POLITECNICO
MILANO 1863

Clustering spatial time series via Bayesian nonparametrics

Bayesian Statistics Project

Carnevali Davide
Gurrieri Davide
Moroni Sofia
Rescalli Sara
Varetti Eugenio
Zelioli Chiara

Tutor: Gianella Matteo
Professor: Guglielmi Alessandra

February 14, 2023

Contents

1 Introduction

Italy's northern cities are rated among the worst in Europe for air pollution, citizens can tell and everyone can notice it by looking at any weather forecast app reporting an air quality map. Many studies and research have been conducted, especially associated with the increasing respiratory diseases. To quantify the level of air pollution, the PM_{10} value is often used. It indicates the concentration of particles of 10 microns-diameter that are inhalable into the lungs and can induce severe health effects.

Specifically, the dataset provides spatial time series reporting PM_{10} concentrations sourced daily in 2018, in 112 georeferenced sensing stations across Northern-Italy regions Lombardia and Emilia-Romagna. This kind of station must be located so that it is as representative as possible of the air quality status of the area in which it is placed. In fact, a detection network must have stations located both in background locations, capable of detecting pollution that is widespread throughout the territory, in peak locations, and areas where the highest concentration levels are reached, to which the population is exposed for a significant period. According to the criteria of the European Environment Agency, air quality measuring stations are classified according to station and area types and characteristics: traffic, industrial, urban, suburban, and rural.

This project aims at finding spatial clusters of locations that exhibit similar behavior in the PM_{10} index measured, which might offer an important insight for a better understanding of the factors that contribute to air pollution. To do so, a Bayesian approach has been followed, by means of nonparametric mixture methods, which are essential where the number of clusters is unknown. Therefore, another goal worth of attention is the broadening of the C++ library *Bayesmix*.

2 Preliminary Data Analysis

At first, exploratory data analysis has been conducted to clean, visualize, and especially choose the likelihood model for the data. The project is focused on 2018 daily values of PM_{10} concentrations, for 64 locations in Lombardia and 48 in Emilia-Romagna (Figure 1 and Figure 2). All the work has been done by considering both regions together. Moreover, the 2017 Emilia-Romagna data cleaning has been performed. This data has been used for prior hyperparameters estimation.

Several observations were missing and have been replaced with NA values. To remove data noise, each time series has been weekly averaged, obtaining 52 observations for each location. Locations with more than 10% of NA have been removed, while the ones with few NA have been filled using interpolation and prediction strategies. Moreover, aiming at a 0-mean likelihood, the sample mean of each time series has been subtracted from it.

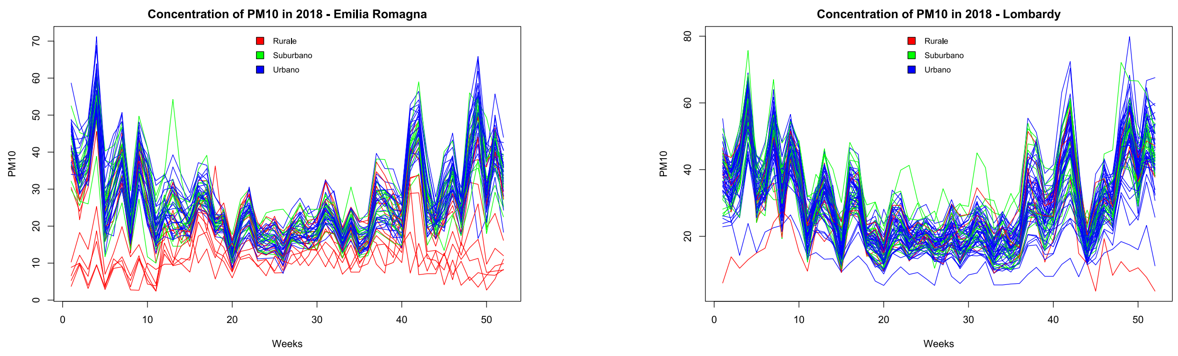


Figure 1: PM_{10} concentrations colored by area (rural, suburban, urban).

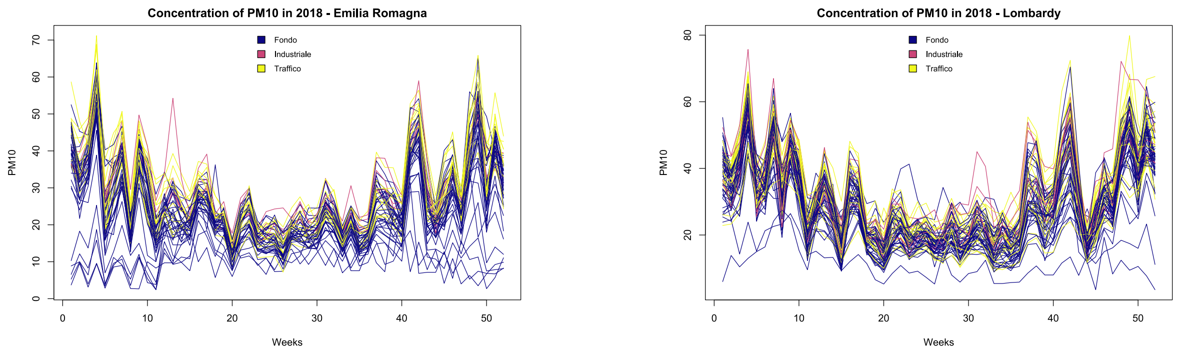


Figure 2: PM_{10} concentrations colored by type (background, factories, traffic).

2.1 ARMA model selection

This section delineates the main steps taken to model the likelihood of 2018 data. A time series $\{Y_t : t \in T\}$ can be written as, according to the classical decomposition form, $Y_t = m_t + u_t + \varepsilon_t$, where m_t stands for the trend component, u_t the seasonal one, and ε_t for the residuals. Residuals represent the non-deterministic component and are usually modeled as white noise. Modeling time series requires the selection of proper parameters (p,q) for an Auto-Regressive Moving-Average model (ARMA(p,q)), which has the following general form:

$$y_t = \rho_1 y_{t-1} + \cdots + \rho_p y_{t-p} + \tau_o z_t + \tau_1 z_{t-1} + \cdots + \tau_q z_{t-q},$$

$$\{z_t\} \sim WN(0, \sigma^2), \quad \boldsymbol{\rho} = (\rho_1, \ \ldots, \ \rho_p), \quad \boldsymbol{\tau} = (\tau_0, \ \tau_1, \ \ldots, \ \tau_q)$$

In the case of interest, data are modeled by a cluster-dependent ARMA process:

$$y_{it} \mid (\rho_h)_{h=1:H}, \quad c_i = h \sim \text{ARMA}(\rho_h)$$

$i = 1, \ldots, N$ locations, $t = 1, \ldots, T$ weeks, $h = 1, \ldots, H$ clusters.

To choose the right orders p and q, Auto-Correlation Function (ACF) and Partial Auto-correlation Function (PACF) are accounted for. In particular, a sharp cut-off after q lags in the ACF plot suggests a Moving Average model of order q, while a sharp cut-off after p lags in the PACF plot suggests an Auto-Regressive model of order p. If none of these behaviors is observed, a complete ARMA process must be considered. Having numerous time series, random locations have been picked to observe their PACF and ACF. The PACF plots in Figure 5 lead to an AR(1) or AR(3) model.

A comparison of the AIC of AR(1) models and the best model at each location, chosen with the help of the R built-in function `auto.arima`, has been made to select the best AR(p) process. Looking at Figure 3 and keeping in mind the principle "the simpler the better", the best model is the AR(1):

$$y_{i,t} = \rho_i y_{i,t-1} + \varepsilon_i, \quad \varepsilon_i \sim WN\left(0, \sigma_i^2\right)$$

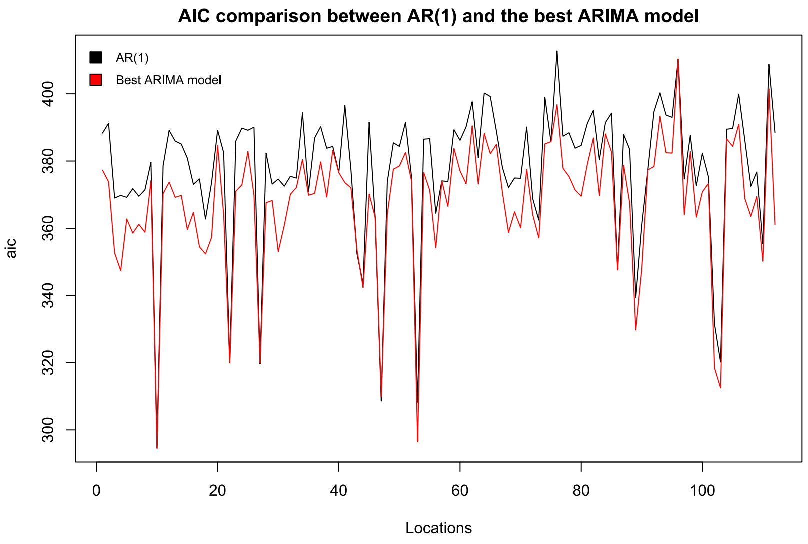


Figure 3

Furthermore, a normality test on the residuals reveals their gaussianity, and this is the reason why the likelihood assigned to each time series $\{y_{i,t}\}$ is uni-variate normal (see subsection 3.2). Then, the final form of the AR(1) process for the cluster h = 1, ..., H is:

$$y_{h,t} = \rho_h y_{h,t-1} + \varepsilon_h, \quad \varepsilon_h \sim \mathcal{N}_1\left(0, \sigma_h^2\right)$$

Figure 4 displays the estimation of the coefficient for each AR(1) model for the 2018 Emilia-Romagna and Lombardia time series, at each numbered location, where the different colors indicate whether the location belongs to a rural, suburban, or urban area.

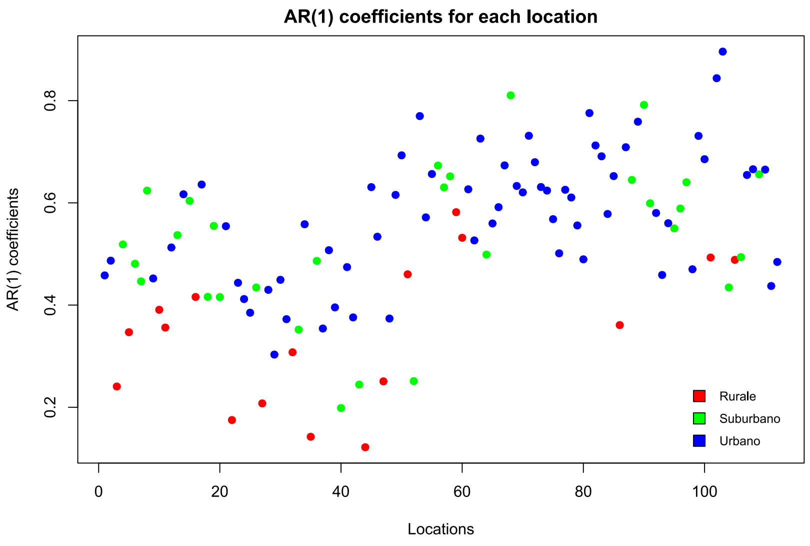


Figure 4

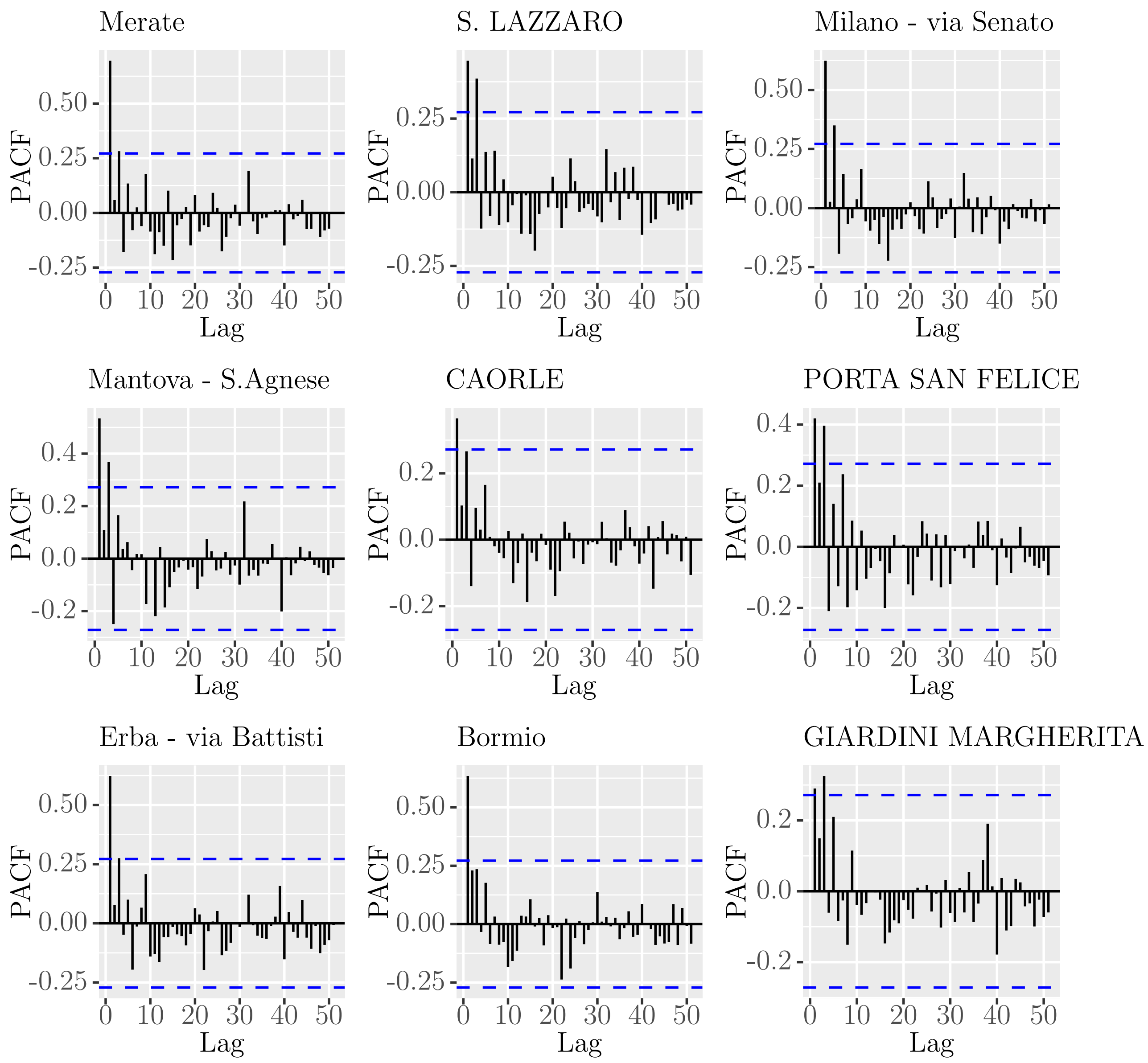

Merate
S. LAZZARO
Milano - via Senato
Mantova - S.Agnese
CAORLE
PORTA SAN FELICE
Erba - via Battisti
Bormio
GIARDINI MARGHERITA
PACF
Lag

Figure 5

3 Model

After model identification for the given time series, a more complex and powerful model is needed to perform the clustering of locations. Modeling a distribution as a mixture of simpler distributions is useful both as a non-parametric density estimation method and as a way of identifying latent classes that can explain the dependencies observed between variables. Mixtures with a countably infinite number of components can reasonably be handled in a Bayesian framework by employing a Dirichlet process $DP(\alpha, G_0)$ as prior distribution for mixing proportions. The use of countably infinite mixtures bypasses the need to determine the "correct" number of components in a finite mixture model.

An interesting point to note is that it is possible to give an alternative and equivalent formulation of a Mixture Model based on the Dirichlet Process. This formulation is given by the Partition Probability Function (*ppf*), which determines the distribution of a random partition:

$$L(S_1, \dots, S_H) \propto \prod_{h=1}^{H} \alpha \, \Gamma(|S_h|)$$

where:

- $S_1 \dots S_H$: partitioning of locations $s_1, \dots, s_N$ into H subsets such that $i \in S_h$ implies that location i belongs to cluster h.
- α: concentration parameter of the Dirichlet Process

In particular, this *ppf* has the shape of a Product Partition Model, defined as:

$$\pi(S_1, \dots, S_H) \propto \prod_{h=1}^{H} C(S_h)$$

where $C(S_h)$ is the Cohesion Function that measures how likely elements of S_h are clustered a priori. Thus, given $\boldsymbol{s_h^*} = \{s_i : i \in S_h\}$, it is possible to perform Bayesian clustering by means of a spatial Product partition model and the model turns out to be:

$$y_{it} \mid (\theta_h)_{h=1:H}, c_i = h \overset{\text{ind}}{\sim} \text{ARMA}(\boldsymbol{\theta}_h)$$

$$\pi(S_1, \dots, S_H) \propto \prod_{h=1}^{H} C(S_h, \boldsymbol{s}_h^*)$$

$$\boldsymbol{\theta}_h \overset{\text{iid}}{\sim} G_0 \quad h = 1, \dots, H$$

3.1 Spatial Product Partition Model and cohesion function

When modeling areal data, it is usual practice to account for spatial structure by using a neighborhood structure. Indeed, making the PPM location dependent is necessary in a spatial setting because if not, then locations that are very far apart could be assigned to the same cluster with high probability. In order to clearly represent the division of locations into geographically dependent clusters, product partition models are expanded to a spatial environment.

Consider n distinct locations denoted by $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$. Let $\rho_n = \{S_1, \ldots, S_{k_n}\}$ denote a partitioning (or clustering) of the n locations into k_n subsets such that $i \in S_h$ implies that location i belongs to cluster h. Alternatively, denote cluster membership using the cluster allocation variables $c_1, \ldots, c_n$ where $c_i = h$ implies $i \in S_h$. Then the PPM prior for ρ is:

$$\Pr(\rho) \propto \prod_{h=1}^{k_n} C\left(S_h\right), \tag{1}$$

where $C\left(S_h\right) \geq 0$ for $S_h \subset \{1, \ldots, n\}$ is a cohesion function that measures how likely elements of S_h are clustered a priori. A popular cohesion function that connects (1) to the marginal prior distribution on partitions induced by a Dirichlet process (DP) is $C(S) = M \times \Gamma(|S|)$. This cohesion produces a PPM that encourages partitions with a small number of large clusters and also a few smaller clusters and will be helpful in avoiding the creation of many singleton clusters when extending PPMs to a spatial setting. Extending the PPM to incorporate spatial information requires making the cohesion of (1) a function of location. Keeping this in mind, consider:

$$\Pr(\rho) \propto \prod_{h=1}^{k_n} C\left(S_h, s_h^{\star}\right).$$

According to the project's goal, the cohesion function chosen is such that it provides a hard cluster boundary for some pre-specified $a > 0$:

$$C\left(S_h, s_h^{\star}\right) = M \times \Gamma\left(|S_h|\right) \times \prod_{i,j \in S_h} \mathbb{I}\left[\|s_i - s_j\| \leq a\right]$$

$M \times \Gamma\left(|S_h|\right)$ is included to favor a small number of large clusters with the number of clusters being regulated by M.

3.2 Likelihood

The model for $\boldsymbol{y_i}$ belonging to cluster $h = 1, \ldots, H$ is defined as:

$$\begin{cases} y_{i,t} \mid y_{i,t-1},\ c_i = h,\ \rho_h,\ \sigma_h^2 \overset{ind}{\sim} \mathcal{N}_1(\rho_h y_{i,t-1},\ \sigma_h^2) & t = 2, \ldots, T \\ y_{i,1} \mid c_i = h,\ \sigma_h^2 \sim \mathcal{N}_1(0,\ \sigma_h^2) & t = 1 \end{cases}$$

Then compute the joint likelihood of the observation $\boldsymbol{y_i}$ in cluster h:

$$\pi\left(\boldsymbol{y_i} \mid c_i = h, \rho_h, \sigma_h^2\right) \propto \exp\left\{-\frac{1}{2\sigma_h^2}\left[y_{i,1}^2 + \sum_{t=2}^{T}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2\right]\right\}$$

Equivalently, assign to $\boldsymbol{y_i}$ the multinormal likelihood with T components:

$$\boldsymbol{y_i} \mid c_i = h, \rho_h, \sigma_h^2 \sim \mathcal{N}_T\left(\mathbf{0}, \Sigma_h\right)$$

Where the precision matrix is:

$$\Sigma_h^{-1} = \frac{1}{\sigma_h^2}\begin{bmatrix} 1+\rho_h^2 & -\rho_h & 0 & 0 & \cdots & 0 \\ -\rho_h & 1+\rho_h^2 & -\rho_h & 0 & & \vdots \\ 0 & -\rho_h & 1+\rho_h^2 & -\rho_h & & \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & -\rho_h & 1+\rho_h^2 & -\rho_h \\ 0 & \cdots & & 0 & -\rho_h & 1 \end{bmatrix}$$

For calculation see Appendix [8.2]

3.3 Prior

The prior distribution for the unique values $\theta_h = (\rho_h, \sigma_h)$ has been set as a Normal-Inverse-Gamma:

$$\begin{cases} \rho_h \mid \sigma_h^2 \sim \mathcal{N}_1\left(\rho_0, \frac{\sigma_h^2}{\lambda}\right) \\ \sigma_h^2 \sim \mathcal{IG}\left(\alpha, \beta\right) \end{cases} \quad \Rightarrow \quad \left(\rho_h, \sigma_h^2\right) \sim \mathcal{N}\left(\rho_0, \frac{\sigma_h^2}{\lambda}\right) \times \mathcal{IG}\left(\alpha, \beta\right)$$

which has density:

$$\pi(\rho_h, \sigma_h^2) = \pi(\rho_h \mid \sigma_h^2) \times \pi(\sigma_h^2) = \sqrt{\frac{\lambda}{2\pi\sigma_h^2}} \exp\left\{-\frac{\lambda}{2\sigma_h^2}(\rho_h - \rho_0)^2\right\} \frac{\beta^\alpha}{\Gamma(\alpha)}(\sigma_h^2)^{(-\alpha-1)} \exp\left\{\frac{-\beta}{\sigma_h^2}\right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2)$$

3.3.1 Prior hyperparameters estimation

Prior hyperparameters have been fixed according to the equivalent sample principle, considering 2017 Emilia Romagna data as an *old* dataset. To ease the reading, let

- $\hat{\rho_h}$ be the sample mean of AR(1) coefficients
- σ_{avg}^2 the average of the differentiated time series' variances
- σ_{var}^2 the variance of the differentiated time series' variances
- $\hat{\sigma}_{\rho_h}^2$ the sample variance of AR(1) coefficients.

Then, set the hyperparameters as:

- $\rho_0 = \hat{\rho_h}$
- $\lambda = \sigma_{avg}^2 / \hat{\sigma}_{\rho_h}^2$
- $\alpha = (\sigma_{avg}^2)^2 / \sigma_{var}^2$
- $\beta = (\alpha - 1) * \sigma_{avg}^2$

3.4 Posterior

The posterior distribution given all the time series in cluster h is the following:

$$\rho_h, \sigma_h^2 \mid \underline{\boldsymbol{y_h}} \sim \mathcal{N}\left(\rho_{0,\text{ post}}\ , \frac{\sigma_h^2}{\lambda_{\text{post}}}\right) \times \mathcal{IG}\left(\alpha_{\text{post}}\ , \beta_{\text{post}}\ \right)$$

where:

$$\rho_{0,\text{ post}} = \frac{B}{A}; \quad \lambda_{\text{post}} = A; \quad \alpha_{\text{post}} = \alpha + \frac{n_h T}{2}; \quad \beta_{\text{post}} = \beta - \frac{1}{2}\left(\frac{B^2}{A} - C\right)$$

And the quantities A, B, and C are defined as follows:

$$A = \sum_{i=1}^{n_h}\sum_{t=1}^{T-1} y_{i,t}^2 + \lambda$$

$$B = \sum_{i=1}^{n_h}\sum_{t=1}^{T-1} y_{i,t} y_{i,t+1} + \lambda\rho_0$$

$$C = \sum_{i=1}^{n_h}\sum_{t=1}^{T} y_{i,t}^2 + \lambda\rho_0^2$$

For calculation see Appendix [8.4].

3.5 Marginal density

The **marginal density** for a single time series is:

$$\pi(\boldsymbol{y_i}) = \frac{\Gamma(\alpha + \frac{T}{2})}{\Gamma(\alpha)\pi^{\frac{T}{2}}(2\alpha)^{\frac{T}{2}}\left(\frac{\beta^T A}{\alpha^T \lambda}\right)^{\frac{1}{2}}}\left\{1 + \frac{1}{2\alpha}\left[\frac{\alpha}{\beta}\left(C - \frac{B^2}{A}\right)\right]\right\}^{-\frac{T}{2}-\alpha}$$

where:

$$C - \frac{B^2}{A} = \sum_{t=1}^{T} y_{i,t}^2 + \lambda\ \rho_0^2 - \frac{\left(\sum_{t=1}^{T-1} y_{i,t}\ y_{i,t+1} + \lambda\ \rho_0\right)^2}{\sum_{t=1}^{T-1} y_{i,t}^2 + \lambda}$$

For calculation see Appendix [8.5].

3.6 Choice of M and a

Setting the parameters M and a of the cohesion function is a crucial step in the spatial clustering process.

The value of M determines the strength of the spatial dependence among the observations and has a direct impact on the clustering results. In general, a higher value of M encourages stronger spatial dependence among the observations, which can lead to a greater number of clusters and potential overfitting of the data. Conversely, a lower value of M reduces the spatial dependence and can result in fewer clusters and potential underfitting of the data. As a first attempt, one can set M by exploiting prior knowledge of the data. For example, it is fairly well known that the expected number of clusters a priori under the DP is, approximately:

$$M \times log\left(\frac{M + \text{number of obs}}{M}\right)$$

So, setting the expected number of clusters at 3, it turns out to be M= 0.567.

As regards a, it is the parameter that controls the spatial dependence between neighboring locations in the clustering and that defines the hard cluster boundary of the cohesion function. In particular, when the distance between two locations is greater than a, then these locations cannot be clustered together. So, it is important to adopt a in a way that is consistent with the geographical area in consideration. As a first attempt, all pairwise distances are computed and then a is chosen to be:

$$a = 250km$$

4 Sampling Strategy

The sampling strategy follows the Neal 2 algorithm, a method relying on Gibbs sampling for models based on conjugate prior distributions, aiming at sampling from the posterior distribution of a Dirichlet process mixture model. The algorithm is described below.

At each iteration:

1. **Sample Cluster Allocation Variables**
   For each observation $i = 1, \ldots, n$:
   - Suppose $c_i = \bar{h}$. Delete observation i from cluster $\bar{h}$. If the present value of c_i is not associated with other observations, remove $\left(\rho_{\bar{h}}, \sigma_{\bar{h}}^2\right)$ from the state.
   - Draw a new value c_i (see subsection 4.1).
   - If the new c_i is not associated with any other observations, draw a value for $\left(\rho_h, \sigma_h^2\right)$ from the posterior and add it to the state.

2. **Sample Cluster Parameters**
   For $j = c_1, \ldots, c_H$:
   - Draw new value for $\left(\rho_j, \sigma_j^2\right)$ from the posterior distribution based on all the data points currently associated with latent class c.

4.1 Draw a new value c_i

To draw a new value c_i:

- Compute the probability of belonging to cluster h for $\forall h = 1, \ldots, H+1$:

$$P\left(c_i = h \mid \text{ rest }\right) \propto w_h L\left(y_i, \theta_h\right)$$
$$P\left(c_i = H+1 \mid \text{ rest }\right) \propto w_{H+1} \pi\left(y_i\right)$$

- Sample a new value for the label h_{new} from a categorical distribution (using the probabilities already calculated) and let $c_i = h_{\text{new}}$.

To compute the weights w_h $\forall h = 1, \ldots, H+1$:

$$w_h = \frac{ppf\left(S_1^{-i}, \ldots, S_h^{-i} \cup \{i\}, S_H^{-i}\right)}{ppf\left(S_1^{-i}, \ldots, S_H^{-i}\right)} \qquad w_{H+1} = \frac{ppf\left(S_1^{-i}, \ldots, S_H^{-i}, \{i\}\right)}{ppf\left(S_1^{-i}, \ldots, S_H^{-i}\right)}$$

Through calculation in Appendix [8.6], the weights turn out to be:

$$w_h = \begin{cases} 0, \text{ if } \exists j^* \in S_h^{-i} \text{ s.t. } \|s_i - s_{j^*}\| = d_{ij^*} > a \\ n_h^{-i}, \text{ otherwise} \end{cases}$$

$$w_{H+1} = M$$

5 Implementation

For a better understanding of the sampling strategy and preliminary results visualization, a first implementation has been coded in Python (it can be found in the dedicated GitHub folder 'python_implementation/'). Subsequently, the main code has been developed extending Bayesmix Library [1], an already existing C++ library for running MCMC simulations in Bayesian mixture models.

5.1 Bayesmix main steps

In order to implement the model reported in Chapter [3], an extension of the existing Bayesmix library has been made. The main additions are reported in the following chapter with their corresponding comments. For further details, please refer to the GitHub repository.

To fulfill the objective, a new `Likelihood` object, the **AR1Likelihood**, has been developed. This class includes all relevant information concerning the likelihood of the model. In addition, the **NIGPriorModel**, which is based on the Normal-Inverse-Gamma distribution, has been exploited to create a new `Hierarchy`, the **AR1NIGHierarchy**. This hierarchy merges the two previously mentioned objects and represents the cluster set present at any given iteration of the MCMC process. To efficiently manage the cluster update process outlined in subsection 4.1, a new `Mixing` object, referred to as **DirichletC2Mixing**, has been implemented. This object holds both the prior distribution on the weights w and the resulting induced EPPF. This addition will streamline the cluster update process, allowing for more accurate and efficient results. Some of the newly written code is reported below.

AR (1) is a particular likelihood that works as a multivariate normal for each statistical unit (see subsection 3.2) and that depends on two univariate parameters. It has therefore been decided to use a state `UniLS` that stores the pair (ρ_h, σ_h^2), with the help of two data structures `Eigen`, *mean* and *prec_chol* that transforms the univariate state into multivariate. This choice has been taken for computational reasons to calculate once per MCMC cycle large-dimension matrices without the need to modify the basic methods of the inherited classes. The *set_state* method is useful for this purpose.

```

class AR1Likelihood
    : public BaseLikelihood<AR1Likelihood, State::UniLS> {
 public:
  ...
  void set_state(const State::UniLS &state_, bool update_card = true);
  ...
 protected:
  double compute_lpdf(const Eigen::RowVectorXd &datum) const override;
  ...
  unsigned int dim;
  ...
```

```cpp
  Eigen::VectorXd mean;
  Eigen::MatrixXd prec_chol;
  double prec_logdet;

};
```

The methods are implemented in the .cpp file:

```cpp
void AR1Likelihood::set_state(const State::UniLS &state_, bool update_card) {
  state = state_;
  if (update_card) {
    set_card(state.card);
  }

  mean = Eigen::VectorXd::Zero(dim);


  Eigen::VectorXd diag = (1.0+state.mean*state.mean)/state.var * Eigen::VectorXd::Ones(dim);
  diag[dim-1] = 1.0/state.var;
  Eigen::VectorXd codiag = -state.mean/state.var * Eigen::VectorXd::Ones(dim-1);
  Eigen::MatrixXd Omega = Eigen::MatrixXd::Zero(dim,dim);
  Omega.diagonal(0) = diag;
  Omega.diagonal(1) = codiag;
  Omega.diagonal(-1) = codiag;

  prec_chol = Eigen::LLT<Eigen::MatrixXd>(Omega).matrixL();
  Eigen::VectorXd diagL = prec_chol.diagonal();
  prec_logdet = 2 * log(diagL.array()).sum();
}

double AR1Likelihood::compute_lpdf(
    const Eigen::RowVectorXd &datum) const {
  return bayesmix::multi_normal_prec_lpdf(datum, mean, prec_chol,
                                          prec_logdet);
}
```

Using the recently described `AR1Likelihood` and the already existing `NIGPriorModel`, the new hierarchy `AR1NIGHierarchy` has been coded. Worth mentioning is the *marg_lpdf* method, which evaluates the marginal density subsection 3.5:

```cpp
class AR1NIGHierarchy
    : public BaseHierarchy<AR1NIGHierarchy, AR1Likelihood, NIGPriorModel> {
 public:
  ...
 % DEFINITION
  double marg_lpdf(ProtoHypersPtr hier_params,
                   const Eigen::RowVectorXd &datum) const override {
    auto params = hier_params->nnig_state();
    Eigen::MatrixXd data_sum_squares = datum.transpose() * datum;
    unsigned dim = datum.size();
    double a_p = params.var_scaling() + (data_sum_squares.diagonal(0).sum↩
       () - data_sum_squares.diagonal(0)[dim-1]) ;
    double b_p = params.var_scaling() * params.mean() + data_sum_squares.↩
       diagonal(1).sum();
    double c_p = params.var_scaling() * (params.mean() * params.mean()) +↩
        data_sum_squares.trace();
    double const_num = stan::math::tgamma(params.shape() + 0.5*dim);
    double det_sigma = pow(params.scale()/params.shape(), dim) * a_p/↩
       params.var_scaling();
    double const_den = stan::math::tgamma(params.shape()) *
              pow(stan::math::pi()*2*params.shape(), 0.5*dim) * pow(↩
                  det_sigma,0.5);

    double factor3 = 1 + 1.0/(2.0*params.scale()) * (c_p - b_p*b_p/a_p);
    double c = const_num/const_den;

    double out = c * pow(factor3,-0.5*dim - params.shape());

    return std::log(out) ;
  };
};
```

The update of the cluster parameters (ρ_j, σ_j^2) is cluster-specific. The sampling from the full conditional is done in the **AR1NIGUpdater** class:

```
class AR1NIGUpdater
    : public SemiConjugateUpdater<AR1Likelihood, NIGPriorModel> {
 public:
 ...
  ProtoHypersPtr compute_posterior_hypers(AbstractLikelihood &like,
                                          AbstractPriorModel &prior) ↩
                                              override;
 ...
};
```

The most significant method is compute_posterior_hypers, that reflects subsection 3.4:

```
AbstractUpdater::ProtoHypersPtr AR1NIGUpdater::compute_posterior_hypers(
    AbstractLikelihood& like, AbstractPriorModel& prior) {
  // Likelihood and Prior downcast
  auto& likecast = downcast_likelihood(like);
  auto& priorcast = downcast_prior(prior);

  // Getting required quantities from likelihood and prior
  int card = likecast.get_card();
  int dim = likecast.get_dim();

  Eigen::MatrixXd data_sum_squares = likecast.get_data_sum_squares();
  auto hypers = priorcast.get_hypers();

  // No update possible
  if (card == 0) {
    return priorcast.get_hypers_proto();
  }

  // Compute posterior hyperparameters
  double mean, var_scaling, shape, scale;

  double a_p = hypers.var_scaling + (data_sum_squares.diagonal(0).sum() -
      data_sum_squares.diagonal(0)[dim-1]) ;
  double b_p = hypers.var_scaling * hypers.mean + data_sum_squares.
     diagonal(1).sum();
  double c_p = hypers.var_scaling * (hypers.mean * hypers.mean) +
     data_sum_squares.trace();

  mean = b_p/a_p;
  var_scaling = a_p;
  shape = hypers.shape + 0.5 * card * dim ;
  scale = hypers.scale - 0.5 * (b_p*b_p/a_p - c_p);

  // Proto conversion
  ProtoHypers out;
  out.mutable_nnig_state()->set_mean(mean);
  out.mutable_nnig_state()->set_var_scaling(var_scaling);
  out.mutable_nnig_state()->set_shape(shape);
  out.mutable_nnig_state()->set_scale(scale);
  return std::make_shared<ProtoHypers>(out);
}
```

To efficiently manage the cluster update process outlined in subsection 4.1, the `DirichletC2Mixing` has been implemented, whose methods *mass_existing_cluster* and *mass_new_cluster* allow to calculate the weight assigned to each cluster for the i-th unit at a generic MCMC iteration.

```
class DirichletC2Mixing
    : public BaseMixing<DirichletC2Mixing, DirichletC2::State, bayesmix::
      DPC2Prior> {
 public:
 ...
 protected:
  ...
  double mass_existing_cluster(
      const unsigned int n, const unsigned int n_clust, const bool log,
      const bool propto,
      const std::shared_ptr<AbstractHierarchy> hier,
      const Eigen::RowVectorXd &covariate) const override;

  ...
  double mass_new_cluster(const unsigned int n, const unsigned int
     n_clust,
                          const bool log, const bool propto,
                              const Eigen::RowVectorXd &covariate) const ↩
                                override;
  ...
};
```

```
double DirichletC2Mixing::mass_existing_cluster(
    const unsigned int n, const unsigned int n_clust, const bool log,
    const bool propto, const std::shared_ptr<AbstractHierarchy> hier,
    const Eigen::RowVectorXd &covariate) const {
  double out;
  // flag on the closeness
  bool is_near = 1;
  std::set<int> index_clusters = hier->get_data_idx();
  for(const auto i : index_clusters)
  {
    if(haversine_formula(covariates_ptr->row(i),covariate) > state.a)
    {
      is_near = 0;
      break;
    }
  }

  if (log) {
    out = hier->get_log_card();
    if (!propto) out -= std::log(n + state.totalmass);
    return is_near ? out : std::log(0);
  } else {
    out = 1.0 * hier->get_card();
    if (!propto) out /= (n + state.totalmass);
    return is_near ? out : 0;
  }
  return std::numeric_limits<double>::quiet_NaN();
}

double DirichletC2Mixing::mass_new_cluster(const unsigned int n,
                                          const unsigned int n_clust,
                                          const bool log,
                                          const bool propto,
                                          const Eigen::RowVectorXd &↩
                                            covariate) const {
  double out;
  if (log) {
    out = state.logtotmass;
    if (!propto) out -= std::log(n + state.totalmass);
  } else {
    out = state.totalmass;
    if (!propto) out /= (n + state.totalmass);
  }
  return out;
}
```

For further details about the code, see Appendix [8.7].

5.2 Performance and comparison

Preliminary analyses of the algorithm were first carried out in Pyhton, using only time series from Emilia-Romagna. The aim was to check the consistency of the choice of model parameters and to do some initial debugging. The average execution time was about 5 minutes for 300 iterations. Once the Bayesmix extension was completed, it was able to run a grid search (a and M) in the same time interval, for a total of 80 thousand MCMC iterations. This comparison between the two refers to the implementation with coordinates expressed in latitude and longitude. It clearly justifies the choice of this library and confirms the expected speed, even (in this specific case) when dealing with time series and taking into account covariates.

The final implementation takes as input a expressed in kilometers, for better interpretability. Thus, the coordinates have to be converted to express distances in the same unit of measurement during execution. However, this did not affect the immediacy of the results. Using all the data from both regions, fixed a and M, it is able to run a full MCMC of 2000 iterations in 10 seconds.

6 Results and interpretation

The following results have been obtained by running MCMC with $n_{iter} = 2000$ and $n_{burnin} = 1000$.

6.1 Optimal partition selection

The posterior distribution of the partition given the data has been acquired using the MCMC sampling strategy described above. The algorithm's output consists of a matrix that has as rows the iterations of the MCMC and as columns the locations. In particular, for each iteration, it provides the cluster membership of each location. In order to select the optimal partition from the set of posterior samples the Variance of Information criterion has been used. VI is a measure of the distance between two partitions, and it is based on the information-theoretic concept of mutual information.

Setting the parameters a and M as described above, the optimal partition from the MCMC sampling according to the VI criterion is shown in Figure 6. The model groups the stations into five distinct clusters, while it also takes into account their geographic structure. It effectively categorizes locations based on their position and pollution.

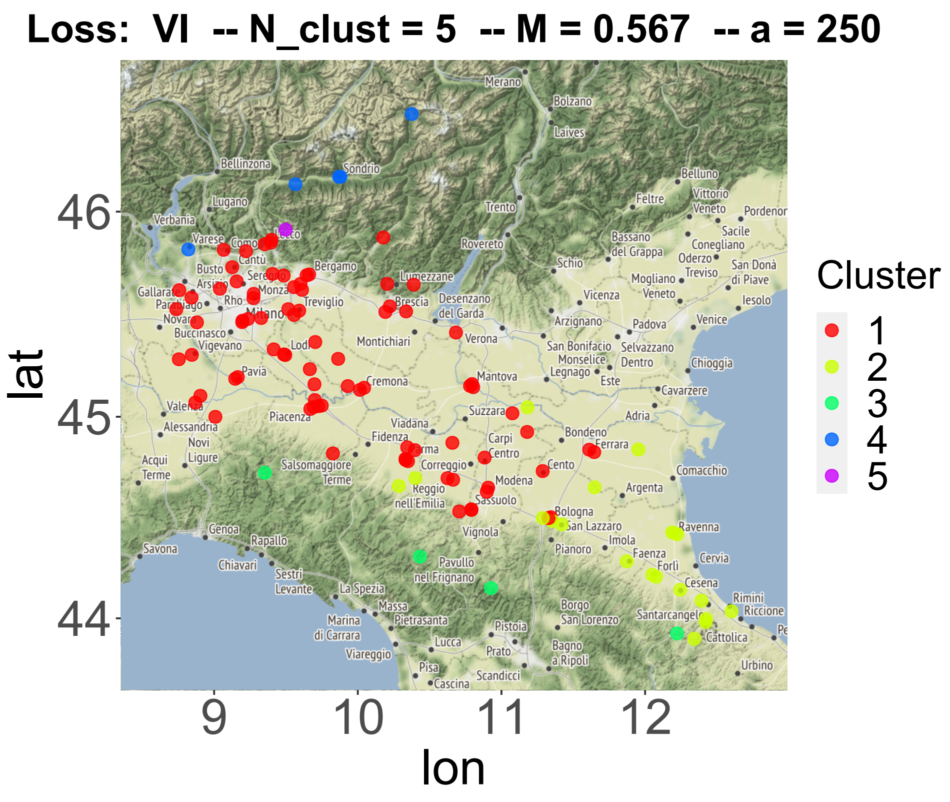


Figure 6: optimal partition from a MCMC sampling setting a=250km and M=0.567

Cluster 1 joints the stations of Pianura Padana, which is one of the most densely populated and economically important areas in Italy. Indeed, it is a highly developed industrial area, with a concentration of factories and production facilities that are involved in a wide range of manufacturing activities. Cluster 2 represents the lower Apennines and the region near the Adriatic Sea; in this area, the concentration of factories is smaller. Cluster 3 puts together the stations of the Apennines, where there are almost no factories and a very small population density. Cluster 4 cluster represents the station of Valtellina (Morbegno, Sondrio, Bormio) and Varese. Thanks to the conformation of the landscape, these regions have a fair number of industries and farms. It is interesting to see that Moggio's station is clustered as a singleton (Cluster 5). This is due to the fact that the station is located far from the village center, at an altitude of more than 1200m in a very isolated location.

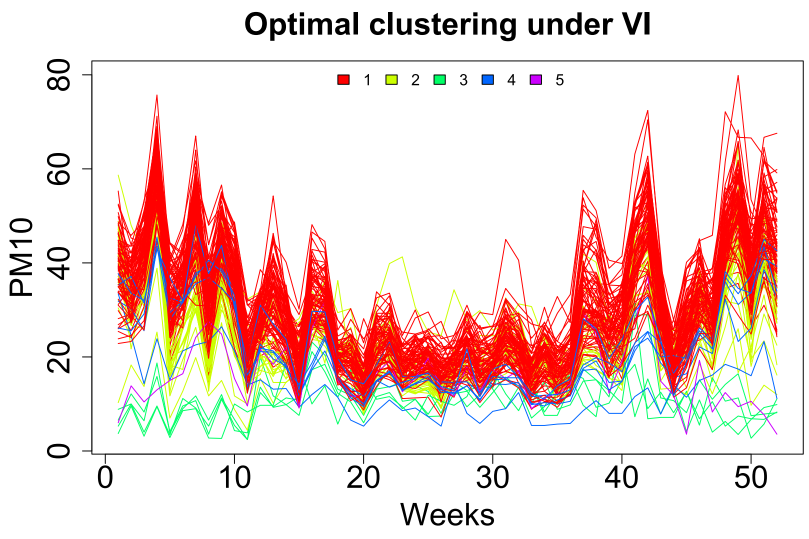


Figure 7: Optimal clustering under VI from a MCMC sampling setting a=250km and M=0.567

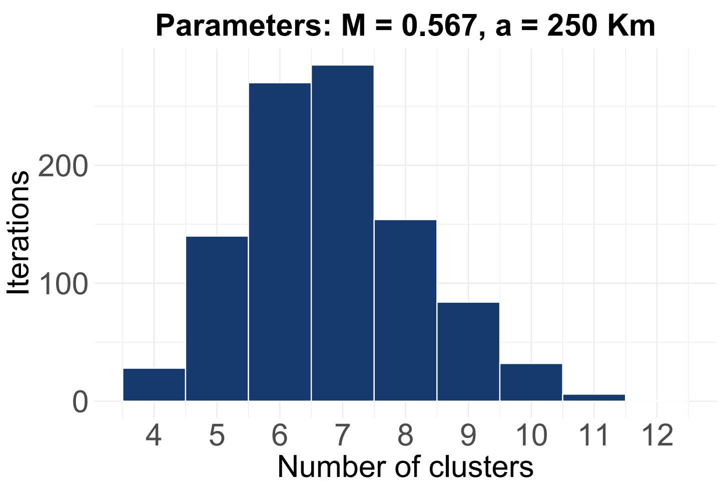


Figure 8: Number of clusters from a MCMC sampling setting a=250km and M=0.567

6.2 Unique values of parameters

The unique values of the parameters can provide valuable information about the underlying structure of the data. In particular, the unique values of the cluster-specific parameters can reveal the nature of the clusters. They are the coefficients of AR(1) models, that usually determine the strength and the sign of the autocorrelation function. Table 1 provides the unique values of the cluster-specific parameter ρ_h of the optimal partition of Figure 6. It is possible to notice that some clusters have a similar unique value ρ_h, meaning that there may not be a significant difference in temporal autocorrelation structure among these clusters. This could suggest that the clusters represent different regions of pollution that have similar temporal patterns of emission and diffusion.

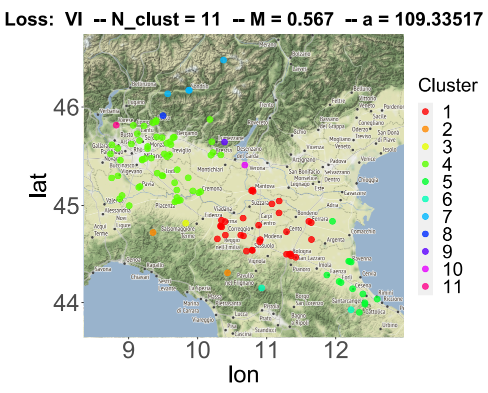
Clusters	1	2	3	4	5
ρ_h	0.5179	0.5177	0.5087	0.4953	0.5438

Table 1: Unique values of parameter ρ_h

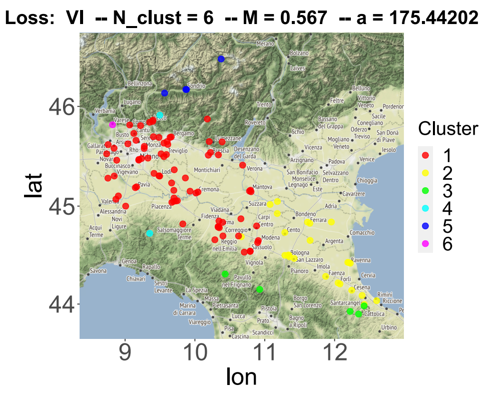
6.3 Varying parameter a

Since a has a strong influence on the number of clusters that the algorithm outputs, additional MCMC simulations with varying values of a are conducted. The general patterns observed in the simulations are consistent with theoretical results. Indeed, for small values of a, cluster boundaries are more tightly defined, resulting in an increase in the number of clusters and including more singletons. As a increases, it is possible to observe that fewer and larger clusters are favored. In Figure 9 three optimal partitions of some MCMC are shown.

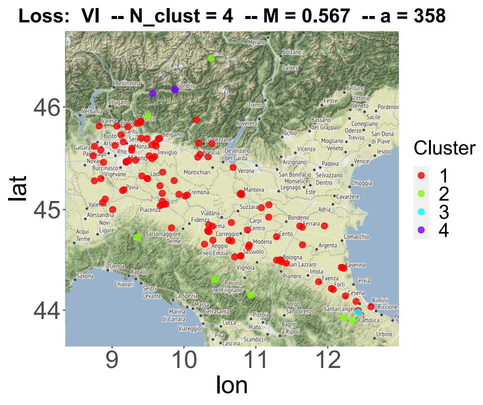
In particular, Figure 9a is the optimal partition of a MCMC with parameter a set equal to the median of the pairwise distances among all the locations. As expected, decreasing the value of a increases the relevance of the spatial setting and leads to a bigger number of clusters. For example, the stations of the plain that before were clustered together, now are split in three groups: Lombard plain, Emilian plain and Romagna plain.
Figure 9b depicts the optimal partition with a equal to the third quantile. As the value of a is increased, the number of clusters decreases. The locations of Pianura Padana are split into two groups, and there is a singleton in the Ligurian Apennine (Corte Brugnatella).
As regards Figure 9c, it is the optimal partition of a MCMC with parameter a equal to the maximum distance between two stations, meaning that the spatial structure of data is not taken into account. In this case, the model clustered the location in mainly three groups (and a singleton for Verucchio location) that can be interpreted as: plain (Cluster 1), mountains (Cluster 2), and valley (Cluster 4).



(a) a=109.36km and M=0.567



(b) a=175.44km and M=0.567



(c) a=358km and M=0.567

Figure 9: Optimal partitions of MCMC with different values of a and M: (a) number of clusters = 11; (b) number of clusters = 6; (c) number of clusters = 4

7 Conclusion and further work

The results of the analysis are plausible and interesting. They demonstrate the effectiveness of the spatial product partition model for clustering PM_{10} data, as evidenced by the discovery of distinct clusters with clear spatial patterns.

One possible direction for improvement is to incorporate additional features or covariates that may be relevant for clustering PM_{10} data, such as meteorological or land-use data. Another potential improvement would be to explore alternative cohesion functions that may offer better performance for this type of data.

8 Appendix

The following appendix presents the full set of calculations used to derive our results.

8.1 Preliminary Calculations

First of all, we consider the following equivalence, which is turning out to be useful in the next steps:

$$\sum_{i=1}^{n_h}\left(\sum_{t=2}^{T}\left(y_{i,t}-\rho_h y_{i,t-1}\right)^2+y_{i,1}^2\right)+\lambda\left(\rho_h-\rho_0\right)^2=A\left(\rho_h-\frac{B}{A}\right)^2-\left(\frac{B^2}{A}-C\right) \tag{2}$$

Where:

$$A=\sum_{i=1}^{n_h}\sum_{t=1}^{T-1}y_{i,t}^2+\lambda$$

$$B=\sum_{i=1}^{n_h}\sum_{t=1}^{T-1}y_{i,t}y_{i,t+1}+\lambda\rho_0$$

$$C=\sum_{i=1}^{n_h}\sum_{t=1}^{T}y_{i,t}^2+\lambda\rho_0^2$$

Proof:

$$\sum_{i=1}^{n_h}\left(\sum_{t=2}^{T}\left(y_{i,t}-\rho_h y_{i,t-1}\right)^2+y_{i,1}^2\right)+\lambda\left(\rho_h-\rho_0\right)^2=$$

$$=\sum_{i=1}^{n_h}\left[\sum_{t=2}^{T}\left(y_{i,t}^2+\rho_h^2y_{i,t-1}^2-2y_{i,t}\rho_h y_{i,t-1}\right)+y_{i,1}^2\right]+\lambda\left(\rho_h^2+\rho_0^2-2\rho_h\rho_0\right)=$$

$$=\sum_{i=1}^{n_h}\sum_{t=1}^{T}y_{i,t}^2+\rho_h^2\sum_{i=1}^{n_h}\sum_{t=2}^{T}y_{i,t-1}^2-2\rho_h\sum_{i=1}^{n_h}\sum_{t=2}^{T}y_{i,t}y_{i,t-1}+\lambda\rho_h^2+\lambda\rho_0^2-2\lambda\rho_0\rho h=$$

$$=\left(\sum_{i=1}^{n_h}\sum_{t=2}^{T}y_{i,t-1}^2+\lambda\right)\rho_h^2-2\left(\sum_{i=1}^{n_h}\sum_{t=2}^{T}y_{i,t}y_{i,t-1}+\lambda\rho_0\right)\rho_h+\sum_{i=1}^{n_h}\sum_{t=1}^{T}y_{i,t}^2+\lambda\rho_0^2=$$

$$=A\rho_h^2-2B\rho_h+C=$$

$$=A\left(\rho_h-\frac{B}{A}\right)^2-\left(\frac{B^2}{A}-C\right)$$

8.2 Likelihood

Here are the computations of the joint likelihood of observation y_i in cluster h:

$$\pi\left(\boldsymbol{y_i} \mid c_i = h, \rho_h, \sigma_h^2\right) = \pi\left(y_{i,1} \mid c_i = h, \sigma_h^2\right) \prod_{t=2}^{T} \pi\left(y_{i,t} \mid y_{i,t-1} c_i = h, \rho_h, \sigma_h^2\right)$$

$$= \sqrt{\frac{1}{2\pi\sigma_h^2}} \exp\left\{-\frac{1}{2\sigma_h^2} y_{i,1}^2\right\} \prod_{t=2}^{T} \left\{\sqrt{\frac{1}{2\pi\sigma_h^2}} \exp\left\{-\frac{1}{2\sigma_h^2}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2\right\}\right\}$$

$$\propto \exp\left\{-\frac{1}{2\sigma_h^2}\left[y_{i,1}^2 + \sum_{t=2}^{T}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2\right]\right\}$$

Focusing on the exponent of the likelihood:

$$\frac{1}{\sigma_h^2}\left(y_{i,1}^2 + \sum_{t=2}^{T}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2\right) =$$

$$= \frac{1}{\sigma_h^2} y_{i,1}^2 + \sum_{t=2}^{T}\left(y_{i,t}^2 - \rho_h 2 y_{i,t-1}^2 - 2 y_{i,t}\rho_h y_{i,t-1}\right) =$$

$$= \sum_{t=1}^{T-1} \frac{1}{\sigma_h^2} y_{i,t}^2 + \frac{1}{\sigma_h^2} y_{i,T}^2 + \sum_{t=1}^{T-1} \frac{1}{\sigma_h^2} 2(-\rho_h) y_{i,t} y_{i,t+1} + \sum_{t=1}^{T-1} \frac{1}{\sigma_h^2} \rho_h^2 y_{i,t}^2$$

$$= \sum_{t=1}^{T-1} \frac{1}{\sigma_h^2}\left(1 + \rho_h^2\right) y_{i,t}^2 + \frac{1}{\sigma_h^2} y_{i,T}^2 + \sum_{t=1}^{T-1} \frac{1}{\sigma_h^2} 2(-\rho_h) y_{i,t} y_{i,t+1}$$

The above quantity is equal to the quadratic form $\boldsymbol{y_i}^T \Sigma_h^{-1} \boldsymbol{y_i} = \sum_{t=1}^{T}\sum_{s=1}^{T}\left(\Sigma_h^{-1}\right)_{t,s} y_{i,t} y_{i,s}$ where:

$$\left(\Sigma_h^{-1}\right)_{t,s} = \begin{cases} \frac{1}{\sigma_h^2}\left(1 + \rho_h^2\right) & \text{if } t = s \neq T \\ \frac{1}{\sigma_h^2} & \text{if } t = s = T \\ \frac{1}{\sigma_h^2}\left(-\rho_h\right) & \text{if } \mid t - s \mid = 1 \\ 0 & \text{if } \mid t - s \mid > 1 \end{cases}$$

8.3 Joint Likelihood

Here are the computations of the joint likelihood of $\underline{\boldsymbol{y_h}}$, that is the vector of all the observations belonging to cluster h :

$$\pi\left(\underline{\boldsymbol{y_h}} \mid \rho_h, \sigma_h^2\right) = \prod_{i=1}^{n_h} \pi\left(\boldsymbol{y_i} \mid c_i = h, \rho_h, \sigma_h^2\right) =$$

$$= \prod_{i=1}^{n_h} \sqrt{\frac{1}{2\pi\sigma_h^2}} \exp\left\{-\frac{1}{2\sigma_h^2} y_{i,1}^2\right\} \prod_{t=2}^{T}\left[\sqrt{\frac{1}{2\pi\sigma_h^2}} \exp\left\{-\frac{1}{2\sigma_h^2}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2\right\}\right]$$

8.4 Posterior

The following calculation demonstrates that the posterior distribution is a normal distribution. The details of this calculation are presented below.

$$
\begin{aligned}
&\pi\left(\rho_h, \sigma_h^2 \mid \underline{\boldsymbol{y_h}}\right) \propto \pi\left(\underline{\boldsymbol{y_h}} \mid \rho_h, \sigma_h^2\right) \pi\left(\rho_h, \sigma_h^2\right) = \\
&= \prod_{i=1}^{n_h} \left\{ \frac{1}{\sqrt{2\pi\sigma_h^2}} \exp\left\{ -\frac{1}{2\sigma_h} y_{i,1}^2 \right\} \prod_{t=2}^{T} \frac{1}{\sqrt{2\pi\sigma_h^2}} \exp\left\{ -\frac{1}{2\sigma_h^2} \left(y_{i,t} - \rho_h y_{i,t-1}\right)^2 \right\} \right\} * \\
&* \sqrt{\frac{\lambda}{2\pi\sigma_h^2}} \exp\left\{ -\frac{\lambda}{2\sigma_h^2} \left(\rho_h - \rho_0\right)^2 \right\} * \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\sigma_h^2\right)^{-\alpha-1} \exp\left\{ -\frac{\beta}{\sigma_h^2} \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2) \propto \\
&\propto \left(\sigma_h^2\right)^{-\frac{n_h T}{2}} \exp\left\{ -\frac{1}{2\sigma_n^2} \sum_{i=1}^{n_h} \left[\sum_{t=2}^{T} \left(y_{it} - \rho_h y_{i,t-1}\right)^2 + y_{i,1}^2 \right] \right\} * \left(\sigma_h^2\right)^{-\frac{1}{2}} \exp\left\{ -\frac{\lambda}{2\sigma_h^2} \left(\rho_h - \rho_0\right)^2 \right\} * \\
&* \left(\sigma_h^2\right)^{-\alpha-1} \exp\left\{ -\frac{\beta}{\sigma_h^2} \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2) \propto \\
&\propto \left(\sigma_h^2\right)^{-\frac{n_h T}{2} - \frac{1}{2} - \alpha - 1} \exp\left\{ -\frac{1}{2} \left[\frac{1}{\sigma_h^2} \sum_{i=1}^{n_h} \left(\sum_{t=2}^{T} \left(y_{i,t} - \rho_h y_{i,t-1}\right)^2 + y_{i,1}^2 \right) + \frac{\lambda}{\sigma_h^2} \left(\rho_h - \rho_0\right)^2 + \frac{2\beta}{\sigma_h^2} \right] \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2)
\end{aligned}
$$

To compute explicitly the Posterior, we exploit:

$$
\pi\left(\rho_h, \sigma_h^2 \mid \underline{\boldsymbol{y_h}}\right) = \pi\left(\rho_h \mid \sigma_h^2, \underline{\boldsymbol{y_h}}\right) \pi\left(\sigma_h^2 \mid \underline{\boldsymbol{y_h}}\right)
$$

First term:

$$
\begin{aligned}
&\pi\left(\sigma_h^2 \mid \underline{\boldsymbol{y_h}}\right) = \int_{\mathbb{R}} \pi\left(\rho_h, \sigma_h^2 \mid \underline{\boldsymbol{y_h}}\right) d\rho_h \propto \\
&\propto \int_{\mathbb{R}} \left(\sigma_h^2\right)^{-\frac{n_h T}{2} - \frac{1}{2} - \alpha - 1} \exp\left\{ -\frac{\beta}{\sigma_h^2} \right\} * \\
&* \exp\left\{ -\frac{1}{2\sigma_h^2} \left[\sum_{i=1}^{n_h} \left(\sum_{t=2}^{T} \left(y_{i,t} - \rho_h y_{i,t-1}\right)^2 + y_{i,1}^2 \right) + \lambda \left(\rho_h - \rho_0\right)^2 \right] \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2) d\rho_h = \\
&\overset{(2)}{=} \left(\sigma_h^2\right)^{-\frac{n_h T}{2} - \frac{1}{2} - \alpha - 1} \exp\left\{ -\frac{\beta}{\sigma_h^2} \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2) \int_{\mathbb{R}} \exp\left\{ -\frac{1}{2\sigma_h^2} \left[A \left(\rho_h - \frac{B}{A} \right)^2 - \left(\frac{B^2}{A} - C \right) \right] \right\} d\rho_h = \\
&= \left(\sigma_h^2\right)^{-\frac{n_h T}{2} - \frac{1}{2} - \alpha - 1} \exp\left\{ -\frac{\beta}{\sigma_h^2} \right\} \exp\left\{ \frac{1}{2\sigma_h^2} \left(\frac{B^2}{A} - C \right) \right\} \frac{\sqrt{2\pi\sigma_h^2}}{\sqrt{A}} \mathbb{1}_{[0,\infty)}(\sigma_h^2) * \\
&* \int_{\mathbb{R}} \frac{\sqrt{A}}{\sqrt{2\pi\sigma_h^2}} \exp\left\{ -\frac{A}{2\sigma_h^2} \left(\rho_h - \frac{B}{A} \right)^2 \right\} d\rho_h \propto \\
&\propto \left(\sigma_h^2\right)^{-\left(\frac{n_h T}{2} + \alpha\right) - 1} \exp\left\{ -\frac{1}{\sigma_h^2} \left[\beta - \frac{1}{2} \left(\frac{B^2}{A} - C \right) \right] \right\} \mathbb{1}_{[0,\infty)}(\sigma_h^2)
\end{aligned}
$$

This is the kernel of an Inverse Gamma $\mathcal{IG}\left(\alpha_{\text{post}}, \beta_{\text{post}}\right)$ whose parameters are:

$$\alpha_{\text{post}} = \alpha + \frac{n_h T}{2}; \quad \beta_{\text{post}} = \beta - \frac{1}{2}\left(\frac{B^2}{A} - C\right)$$

Second term:

$\pi\left(\rho_h \mid \sigma_h^2, \underline{\boldsymbol{y_h}}\right) \propto \pi\left(\rho_h, \sigma_h^2, \underline{\boldsymbol{y_h}}\right) \propto$

$$\propto \exp\left\{-\frac{1}{2}\left[\frac{1}{\sigma_h^2}\sum_{i=1}^{n_h}\left(\sum_{t=2}^{T}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2 + y_{i,1}^2\right) + \frac{\lambda}{\sigma_h^2}\left(\rho_h - \rho_0\right)^2 + \frac{2\beta}{\sigma_h^2}\right]\right\}\mathbb{1}_{[0,\infty)}(\sigma_h^2)$$

$$\propto \exp\left\{-\frac{1}{2\sigma_h^2}\left[\sum_{i=1}^{n_h}\left(\sum_{t=2}^{T}\left(y_{i,t} - \rho_h y_{i,t-1}\right)^2 + y_{i,1}^2\right) + \lambda\left(\rho_h - \rho_0\right)^2\right]\right\}$$

$$\overset{(2)}{=} \exp\left\{-\frac{1}{2\sigma_h^2}\left[A\left(\rho_h - \frac{B}{A}\right)^2 - \left(\frac{B^2}{A} - C\right)\right]\right\}$$

$$\propto \exp\left\{-\frac{A}{2\sigma_h^2}\left(\rho_h - \frac{B}{A}\right)^2\right\}$$

This is the kernel of a Normal $\mathcal{N}\left(\rho_{0,\text{ post}}, \frac{\sigma_h^2}{\lambda_{\text{post}}}\right)$ whose parameters are:

$$\rho_{0,\text{ post}} = \frac{B}{A}; \quad \lambda_{\text{post}} = A;$$

8.5 Marginal Density

The following computations show how to derive the marginal density for a single time series.

$\pi\left(\boldsymbol{y_i}\right) = \int_{\mathbb{R}\times\mathbb{R}^+} \pi\left(\boldsymbol{y_i}, \rho_i, \sigma_i^2\right) d\rho_i d\sigma_i^2 =$

$$= \int_{\mathbb{R}\times\mathbb{R}^+} \pi\left(\boldsymbol{y_i} \mid \rho_i, \sigma_i^2\right)\pi\left(\rho_i, \sigma_i^2\right) d\rho_i d\sigma_i^2 =$$

$$= \int_{\mathbb{R}\times\mathbb{R}^+} \frac{1}{\sqrt{2\pi\sigma_i^2}}\exp\left\{-\frac{1}{2\sigma_i^2}y_{i,1}^2\right\}\prod_{t=2}^{T}\frac{1}{\sqrt{2\pi\sigma_i^2}}\exp\left\{-\frac{1}{2\sigma_i^2}\left(y_{i,t} - \rho_i y_{i,t-1}\right)^2\right\} *$$

$$* \sqrt{\frac{\lambda}{2\pi\sigma_i^2}}\exp\left\{-\frac{\lambda}{2\sigma_i^2}\left(\rho_i - \rho_0\right)^2\right\} * \frac{\beta^\alpha}{\Gamma(\alpha)}\left(\sigma_i^2\right)^{-\alpha-1}\exp\left\{-\frac{\beta}{\sigma_i^2}\right\} d\rho_i d\sigma_i^2 =$$

$$\overset{(2)}{=} \sqrt{\lambda}\left(2\pi\right)^{-\frac{T+1}{2}}\frac{\beta^\alpha}{\Gamma\left(\alpha\right)}\int_{\mathbb{R}\times\mathbb{R}^+}\left(\sigma_i^2\right)^{-\frac{T+1}{2}-\alpha-1}\exp\left\{-\frac{\beta}{\sigma_i^2}\right\} *$$

$$* \exp\left\{-\frac{1}{2\sigma_i^2}\left[A\left(\rho_i - \frac{B}{A}\right)^2 - \left(\frac{B^2}{A} - C\right)\right]\right\} d\rho_i d\sigma_i^2 =$$

$$= \sqrt{\lambda}\left(2\pi\right)^{-\frac{T+1}{2}}\frac{\beta^\alpha}{\Gamma\left(\alpha\right)}\int_{\mathbb{R}\times\mathbb{R}^+}\left(\sigma_i^2\right)^{-\frac{1}{2}}\left(\sigma_i^2\right)^{-\left(\frac{T}{2}+\alpha\right)-1}\exp\left\{-\frac{1}{\sigma_i^2}\left[\beta - \frac{1}{2}\left(\frac{B^2}{A} - C\right)\right]\right\} *$$

$$* \exp\left\{-\frac{A}{2\sigma_i^2}\left(\rho_i - \frac{B}{A}\right)^2\right\} d\rho_i d\sigma_i^2$$

We can notice that the integrand function is the kernel of a Normal-InverseGamma:

$$\left(\rho,\sigma^2\right) \sim \mathcal{N}-\mathcal{IG}\left(\bar{\rho_0},\bar{\lambda},\bar{\alpha},\bar{\beta}\right)$$

$$\bar{\rho_0}=\frac{B}{A} \qquad \bar{\lambda}=A \qquad \bar{\alpha}=\alpha+\frac{T}{2} \qquad \bar{\beta}=\beta-\frac{1}{2}\left(\frac{B^2}{A}-C\right)$$

So we obtain:

$$\pi(\boldsymbol{y_i})=\sqrt{\lambda}\,(2\pi)^{-\frac{T+1}{2}}\,\frac{\beta^\alpha}{\Gamma(\alpha)}*\sqrt{\frac{2\pi}{A}}\frac{\Gamma\left(\alpha+\frac{T}{2}\right)}{\left[\beta-\frac{1}{2}\left(\frac{B^2}{A}-C\right)\right]^{\alpha+\frac{T}{2}}}$$

Now we can rearrange this expression and obtain:

$$\pi(\boldsymbol{y_i})=\frac{\Gamma(\alpha+\frac{T}{2})}{\Gamma(\alpha)\pi^{\frac{T}{2}}(2\alpha)^{\frac{T}{2}}\left(\frac{\beta^T A}{\alpha^T\lambda}\right)^{\frac{1}{2}}}\left\{1+\frac{1}{2\alpha}\left[\frac{\alpha}{\beta}\left(C-\frac{B^2}{A}\right)\right]\right\}^{-\frac{T}{2}-\alpha}$$

8.6 Weights computation

The following computations show how to derive the weights from:

$$w_h=\frac{ppf\left(S_1^{-i},\dots,S_h^{-i}\cup\{i\},S_H^{-i}\right)}{ppf\left(S_1^{-i},\dots,S_H^{-i}\right)} \qquad w_{H+1}=\frac{ppf\left(S_1^{-i},\dots,S_H^{-i},\{i\}\right)}{ppf\left(S_1^{-i},\dots,S_H^{-i}\right)}$$

- Denominator:

$$ppf\left(S_1^{-i},\dots,S_H^{-i}\right)\propto\prod_{m=1}^{H}C\left(S_m\right)\propto\prod_{m=1}^{H}M\times\Gamma\left(n_m^{-i}\right)\times\prod_{k,j\in S_m^{-i}}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]=$$

$$=M^H\prod_{m=1}^{H}\left(n_m^{-i}-1\right)!\prod_{k,j\in S_m^{-i}}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]$$

- Numerator for w_h:

$$ppf\left(S_1^{-i},\dots,S_h^{-i}\cup\{i\},\dots,S_H^{-i}\right)\propto\prod_{m=1}^{H}M\times\Gamma\left(n_m^{-i}\right)\times\prod_{k,j\in S_m^{-i}}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]=$$

$$=M^H\left\{\prod_{m\neq h}^{H}\left(n_m^{-i}-1\right)!\prod_{k,j\in S_m^{-i}}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]\right\}(n_h-1)!\prod_{k,j\in S_h}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]$$

- Numerator for w_{H+1}:

$$ppf\left(S_1^{-i},\dots,S_H^{-i},\{i\}\right)\propto$$

$$\propto\left\{\prod_{m=1}^{H}M\times\Gamma\left(n_m^{-i}\right)\prod_{k,j\in S_m^{-i}}\mathbb{I}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]\right\}\times M\times\Gamma(1)\mathbb{I}\left[\|\boldsymbol{s}_i-\boldsymbol{s}_i\|\leq a\right]=$$

$$=M^{H+1}\prod_{m=1}^{H}\left(n_m^{-i}-1\right)!\prod_{k,j\in S_m^{-i}}\left[\|\boldsymbol{s}_k-\boldsymbol{s}_j\|\leq a\right]$$

Now we can compute w_h:

$$
\begin{aligned}
w_h &= \frac{ppf\left(S_1^{-i}, \ldots, S_h^{-i} \cup \{i\}, S_H^{-i}\right)}{ppf\left(S_1^{-i}, \ldots, S_H^{-i}\right)} = \\
&= \frac{M^H \left\{\prod_{m \neq h}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]\right\} \left(n_h^{-i}\right)! \prod_{k,j \in S_h} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]}{M^H \prod_{m=1}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]} = \\
&= \frac{M^H \left\{\prod_{m \neq h}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]\right\} \left(n_h^{-i}\right)! \prod_{k,j \in S_h} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]}{M^H \left\{\prod_{m \neq h}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]\right\} \left(n_h^{-i} - 1\right)! \prod_{k,j \in S_h^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]} = \\
&= n_h^{-i} \times \frac{\prod_{k,j \in S_h} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]}{\prod_{k,j \in S_h^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]} = \\
&= n_h^{-i} \left\{\prod_{j \in S_h^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_i - \boldsymbol{s}_j\| \leq a\right]\right\}^2 = \\
&= \begin{cases} 0, \text{ if } \exists j^* \in S_h^{-i} \text{ s.t. } ||s_i - s_{j^*}|| = d_{ij^*} > a \\ n_h^{-i}, \text{ otherwise} \end{cases}
\end{aligned}
$$

And w_{H+1}:

$$
\begin{aligned}
w_{H+1} &= \frac{ppf\left(S_1^{-i}, \ldots, S_H^{-i}, \{i\}\right)}{ppf\left(S_1^{-i}, \ldots, S_H^{-i}\right)} = \\
&= \frac{M^{H+1} \prod_{m=1}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]}{M^H \prod_{m=1}^H \left(n_m^{-i} - 1\right)! \prod_{k,j \in S_m^{-i}} \mathbb{I}\left[\|\boldsymbol{s}_k - \boldsymbol{s}_j\| \leq a\right]} \\
&= M
\end{aligned}
$$

8.7 Some more code from Bayesmix

ar1_likelihood.h

```cpp
class AR1Likelihood
    : public BaseLikelihood<AR1Likelihood, State::UniLS> {
 public:
  AR1Likelihood() = default;
  ~AR1Likelihood() = default;
  bool is_multivariate() const override { return true; };
  bool is_dependent() const override { return false; };
  void clear_summary_statistics() override;

  // dim reflects the length of the time series
  void set_dim(unsigned int dim_) {
    dim = dim_;
    clear_summary_statistics();
  };

  void set_state(const State::UniLS &state_, bool update_card = true);

  unsigned int get_dim() const { return dim; };

  Eigen::MatrixXd get_data_sum_squares() const { return data_sum_squares;↩
      };

 protected:
  double compute_lpdf(const Eigen::RowVectorXd &datum) const override;
  void update_sum_stats(const Eigen::RowVectorXd &datum, bool add) ↩
      override;

  unsigned int dim;
  //this matrix is useful to calculate A,B,C
  Eigen::MatrixXd data_sum_squares;

  Eigen::VectorXd mean;
  Eigen::MatrixXd prec_chol;
  double prec_logdet;

};
```

ar1_likelihood.cc:

```
void AR1Likelihood::update_sum_stats(const Eigen::RowVectorXd &datum,
                                      bool add) {
  // Check if dim is not defined yet (this usually doesn't happen if the
  // hierarchy is initialized)
  if (!dim) set_dim(datum.size());
  // Updates
  if (add) {
    data_sum_squares += datum.transpose() * datum;
  } else {
    data_sum_squares -= datum.transpose() * datum;
  }
}

void AR1Likelihood::clear_summary_statistics() {
  data_sum_squares = Eigen::MatrixXd::Zero(dim, dim);
}

void AR1Likelihood::set_state(const State::UniLS &state_, bool ↩
    update_card) {
  state = state_;
  if (update_card) {
    set_card(state.card);
  }

  mean = Eigen::VectorXd::Zero(dim);


  Eigen::VectorXd diag = (1.0+state.mean*state.mean)/state.var * Eigen::VectorXd::Ones(dim);
  diag[dim-1] = 1.0/state.var;
  Eigen::VectorXd codiag = -state.mean/state.var * Eigen::VectorXd::Ones(dim-1);
  Eigen::MatrixXd Omega = Eigen::MatrixXd::Zero(dim,dim);
  Omega.diagonal(0) = diag;
  Omega.diagonal(1) = codiag;
  Omega.diagonal(-1) = codiag;

  prec_chol = Eigen::LLT<Eigen::MatrixXd>(Omega).matrixL();
  Eigen::VectorXd diagL = prec_chol.diagonal();
  prec_logdet = 2 * log(diagL.array()).sum();
}

double AR1Likelihood::compute_lpdf(
    const Eigen::RowVectorXd &datum) const {
  return bayesmix::multi_normal_prec_lpdf(datum, mean, prec_chol,
                                          prec_logdet);
}
```

ar1nig_hierarchy.h:

```
class AR1NIGHierarchy
    : public BaseHierarchy<AR1NIGHierarchy, AR1Likelihood, NIGPriorModel> {
 public:
  AR1NIGHierarchy() = default;
  ~AR1NIGHierarchy() = default;

  //! Returns the Protobuf ID associated to this class
  bayesmix::HierarchyId get_id() const override {
    return bayesmix::HierarchyId::AR1NIG;
  }

  //! Sets the default updater algorithm for this hierarchy
  void set_default_updater() { updater = std::make_shared<AR1NIGUpdater>(); }

  //! Initializes state parameters to appropriate values
  void initialize_state() override {
    // Initialize likelihood dimension to the time series length
    unsigned int ts_length = like->get_dataset()->cols();
    like->set_dim(ts_length);

    // Get hypers
    auto hypers = prior->get_hypers();
    // Initialize likelihood state
    State::UniLS state;
    state.mean = hypers.mean;
    state.var = hypers.scale / (hypers.shape + 1.0);
    like->set_state(state);
  };

  //! Evaluates the log-marginal distribution of data in a single point
  //! @param hier_params  Pointer to the container of (prior or posterior)
  //! hyperparameter values
  //! @param datum        Point which is to be evaluated
  //! @return             The evaluation of the lpdf
  double marg_lpdf(ProtoHypersPtr hier_params,
                   const Eigen::RowVectorXd &datum) const override {
    auto params = hier_params->nnig_state();
    Eigen::MatrixXd data_sum_squares = datum.transpose() * datum;
    unsigned dim = datum.size();
    double a_p = params.var_scaling() + (data_sum_squares.diagonal(0).sum() - data_sum_squares.diagonal(0)[dim-1]) ;
    double b_p = params.var_scaling() * params.mean() + data_sum_squares.diagonal(1).sum();
    double c_p = params.var_scaling() * (params.mean() * params.mean()) + data_sum_squares.trace();
    double const_num = stan::math::tgamma(params.shape() + 0.5*dim);
    double det_sigma = pow(params.scale()/params.shape(), dim) * a_p/params.var_scaling();
    double const_den = stan::math::tgamma(params.shape()) *
              pow(stan::math::pi()*2*params.shape(), 0.5*dim) * pow(det_sigma,0.5);

    double factor3 = 1 + 1.0/(2.0*params.scale()) * (c_p - b_p*b_p/a_p);
    double c = const_num/const_den;

    double out = c * pow(factor3,-0.5*dim - params.shape());

    return std::log(out) ;
  };

};
```

ar1nig_updater.h:

```
class AR1NIGUpdater
    : public SemiConjugateUpdater<AR1Likelihood, NIGPriorModel> {
 public:
  AR1NIGUpdater() = default;
  ~AR1NIGUpdater() = default;



  bool is_conjugate() const override { return true; };

  ProtoHypersPtr compute_posterior_hypers(AbstractLikelihood &like,
                                          AbstractPriorModel &prior) override;

  std::shared_ptr<AbstractUpdater> clone() const override {
    auto out =
        std::make_shared<AR1NIGUpdater>(static_cast<AR1NIGUpdater const ↩
            &>(*this));
    out->clear_hypers();
    return out;
  }
};
```

ar1nig_updater.cc:

```
AbstractUpdater::ProtoHypersPtr AR1NIGUpdater::compute_posterior_hypers(
    AbstractLikelihood& like, AbstractPriorModel& prior) {
  // Likelihood and Prior downcast
  auto& likecast = downcast_likelihood(like);
  auto& priorcast = downcast_prior(prior);

  // Getting required quantities from likelihood and prior
  int card = likecast.get_card();
  int dim = likecast.get_dim();

  Eigen::MatrixXd data_sum_squares = likecast.get_data_sum_squares();
  auto hypers = priorcast.get_hypers();

  // No update possible
  if (card == 0) {
    return priorcast.get_hypers_proto();
  }

  // Compute posterior hyperparameters
  double mean, var_scaling, shape, scale;

  double a_p = hypers.var_scaling + (data_sum_squares.diagonal(0).sum() -↩
      data_sum_squares.diagonal(0)[dim-1]) ;
  double b_p = hypers.var_scaling * hypers.mean + data_sum_squares.↩
     diagonal(1).sum();
  double c_p = hypers.var_scaling * (hypers.mean * hypers.mean) + ↩
     data_sum_squares.trace();

  mean = b_p/a_p;
  var_scaling = a_p;
  shape = hypers.shape + 0.5 * card * dim ;
  scale = hypers.scale - 0.5 * (b_p*b_p/a_p - c_p);

  // Proto conversion
  ProtoHypers out;
  out.mutable_nnig_state()->set_mean(mean);
  out.mutable_nnig_state()->set_var_scaling(var_scaling);
  out.mutable_nnig_state()->set_shape(shape);
  out.mutable_nnig_state()->set_scale(scale);
  return std::make_shared<ProtoHypers>(out);
}
```

dirichlet_c2_mixing.h:

```cpp
class DirichletC2Mixing
    : public BaseMixing<DirichletC2Mixing, DirichletC2::State, bayesmix::DPC2Prior> {
 public:
  DirichletC2Mixing() = default;
  ~DirichletC2Mixing() = default;

  //! Performs conditional update of state, given allocations and unique values
  //! @param unique_values  A vector of (pointers to) Hierarchy objects
  //! @param allocations    A vector of allocations label
  void update_state(
      const std::vector<std::shared_ptr<AbstractHierarchy>> &unique_values,
      const std::vector<unsigned int> &allocations) override;

  //! Read and set state values from a given Protobuf message
  void set_state_from_proto(const google::protobuf::Message &state_) override;

  //! Writes current state to a Protobuf message and return a shared_ptr
  //! New hierarchies have to first modify the field 'oneof val' in the
  //! MixingState message by adding the appropriate type
  std::shared_ptr<bayesmix::MixingState> get_state_proto() const override;

  //! Returns the Protobuf ID associated to this class
  bayesmix::MixingId get_id() const override { return bayesmix::MixingId::DPC2; }

  //! Returns whether the mixing is conditional or marginal
  bool is_conditional() const override { return false; }
  bool is_dependent() const override {return true; }

 protected:
  //! Returns probability mass for an old cluster (for marginal mixings only)
  //! @param n          Total dataset size
  //! @param n_clust    Number of clusters
  //! @param log        Whether to return logarithm-scale values or not
  //! @param propto     Whether to include normalizing constants or not
  //! @param hier       `Hierarchy` object representing the cluster
  //! @return           Probability value
  double mass_existing_cluster(
      const unsigned int n, const unsigned int n_clust, const bool log,
      const bool propto,
      const std::shared_ptr<AbstractHierarchy> hier,
      const Eigen::RowVectorXd &covariate) const override;

  //! Returns probability mass for a new cluster (for marginal mixings only)
  //! @param n          Total dataset size
  //! @param log        Whether to return logarithm-scale values or not
  //! @param propto     Whether to include normalizing constants or not
  //! @param n_clust     Current number of clusters
  //! @return            Probability value
  double mass_new_cluster(const unsigned int n, const unsigned int ↩
      n_clust,
                          const bool log, const bool propto,
                          const Eigen::RowVectorXd &covariate) const ↩
                             override;

  //! Initializes state parameters to appropriate values
  void initialize_state() override;
};
```

dirichlet_c2_mixing.cc:

```
void DirichletC2Mixing::update_state(
    const std::vector<std::shared_ptr<AbstractHierarchy>> &unique_values,
    const std::vector<unsigned int> &allocations) {
  auto &rng = bayesmix::Rng::Instance().get();
  auto priorcast = cast_prior();
  unsigned int n = allocations.size();

  if (priorcast->has_fixed_value()) {
    return;
  }
  else {
    throw std::invalid_argument("Unrecognized mixing prior");
  }
}



double DirichletC2Mixing::mass_existing_cluster(
    const unsigned int n, const unsigned int n_clust, const bool log,
    const bool propto, const std::shared_ptr<AbstractHierarchy> hier,
    const Eigen::RowVectorXd &covariate) const {
  double out;
  // flag on the closeness
  bool is_near = 1;
  std::set<int> index_clusters = hier->get_data_idx();
  for(const auto i : index_clusters)
  {
    if(haversine_formula(covariates_ptr->row(i),covariate) > state.a)
    {
      is_near = 0;
      break;
    }
  }

  if (log) {
    out = hier->get_log_card();
    if (!propto) out -= std::log(n + state.totalmass);
    return is_near ? out : std::log(0);
  } else {
    out = 1.0 * hier->get_card();
    if (!propto) out /= (n + state.totalmass);
    return is_near ? out : 0;
  }
  return std::numeric_limits<double>::quiet_NaN();
}

double DirichletC2Mixing::mass_new_cluster(const unsigned int n,
                                           const unsigned int n_clust,
                                           const bool log,
                                           const bool propto,
                                           const Eigen::RowVectorXd &↩
                                               covariate) const {
  double out;
  if (log) {
    out = state.logtotmass;
    if (!propto) out -= std::log(n + state.totalmass);
  } else {
    out = state.totalmass;
    if (!propto) out /= (n + state.totalmass);
  }
  return out;
}



void DirichletC2Mixing::set_state_from_proto(
    const google::protobuf::Message &state_) {
  auto &statecast = downcast_state(state_);
  state.totalmass = statecast.dpc2_state().totalmass();
  state.logtotmass = std::log(state.totalmass);
  state.a = statecast.dpc2_state().a();
}


std::shared_ptr<bayesmix::MixingState> DirichletC2Mixing::get_state_proto↩
    ()
    const {
  bayesmix::DPC2State state_;
  state_.set_totalmass(state.totalmass);
  state_.set_a(state.a);
  auto out = std::make_shared<bayesmix::MixingState>();
  out->mutable_dpc2_state()->CopyFrom(state_);
  return out;
}

void DirichletC2Mixing::initialize_state() {
  auto priorcast = cast_prior();
  if (priorcast->has_fixed_value()) {
    state.totalmass = priorcast->fixed_value().totalmass();
    state.logtotmass = std::log(state.totalmass);
    state.a = priorcast->fixed_value().a();
    if (state.totalmass <= 0) {
      throw std::invalid_argument("Total mass (or a) parameter must be > ↩
          0");
```

```cpp
    }
    if (state.a <= 0) {
      throw std::invalid_argument("Distance parameter a must be > 0");
    }
  }
  else {
    throw std::invalid_argument("Unrecognized mixing prior");
  }
}

// This function compute the haversine formula (distance in km) between two points
// given as a vector (lat,lon)
double haversine_formula(const Eigen::RowVectorXd& point1, const Eigen::RowVectorXd& point2)
{
  // check if the coordinate is bidimensional
  if(point1.cols() != 2 || point2.cols() != 2)
  {
    std::cerr << "Coordinates are not in the right format!" << std::endl;
  }
  const double earth_radius = 6371.0;
  double lat1 = point1(0) * M_PI / 180.0;
  double lat2 = point2(0) * M_PI / 180.0;
  double delta_lat = (lat2 - lat1);
  double delta_long = (point2(1) - point1(1)) * M_PI / 180.0;

  double a = std::sin(delta_lat / 2.0) * std::sin(delta_lat / 2.0) +
          std::cos(lat1) * std::cos(lat2) *
          std::sin(delta_long / 2.0) * std::sin(delta_long / 2.0);

  double c = 2 * std::atan2(std::sqrt(a), std::sqrt(1 - a));

  return earth_radius * c;
}
```

References

[1] Mario Beraha, Bruno Guindani, Matteo Gianella, and Alessandra Guglielmi. Bayesmix: Bayesian mixture models in c++. 2022.

[2] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting.* Springer Nature, 2016.

[3] Radford Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9, 01 2000.

[4] Garritt Page and Fernando Quintana. Spatial product partition models. *Bayesian Analysis*, 11, 04 2015.

[5] Fernando Quintana and Pilar Iglesias. Bayesian clustering and product partition models. *Journal of the Royal Statistical Society Series B*, 65:557–574, 05 2003.

[6] Fernando Quintana, Rosângela Loschi, and Garritt Page. *Bayesian Product Partition Models*, pages 1–15. 11 2018.

[1] [4] [6] [3] [5] [2]

Link to GitHub repository

- Project implamentation:https://github.com/eugeniovaretti/PM10_BAYESIAN
- Original Bayesmix Library [1]: https://github.com/bayesmix-dev/bayesmix
- Extended Bayesmix Library: https://github.com/eugeniovaretti/bayesmix