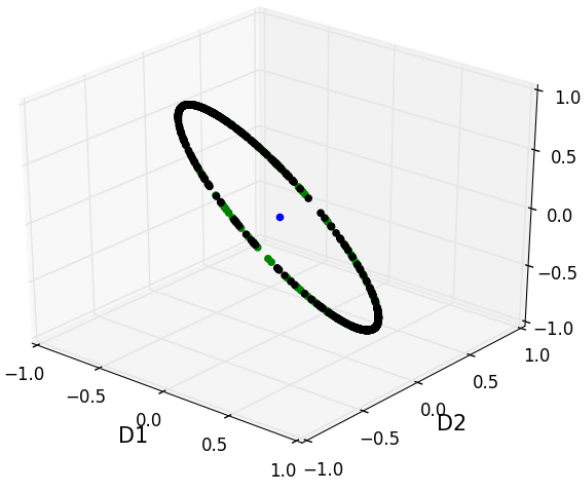


Fig. 11: Effect of the removal of the mean in 3D data

As a matter of a fact, the last eigenvalue of the estimation is always the uniform vector, and since the mean of each sample vector is 0 then all the points are projected to 0 when using the uniform eigenvector and therefore the variance explained by that eigenvector is $r = 0$. We conclude that in this case we cannot compute any negative κ due to the preprocessing performed in the data.

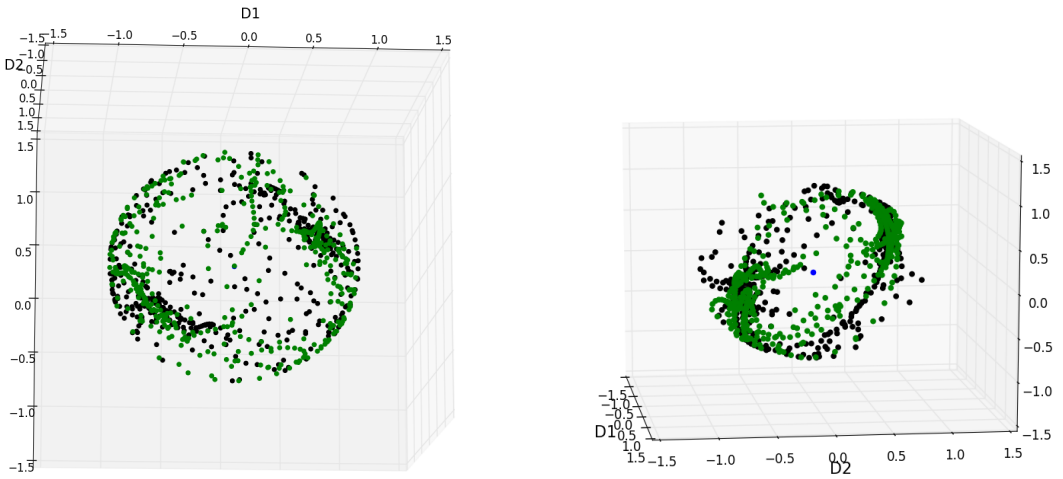
8.3. Exploratory analysis: subset of channels

Inspired by the limitations that the preprocessing and the high dimensionality generates to find clusters with negative κ, we have made some preliminary analysis of the data for a subset of channels. In this case, the explained variance of the last eigenvector is enough to be able to compute the negative κ. This selection of channels is done after the removal of the mean of the whole vector, but now our vector will not have 0 mean.

We could eventually create a complete system that selects them or combine them or we could just select them randomly. We can create several systems with different channels and combine the responses (like random patches (bagging)). This option is naturally parallelizable.

If we choose some 3 channels, in this case number 21,22 and 23 we can see generally the following distributions of the preprocessed samples. Figure 12 (a) shows the distribution of the 3 channels for a trial of the first class and a trial of the second class. We can observe what appears to be 2 Watson distributions, one with a positive kappa and the other with a negative kappa. The one with positive kappa is not perfectly spherical and the one with negative kappa actually pretty much always goes through the clusters with the positive kappa, taking into account that the is a lot of noise in the trials. It would make sense to fit a Fisher–Bingham distribution instead since the concentration is different along the dimensions. Different trials have different orientation of the negative kappa cluster.

Figure 12 (b) shows the same information but for the mean trial of both classes. We can appreciate a less noisy version of (a). It also seems like there is not a lot of discriminative information in these 3 channels, but other combinations might have better discriminative information.



(a) Scattering of points for all time instances of 1 trial of each class

(b) Scattering of points for all time instances of the average trial of each class

Fig. 12: Scattering of the data samples for the channels 21,22 and 23

Finally Figure **??** (c) shows, for a fixed time instant $t = \tau$ the scatter of the points for all the individual trials for both classes. We once again can observe this 2 Watson components behaviour.

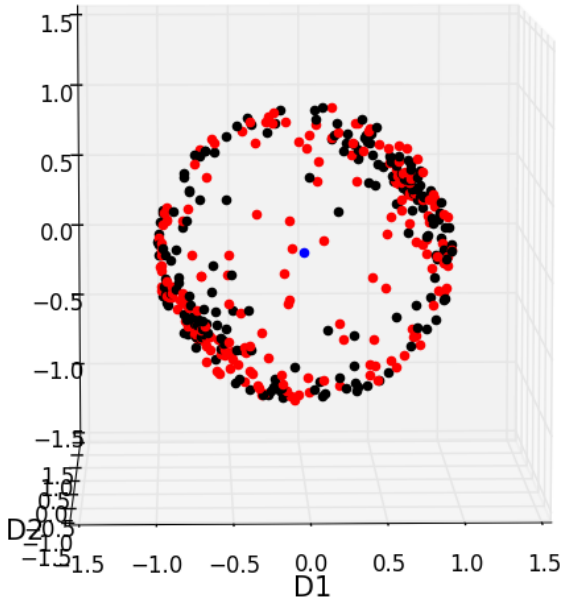


Fig. 13: Scattering of points for a given time instant of all the individual trials of both classes

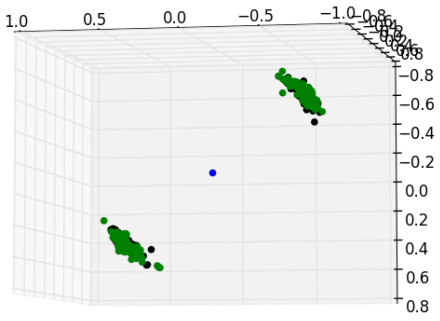
In order to check if the EM algorithm could compute these apparent 2 clusters of the data, we have run the EM algorithm for $K = 2$ clusters for the different time instances. That is, given a time instance $t = \tau$, we compute a 2 clusters EM with the 295 samples of the 295 trials of each class, using for example the data in the Figure **??** (c). We perform this operation for the 451 time instances and then plot the centroids found. The following figures show this centroids, (a) shows the centroid of the cluster with the highest κ and (b) shows the centroid of the cluster with lowest κ which is usually negative.

As we can appreciate, the centroids obtained for each of the clusters are very similar for the different time instances, which could in principal indicate some ergodicity or at least that the distribution at all time instances folows somewhat a 3 component EM distribution. There many possible reasons for this, the simplest one that the process if simply ergodic but we will rule this one out since it would make the analysis of discriminate information imposible.

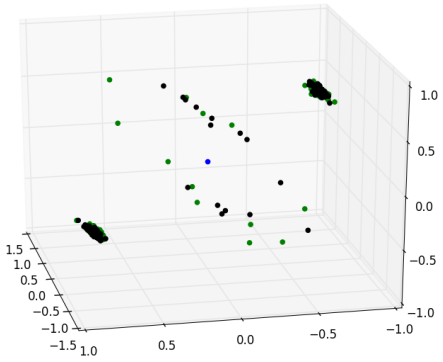
Maybe the apparent ergodicity is do the fact that, once we remove the scale information, the signal at all times is composed by zones of peaks and zones of sign conmutation, like in a sinusoidal function. So we have 2 main stages, when we are the spikes and when we are not, and since the signal conmutes the whole time,

and the distribution is scale invariant, that why we see that. For time instances in the normal place, this could explain the results.

Another possible explanation could be that the time instances are not perfectly aligned (the event we want to identify does not happen at the same time in the trials). We can observe also that they are apparently not discriminative for separating the clases. Of course it can be argued that this is just 3 channels, and that the discrimination between the classes has a high spacial component (some specific channels are active for those labels and are very discriminative). To solve this we could choose random channels until finding some combination that is discriminative.



(a) Trials for the same time instant for different classes



(b) Time evolution of trials differnt classes

9. MICRO-STATE MODELING RESULTS

9.1. Results for a single subject with 2-fold cross validation

In this section we will obtain the optimal number of clusters needed to express the data of single subject for both classes by means of 2-fold cross-validation (CV) of the data. We will compare the results obtained for the EM and HMM algorithms and then we will analyze the clusters found using the scalp maps and the time information

In order to perform this CV, we first split the trials into 2 balanced sets, one set of trials for training and the other for validations, then we apply to each set the the preprocessing explained in section 8.1. Using the mean trial obtained from the training set, we train the EM and HMM algorithms with 20 random initialization of the clusters, keeping the results that has the highest likelihood of the training set. This is done to avoid suboptimal results due to singularities and local minimums. We then obtain the likelihood of and validations mean trial as well. This processes is repeated changing the roles of the train and validations samples to complete the cross validation.

We perform this CV for different number of clusters K for the EM and HMM algorithm and for each class. The training and validation likelihoods are shown in Figures 15 and 16. As we can see the validations likelihood of both EM and HMM converge for $\sim$4 when characterizing "Scramble face" while it needs $\sim$6 for "Famous face".

The likelihood curves are noisy due to the local optimal problem that Expectation Maximization algorithm suffers from. This could be reduced using a higher number of initializations. We can observe how the training likelihood is always higher than the validation likelihood and monotically increasing with the number of clusters (or states) whereas the validation likelihood saturates at a certain number of clusters.

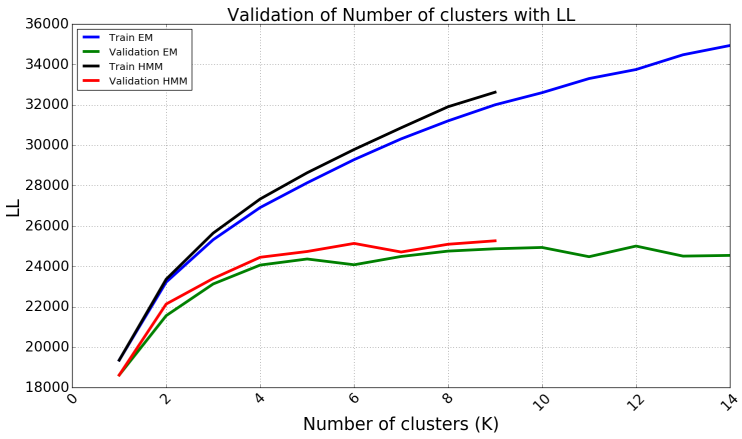


Fig. 15: LL HMM-EM for class "Scramble face"

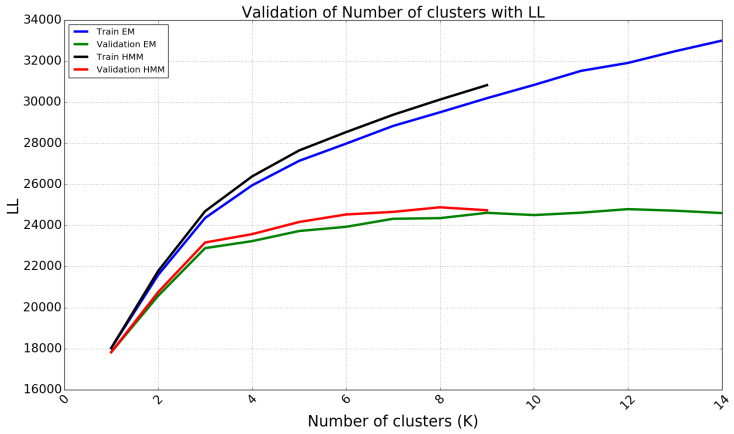


Fig. 16: LL HMM-EM for class "Famous face"

We can also appreciate how the likelihood of the HMM model is higher for both training and validation sets meaning that this model expresses better the data without further overfitting. The difference appears small in the graph due to the fact the probability density of the Watson samples is very high, this dominance effect was explained in section where we analyzed the generated data. The difference in likelihood grows with the number of clusters and around the optimal number of clusters is about 400. Taking into account that we only have 451 samples, we consider this a significant enough increase in likelihood.

9.2. Scalp maps of the Clusters

In this Section we will analyze the scalp maps of the clusters obtained for each of the models (EM and HMM) and each of the classes (Scramble face and Famous face). Figures 17 and 18 show these scalp maps.

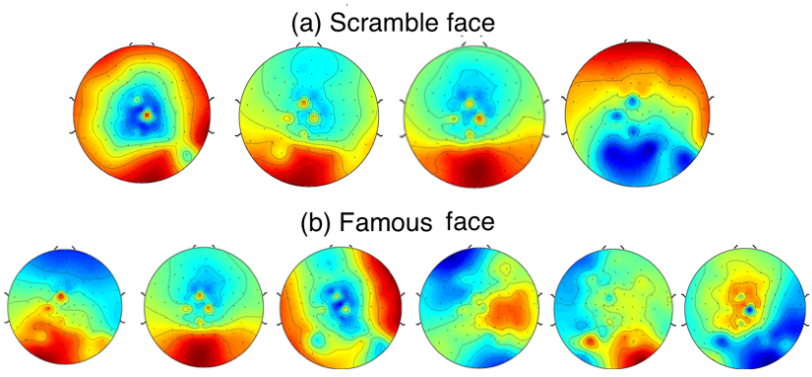


Fig. 17: Scalp maps from each class generated by EM

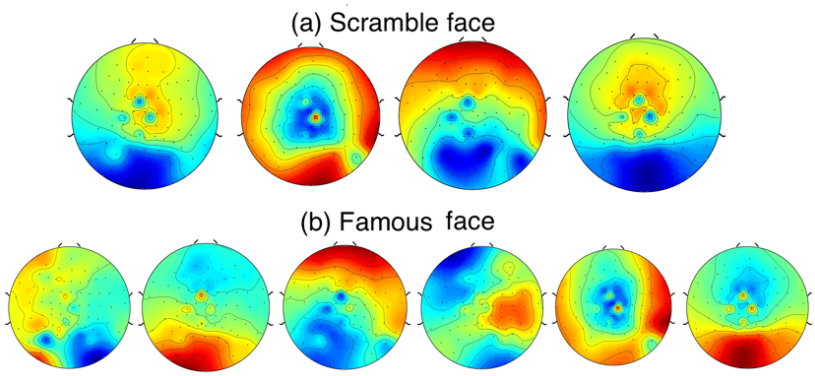


Fig. 18: Scalp maps from each class generated with HMM

The cluster sets obtained from both algorithms are very similar. The EM and EM-HMM converge at four clusters for the scrambled and six for the famous face class. This might be due to the different natures of the classes, but it also might be a sign of noisy electrodes. All clusters have heavy reliance on the visual cortex and the frontal lobe, which indicates a visually evoked reponse. Both algorithms put more emphasis on the right-Parietal lobe for the famous face, which could indicate recognition. Also, the famous face seems to get a stronger response from the right temporal lobe, which is connected with emotional memory.

However, there is one cluster in the famous face class that differs between EM and EM-HMM results. The cluster b6 in EM, and cluster b6 in EM-HMM don't have matches: b6-EM puts more emphasis on the right temporal lobe, while b2-EM-HMM on the visual cortex.

9.3. Time analysis of the Clusters

In order to perform an statistical analysis of the clusters, we will show, for the validation trial, the responsibilities of each of the clusters for the 451 temporal samples of the trial. In the case of the EM algorithm we will show the $r_t(i)$ values, and for the HMM algorithm we will use the $\gamma_t(i)$ which both have the same statistical meaning $P(s_t = i|Y, \theta)$.

The next Figure shows this temporal ocurrence of the clusters for the validation trial of the class "Scrambled face" for the EM algorithm, along with the initial time series for comparison.

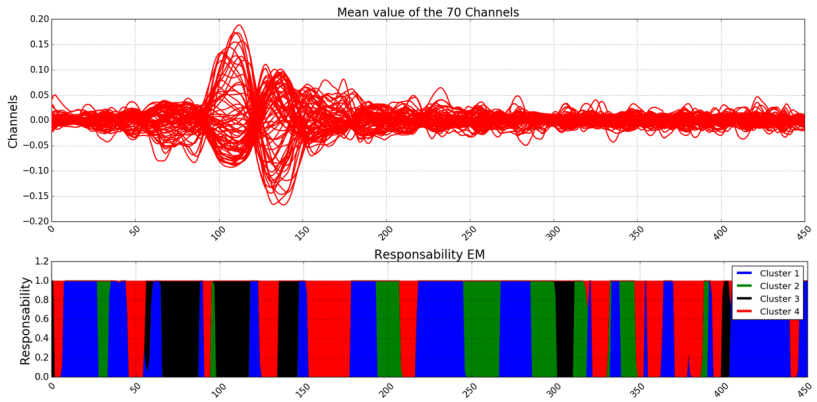


Fig. 19: Responsibilities of the EM algorithm for the scrambled face class using the average trial

As we can observe, responsibilities are either very close 0 or to 1, they are almost like the hard decision value of the cluster they were generated from. This means that the likelihood of the individual clusters is very unbalanced from cluster to cluster, most of the time there is a highly dominant cluster, which could mean that the clusters are apart from each other. Once again, since we have only 451 samples in a 70 dimensional space, the sparsity of the clusters is expected.

We can also appreciate time dependent information. Some clusters, for example the cluster 3 (black) almost only happen in a very specific area of the time series. Moreover, we are not constantly commuting from one cluster to the other as it would happen in a normal EM. Once we are in a cluster we stay in it for some time, this information would be better captured with the transition matrix of the HMM model as we will see next.

Similar graphs are generated for the rest of classes and models and their time series is shown in the next figures.

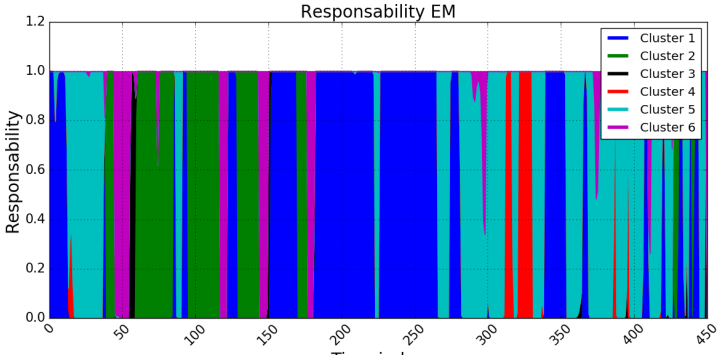


Fig. 20: Responsibilities of the EM algorithm for the famous face class using the average trial

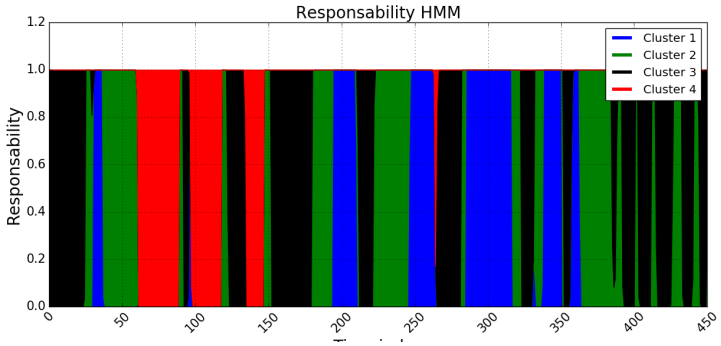


Fig. 21: Responsibilities of the HMM algorithm for the scrambled face class using the average trial

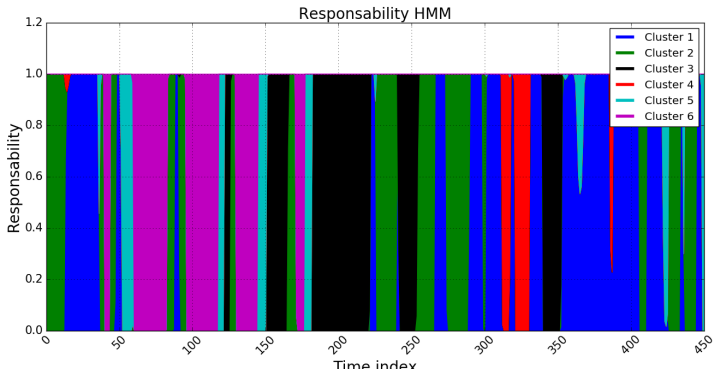


Fig. 22: Responsibilities of the HMM algorithm for the famous face class using the average trial

The previous figures show the same temporal behaviour explained before. Notice that in all of them there is a cluster that always mainly happens between the samples 50-150, this cluster is almost identical in all the scalp maps.

For the HMM models we can observe the time dependency in the trainisition matrix A, which has high values in the diagonal, meaning that once we are in a cluster (micro-state), we tend to stay in it. The next matrix shows the transition matrix for the "Scrambled face" class.

$$A = \begin{bmatrix} \mathbf{0.93} & 0.04 & 0 & 0.03 \\ 0.03 & \mathbf{0.88} & 0.08 & 0.01 \\ 0.01 & 0.08 & \mathbf{0.89} & 0.02 \\ 0.01 & 0.05 & 0.01 & \mathbf{0.93} \end{bmatrix} \tag{9.1}$$

9.4. Results for 5-fold for 1 person

We also tried to cross-validate the number of clusters needed to express the statistical distribution of a single person using a 5-fold scheme in which we divide the trials into 5 sets of the same size, average them and then

train the EM algorithm with 4 of them, validating with the remaining average sample. Of course this process is repeated 5 times, permutating the validation average sample.

The results of such CV for both classes (class 0 is the ”famous face” class and class 1 is the ”scrambled face” ”class) is shown in the next two figures. As we can observed, the number of clusters needed converge in a much higher number than in 2-fold, around 22 clusters are needed, roughly 4-5 times more than when training the EM algorithm with just one average sample.

Our first inference about this difference is that since now we are training the EM algorithm with 4 average samples that are more noisy (since they come from the average of a lower number of trials), the noise makes the clusters to expand (lower κ) and rotate, thus creating more potential Watson Distribution components that would be explained better using more clusters in the EM algorithm.

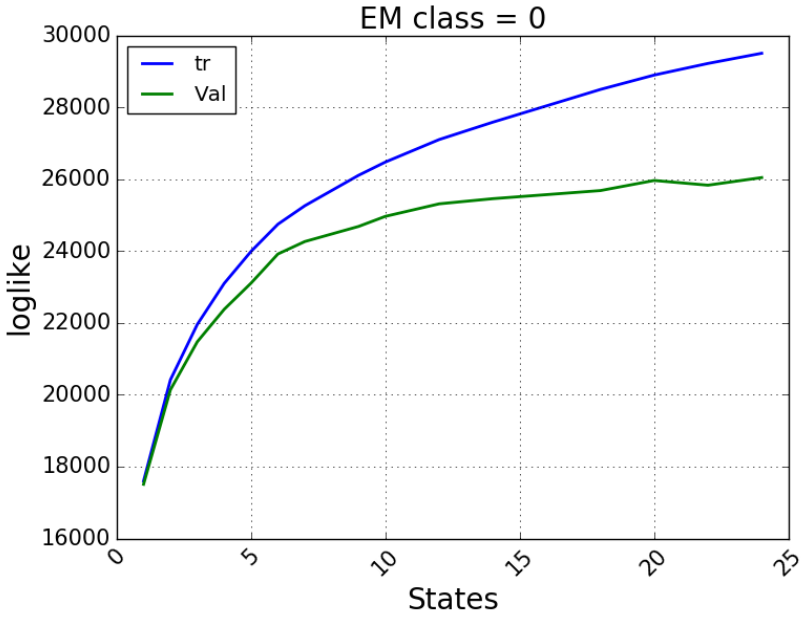


Fig. 23

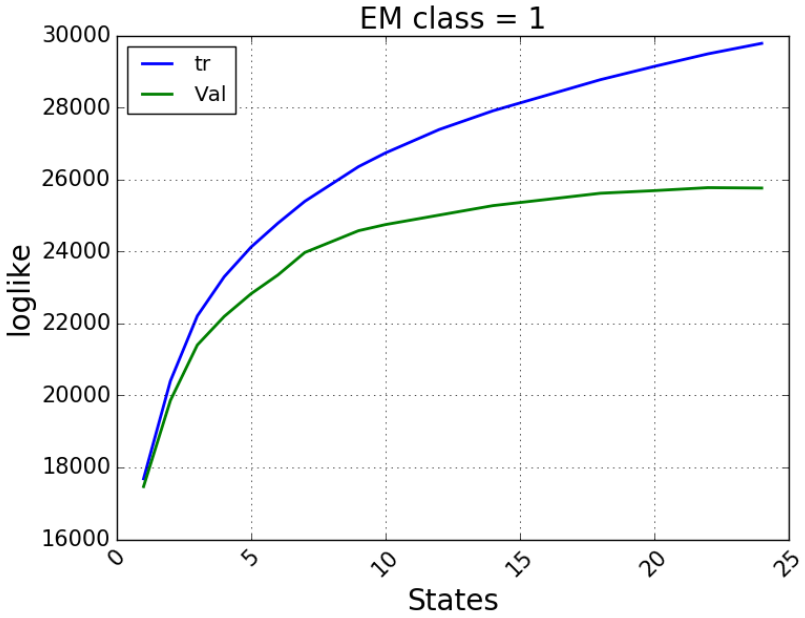


Fig. 24

We did not further analyzed this case, as we decided to focus on the 2-fold realization that has more interpretable results.

9.5. Results for several subjects

We also tried to cross-validate the number clusters needed to express the data using the information from all the available 16 subjects. In this scheme, we compute the average trial of all the subjects using the preprocessing described. Then we perform a leave-one-out cross-validation with the 16 mean trials. The results of such crosvalidation for the EM

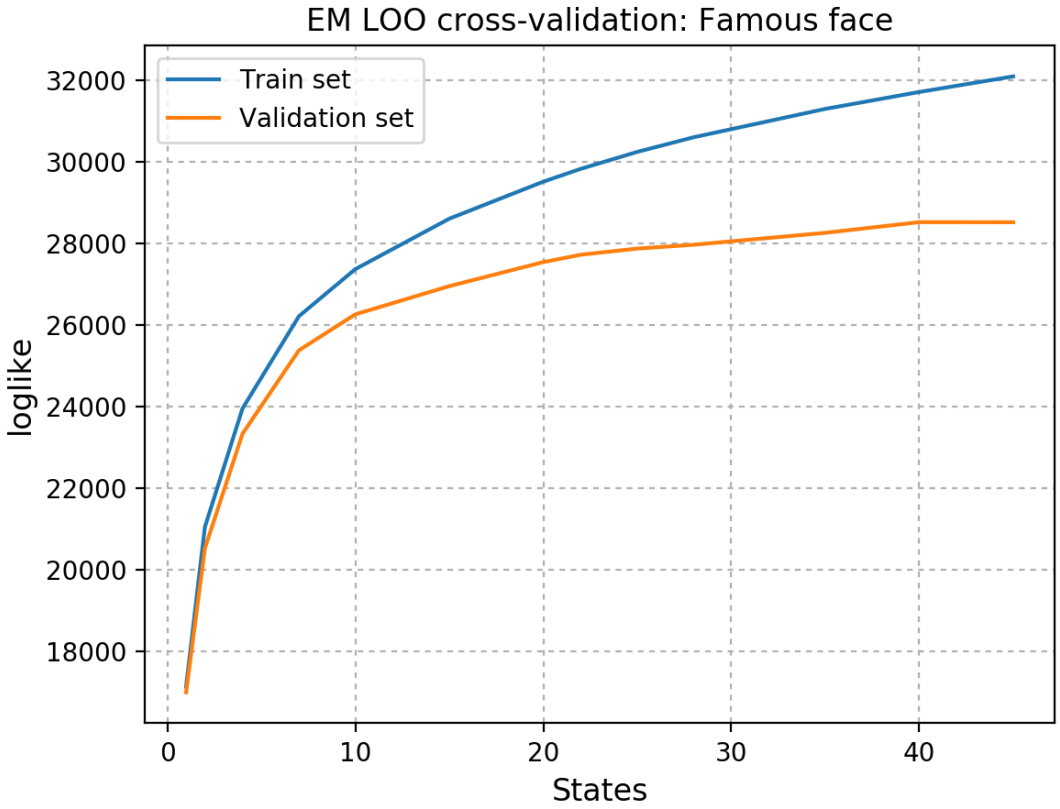


Fig. 25: EM Leave One Out cross cross-validation for several subjects

10. CLASSIFICATION RESULTS

In this section we will describe our attempts to use the information of the generated clusters to try to classify the individual noisy trials respect to the two classes, ”famous face” and ”scrambled face”.

First, a basic Bayesian approach was implemented where, for a given trial, we compute its likelihood for both sets of clusters (one set of cluster per class). The classifier will then classify the trial as belonging to the class with the highest likelihood, if we have priors on the classes then we use them to weight the probability, obtaining a Maximum a posteriori classifier. In practice we had the same number of trials for each class so the priors are non-informative.

The accuracy of this system is very low, always around 50% classification error. One possible hypostesis (appart from the obvious ones that the prepossessing, transformation, Watson Distribution and EM algorithm are not able to find discriminative information) is that many of the clusters obtained are non-informative (sometimes almost identical for both classes), and therefore they do not provide discriminative information for the classification problem. Their contribution to the overall likelihood makes it more a ”noisy” feature regarding the classifcation problem, so it ”masks” the constribution of the specific clusters that are class-discriminative.

So one possible idea would be to, instead of computing the likelihood of the trial, we compute some other features from the individual clusters. For example, if the discrimination could be on how much percentage of the time we are in a given cluster, we can just get compute the number of times that we are in that state. Or if the discrimination would be on how big is the likelihood of a given cluster (maybe also using time windows (only computing the likelihood for certain samples). Maybe we should just compute the likelihood of a sample for a given cluster if that sample belongs to that cluster. There are many possible options.

Intuitions says that the information extracted from the clusters should be according to the pattern that we are trying to find, but we lack this kind of information ourselves. For example, if we plot the mean trial for the 70 dimensions for the different classes as we did in the data preprocessing section, we can appreciate how the peak

values in the samples 50-150 are higher for one class than for the other on average. If we plot the "normalized in module" data, we cannot really identify any patterns visually, but that is probably a representation where we cannot appreciate them easily anyway.

Regardless of how we will compute the clusters and how we obtain characteristics from them, after obtaining this information (new features) from the individual clusters, then we create a vector of these features and formulate a normal ML classification problem. Several classifiers have been used from different libraries: Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Gaussian Naive Bayed, Linear Support Vector Machine (LSVM), Gaussian Kernel SVM, K Nearest Neighbours, Classification Trees, Random Forest, Extremely Randomized Trees. So far, a solution with a number of clusters $K = 3$ and the LSVM classifier could obtain some 60% accuracy in both train and validation. The features are the product of responsibilities of the samples for a given trial, for all of the clusters. It can be a baseline for the future.

11. CONCLUSION

The main fruitful conclusions are obtained from the analysis of the experiments for 1 person using a 2-fold cross-validation scheme to obtain the clusters. They are summarized in the following points.

- The moW and EM-HMM can successfully model the data better than the simpler modified K-means clustering. This models offer a bigger expressivity without causing over-fitting.
- The number of Watson distribution clusters needed to express the EEG data for a single person is in the range 4-6.
- The data reflects temporal correlation indicated by the EM-HMM transition matrix and likelihood of both methods. This indicates that the EM-HMM algorithm can model the data better.
- The clusters obtained for the EM and EM-HMM are very similar, since the data is sparse, the clusters are very distinguishable.
- The scalp map indicates high visual perception for both classes and "famous face" class has an emotional response.
- Due to the highly noise that individual trials contain we could not build a successful classification system using the generated clusters.

12. OTHER IMPLEMENTATIONS AND FUTURE WORK

There are many future lines of research throughout the report, in this section we will gather some of them along with some basic implementation of some of them.

12.1. Using a similarity measure for the clusters

From this paper we could see that many of the clusters for the classes where very similar, they have similar κ and their μ have about 92% cosine similarity. We could use this knowledge to use cosine similarity measures to detect them and treat the clusters accordingly. Maybe we should not use the samples that belonging to them in the discriminative approach. We could also use this information during the training of the EM algorithm to avoid have similar clusters with in the same class.

12.2. Using the violation of the EM as discriminative information

As we saw, the data has time information that violated the EM algorithm assumtions, maybe we can quantize and use these violations to obtain discrinatory information.

12.3. Modifications of the EM algorithm

Observing this time dependency of the clusters,they usually happen in the same time intervals, we could try to implement a "Time windowed EM". Since apparently we want our clusters to be time-domain specific, we can try to put that information into the EM. Probably one of the most straight forward ways to do so is to just divide the time series into time intervals and train each interval with a different EM. But maybe we cannot train the position and length of the windows. Also, probably the clusters on the windows would have common clusters (like a residual noise cluster). We can also do a smoothed version of the previous where we give each sample a time dependent weight, a contribution to the parameters of the cluster taken into account the time information, and this weights would follow some time-distribution. A Bayesian approach would be suitable. At the end, to

compute the probability of a sample belonging to a cluster and then we multiply it for the time index weight. For the case of fixed time intervals, the weight is 0 when the sample does not belong to the interval. This is just a draft idea, but it seems reasonable, maybe there is some work done on that we do not know of. For sure there must be other mathematical frameworks that contain this information about changing the state over time, maybe some flavour of Markov Switching models.

Using a Batch for of the EM to give more randomness to the EM and scape local minima. We could train with the means as we did before but in each iteration, we train only with one of them and then we change.

We could use a "Time - windowed EM". Like in communcation systems where the distribution changes over time and for each new sample that we get we perform an iteration of the EM. We select the clusters when they are stable in a time range.

Use Improved initialization and Variational Inference.

12.4. Using other distributions

From looking at the data, it can also be noticed that the clusters are more narrowed in one dimension than the other, probably there is an extension of the Watson distribution that allows that, it would be very nice, probably is would be way more complicated of course. Probably the Bingham and Fisher–Bingham but we havent done too much study about them.

12.5. Other data pre-processing

We could work with correlations instead of the original data. They could also vary in scale and sign so Watson distribution could still make sense.

We could also only filter samples by correlation or other parameter.

Instead of averaging all samples, we could get several averages of subsets for training. Instead of training the EM of each label with the average of all its trials, we randomly create N subsets of trials and do a number of N different mean trials, so that we have more (less- noisy) data point to train the clusters.

12.6. Using the Principal Component Analysis decomposition

We also tried to reduce the dimensionality of the data using a PCA decomposition but since the new set of features would be a linear combination of the sensors, then it would be impossible to make the same analysis for the scalp maps. The main first components of the PCA projection are shown in the following:

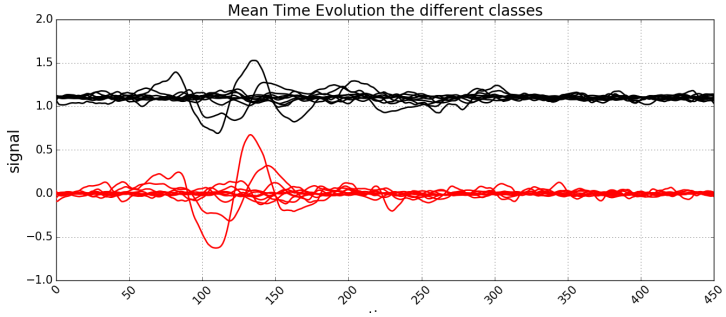


Fig. 26

Other possible way of the PCA could be to initialize the cluster for the EM algorithm. In the Watson distribution, the way to obtain the μ parameter could actually be seen as some kind of weighted PCA where we take as μ the eigenvector that has the most explained variance. So we could use some vectors of an initial PCA to initialize the clusters.

References

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

Makeig S. Delorme, A. Eeglab: An open source toolbox for analysis of single-trial eeg dynamics including independent component analysis. URL `http://sccn.ucsd.edu/eeglab/download/eeglab_jnm03.pdf`.

M. Gschwind, M.; Hardmeier. Fluctuations of spontaneous eeg topographies predict disease state in relapsing-remitting multiple sclerosis. *NeuroImage*, 2016. doi: https://dx.doi.org/10.1016\%2Fj.nicl.2016.08.008.

M. Hatz, F.; Hardmeier. Microstate connectivity alterations in patients with early alzheimer's disease. *Alzheimer's Research Therapy*, 2015. doi: https://alzres.biomedcentral.com/articles/10.1186/s13195-015-0163-9.

Lelic D. Petrini L. Hennings, K. An automated method for micro-state segmentation of evoked potentials. *Journal of Neuroscience Methods*, 2008. doi: https://www.researchgate.net/deref/dx.doi.org\%2F10.1016\%2Fj.jneumeth.2008.09.029.

Pascual-Leone A. Khanna, A. Reliability of resting-state microstate features in electroencephalography. 2014. doi: https://doi.org/10.1371/journal.pone.0114163.

D.; Merlo M. C. G.; Kochi K.; Hell D.; Koukkou M. Koenig, T.; Lehmann. A deviant eeg brain microstate in acute, neuroleptic-naive schizophrenics at rest. *European Archives of Psychiatry and Clinical Neuroscience*, 249(4):205–211, 1999. doi: https://doi.org/10.1007\%2Fs004060050088.

Prichep L.S. Lehmann D. Koenig, T. Millisecond by millisecond, year by year: normative eeg microstates and developmental stages. *NeuroImage*, 2002. doi: https://doi.org/10.1006/nimg.2002.1070.

D. Lehmann, 2009.

Michel M. Christoph Lehmann, D. Eeg-defined functional microstates as basic building blocks of mental processes. *Clinical Neurophysiology*, 122(6):1073–1074, 2010. doi: https://doi.org/10.1016/j.clinph.2010.11.003.

N. Michalopoulos, K.; Bourbakis. Microstate analysis of the eeg using local global graphs. *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, 2013. doi: https://doi.org/10.1109/BIBE.2013.6701583.

Michel C. M. Lehmann D. Pascual-Marqui, R. D. Segmentation of brain electrical activity into microstates: model estimation and validation. *IEEE Transactions on Biomedical Engineering*, 1995. doi: https://doi.org/10.1109/10.391164.

Pedroni A. Langer N. Hansen L. K. Poulsen, A. T. Microstate eeglab toolbox: An introductionary guide, 2017. URL `http://dx.doi.org/10.4249/scholarpedia.7632`.

Karp Dimitrii Sra Suvrit. *The multivariate Watson distribution: Maximum-likelihood estimation and other aspects*. 2011.

Henson Richard N Wakeman, Daniel G, 2015.

F. Zdyb. A probabilistic model for segmenting eeg into microstates. 2016. doi: https://semisupervised-learning.compute.dtu.dk/wp-content/uploads/2016/08/sciposter.pdf.