Distributed Learning and Inference for Gaussian Mixtures

Pedro Valdeira
pedromvaldeira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

Abstract

Distributed setups consist of a team of agents linked by a sparse communication network: the network does not link each agent to all others; it only links each agent to a reduced number of nearby agents. After each agent measures local data, they wish to collaborate over the communication network to learn from the collective dataset. What enables such learning is a distributed algorithm—an algorithm that lays down the specific messages that each agent sends through its channels. Distributed setups can model applications such as distributed cooperative target localization by a team of Unmanned Aerial Vehicles (UAV) communicating via wireless channels, having access to local measurements. We design distributed algorithms that address two questions in distributed setups: (1) How to fit a mixture of K Gaussians to a measured dataset? And (2) how to classify a measured data point into one of K given Gaussians? For (1), we design three distributed approximate, yet scalable, expectation-maximization (EM) algorithms. Our algorithms enable the agents to fit a Gaussian mixture model (GMM) to a dataset that is scattered across the network by features. As the literature considers datasets scattered instead by data points, our algorithms expand the state-of-art in a direction more useful in practice. For (2), we design a distributed detector that performs close to the optimum centralized detector and, more importantly, is scalable. As the literature considers Gaussians with particular structures (such as diagonal covariances), we expand on the state-of-art by allowing arbitrary structures.

Keywords: Distributed algorithms, Gaussian mixture model, distributed EM algorithm, distributed detector

1. Introduction

In distributed setups, agents start by measuring data with onboard sensors and then wish to collaborate with each other—over an underlying communication network—so that all agents learn from the collective dataset. An example of a distributed setup with nine agents, linked by a communication network with eleven channels, is given in Figure 1.

Agents learn from the collective dataset through a distributed algorithm: an algorithm that lays down which messages should be passed between neighbor agents (agents linked by a channel).

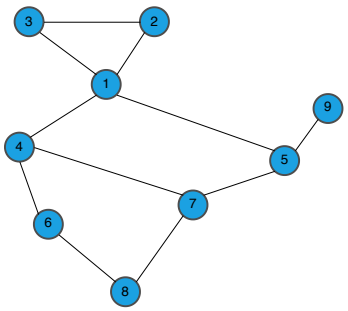

Figure 1: A distributed setup as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: a node in $\mathcal{V}$ is an agent measuring data; an edge in $\mathcal{E}$ is a channel linking two agents. Such setup models, for instance, a set of unmanned aerial vehicles (UAV) linked by wireless channels, the UAVs taking local measurements and wishing to collaborate to infer the position of an intruder.

Why distributed algorithms? Distributed algorithms dispense with a central agent, the agent that, in classic centralized setups, receives all raw data from the network and carries out the learning alone. In distributed setups such central agent is unfit. When it malfunctions, the whole apparatus is brought to a halt. But, even when the central agent is operating normally, in large networks the information flow routed from all agents to the central agent easily overwhelms the capacities of the

channels, causing acute packet jams and high delays. A central agent impedes the networks to scale. Distributed setups demand distributed algorithms.

The distributed algorithms designed in this thesis. We design distributed algorithms that answer two questions in distributed setups:

- How to fit a mixture of K Gaussians to a measured dataset?
- How to classify a measured data point into one of K given Gaussians?

2. How to fit a mixture of K Gaussians to a measured dataset?

As a motivation example, consider a large number N of agents deployed over a region to study the field of values of some physical quantity, say, the concentration of a dangerous gas. Each agent can measure the quantity at its location, carry out a modest amount of computations, and communicate via wireless with nearby agents. Suppose the agents take M measurements in sync, along a time interval, the mth measurement being the vector $x_m = (x_{m1}, \ldots, x_{mn}, \ldots, x_{mN})$.

Dataset is scattered by features. In each vector x_m (a snapshot of the field), agent n knows only the component x_{mn}. Commonly, the components of a data vector are called *features*. So, the matrix dataset

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \\ \vdots \\ x_M^T \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1n} & \cdots & x_{1N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & \cdots & x_{mN} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{M1} & \vdots & x_{Mn} & \cdots & x_{MN} \end{bmatrix},$$

is scattered across the N agents by features. Agent n knows only feature n (the column n of X).

Gaussian mixture model (GMM). In modelling the behavior of a random vector, a widespread approach is to fit to the dataset a gaussian mixture model (GMM). A GMM postulates that each data point of the dataset is an independent draw of a mixture of K Gaussians. Specifically, the GMM assigns a probability density function that, for a generic data point $x = (x_1, \ldots, x_N)$, evaluates to

$$p(x) = \sum_{k=1}^{K} \pi_k N\left(x | \mu_k, \Sigma_k\right), \tag{1}$$

where $N(\cdot|\mu, \Sigma)$ denotes a gaussian with center μ and covariance Σ. The prior probabilities for the K Gaussians are $\pi = \{\pi_k \colon 1 \leq k \leq K\}$, their centers $\mu = \{\mu_k \colon 1 \leq k \leq K\}$, and their covariances $\Sigma = \{\Sigma_k \colon 1 \leq k \leq K\}$. We refer to $\theta = (\pi, \mu, \Sigma)$ as the GMM parameter.

Expectation-Maximization (EM) algorithm. Fitting a GMM to a dataset is usually carried out by the expectation-maximization (EM) algorithm [1]. We review the details of the EM algorithm in chapter 2 of the thesis. Here, we just single out the main computations of the E- and M-steps.

Let $\theta^{(t)} = \left\{\pi^{(t)}, \mu^{(t)}, \Sigma^{(t)}\right\}$ be the current guess of the GMM parameters. The E-step computes, for each data point x_m, the K *responsibilities* γ_{mk} (which are nonnegative and sum to one), where

$$\log \gamma_{mk} \propto \log \pi_k^{(t)} - (1/2) \log \left|\Sigma_k^{(t)}\right| \tag{2}$$
$$-(1/2) \left(x_m - \mu_k^{(t)}\right)^T \left(\Sigma_k^{(t)}\right)^{-1} \left(x_m - \mu_k^{(t)}\right).$$

The M-step updates the GMM parameter to $\theta^{(t+1)} = \left\{\pi^{(t+1)}, \mu^{(t+1)}, \Sigma^{(t+1)}\right\}$ as follows:

$$\pi_k^{(t+1)} = \frac{\sum_{m=1}^{M} \gamma_{mk}}{M}, \tag{3}$$

$$\mu_k^{(t+1)} = \frac{\sum_{m=1}^{M} \gamma_{mk} x_m}{\sum_{m=1}^{M} \gamma_{mk}}, \tag{4}$$

$$\Sigma_k^{(t+1)} = \frac{\sum_{m=1}^{M} \gamma_{mk} \zeta_{mk}^{(t+1)} \left(\zeta_{mk}^{(t+1)}\right)^T}{\sum_{m=1}^{M} \gamma_{mk}}, \tag{5}$$

where $\zeta_{mk}^{(t+1)} = \left(x_m - \mu_k^{(t+1)}\right)$.

The problem addressed: a distributed EM. The standard EM, however, can run only at a central agent that knows the whole dataset. In this thesis we want to design a distributed EM.

Although many researchers have designed distributed EM algorithms (of which [2] and [3] are but two examples, the thesis enumerating more), they assumed that the dataset X is scattered across agents in a different way. Specifically, they assume X is scattered by rows, whereas we have to deal with an X scattered by columns—the natural setting of many applications, as illustrated above. As detailed in chapter 3 of the thesis, dealing with a dataset that is scattered by columns is intrinsically harder.

Our contribution: Three distributed approximate EM algorithms. As we show in chapter 3 of the thesis, we can design a distributed exact EM algorithm for our setup. This exact EM, however, claims such inordinate amounts of communications and memory capacities across the network that it scales badly with network size. There-

fore, we design three approximate yet scalable alternatives: the *Diagonal Correlation Distributed EM* (EM-DiagC); the *Block Diagonal Correlation Distributed EM* (EM-BlockDiagC); and the *Full Correlation Distributed EM*, (EM-FullC). In the rest of this section, for lack of space, we outline only EM-BlockDiagC and EM-FullC.

2.1. Block Diagonal Correlation Distributed EM (EM-BlockDiagC).

To design EM-BlockDiagC, we restrict ourselves to a *block-diagonal* GMM. Specifically, in the parameter $\theta = \{\pi, \mu, \Sigma\}$, each of the K covariance matrices in $\Sigma = \{\Sigma_k \colon 1 \le k \le K\}$ is block diagonal,

$$\Sigma_k = \begin{bmatrix} \Sigma_{k1} & 0 & \dots & 0 \\ 0 & \Sigma_{k2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \Sigma_{kB} \end{bmatrix}, \tag{6}$$

all K covariance matrices sharing the same block-diagonal structure.

Which block-diagonal structure? The structure must be chosen carefully; otherwise, the ensuing distributed algorithm will not scale well. We choose a sparsity pattern that is submissive to the communication network: entry (i, j) of Σ_k can be nonzero *only if* agents i and j are neighbors. Other than that, we want the set of nonzero entries of the covariances Σ_k to be as large as possible, so as many correlations as possible are captured by the model.

The hub algorithm. To choose a block-diagonal structure, we devised the *hub algorithm*, which runs once, as detailed in chapter 3 of the thesis. The hub algorithm accepts as input a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and returns as output a block-diagonal structure such as (6). In a nutshell, the hub algorithm extracts hubs—subgraphs consisting of a node along with its neighbors—one hub at a time from the communication graph, with the goal of extracting hubs as large as possible because each hub induces a block in (6). Figure 2 shows an example.

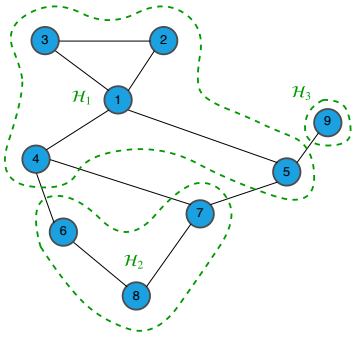

Figure 2: The hub algorithm extracts $B = 3$ hubs, $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$. Each hub is a tiny tree, a node standing at the root, the remaining nodes surrounding the root. The *root* of hub $\mathcal{H}_1$, denoted by $r(\mathcal{H}_1)$, is node 1; likewise, $r(\mathcal{H}_2) = 8$ and $r(\mathcal{H}_3) = 9$. The *leaves* of hub $\mathcal{H}_1$, denoted by $\mathcal{L}(\mathcal{H}_1)$, are the nodes surrounding the root, that is, $\{2, 3, 4, 5\}$; likewise, $\mathcal{L}(H_2) = \{6, 7\}$ and $\mathcal{L}(\mathcal{H}_3) = \emptyset$.

Memory organization. The hub algorithm divides the network in B hubs and agents into *roots* or *leaves*, as illustrated in Figure 2. Implicitly, each generic data vector $x \in \mathbf{R}^N$ is also divided (after a suitable permutation) in B subvectors: $x = (x_{\mathcal{H}_1}, \dots, x_{\mathcal{H}_B})$, where $x_{\mathcal{H}_b}$ contains the components x_n for $n \in \mathcal{H}_b$.

We assume that, at the start of the tth iteration of EM-BlockDiagC, the following information is in the memory of each root agent $r(\mathcal{H}_b)$: the dataset of hub $\mathcal{H}_b$, that is, $\{x_{m\mathcal{H}_b} \colon 1 \le m \le M\}$; the bth subvector of the K centers $\mu_k^{(t)} = \left(\mu_{k\mathcal{H}_1}^{(t)}, \dots, \mu_{k\mathcal{H}_B}^{(t)}\right)$, that is, $\left\{\mu_{k\mathcal{H}_b}^{(t)} \colon 1 \le k \le K\right\}$; the bth block of the K covariance matrices

$$\Sigma_k^{(t)} = \begin{bmatrix} \Sigma_{k\mathcal{H}_1}^{(t)} & & \\ & \ddots & \\ & & \Sigma_{k\mathcal{H}_B}^{(t)} \end{bmatrix},$$

that is, $\left\{\Sigma_{k\mathcal{H}_b}^{(t)} \colon 1 \le k \le K\right\}$; and the parameters $\{\pi_k^{(t)} \colon 1 \le k \le K\}$. Agents that are leaves need no memory storage.

E-step. We need to compute $\log \gamma_{mk}$ in (2). For our block-diagonal GMM, update (2) is as follows:

$$\log \gamma_{mk} \propto \log \pi_k - \frac{1}{2} \sum_{b=1}^{B} \log |\Sigma_{k\mathcal{H}_b}| \tag{7}$$
$$- \frac{1}{2} \sum_{b=1}^{B} \left(x_{m\mathcal{H}_b} - \mu_{k\mathcal{H}_b}\right)^T \Sigma_{k\mathcal{H}_b}^{-1} \left(x_{m\mathcal{H}_b} - \mu_{k\mathcal{H}_b}\right).$$

(We dropped the superscripts $(\cdot)^{(t)}$ for clarity.) This quantity can be delivered to all agents as follows:

first, each bth term

$$\log|\Sigma_{k\mathcal{H}_b}| + (x_{m\mathcal{H}_b} - \mu_{k\mathcal{H}_b})^T \Sigma_{k\mathcal{H}_b}^{-1} (x_{m\mathcal{H}_b} - \mu_{k\mathcal{H}_b}) \tag{8}$$

is computed at the corresponding root agent $r(\mathcal{H}_b)$; then, each root agent $r(\mathcal{H}_b)$ passes this bth term to all agents (if any) that are leaves of its hub; finally, all agents access (7) by resorting to the distributed *consensus* algorithm (as detailed in chapter 3 of the thesis). The E-step thus needs communications between agents.

M-step. We need to update the GMM parameter to $\theta^{(t+1)} = \left\{\pi^{(t+1)}, \mu^{(t+1)}, \Sigma^{(t+1)}\right\}$ as follows:

$$\pi_k^{(t+1)} = \frac{\sum_{m=1}^M \gamma_{mk}}{M}, \tag{9}$$

$$\mu_{k\mathcal{H}_b}^{(t+1)} = \frac{\sum_{m=1}^M \gamma_{mk} x_{m\mathcal{H}_b}}{\sum_{m=1}^M \gamma_{mk}}, \tag{10}$$

$$\Sigma_{k\mathcal{H}_b}^{(t+1)} = \frac{\sum_{m=1}^M \gamma_{mk} \zeta_{mk\mathcal{H}_b}^{(t+1)} \zeta_{mk\mathcal{H}_b}^{T,(t+1)}}{\sum_{m=1}^M \gamma_{mk}}, \tag{11}$$

where $\zeta_{mk\mathcal{H}_b}^{(t+1)} = \left(x_{m\mathcal{H}_b} - \mu_{k\mathcal{H}_b}^{(t+1)}\right)$. At each hub $\mathcal{H}_b$, the root agent $r(\mathcal{H}_b)$ is able to compute these updates. The M-step thus needs no communications.

The EM-BlockDiagC algorithm is outlined in Algorithm 1.

Algorithm 1 The EM-BlockDiagC algorithm

1. Choose initial parameters $\theta^{(0)}$ and set $t = 0$
2. E-step: root agent $r(\mathcal{H}_b)$ computes (8), then all agents communicate to jointly find γ_{mk} (see (7))
3. M-step: root agent $r(\mathcal{H}_b)$ updates to $\theta^{(t+1)}$ as in (9), (10), and (11)
4. update $t \leftarrow t + 1$ and repeat steps 2 and 3

2.2. Full Correlation Distributed EM (EM-FullC).

In EM-FullC, all agents act as root agents. Specifically, EM-FullC has an initial step (ran once), which we denominate as *prelude*. In this prelude, each agent n is turned into a root agent: agent n becomes the root agent of hub $\mathcal{H}_n$ whose leaves are the neighbors of agent n. So, for EM-FullC, there are as many hubs as agents. As in the EM-BlockDiagC algorithm, root agents immediately receive the datasets of their leaves, which they add to their starting dataset: for agent n, this means that its initial dataset $\{x_{mn} : 1 \le m \le M\}$ is widened to $\mathcal{D}_n = \{x_{m\mathcal{H}_n} : 1 \le m \le M\}$,where $x_{m\mathcal{H}_n}$ contains the features measured by agent n and its neighbors on the mth data point.

Overview of EM-FullC. The prelude completed, each agent n starts running the standard EM on its dataset $\mathcal{D}_n$, in the private setting of its memory. But with a difference. In the E-step, agents pause their secluded computations to socialize with neighbors in order to agree on the responsibilities γ_{mk}; after agreement is reached, they resume their private computations, as set down by the M-step.

Memory organization. Besides the dataset $\mathcal{D}_n$, each agent n keeps in memory the current parameter $\theta_{\mathcal{H}_n}^{(t)} = \left(\pi_{\mathcal{H}_n}^{(t)}, \mu_{\mathcal{H}_n}^{(t)}, \Sigma_{\mathcal{H}_n}^{(t)}\right)$ of its private GMM (the GMM of hub $\mathcal{H}_n$).

E-step. Each agent n computes the responsibilities for each data point in its dataset $\mathcal{D}_n$:

$$\gamma_{mk\mathcal{H}_n} = \frac{\pi_{k\mathcal{H}_n}^{(t)} N\left(x_{m\mathcal{H}_n} \mid \mu_{k\mathcal{H}_n}^{(t)}, \Sigma_{k\mathcal{H}_n}^{(t)}\right)}{\sum_{l=1}^K \pi_{l\mathcal{H}_n}^{(t)} N\left(x_m \mid \mu_{l\mathcal{H}_n}^{(t)}, \Sigma_{l\mathcal{H}_n}^{(t)}\right)}. \tag{12}$$

Agents compute these responsibilities in parallel.

Let the vector

$$\gamma_{m\mathcal{H}_n} = (\gamma_{m1\mathcal{H}_n}, \ldots, \gamma_{mK\mathcal{H}_n}) \tag{13}$$

be the vector of responsibilities computed by agent n for the mth datapoint. Note that $\gamma_{m\mathcal{H}_n}$ is a probability mass function because its components are nonnegative and sum to one. In fact, this vector represents a belief—how agent n believes the mth data point came about, according to its view of the world, that is, the view of the world that agent n sees through the lenses of its private GMM: the mth data point came about with probability $\gamma_{m1\mathcal{H}_n}$ from gaussian $N\left(x_{m\mathcal{H}_n} \mid \mu_{1\mathcal{H}_n}^{(t)}, \Sigma_{1\mathcal{H}_n}^{(t)}\right)$, with probability $\gamma_{m2\mathcal{H}_n}$ from gaussian $N\left(x_{m\mathcal{H}_n} \mid \mu_{2\mathcal{H}_n}^{(t)}, \Sigma_{2\mathcal{H}_n}^{(t)}\right)$, and so on.

Merging all beliefs. Different agents, operating on different datasets, arrive naturally at different beliefs per data point. We suggest to bring all these disparate beliefs into agreement by having the N agents engaging in distributed *consensus* so that their N individual beliefs $\gamma_{m\mathcal{H}_n}$, $1 \le n \le N$, are replaced by a common, merged belief $\overline{\gamma}_m$. We suggest to merge the beliefs either by an arithmetic mean or a geometric mean, the details of both we refer to the thesis (for lack of space here). The E-step thus needs communications between agents.

M-step. In this step, each agent n simply carries out the standard update of the M-step:

$$\pi_{k\mathcal{H}_n}^{(t+1)} = \frac{\sum_{m=1}^{M} \overline{\gamma}_{mk}}{M}, \quad (14)$$

$$\mu_{k\mathcal{H}_n}^{(t+1)} = \frac{\sum_{m=1}^{M} \overline{\gamma}_{mk}\, x_{m\mathcal{H}_n}}{\sum_{m=1}^{M} \overline{\gamma}_{mk}}, \quad (15)$$

$$\Sigma_{k\mathcal{H}_n}^{(t+1)} = \frac{\sum_{m=1}^{M} \overline{\gamma}_{mk} \zeta_{mk\mathcal{H}_n}^{(t+1)} \zeta_{mk\mathcal{H}_n}^{T,(t+1)}}{\sum_{m=1}^{M} \overline{\gamma}_{mk}}. \quad (16)$$

where $\zeta_{mk\mathcal{H}_n}^{(t+1)} = \left(x_{m\mathcal{H}_n} - \mu_{k\mathcal{H}_n}^{(t+1)}\right)$. The M-step thus needs no communications.

The EM-FullC algorithm is outlined in Algorithm 2.

Algorithm 2 The EM-FullC algorithm

1. Choose initial parameters $\theta_{\mathcal{H}_n}^{(0)}$ and set $t = 0$
2. E-step: each agent n computes the belief vectors $\gamma_{m\mathcal{H}_n}$ (13), then all agents communicate to jointly find the merged beliefs $\overline{\gamma}_m$
3. M-step: each agent n updates to $\theta_{\mathcal{H}_n}^{(t+1)}$ as in (14), (15), and (16)
4. update $t \leftarrow t + 1$ and repeat steps 2 and 3

2.3. Numerical simulations

The general procedure. To compare the algorithms, we use two metrics: the log-likelihood of the GMM fit because it is the function that EM seeks to maximize, although an algorithm more prone to overfitting may achieve a higher log-likelihood while actually making a poorer estimation of the parameters; the error between the GMM parameter fitted by an algorithm and the true GMM parameter (ground truth) used to generate the data.

Because our tests focus on cases where $K = 2$ with equal priors, $\pi_0 = \pi_1 = 0.5$, the value of the GMM parameter $\pi = (\pi_0, \pi_1)$ fitted by each algorithm is reported directly, as it is immediately interpretable. The remaining GMM parameters μ and Σ, being multi-dimensional, are not reported directly; instead, we report their error with respect to the true parameters. In fact, we do not evaluate directly the error in the covariance Σ_k but rather in its inverse $\Upsilon = \Sigma_k^{-1}$, also known as an information matrix. Specifically, we report π, $\varepsilon_\mu = \sum_k \frac{||\mu_{c,k} - \mu_k||}{K||\mu_{c,k}||}$, and $\varepsilon_\Upsilon = \sum_k \frac{||\Upsilon_{c,k} - \Upsilon_k||}{K||\Upsilon_{c,k}||}$.

Furthermore, we tested the algorithms on different kinds of networks—chain networks, geometric networks, and anti-hubs networks, whose detailed description can be found in chapter 3 of the thesis.

Main conclusions. From the several simulations carried out in chapter 3 (too many to dissect here), we draw three main conclusions:

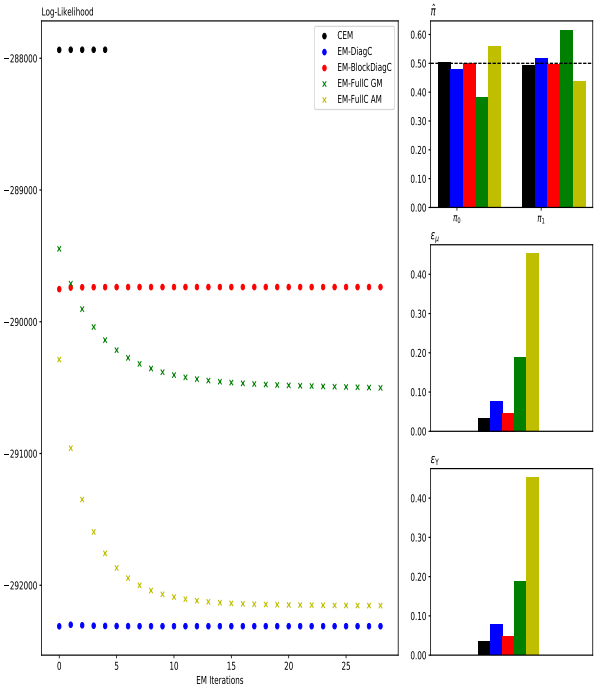

Figure 3: EM-DiagC and EM-BlockDiagC outperform EM-FullC.

- In most situations, EM-DiagC and EM-BlockDiagC outperform EM-FullC. Such an example, concerning a geometric network, is given in Figure 3;
- However, in the experiment we called the *cross correlation* experiment, the reverse happens: EM-FullC outperforms both EM-DiagC and EM-BlockDiagC, as illustrated in Figure 4;
- EM-BlockDiagC always outperforms EM-DiagC, an expected outcome since EM-DiagC can be seen as a special case of EM-BlockDiagC;
- The EM-FullC GM, which merges beliefs by the geometric mean (GM), nearly always outperforms the EM-FullC AM, which merges beliefs by the arithmetic mean (AM).

3. How to classify a measured data point into one of K given Gaussians?

Detection is the categorization of each new data point on a set of hypotheses. Although we assume that each node knows a single entry of x, it is simple to generalize for the case where each node gets chunks of features of varied dimensions for each observation.

We consider a set of K subpopulations, each parameterized by the sufficient statistics μ_k and Σ_k, with a prior probability of π_k of having generated the sampled point. While the responsibility γ_k is

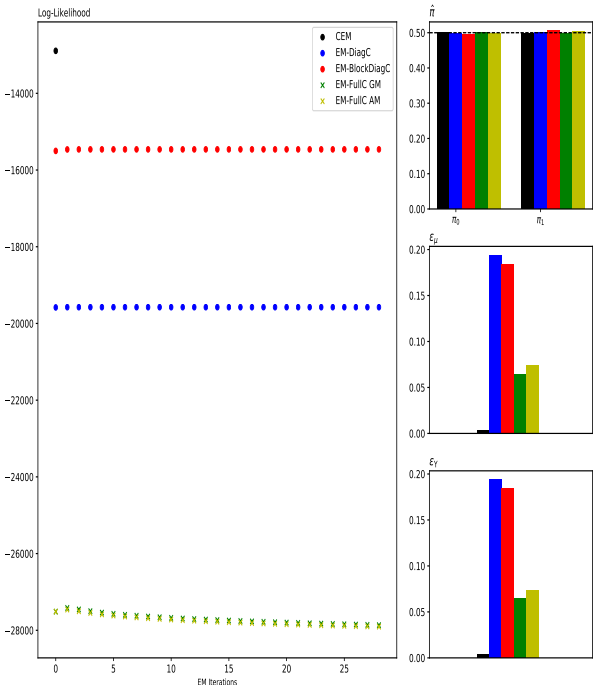

Figure 4: EM-FullC outperforms EM-DiagC and EM-BlockDiagC.

the probability that data point x belongs to cluster k, the prior π_k is the probability that any point will belong to class k, independently of the observation. Considering H_k to be the hypothesis of x belonging to subpopulation k, and denoting $\theta = \{\pi, \mu, \Sigma\}$, each responsibility can be written as

$$\begin{aligned} \gamma_k &= p\left(H_k \text{ valid} \mid x; \theta\right) \\ &= \frac{p\left(x \mid H_k \text{ valid}; \theta\right) p(H_k \text{valid} \mid \theta)}{p\left(x \mid \theta\right)} \\ &= \frac{\pi_k N\left(x \mid \mu_k, \Sigma_k\right)}{\sum_{l=1}^{K} \pi_l N\left(x \mid \mu_l, \Sigma_l\right)}. \end{aligned} \tag{17}$$

Note that (17) leads to an expression for the responsibility matching the one encoded by a GMM in the E-step of EM considered previously on this work. Following the same reasoning as earlier, evaluating (17) boils down to a expression similar to (2). However, in this chapter it is more convenient to write the expressions as a function of the precision matrix $\Upsilon_k = \Sigma_k^{-1}$, also a positive semi-definite (PSD) matrix. As such we write the log-responsibility as

$$\begin{aligned} \log \gamma_k \propto \log \pi_k + \frac{1}{2} \log \left|\Upsilon_k\right| \\ -\frac{1}{2}\left(x-\mu_k\right)^T \Upsilon_k \left(x-\mu_k\right). \end{aligned} \tag{18}$$

Figure 5 illustrates the overall goal of the proposed method resorting to an example network.

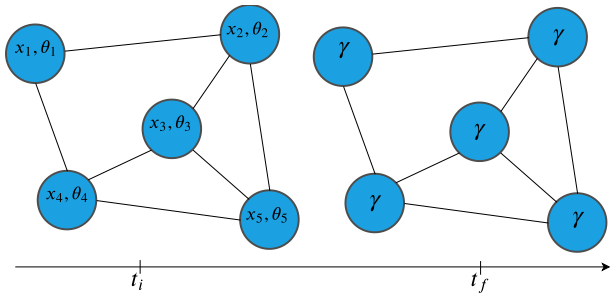

Figure 5: Desired operation of the distributed detection algorithm: decentralized computation of responsibilities, in a setup with feature-wise fragmented data points. Starting with x_n in node n, together with the required parameters, θ_n. At completion we want $\gamma = \{\gamma_1, \ldots, \gamma_K\}$ in every node (or its estimate).

The memory required in each node should not depend on the size of the network, N. We now make the key observation that the quadratic factor in (18) is the only one that does not scale with N in storage and, so, we will partition both Υ_k and $(x - \mu_k)$ throughout the network. A more in-depth analysis can be found in Chapter 4 of the thesis.

In the thesis we present three algorithms, two baseline, standard distribution techniques, but here we will cover the most promissing in terms of complexity.

Hop + Consensus. The key factor allowing scalability in our setting is the precision matrix approximation following the restrictions of the communication graph between nodes. The adjacency matrix, $A \in \mathbb{R}^{N \times N}$, describes the topology of a graph, where each entry A_{ij} takes the value 1 if i and j are adjacent, $i \sim j$, and 0 if they are not, $i \not\sim j$. The main diagonal of A will be set to 1 because each node incorporates information from its own measurements.

Each of the heuristics developed will have a different approximated estimator, $\hat{\Upsilon}$. Their entries correspond to pairs of features whose associated nodes, i.e., the ones measuring the features, are not connected, will be considered conditionally independent, given all the other features [4].

Where zero entries of the adjacency matrix A will be null in the precision matrices estimators too, formalized as

$$\mathbf{c}_1\text{:} \quad \hat{\Upsilon}_{ij} = 0, \quad i \not\sim j. \tag{19}$$

With the introduction of this constraint, instead of using a $N \times N$ fully unknown precision matrix Υ_k, we now assume that only entries corresponding to connected nodes have non-zero values. This is a strong assumption, namely on the conditional independence given all other features. Nevertheless,

despite this being a strong approximation, it is useful computationally.

Before presenting the several candidates to be used as $\hat{\Upsilon}$, discussed in the next section, let us see the impact using such an estimator has on the algorithm. To that end, our current analysis of the decentralization benefits from expanding

$$(x-\mu_k)^T\Upsilon_k(x-\mu_k) = x^T\Upsilon_k x - 2x^T\Upsilon_k\mu_k + \mu_k^T\Upsilon_k\mu_k. \tag{20}$$

Let us analyze the different parts of the expansion above individually. Firstly, the rightmost term, $\mu_k^T\Upsilon_k\mu_k$, is a mere scalar which depends only on the K parameters of the model, and not on the measurement, x. Therefore, it suffices to store K scalar values in each of the nodes of the network. When it comes to $x^T\Upsilon_k\mu_k$, we can rewrite it as

$$x^T \begin{bmatrix} \sum_{n=1}^N \Upsilon_{k1n}\mu_{kn} \\ \vdots \\ \sum_{n=1}^N \Upsilon_{kNn}\mu_{kn} \end{bmatrix},$$

underlining the fact that, even using the exact precision matrix, each row $(\Upsilon_k\mu_k)_n$ is a single scalar parameter. For this reason, it suffices for each node n to store $(\Upsilon_k\mu_k)_n$, for $1 \leq n \leq N$. This way, it is straightforward to apply the inner product consensus $x_1(\Upsilon_k\mu_k)_1 + \cdots + x_1(\Upsilon_k\mu_k)_N$, where each node n naturally has x_n since it was the node measuring it, and stores $(\Upsilon_k\mu_k)_n$. $\psi_n = Nx_n(\Upsilon_k\mu_k)_n$, resulting in $\bar{\psi} = \frac{1}{N}\sum_{n=1}^N \psi_n = \sum_{n=1}^N x_n(\Upsilon_k\mu_k)_n = x^T\Upsilon_k\mu_k$. That is, there is no need to include $\hat{\Upsilon}$ in an approximation to $x^T\Upsilon_k\mu_k$. Finally, we arrive at $x^T\Upsilon_k x$, the trouble-maker, when using the exact precision matrix Υ_k. In order to avoid using the N+1 consensus scheme, we replace the use of Υ_k in $x^T\Upsilon_k x$ by an estimator $\hat{\Upsilon}$ respecting (19), allowing us to simplify the computations required. More concretely, notice how now, for every n such that $1 \leq n \leq N$, $x_n(\hat{\Upsilon}_{n1}x_1 + \cdots + \hat{\Upsilon}_{nN}x_N)$ will have all the entries in row Υ_n null apart from those corresponding to $\mathcal{N}_{[n]}$, that is, $x_n\left(\hat{\Upsilon}_{n1}x_1 + \cdots + \hat{\Upsilon}_{nN}x_N\right) = x_n\left(\sum_{i\in\mathcal{N}_{[n]}} \hat{\Upsilon}_{ni}x_i\right)$. With this approximation, the whole expression can be computed in node n requiring only a single *hop* of comunication simultaneous for all nodes. Additionally, each $\left(\hat{\Upsilon}x\right)_n$ depends only on node n and its D_n neighbors measurements, as long as each node n stores Υ_{kni}, for all $i \in \mathcal{N}_{[n]}$. After that, we can use these terms to obtain $x^T\hat{\Upsilon}_k x$ via a single consensus, $\psi_n = Nx_n\left(\hat{\Upsilon}x\right)_n$, resulting in $\bar{\psi} = \frac{1}{N}\sum_{n=1}^N \psi_n = \sum_{n=1}^N x_n\left(\hat{\Upsilon}x\right)_n = x^T\hat{\Upsilon}x$. Having concluded that only one of the three terms being added requiring an approximation is the leftmost one of (20), note that, the hop and the consensus required by the terms $x^T\Upsilon_k\mu_k$ and $x^T\hat{\Upsilon}x$ can actually be the same, as long as we we resort to the consensus states $\psi_n = N\left[x_n\left(\hat{\Upsilon}x\right)_n + x_n(\Upsilon_k\mu_k)_n\right]$, which lead to

$$\begin{aligned}\bar{\psi} = \frac{1}{N}\sum_{n=1}^N \psi_n &= \sum_{n=1}^N \left[x_n\left(\hat{\Upsilon}x\right)_n + x_n(\Upsilon_k\mu_k)_n\right] \\ &= x^T\hat{\Upsilon}x + x^T\Upsilon_k\mu_k,\end{aligned}$$

thus approximating the responsibility estimator, as desired.

This algorithm requires each node n to store Υ_{kni}, $\forall i \in \mathcal{N}_{[n]}$, hence a memory complexity of $O(d_{max})$, which is not a problem, since degree usually not associated to size of network, meaning this does not compromise scalability. In conclusion, the complete set of parameters required to be present in each node n initially, on top of its measurement x_n, is given by $\{\pi_k, |\Upsilon_k|, \mu_k^T\Upsilon_k\mu_k, (\Upsilon_k\mu_k)_n, \Upsilon_{kni}\}, k = 1, .., K, \forall i \in \mathcal{N}_{[n]}$.

In the following section we present several different heuristics designed for the distributed computation of the responsibilities, resorting to different distributed and scalable estimators $\hat{\Upsilon}_k$, to approximate the centralized detector.

3.1. Naïve estimators

Baseline. With a brute-force approach, this estimator forces all elements corresponding to $i \not\sim j$ of Υ_k to zero, allowing for the Hop+Consensus algorithm to run.

Least F-norm. This estimator both assures $\mathbf{c}_1$ to be respected and guarantees $\hat{\Upsilon}$ to be positive semi-definite (PSD) and symmetric, $\mathbf{c}_2$**:** $\hat{\Upsilon} = \hat{\Upsilon}^T \succeq 0$. All the estimators that follow respect $\mathbf{c}_1$ and $\mathbf{c}_2$. The divergence measuring objective function $F_h(\hat{\Upsilon})$, where h is the heuristic being used:

$$\begin{aligned} \underset{\hat{\Upsilon}_k}{\arg\min} \quad & F_h(\hat{\Upsilon}) \\ \text{subject to} \quad & \hat{\Upsilon} = \hat{\Upsilon}^T \succeq 0, \\ & \hat{\Upsilon}_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{21}$$

The Least F-norm differs from the subsequent estimators in using $F_h(\hat{\Upsilon}) = ||\hat{\Upsilon} - \Upsilon_k||_F^2$.

The problem is convex more specifically, given the inequality constraint in question, with Semidefinite Programming (SDP). Further details are provided in [5, pp. 136 and 168].

The fact that we are dealing with a convex optimization problem is important, as it allows us to use the techniques already developed for this kind of problem. For instance, in our implementation we use the Python library *cvxpy* to solve this and the following problems.

3.2. KullbackLeibler (KL) estimators

Mode Seeking KL. Resorting to the optimization problem in (21), the Mode Seeking KL uses as the objective function the KullbackLeibler divergence [1, Section 10.1.2], D_{KL}, between a normal distribution parameterized by the true mean and the optimization variable, $N\left(\mu_k, \hat{\Upsilon}\right)$, and the true normal distribution, parameterized by the true mean and precision, $N\left(\mu_k, \Upsilon_k\right)$:

$$F\left(\hat{\Upsilon}\right) = D_{KL}\left(N\left(\mu_k, \hat{\Upsilon}\right), N\left(\mu_k, \Upsilon_k\right)\right).$$

The Kullback-Leibler divergence between the two Gaussians will be

$$\begin{aligned} F(\hat{\Upsilon}) &= \frac{1}{2}\left\{\mathrm{tr}\left(\Upsilon_k \hat{\Upsilon}^{-1}\right) - d + \log\frac{\left|\Upsilon_k^{-1}\right|}{\left|\hat{\Upsilon}^{-1}\right|}\right\} \\ &\propto \mathrm{tr}\left(\Upsilon_k \hat{\Upsilon}^{-1}\right) + \log\left|\hat{\Upsilon}\right|, \end{aligned} \tag{22}$$

where the proportionality corresponds to the removal of the terms that are constant with respect to the optimization variable, $\hat{\Upsilon}$, and therefore do not impact the optimization problem.

So, from (22), we have that the problem to be solved in order to obtain the Mode Seeking KL estimator is:

$$\begin{aligned} \underset{\hat{\Upsilon}}{\arg\min} \quad & \mathrm{tr}\left(\Upsilon_k \hat{\Upsilon}^{-1}\right) + \log\left|\hat{\Upsilon}\right| \\ \text{subject to} \quad & \hat{\Upsilon} = \hat{\Upsilon}^T \succeq 0, \\ & \hat{\Upsilon}_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{23}$$

However, it is important to notice that, unlike in the previous heuristic, this cost function is not convex, as it results from the sum of a first convex term and the second, which is concave. In order to perform this optimization we employ the Majorization-Minimization (MM) algorithm, described in the thesis document.

Lastly, by applying the MM algorithm to the initial nonconvex optimization problem (23), we iterate the solution of the convex problem

$$\begin{aligned} \underset{\hat{\Upsilon}}{\arg\min} \quad & \mathrm{tr}\left(\Upsilon_k \hat{\Upsilon}^{-1}\right) + \mathrm{vec}\left(\hat{\Upsilon}_r^{-1}\right)^T \mathrm{vec}\left(\hat{\Upsilon}\right) \\ \text{subject to} \quad & \hat{\Upsilon} = \hat{\Upsilon}^T \succeq 0, \\ & \hat{\Upsilon}_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{24}$$

Mean Seeking KL. Again having as foundation the optimization problem in (21), this estimator resorts to the the KullbackLeibler divergence as the objective function. However, here the arguments to the KL divergence are exchanged, having firstly the estimate distribution and secondly the actual one. The different order of arguments imposes a mean-seeking behavior. Further details regarding the difference between these to applications of the KL divergence can also be seen in [1, Section 10.1.2]. We will be working with

$$F\left(\hat{\Upsilon}\right) = D_{KL}\left(N\left(\mu_k, \Upsilon\right), N\left(\mu_k, \hat{\Upsilon}\right)\right), \tag{25}$$

where $\Upsilon = \{\Upsilon_1, \ldots, \Upsilon_K\}$ and $\hat{\Upsilon} = \left\{\hat{\Upsilon}_1, \ldots, \hat{\Upsilon}_K\right\}$.

Again, we rewrite (25) as

$$\begin{aligned} F\left(\hat{\Upsilon}\right) &= \frac{1}{2}\left\{\mathrm{tr}\left(\hat{\Upsilon}_k \Upsilon_k^{-1}\right) - d + \log\left(\frac{\left|\hat{\Upsilon}_k^{-1}\right|}{\left|\Upsilon_k^{-1}\right|}\right)\right\} \\ &\propto \mathrm{tr}\left(\Upsilon_k^{-1}\hat{\Upsilon}_k\right) - \log\left|\hat{\Upsilon}_k\right|, \end{aligned}$$

ending up with

$$\begin{aligned} \underset{\hat{\Upsilon}_k}{\arg\min} \quad & \mathrm{tr}\left(\Upsilon_k^{-1}\hat{\Upsilon}_k\right) - \log\left|\hat{\Upsilon}_k\right| \\ \text{subject to} \quad & \hat{\Upsilon}_k = \hat{\Upsilon}_k^T \succeq 0, \\ & \left(\hat{\Upsilon}_k\right)_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{26}$$

Note how, unlike (24), it can be shown that the cost function in (26) is convex, allowing for this estimator to be obtained without requiring any further manipulations.

3.3. Bhattacharyya-based estimator

Least B-Distance. In this heuristic the objective cost function to be used in the optimization problem (21) is the Bhattacharyya Distance, D_B, between the distribution described by the true mean and true precision, or equivalently $N(\mu_k, \Sigma_k)$, and the one described by the true mean and the precision estimator, $N\left(\mu_k, \hat{\Upsilon}^{-1}\right)$. For our two Gaussians, the Bhattacharyya distance is

$$\begin{aligned} &D_B\left(N(\mu_k, \Sigma_k), N\left(\mu_k, \hat{\Upsilon}^{-1}\right)\right) \\ &\quad = \frac{1}{2}\log\det\left(\frac{\Sigma_k + \hat{\Upsilon}^{-1}}{2}\right) - \frac{1}{2}\log\sqrt{\det\Sigma_k} \\ &\quad\quad - \frac{1}{2}\log\sqrt{\det\hat{\Upsilon}^{-1}}, \end{aligned}$$

where $\Sigma = \frac{\Sigma_k + \hat{\Upsilon}^{-1}}{2}$. By removing the terms which are constant w.r.t. $\hat{\Upsilon}$, we rewrite our cost function as

$$\frac{1}{2}\log\left|\Sigma_k^{1/2}(I + \Sigma_k^{-1/2}\hat{\Upsilon}^{-1}\Sigma_k^{-1/2})\Sigma_k^{1/2}\right| + \frac{1}{4}\log\left|\hat{\Upsilon}\right|.$$

Where, once more, terms which are constant w.r.t. $\hat{\Upsilon}$ can be removed obtaining as cost function $\log\left|I + \Sigma_k^{-1/2}\hat{\Upsilon}^{-1}\Sigma_k^{-1/2}\right| + \frac{1}{2}\log\left|\hat{\Upsilon}\right|$. However, we know from the WeinsteinAronszajn identity, also known as the Sylvester's determinant theorem [6, p. 271], that $|I + CD| =$

$|I + DC|$, therefore $\log\left|I + \Sigma_k^{-1/2}\hat{\Upsilon}^{-1}\Sigma_k^{-1/2}\right| = -\log\left|\hat{\Upsilon}\right| + \log\left|\hat{\Upsilon} + \Sigma_k^{-1}\right|$, leading us to rewrite the optimization problem as

$$\begin{aligned} \underset{\hat{\Upsilon}}{\arg\min} \quad & -\frac{1}{2}\log\left|\hat{\Upsilon}\right| + \log\left|\hat{\Upsilon} + \Sigma_k^{-1}\right| \\ \text{subject to} \quad & \hat{\Upsilon} = \hat{\Upsilon}^T, \\ & \left(\hat{\Upsilon}\right)_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{27}$$

However, once more, our cost is the sum of a convex term, $f_1(\hat{\Upsilon}) = -\frac{1}{2}\log\left|\hat{\Upsilon}\right|$, and a concave one, $f_2\left(\hat{\Upsilon}\right) = \log\left|\hat{\Upsilon} + \Sigma_k^{-1}\right|$. Thus, we resort once again to the MM algorithm. By linearizing the term $f_2\left(\hat{\Upsilon}\right)$, we arrive at the MM itaration of the convex approxomation

$$\begin{aligned} \underset{\hat{\Upsilon}}{\arg\min} \quad & -\frac{1}{2}\log\left|\hat{\Upsilon}\right| + \mathrm{tr}((\hat{\Upsilon}_r + \Sigma_k^{-1})^{-1}\hat{\Upsilon}) \\ \text{subject to} \quad & \hat{\Upsilon} = \hat{\Upsilon}^T \succ 0, \\ & \left(\hat{\Upsilon}\right)_{ij} = 0, \quad i \not\sim j. \end{aligned} \tag{28}$$

at each step r.

3.4. Estimator based on least expected error

Least MSE. In this heuristic, our estimator comes from the computation of the Least Mean Squared Error (MSE). Therefore, by minimizing it, we seek estimators $\hat{\Upsilon}_k$ such that, for $k = 1, ..., K$: $x^T\hat{\Upsilon}_k x \simeq x^T\Upsilon_k x$, where $x \sim p(x) = \sum_{k=1}^K \pi_k p_k(x)$, and $p_k(x) = N(\mu_k, \Sigma_k)$. We formulate our objective function corresponding to the MSE to be minimized as

$$\begin{aligned} \underset{\hat{\Upsilon}_1, ..., \hat{\Upsilon}_K}{\text{minimize}} \quad & \sum_{k=1}^K \sum_{l=1}^K \pi_l \mathbf{E}_{p_l(x)}\left\{\left(x^T\left(\Upsilon_k - \hat{\Upsilon}_k\right)x\right)^2\right\} \\ \text{subject to} \quad & \hat{\Upsilon}_k = \hat{\Upsilon}_k^T, \\ & \left(\hat{\Upsilon}_k\right)_{ij} = 0, \quad i \not\sim j. \end{aligned}$$

However, we still need to compute the expectation, removing x. To do this, we resort to reference [7, eq. 5.28]. We define $\Delta_k = (\Upsilon_k - \hat{\Upsilon}_k)$, to write the cost function

$$\begin{aligned} F_h(\hat{\Upsilon}) = \sum_{k=1}^K \sum_{l=1}^K \pi_l \Big[& 2\mathrm{tr}\big[\Delta_k\Sigma_l\Delta_k\Sigma_l\big] + \\ & 4\mu_l^T\Delta_k\Sigma_l\Delta_k\mu_l + \\ & \big(\mathrm{tr}\big[\Delta_k\Sigma_l\big] + \mu_l^T\Delta_k\mu_l\big)^2\Big] \end{aligned}$$

used in the same optimization problem as earlier.

3.5. Large Deviations estimator

Large Deviations. This estimator, still preliminary work, is based on *Large Deviations theory*. The design and details can be found in a paper currently in preparation [8]. And some details are provided in the thesis. However, it important to state that this is an ongoing work, and given that it is still in progress, not many results have been obtained yet.

3.6. Numerical simulations

Consider the two main goals one may have when performing detection: computing the responsibilities assigned to each of the K models being considered — soft classification — and selecting the most likely model to have generated the data point — hard classification. Our metrics to evaluate the different heuristics will reflect these two different goals. One of the tests will measure how close is the estimated probability mass function (PMF) to the ground truth; and the other test will focus on the relative detection error.

Description of the simulation setups and test structure. First, we evaluate the responsibility error plot, using the sum of squared errors of the responsibilities estimation error with respect to the true responsibility, that is, the responsibility as obtained from using the true precision, γ_k, resorting to (17). This γ_k, to be used as the ground truth, is the centralized responsibility estimator, where Υ_k is not approximated. For $1 \leq k \leq K$, $\epsilon_\gamma^2 = \sum_{k=1}^K(\gamma_k(x) - \hat{\gamma}_k(x))^2$ is the squared responsibility error.

On the second test, we evaluate the detection error. Here, we are interested in selecting the distribution k taking the largest responsibility for point x, or equivalently, the hypothesis with the largest probability of being correct. We will apply the estimators to data points and consider them altogether, computing

$$\log\epsilon_{rel} = \log_{10}\frac{\#\text{failed detections}}{\#\text{detections performed}},$$

for a large number of Monte Carlo (MC) trials, and we used random precision matrices for both types of tests.

In the *initialization* of the MM algorithm in both Mode Seeking KL and in Least B-Distance, we used the estimator obtained from the Least F-norm heuristic. In all tests, we have $K = 2$, the covariance matrices for both $k = 0$ and $k = 1$ are equal, and $\pi_0 = \pi_1 = 0.5$. On top of these, we used $MC = 10^6$ and therefore we considered parameters such that log-error varied in the interval [0.5,4], for significance purposes. Further details can be found on the thesis.

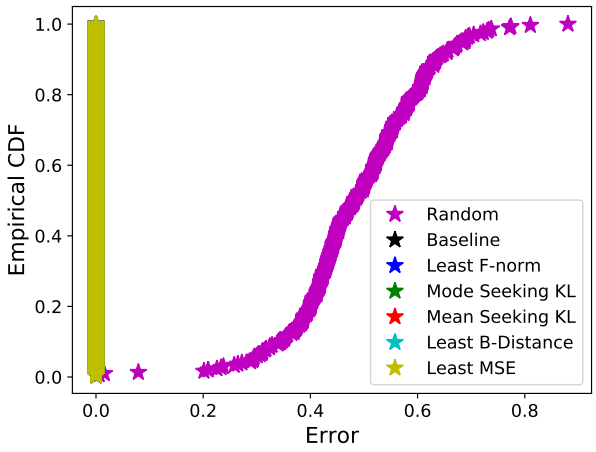

Figure 6: Most often, all heuristics perform well.

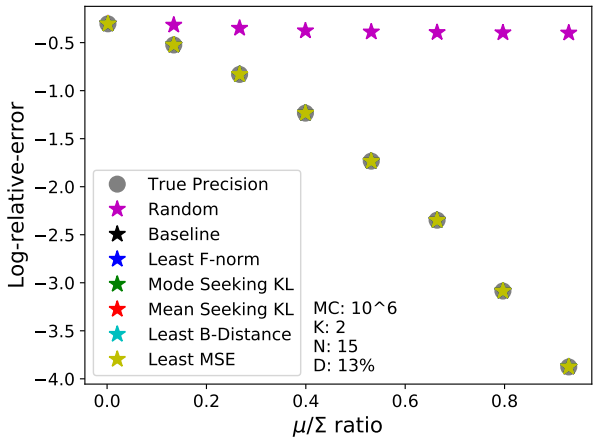

Figure 7: Most often, all heuristics perform detection similarly to centralized approach.

Good performance in standard cases. In standard cases, for most of the randomly initialized parameters, we obtain results overlapping with the centralized ones for all the estimators, other than the random heuristic. See Figures 6 and 7.

4. Main conclusions

We addressed two questions in distributed setups: (1) How to fit a mixture of K Gaussians to a measured dataset? And (2) how to classify a measured data point into one of K given Gaussians?

How to fit a mixture of K Gaussians to a measured dataset? To handle the more difficult setup of data scattered by features, we first derived a distributed exact EM and concluded that it would be unattractive even for modest size networks, due its heavy memory and communications demands. We then developed three distributed approximate EM algorithms scaling well with network size: EM-DiagC, EM-BlockDiagC, and EM-FullC.

The main conclusion is that the three algorithms are pertinent, because, as numerical simulations showed, neither algorithm is always the most accurate—the most accurate one depends on the particular network layout and data scattering. By offering three options, with different trade-offs, we empower the end user with a flexible toolset.

How to classify a measured data point into one of K given Gaussians? After showing that a distributed exact detector does not scale well, we developed the distributed approximate detector Hop+Consensus, which is scalable. To achieve scalability, this approximate detector replaces the information matrices of the K Gaussians by approximate information matrices whose sparsity mirrors that of the underlying communication network. We suggested eight heuristics to obtain the approximate information matrices, heuristics grouped into Naive (2), Kullback-Leibler (3), Bhattacharyya, Expectation, and Large Deviations heuristics.

The main conclusion is that, except for contrived scenarios (in which the heuristics Mean Seeking KL and Least MSE perform better), most heuristics enable the scalable Hop+Consensus distributed detector to perform close to the unscalable exact detector.

References

[1] Christopher M. Bishop. Em for gaussian mixtures. In *Patter Recognition and Machine Learning*, pages 435–439. Springer, 2006.

[2] Junhao Hua and Chunguang Li. Distributed variational bayesian algorithms over sensor networks. *IEEE Transactions on Signal Processing*, 64 (3):783–798, 2015.

[3] S. S. Pereira, R. Lpez-Valcarce, and A. Pags-Zamora. Parameter estimation in wireless sensor networks with faulty transducers: A distributed em approach. *Signal Processing*, 144: 226–237, 2018.

[4] Steffen L. Lauritzen. *Graphical Models*. Clarendon Press, 1996.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] Pozrikidis C. *An Introduction to Grids, Graphs, and Networks*. Oxford University Press, 2014.

[7] Brookes M. *The Matrix Reference Manual*. [online], 2011.

[8] Pedro Valdeira, Cludia Soares, and Joo Xavier. Large deviations estimation of a sparsity constrained precision matrix. *(in preparation)*, 2019.