Homework 2

Topics on Optimization and Machine Learning

Roberto Meroni
email: roberto.meroni@estudiantat.upc.edu

April 2024

Introduction

The goal of the project is to obtain a graph representing the network of eight sensors distributed in Barcelona. The city is characterized by a varied landscape that includes beaches, a dense urban core, parks, and hills.
This diversity can lead to different microclimates within relatively short distances. Differences in temperature, humidity, vegetation, and air pollution are among the factors that make measuring O_3 concentrations challenging.
With this in mind, three different techniques to obtain a graph are explored.

1 Distance-based model

For this model, the only factor taken into account to develop a network topology is the distance between nodes.
The Adjacency Matrix **A** is obtained from:

$$w_{ij} = \begin{cases} e^{-\frac{d_{ij}^2}{2\theta^2}} & \text{if } d_{ij} < T, \\ 0 & \text{otherwise.} \end{cases}$$

The formula uses a Gaussian decay function to model the decrease in connection strength with distance. The exponential part $exp(-(d_{ij}^2)/(2\theta^2))$ rapidly diminishes the connection weight as the distance increases.

Theta

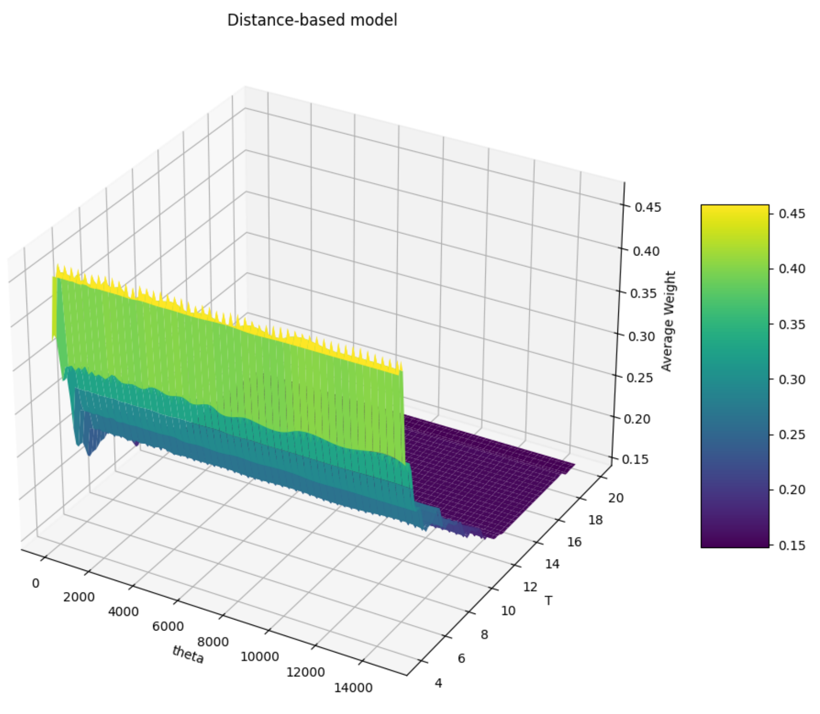


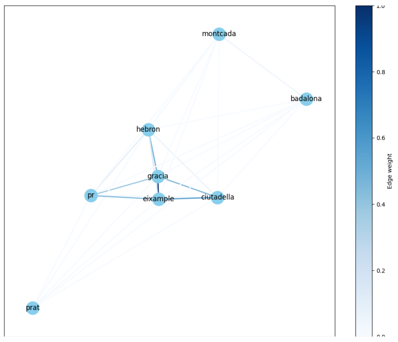
Figure 1: A smaller Threshold will result in fewer edges, but those that remain will have higher weights, increasing the average weight. Also, a higher θ weakens the relationships between nodes.

The parameter θ determines how quickly the weight of an edge between two sensors decreases with increasing distance. It effectively controls the ”spread” of the Gaussian function.

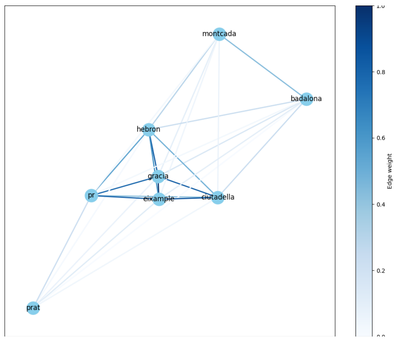
- A smaller value of θ results in a quicker decline in connection weights. This means that only sensors that are very close to each other will have significant weights. The graph will tend to have clusters of highly connected nodes with very sparse connections between these clusters.
- A larger θ smooths out the decay curve, allowing distant nodes to still have considerable influence on each other. The graph will show a more uniform spread of connectivity across a broader area, creating a more integrated network.

To test the Theta parameter, four different θ have been assigned (Threshold is fixed at maximum distance):

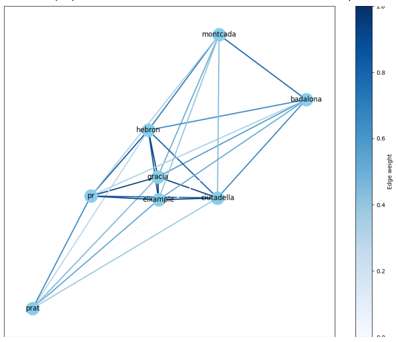
1. $\theta = stdev(distances)/2$
2. $\theta = stdev(distances)$
3. $\theta = stdev(distances) * 2$
4. $\theta = stdev(distances) * 4$



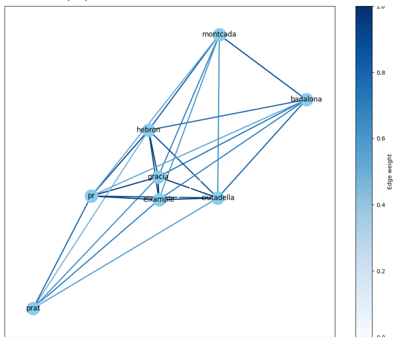
(a) $\theta = StandardDeviation/2$



(b) $\theta = StandardDeviation$



(c) $\theta = StandardDeviation * 2$



(d) $\theta = StandardDeviation * 4$

Figure 2: Comparison between different Theta values for fixed Threshold

Using the Standard Deviation to tune θ allows to capture the inherent variability in the data. The Threshold is kept to the maximum to show the θ effects on all the connections.
As expected, for small Theta values only close connections are considered particularly relevant. Increasing Theta gives more importance to distant connections; for extremely high values, almost all the connections are considered relevant and the distance model loses its purpose. As we don't modify the Threshold, the number of allowed connections remains the same across the different graphs, with only changes in the weights of the connections.

Threshold

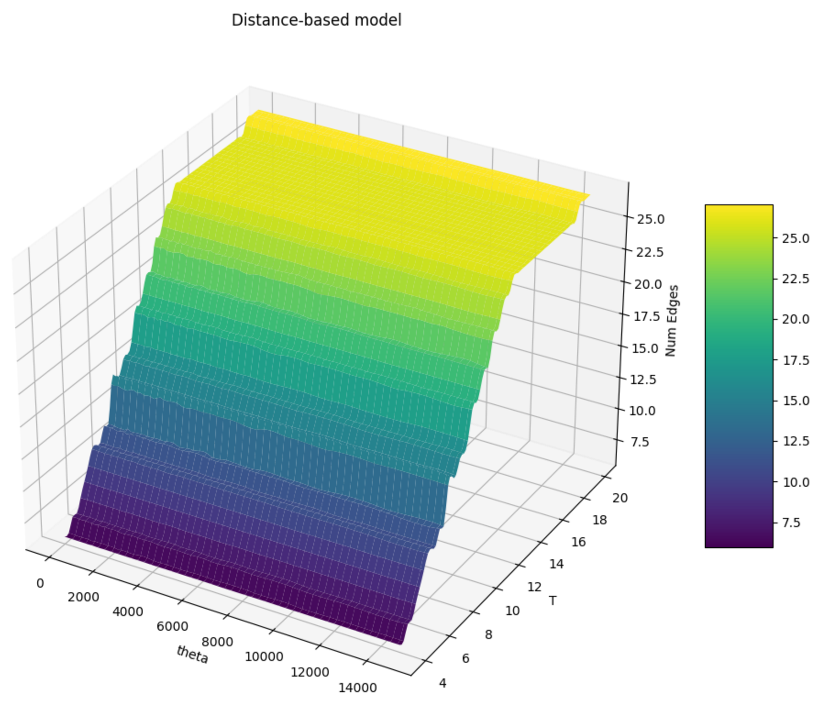


Figure 3: For the distance-based model, the number of edges depends only by the Threshold.

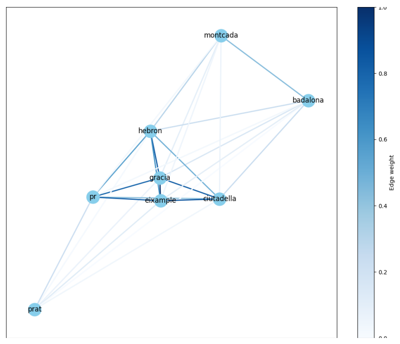
The threshold T sets a hard limit on the maximum distance at which sensors can influence each other, regardless of the weight calculated by the weight function. If the distance between two sensors exceeds T, their connection weight is set to zero.

- A smaller threshold leads to a more localized network where only nearby sensors are connected.
- A larger threshold increases the cluster sizes, allowing sensors that are farther apart to influence temperature measurements.

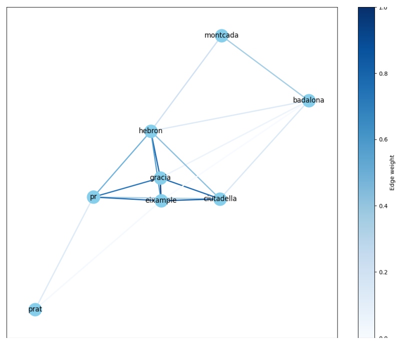
To test the Threshold parameter, four different T have been assigned (Theta is fixed at $\theta = stdev(distances)$):

1. T=max(distances), where all the connections are considered.
2. T=percentile(distances,70), where maximum 70% of the connections are considered.

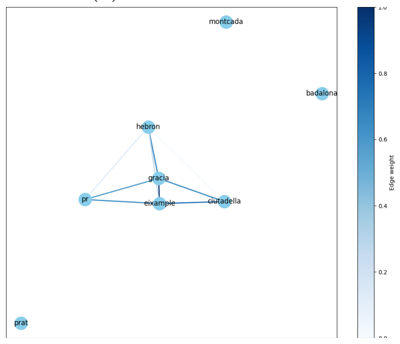
3. T=percentile(distances,40), where maximum 40% of the connections are considered.
4. T=percentile(distances,20), where maximum 20% of the connections are considered. This is a very small value and is utilized just for testing purposes.



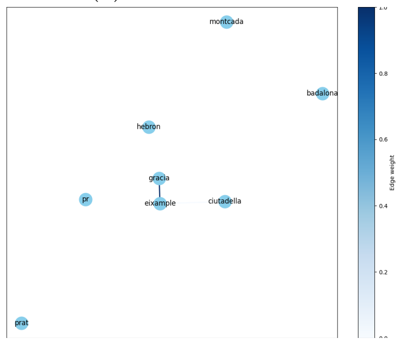
(a) All the connections



(b) 70% of connections



(c) 40% of connections



(d) 20% of connections

Figure 4: Comparison between different Threshold values for fixed θ

As expected, increasing the Threshold results in a broader network, while decreasing it results in small clusters and isolated sensors. An extremely small value, even if it allows for higher precision, invalidates the purpose of finding connections between sensors. As we don't modify θ, the weights between connections that are allowed remain the same across the different graphs.
The most appropriate value for T highly depends on the environment features, and it is not possible to assess the correctness of the value without looking at the data or knowing the specific characteristics of the landscape.

Analysis

The model is easy to implement, and the parameters offer a straightforward approach to tailor the algorithm to a specific problem. However, using only dis-

tance as a factor is very limiting in heterogeneous environments. For example, even if close in distance, the connection between a sensor placed near the sea and a sensor placed on a mountain should have a poor weight.
Although it is possible to adapt the parameters θ and the threshold—for example, by assigning a low θ value to emphasize connections only between sensors that are very close to each other (thus more likely to be in the same environment), and a high threshold to allow connections between sensors that are far apart but may be in the same climatic conditions—it remains extremely difficult to tailor these parameters to a heterogeneous environment. The weight function would require very specific tuning, adapted and customized to the specific area.

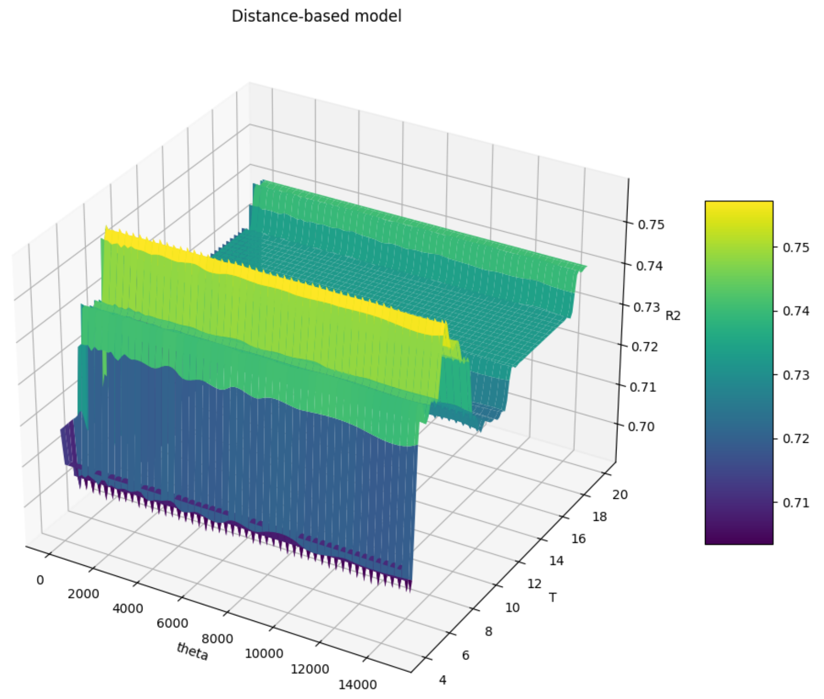


Figure 5: The threshold is the most impactful parameter in the accuracy. Best r-squared are registered for models tuned with a T value around 9.

2 GLASSO method

The Graphical Lasso (GLASSO) problem involves estimating the precision matrix $\bar{\Theta}$, formulated as follows:

$$\min_{\Theta \succeq 0} \left(\operatorname{tr}(S\Theta) - \log\det(\Theta) + \lambda \sum_{j \neq k} |\Theta_{jk}| \right)$$

where:

- Θ is the variable, representing the precision matrix.
- S is the sample covariance matrix of X.
- λ is the regularization parameter that penalizes the L1 norm of the off-diagonal entries of Θ, promoting sparsity.
- $\Theta \succeq 0$ indicates that Θ must be positive semidefinite.

Definition of the Weight Matrix:

$$W = \hat{\Theta}$$

Sparsity in the graph can be controlled by setting a threshold T. If $\hat{\Theta}_{ij} \leq T$, then $\hat{\Theta}_{ij}$ is set to zero, effectively disconnecting the link in the graph for such entries:

$$\hat{\Theta}_{ij} = 0 \quad \text{for} \quad \hat{\Theta}_{ij} \leq T$$

Increasing sparsity is interesting for many reasons, like interpretability and computational efficiency.

Assumptions

To draw reliable conclusions from the Graphical LASSO method, is necessary to test several assumptions to assess whether the data is suitable for this type of analysis. Also, before applying GLASSO, **standardization** is applied to the dataset (mean of 0 and a standard deviation of 1).

Normality

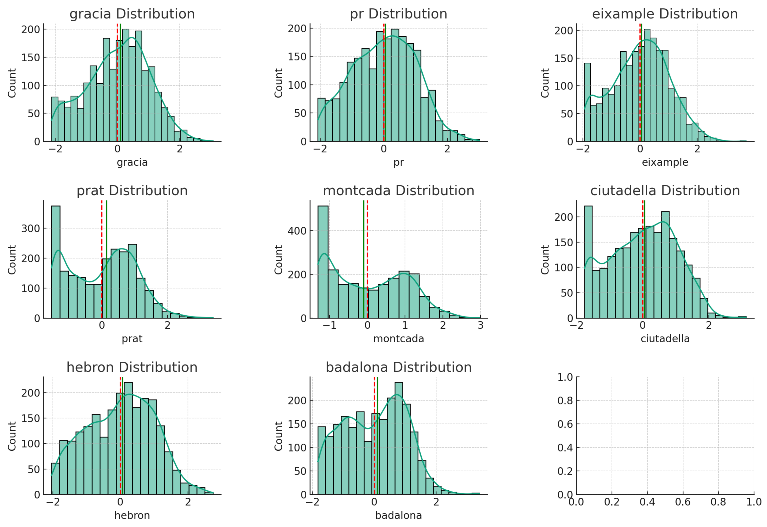


Figure 6: Variables distributions

Most variables show some degree of skewness and do not strictly follow a normal distribution. This might affect the results of GLASSO, as the normality assumption is not completely fulfilled.

Outliers

There are visible indications of potential outliers, particularly for variables like 'Prat', 'Ciutadella' or 'Montcada', where dominant frequencies of low values are observed. Especially when working with sensors, a disproportionate number of values close to 0 (in the reported graph the distributions are centered with mean 0) might indicate an incorrect data collection process, with some measurements not being registered accurately by the sensor.
The outliers not only impact the GLASSO model, but all the models.

Autocorrelation

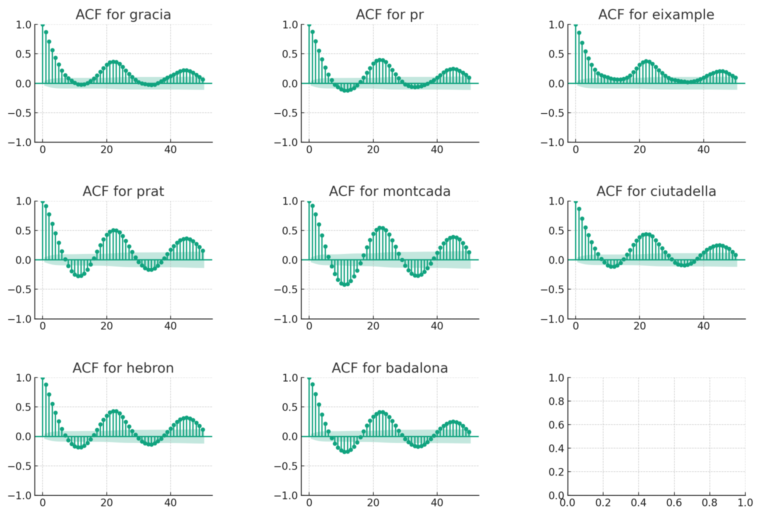


Figure 7: Autocorrelation function plots

Many variables exhibit significant autocorrelation, especially at the initial lags. This indicates that past values have a strong influence on current values, which is typical for time-series data. The presence of autocorrelation in the data means that the assumption of independent observations is violated. This can impact the validity of applying standard GLASSO directly, since the method assumes that the observations are independent.

Dealing with unfulfilled assumptions

Some transformations, like log or CoxBox, can be applied to attempt smoothing the problematic distributions. Also, some methods like Aggregation or Differencing could help solve the Autocorrelation problem. However, given the time constraints I will not attempt to smooth the dataset, and I will apply GLASSO with the current dataset for learning purposes.

Lambda

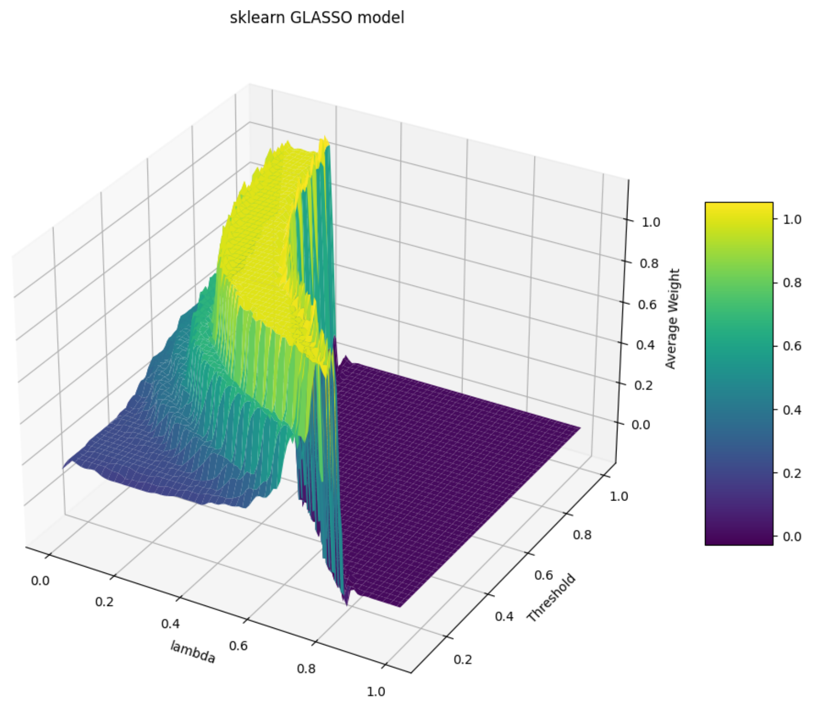


Figure 8: Average weight of edges: There is a clear ratio curve where the average weight reaches its maximum.

The parameter λ determines the level of sparsity in the estimated precision matrix.

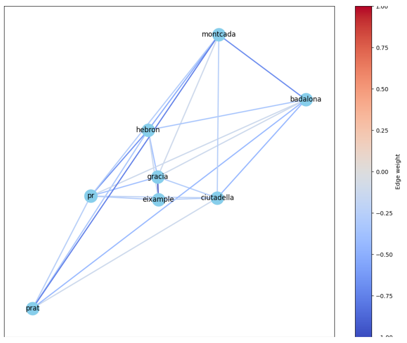
- A smaller value of λ reduces the penalty, which allows more off-diagonal elements to remain non-zero. This results in a denser graph, where even weaker relationships are included.
- A larger λ increases the penalty applied to the off-diagonal elements of the precision matrix. This leads to more zero entries in the matrix, resulting in a sparser graph. In practical terms, high λ values may result in a graph where only the strongest relationships (correlations) between variables are retained.

To test the Lambda parameter, four different λ have been assigned (Threshold is fixed at 0.1):

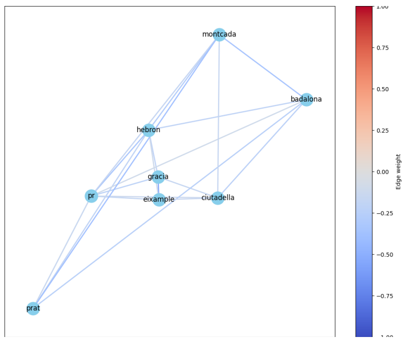
1. $\lambda = 0.2$
2. $\lambda = 0.4$

3. $\lambda = 0.6$

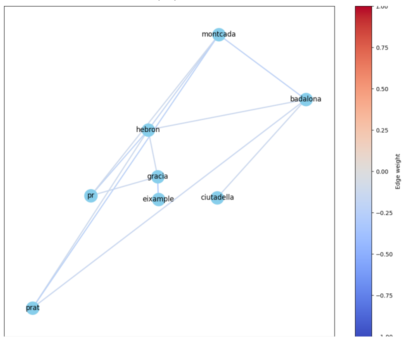
4. $\lambda = 0.8$



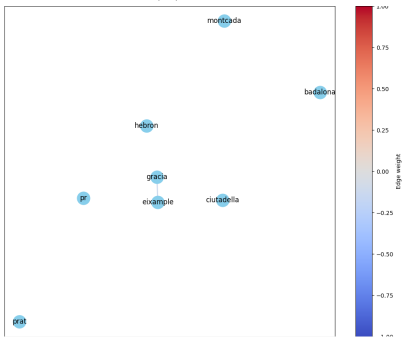
(a) $\lambda = 0.2$



(b) $\lambda = 0.4$



(c) $\lambda = 0.6$



(d) $\lambda = 0.8$

Figure 9: Comparison between different Lambdas values for fixed Threshold (T=0.1)

As expected, increasing λ results in fewer edges, as a consequence of a more sparse adjacency matrix. Also, the relationships that are considered strong for lower λ values become weak for higher λ values. Fixing the Threshold at 0.1:

- For $\lambda \geq 0.9$, no neighbor is found for any node.
- The highest changes in number of edges are observed from $\lambda = 0.5$ to $\lambda = 0.8$, going from 17 edges in the graph to just 1 edge.

Threshold

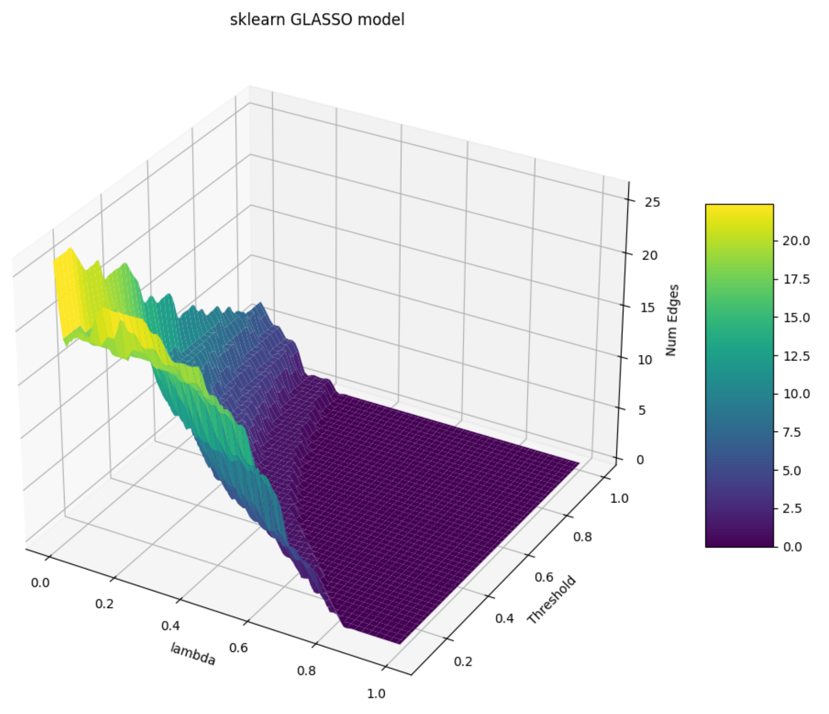


Figure 10: The number of edges is impacted by both λ and T, and it is higher for lower values of the parameters.

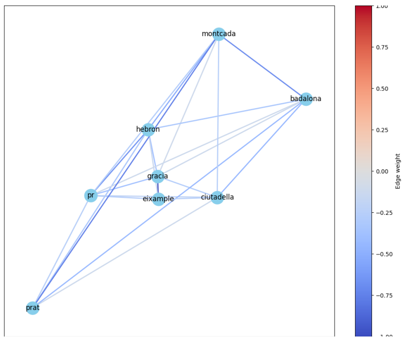
The threshold T is a parameter used post-estimation to control the sparsity of the precision matrix by setting small values to zero. It's a way to further clean up the precision matrix by removing elements that are small enough to be considered negligible, which can help in focusing on the most significant relationships between variables.

- Small values of T lead to minimal changes in the sparsity of the precision matrix. Most of the smaller values in the matrix remain non-zero, maintaining a denser network structure.
- Large values of T create a much sparser precision matrix by zeroing out a greater number of smaller coefficients. This results in fewer relationships being displayed in the graph.

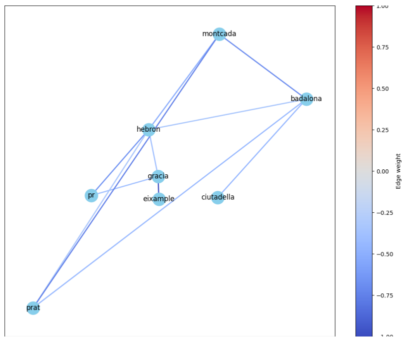
To test the Threshold parameter, four different Ts have been assigned (λ is fixed at 0.2):

1. $T = 0.1$

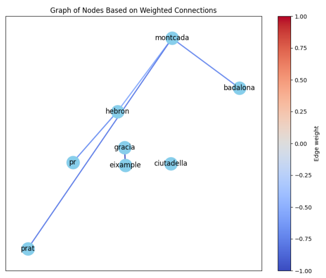
2. $T = 0.3$
3. $T = 0.5$
4. $T = 0.8$



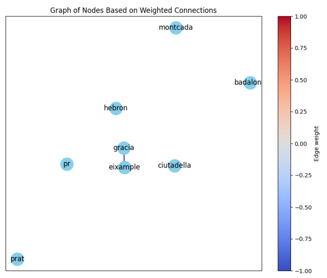
(a) $T = 0.1$



(b) $T = 0.3$



(c) $T = 0.5$



(d) $T = 0.8$

Figure 11: Comparison between different Threshold values for fixed Lambda ($\lambda = 0.2$)

As expected, increasing the Threshold reduces the number of edges in the graph. In fact, relationship with an adjacency coefficient less or equal than the Threshold are discarded. The weights of the remaining edges are constant across all the values of T, since the Threshold only impacts whether an edge should be included or not. For example, both Fig 9d and 11d only display the edge between *eixample* and *gracia*, but the relationship in the second one is stronger. This is due to how the scarcity of edges is determined in the two figures: in figure 9d, λ is the clean up factor, which also weakens the edge weights; in figure 11d, the high Threshold discards all the other edges, but does not impact the edge weights.
For extreme Threshold values (e.g. T=max(weights)), the node would have 0 edges, since no relationship weight would exceed the Threshold.

Comparison with CVXPY optimization of GLASSO problem

Instead of using the *GraphicalLasso* function from *sklearn*, the problem has also been solved with a custom solution: the optimization problem given by GLASSO has been obtained and the solution has been computed using CVXPY. When

Lambda	**Norm Diff.**	**Edges (T=0)**		**Edges (T=1e-5)**	
		Precision	**CVXPY**	**Precision**	**CVXPY**
0.1	0.00282	25	28	25	25
0.2	0.00135	27	28	27	27
0.3	0.00084	28	28	28	28
0.4	0.00062	28	28	28	28
0.5	0.00056	28	28	28	28
0.6	0.00068	27	28	27	27
0.7	0.00119	22	28	22	22
0.8	0.00172	11	28	11	11
0.9	0.00135	1	28	1	1

Table 1: Comparison between Precision Matrices obtained with *sklearn* and CVXPY

the Threshold is 0, $CXVPY$ and *GraphicalLasso* from *sklearn* produce different graph structures across several values of λ. The Frobenius norm of differences between the matrices from these methods remains low, highlighting that the numeric differences are minimal. The key factor in the discrepancies between the graphs obtained from CVXPY and *GraphicalLasso* by *sklearn* lies in the treatment of values close to zero. CVXPY tends to produce small but nonzero values, which are still classified as neighbors. In contrast, the *GraphicalLasso* method in *sklearn* may round these small values to exact zeros, leading to fewer connections in the resulting graphs. In fact, when a Threshold (T = 1e-05) is applied, the two methods return the same graphs for every λ value, even if the Threshold value is extremely low. Also, the Frobenius Norm of difference stays the same for both the Threshold values (up to 1e-05), confirming that the numerical variation deriving from the Threshold application is negligible, and the significant changes are observed only in the graph structures.
Also, the difference in the number of edges is extremely pronounced for $\lambda = 0.9$. Since the penalization factor is very high, the matrix contains many values close to zero, leading to more instances where the CVXPY implementation identifies these values as neighbors, while the *sklearn* implementation does not.

Analysis

The Graphical LASSO allows the construction of a data-driven model, which is particularly important in heterogeneous environments like Barcelona. GLASSO promotes sparsity in the adjacency matrix, highlighting only the most relevant connections between nodes; this is suitable for our objective of obtaining a graph of sensors. The hyperparameter λ is easy to tune, and the implementation is straightforward since it is possible to utilize the *GraphicalLasso* function from the *sklearn* library. Also, writing a customized function as an optimization problem with CVXPY is not extremely challenging and offers a more detailed view of the function implementation. However, GLASSO requires some statistical assumptions that are not easy to fulfill. Autocorrelation might be a problem, since in sensors measurement we often see time-series data. It also suffers from multicollinearity, so its results might not be reliable for denser graphs. [2]

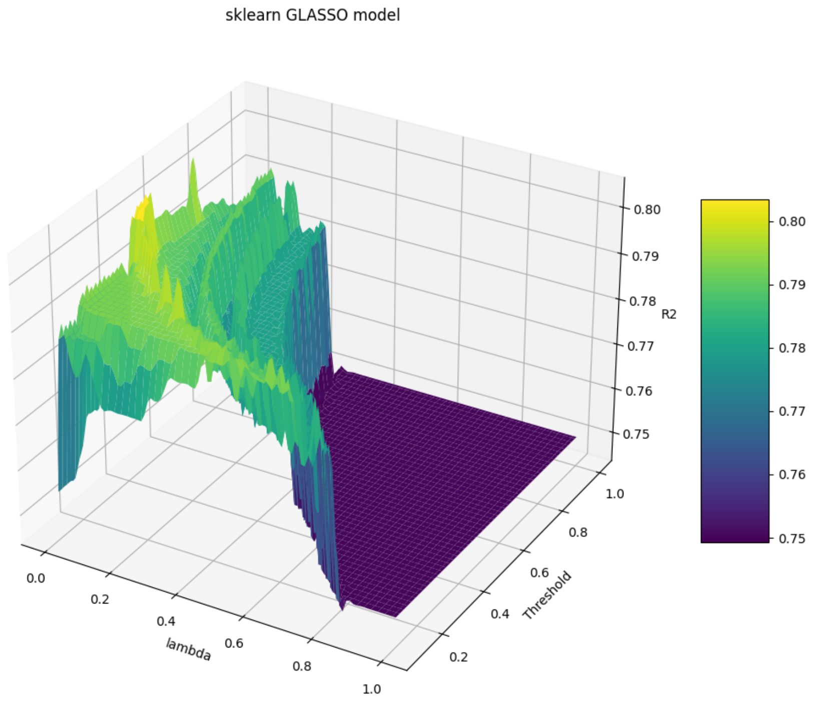


Figure 12: The GLASSO model is more accurate for lower λ values and medium Thresholds. The graph shows some clear curve, highlighting a ratio relationship between the two parameters.

3 GSP with Laplacian matrix

Another way to find a weight matrix (and consequently a graph) is to obtain the Laplacian matrix directly from a non-linear optimization model:

$$\begin{array}{ll} \underset{L,Y}{\text{minimize}} & \|X - Y\|_F^2 + \alpha \text{tr}(Y^T L Y) + \beta \|L\|_F^2 \\ \text{subject to} & \text{tr}(L) = n, \\ & L_{ij} = L_{ji} \leq 0, \quad i \neq j, \\ & L \cdot \mathbf{1} = 0 \\ \text{variable} & L, Y \end{array}$$

where:

- The term $\|X - Y\|_F^2$ represents the *data fidelity*, which measures the Frobenius norm of the difference between the data matrix X and the matrix Y. This term ensures that the matrix Y is a close approximation to the data matrix X.
- The term $\alpha \text{tr}(Y^T L Y)$ imposes *smoothness* on the values of Y. It is weighted by a factor α and uses the trace of the product $Y^T L Y$, where L is the Laplacian matrix representing the graph structure. The smoothness term ensures that connected nodes in the graph have similar values in Y.
- The term $\beta \|L\|_F^2$ enforces *sparsity* in the Laplacian matrix L. The parameter β adjusts the weight of this sparsity regularization term in the objective function. A sparser Laplacian matrix implies a graph where only the strongest relationships are retained.

This optimization problem is in general not convex. In order to solve it, I've followed the approach taken from Dong's paper[1]. Initially, Y is set = X. There

Algorithm 1 Graph Learning for Smooth Signal Representation (GL-SigRep)

```
1: Input: Input signal $X$, number of iterations $iter$, $\alpha$, $\beta$
2: Output: Output signal $Y$, graph Laplacian $L$
3: Initialization: $Y = X$
4: for $t = 1, 2, \ldots, iter$ do
5:     Step to update Graph Laplacian $L$:
6:     Solve the optimization problem of Eq. (17) to update $L$.
7:     Step to update $Y$:
8:     Solve the optimization problem of Eq. (18) to update $Y$.
9: end for
10: $L = L_{iter}, Y = Y_{iter}$.
```

are two different problems, both convex:

Problem with respect to L:

$$\begin{aligned} \min_{L} \quad & \alpha \operatorname{tr}(Y^T L Y) + \beta \|L\|_F^2, \\ \text{s.t.} \quad & \operatorname{tr}(L) = n, \\ & L_{ij} = L_{ji} \leq 0, \quad i \neq j, \\ & L \cdot 1 = 0. \end{aligned} \tag{17}$$

Problem with respect to Y:

$$\min_{Y} \|X - Y\|_F^2 + \alpha \operatorname{tr}(Y^T L Y). \tag{18}$$

At each iteration, a single step of each optimization problem is computed, and the partial solution is utilized in the other step. Differently from Dong's [1] implementation, I set my stopping criterion to be when there is no improvement from the previous step, with a tolerance of $tol = 10^{-9}$. To solve the optimization problem I have used CVXPY, specifically with the **SCS** solver, which is more suitable for large data (but might be less accurate).
In both implementations, the algorithm converges in few iterations.
At the end of the algorithm, the obtained Laplacian matrix is transformed into an Adjacency matrix:

- Each diagonal entry D_{ii} is calculated as the negative of the sum of the off-diagonal elements of the corresponding row in L:

$$D_{ii} = -\sum_{j \neq i} L_{ij}$$

- Given D, the adjacency matrix A is then computed by rearranging the original definition of L:

$$A = D - L$$

This method allows to retrieve A, thereby understanding the interconnections and weights between nodes (e.g., sensors in a network) that are not explicitly evident in the Laplacian matrix.

3.1 Alpha

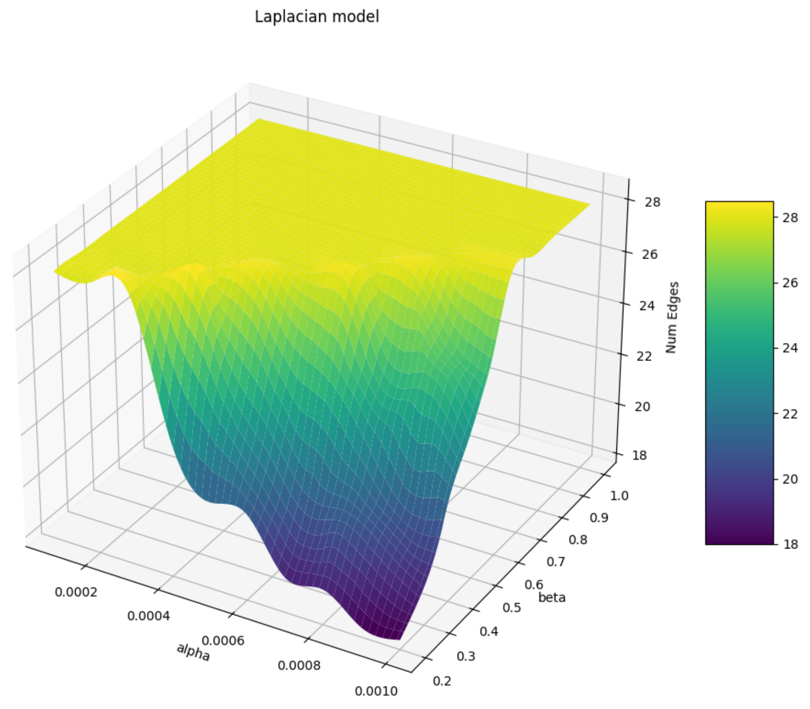


Figure 13: The number of edges in the graph is higher as α decreases and β increases.

α (alpha) multiplies the term $\mathrm{tr}(Y^T LY)$, which is a measure of smoothness or variation of features Y along the edges of the graph defined by L.

- Increasing α leads to a stronger emphasis on minimizing the trace of the quadratic term of L in the objective function. This drives the optimization towards a graph Laplacian L with fewer non-zero off-diagonal entries, effectively reducing the number of edges in the graph.

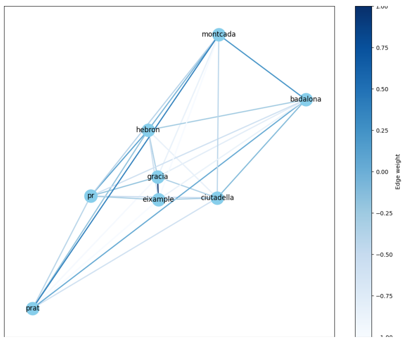
- Decreasing α relaxes the constraint on the trace of the quadratic term, potentially allowing more non-zero off-diagonal entries in L, which could increase the number of edges.

four different α have been assigned (Beta is fixed at 0.2); to draw only relevant relationships, a small Threshold is also applied ($T = 0.01$):

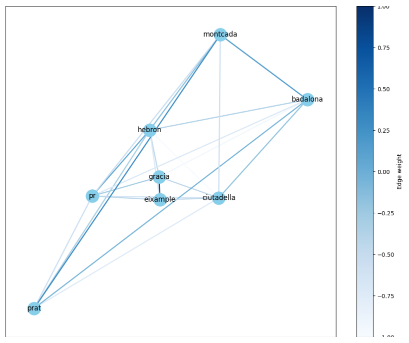
1. $\alpha = 0.0001$

2. $\alpha = 0.0004$

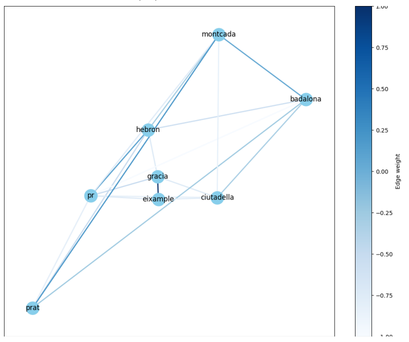
3. $\alpha = 0.0007$
4. $\alpha = 0.001$



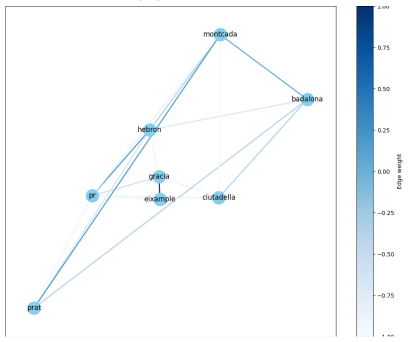
(a) $\alpha = 0.0001$



(b) $\alpha = 0.0004$



(c) $\alpha = 0.0007$



(d) $\alpha = 0.001$

Figure 14: Comparison between different Alphas values for fixed Beta ($\beta = 0.2$)

As expected, increasing α results in fewer edges, as a consequence of a more sparse adjacency matrix. Relationships with a low weight get weakened even more, but relevant bounds become even stronger. Increasing the value of α leads to a graph with fewer connections, where the strongest relationships are highlighted and the weakest relationships get even more attenuated.

3.2 Beta

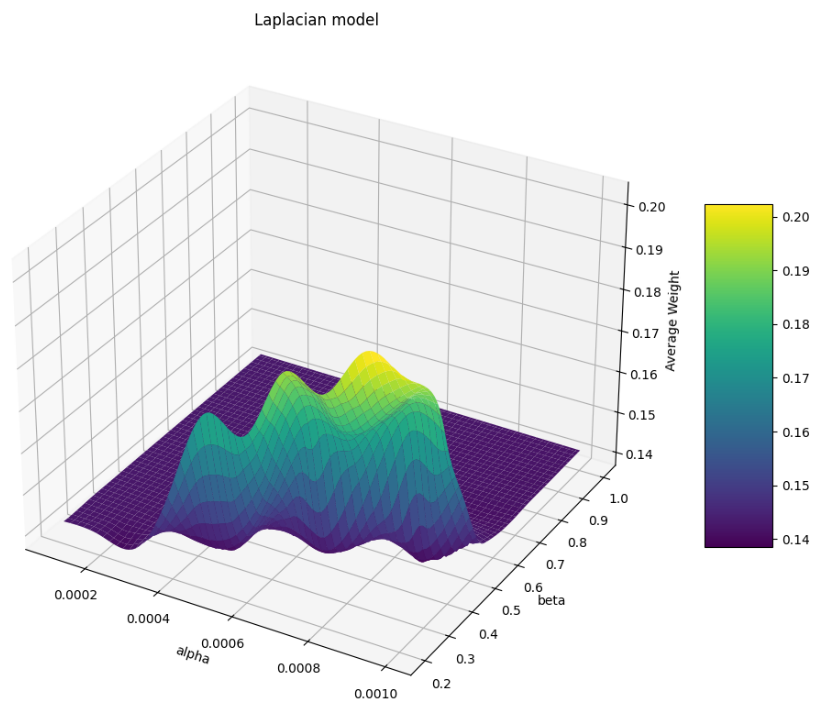


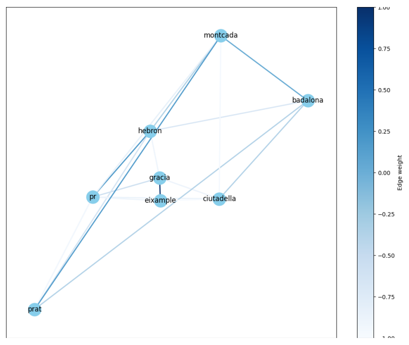
Figure 15: Higher values of α and lower values of β lead to fewer but stronger relationships between the nodes.

β (beta) influences the Frobenius norm of the Laplacian matrix L, $\|L\|_F^2$. The Frobenius norm acts as a regularization term that controls the overall values of the entries in L.

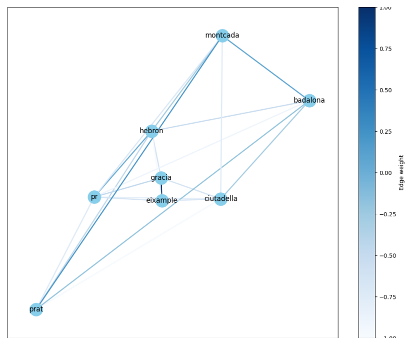
- When β increases, it minimizes the Frobenius norm of the graph Laplacian L, which is a part of the objective function in the optimization process. A smaller Frobenius norm tends to lead to a more uniform distribution of the off-diagonal entries of L but with smaller values. This increases the likelihood of more edges being included in the graph because more entries of L are different from zero.
- Decreasing β results in a less strict minimization of the Frobenius norm, which can lead to fewer, but larger values among the off-diagonal entries of L, effectively decreasing the number of edges in the graph.

four different β have been assigned (Alpha is fixed at 0.001); to draw only relevant relationships, a small Threshold is also applied ($T = 0.01$):

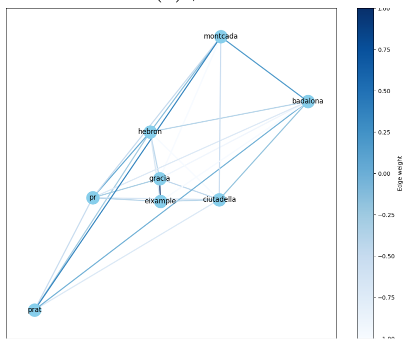
1. $\beta = 0.2$
2. $\beta = 0.4$
3. $\beta = 0.6$
4. $\beta = 0.8$



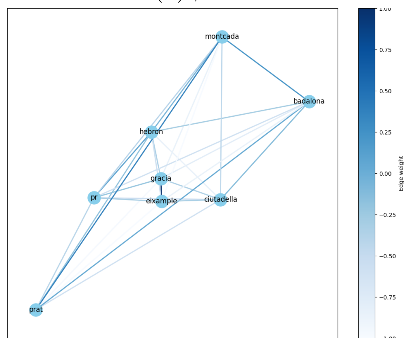
(a) $\beta = 0.2$



(b) $\beta = 0.4$



(c) $\beta = 0.6$



(d) $\beta = 0.8$

Figure 16: Comparison between different Betas values for fixed Alpha ($\alpha = 0.001$)

As expected, increasing β results in more edges, as a consequence of a less sparse adjacency matrix. Weights with a value very close to 0 get more pronounced, while weights with high values get attenuated. For example, looking at *gracia* when going from $\beta = 0.2$ to $\beta = 0.8$:

- Relationships with *montcada*, *badalona* and *prat*, that previously were considered irrelevant, are now found significant.
- Edge with *eixample*, that for $\beta = 0.2$ as a very strong relationship with a weight of 0.81, has now a weight of 0.37 with $\beta = 0.8$

Analysis

In sensor networks, changes between sensor readings are generally gradual rather than abrupt, unless an environmental incident occurs. The smoothness parameter α and the effectiveness in reconstructing signals even when data is missing from the sensors [2] make the Laplacian model a suitable choice for this type of problems. By integrating data from multiple sensors, the Laplacian model can mitigate the effects of individual sensor failures, ensuring that the measurements remain reliable and effective even when some nodes fail. Also, the relationships between nodes are data-driven, which is an important factor in a Heterogeneous environment like Barcelona.

However, the model has been quite difficult to implement, with a multi-step optimization problem that requires specific programming and particular carefulness. Also, finding the optimal parameters for the Laplacian (such as the regularization parameter) is not intuitive, and can be complex and computationally expensive. The choice of these parameters significantly affects the performance and accuracy of the model.

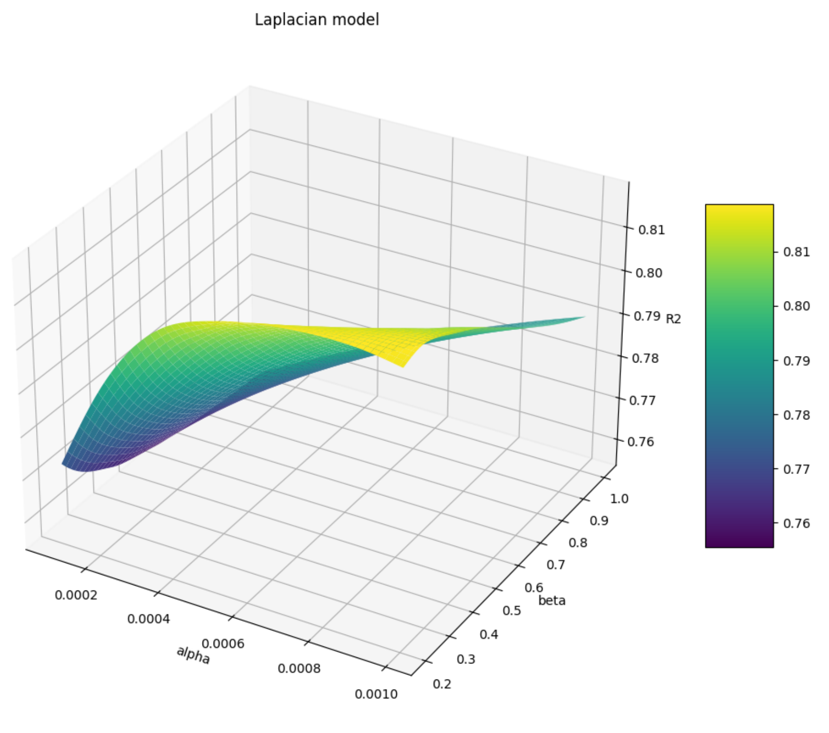


Figure 17: Better accuracy in the Laplacian model is achieved by tuning α to higher values and β to lower values

3.3 Methodology

To assess the accuracy of a model, I have tested its ability to predict the value of a sensor based on the readings from its neighboring sensors.
The impact of each neighbor is weighted based on the adjacency matrix returned by the model. In the neighborhood of a node, the node itself is excluded, so the actual node value has no effect on its predicted value (only the neighborhood of a node contributes in predicting the node value).
In cases where nodes lacked neighboring connections, I have approximated their predicted value as the average of values from all the other nodes. Reasons for this choice:

- If nodes with 0 neighbors are excluded from the analysis, models with few but strong connection would show higher accuracy, just because only the strong relationships will be analyzed, excluding the weak relationships from the analysis.
- If the entire models having a node with 0 neighbors are excluded from the analysis, models with few but strong connection would have been completely excluded, in favour of highly connected models.

I believe that averaging the values from all nodes for those with no neighbors is a fair estimation. If this method results in higher accuracy than defining specific neighbors, then the entire purpose of finding a neighbor graph is invalidated.
The predicted value is calculated as:

$$y_{\text{pred}} = Xw$$

where:

- y_{pred} is the predicted value of the node.
- X is a matrix containing only the sensor readings from the neighbors of the node.
- w is a vector of weights corresponding to each neighbor, derived from the adjacency matrix and normalized such that the sum of the weights equals 1.

Since the objective of the project is to test different models, rather than to obtain the most accurate real measurement, I have decided to exclude the node's own reading when estimating its value. This allows for a clearer vision of the impact of the neighborhood only.

4 Statistical comparison between models

To assess quantitatively which model is more accurate, I have gathered two metrics from each model:

- R-squared: The higher, the better (up to 1)
- Root mean square error (RMSE): The lower, the better (down to 0)

Also, a benchmark model where all the nodes are connected with the same weight is tested. This model can easily be obtained and doesn't need any reasoning or tuning. The Overall Average R^2 returned by the benchmark model is 0.749. The highest R^2 achieved by the distance-based model across different

Table 2: Model Performance Comparison

Feature	**Distance-based Model**	**GLASSO Model**	**Laplacian Model**
Values of the tuned model with the highest R^2 (best case scenario)			
R^2	0.853	0.875	0.882
RMSE	0.383	0.353	0.343
Parameters	$\theta = 1.023$, $T = 3.979$	$\lambda = 0.0001$, $T = 0.614$	$\alpha = 0.0008$, $\beta = 0.2$
Number of Edges	6	12	19
Best Node by Average R^2	Montcada with $R^2 = 0.805$	Montcada with $R^2 = 0.831$	Montcada with $R^2 = 0.854$
Overall Average R^2	0.732	0.766	0.788
Benchmark model: Overall Average $R^2 = 0.749$			

combinations of parameters is 0.853. Looking at the Overall Average R^2 and at Fig 5, is clear that this value is misleading. The result is returned for $\theta = 1.023$ and $T = 3.979$, but R^2 values for parameters in the same areas are significantly lower, labeling the obtained high R-squared value as an outlier. Except for peculiar parameters values, the model scores the lowest accuracy across all the three models. Even if this is a model based on distance, its most accurate node is *montcada*, which is located quite far from other nodes (see Fig 2).
This model has a lower overall accuracy than the benchmark model. One possible explanation for this anomaly could be the following: imagine you have to locate eight expensive measurement stations around Barcelona. If you are going to place two stations close to each other, it is probably because you expect to see different measurements due to environmental differences in the two areas. Given this, the closer two nodes are located, the more likely it is that you will observe different climatic conditions at the two locations. Therefore, a model

where two closer nodes are considered more correlated is not adequate to explain the placement of the sensors. This is my **personal** speculation and is not supported by any real data.
The second most accurate model is the GLASSO model, with a highest R^2 of 0.875 and an Overall Average R^2 of 0.766; it achieves a higher result that of the benchmark model, manifesting the utility of defining a correlation network between the sensors. Also, its parameters are similar to other high accuracy models (see Fig 12)
With the best R^2 of 0.882, the most accurate model is the Laplacian model. The parameter values are also in a graph area yielding accurate predictions (see Fig 17). Its Overall Average R^2 is also significantly higher than the benchmark model.
Notably, all the models identify *montcada* as the most accurate sensor station, and consistently with what has been observed before, the Laplacian model predicts the *montcada* measurements most accurately, followed by the GLASSO model and as last, the Distance-based model.

5 Conclusions

This project embarked on the task of developing a network graph of sensors across Barcelona, a city with complex and diverse microclimates. I have investigated three distinct modeling approaches: a distance-based model, the Graphical LASSO (GLASSO), and a Laplacian model, each presenting unique strengths and challenges in the context of environmental sensor networks.
The analyses revealed that a distance-based models, although straightforward in implementation, struggle with the heterogeneity of Barcelona's landscapes. The highest R^2 observed was an outlier, suggesting that simpler models may not adequately capture the environmental complexities and inter-sensor relationships necessary for accurate predictions in diverse settings.
Despite the dataset not being perfectly suited for the GLASSO model, it showed a notable improvement over the distance-based approach. Achieving a higher average R^2 and providing a robust framework for identifying relevant connections between sensors, the GLASSO model demonstrates its utility in constructing meaningful and sparse sensor networks, highlighting its adaptability even in suboptimal conditions. Its data-driven nature allows for a more nuanced understanding of sensor interdependencies, which is crucial in heterogeneous environments like Barcelona.
The Laplacian model emerged as the most effective, achieving the highest R^2 value. The smoothness parameter α and the effectiveness in reconstructing signals even when data is missing make the Laplacian model a suitable choice for sensor networks. However, the complexity of its implementation and the difficulty in tuning its parameters pose significant challenges.

References

[1] Xiaowen Dong et al. “Learning Laplacian Matrix in Smooth Graph Signal Representations”. In: (2014).

[2] Pau Ferrer-Cid, Jose M. Barcelo-Ordinas, and Jorge Garcia-Vidal. “Graph Learning Techniques Using Structured Data for IoT Air Pollution Monitoring Platforms”. In: (2021).