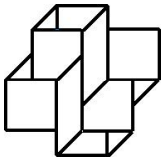


Laboratório Nacional de Computação Científica

Programa de Pós-Graduação em Modelagem Computacional

An Spatial-Temporal Aware Model Selection for Time Series Analysis

Rocío Milagros Zorrilla Coz

Petrópolis, RJ - Brasil

Janeiro de 2021

Rocío Milagros Zorrilla Coz

An Spatial-Temporal Aware Model Selection for Time Series Analysis

Thesis submetida ao corpo docente do Laboratório Nacional de Computação Científica como parte dos requisitos necessários para a obtenção do grau de Doctor em Ciências em Modelagem Computacional.

Laboratório Nacional de Computação Científica

Programa de Pós-Graduação em Modelagem Computacional

Advisor(s): Fábio André Machado Porto and Eduardo Ogasawara

Petrópolis, RJ - Brasil

Janeiro de 2021

Zorrilla Coz, Rocío Milagros
XXXX An Spatial-Temporal Aware Model Selection for Time Series Analysis / Rocío Milagros Zorrilla Coz. – Petrópolis, RJ - Brasil, Janeiro de 2021-
71 p. : il. ; 30 cm.

Advisor(s): Fábio André Machado Porto e Eduardo Ogasawara

Thesis (D.Sc.) – Laboratório Nacional de Computação Científica
Programa de Pós-Graduação em Modelagem Computacional, Janeiro de 2021.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Machado Porto, Fábio André. II. LNCC/MCTIC. III. Title

CDD: XXX.XXX

Rocío Milagros Zorrilla Coz

An Spatial-Temporal Aware Model Selection for Time Series Analysis

Thesis submetida ao corpo docente do Laboratório Nacional de Computação Científica como parte dos requisitos necessários para a obtenção do grau de Doctor em Ciências em Modelagem Computacional.

Approved by:

Prof. Fábio André Machado Porto,
(Chair of the Committee)

Prof. Agma, D. Sc.

Prof. Marta Lima de Queiroz Mattoso, D. Sc.

Prof. João Eduardo Ferreira, D. Sc.

Prof. Arthur Ziviani, Ph.D.

Petrópolis, RJ - Brasil
Janeiro de 2021

Dedication

To all....

to all fools.

Acknowledgements

Gracias totales.

"Título ou frase que serve de tema ao assunto ou para resumir o sentido ou situar a motivação da obra."
(Referência para a epígrafe)

Abstract

Segundo, o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Keywords: latex. abntex. editoração de texto.

Abstract

A Spatio–Temporal Predictive Serving System enables users to express Predictive Queries that specify: a spatio–temporal region; a predictive variable and a evaluation metric. The outcome of a predictive query presents the values of the predictive variable on the specified region computed by predictive models that maximize the evaluation metric.

In Spatio–Temporal domains, where datasets are represented by massive amounts of univariate time–series, traditional data processing and time–series analysis approaches to generate predictive models are costly in time and computational resources, although they present good predictive performance.

In this work we propose a consistent methodology for spatio–temporal query processing, which aims to reduce the computational workload and time that would be consumed if we were to train a model on each element of a spatio–temporal domain, without losing performance of the models that attend the query. The focus of our methodology begins with the domain characterization to find groups represented by an element that generalize their temporal evolution, then using temporal auto-regressive models for these representative elements we analyze their predictive power to process spatio–temporal predictive queries.

The computational experiments performed and a case study developed demonstrates the applicability of this methodology to generate predictive models in a spatio–temporal domain for query processing. We indicate that optimal predictive performance can be found when we evaluate a spatio-temporal query with a combination of shape-based domain discretization and temporal models.

Keywords: Spatio–Temporal, Univariate Time–Series, Dynamic Time Warping (DTW), Auto–Regressive models.

List of Figures

List of Tables

List of abbreviations and acronyms

ABNT	Associação Brasileira de Normas Técnicas
abnTeX	ABsurdas Normas para TeX

List of symbols

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

Contents

1 Introduction

1.1 Motivation

Recent works (GHANTA et al., 2019; CRANKSHAW et al., 2017; POLYZOTIS et al., 2018), highlight some of the challenges encountered in spatio-temporal predictive serving systems. In such systems, values of variables whose behavior vary in space-time are inferred by predictive models built over regions of the spatio-temporal domain. In this context, we are interested in the problem arising in the presence of multiple competing predictive models, i.e. models that predict the same variable but built independently over potentially different regions of the domain. This may be the case in large companies in which autonomously developed models about the same phenomenon (a.k.a. competing models) are deployed and used to answer predictive queries.

A predictive serving system enables users to express predictive queries that specify: a spatio-temporal region; a predictive variable and a evaluation metric. The outcome of a predictive query present the values of the predictive variable on the specified region computed by predictive models that maximize the evaluation metric.

Identifying the predictive models to be used by the system to answer the predictive query is, however, a hard problem. This is due to the fact that competing predictive models present varying predictive behavior through regions of the domain. This maybe due to learners intrinsic learning capabilities and/or variations on spatio-temporal regions used during the construction of independently built models. In this context, our goal is to define a procedure that selects a set of models that computes the query answer and maximizes the evaluation metric, see Figure 1.

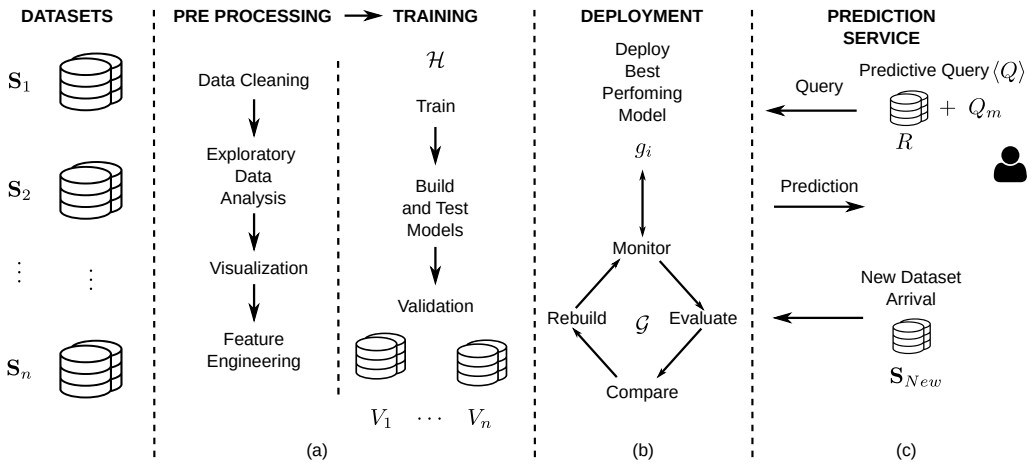


Figure 1: Predictive Serving System Models.

Our approach objective is quantify the predictive quality for auto-regressive models using a shape-based characterization of the domain. In particular, we compute the data distribution on regions of the domain where the models have been trained and compare them against those within the query spatio-temporal region. The intuition is that models behavior in query regions with similar data distribution as the ones used to train the model, shall follow the one observed during training. Conversely, in spatio-temporal regions specified by the query and that have not been used for training the model would exhibit an error computed as a function of the distance between the query non seen data region and the model building data region.

1.1.1 Problem Statement

Given a phenomenon with a spatio-temporal domain, represented by univariate time series, where independent predictive models have been trained, select a model or a composition of models to answer a spatio–temporal predictive query that maximizes an evaluation metric.

1.2 Objective and Contribution

(TODO) Our approach aims to quantify the predictive quality of models, by characterizing patterns within the predictive variable domain data. In particular, we identify some patterns on domain regions where the models have been trained, and compare them against patterns within the query spatio-temporal region. Our intuition is that, if a query region has similar data patterns as the regions used to train the model, then the behavior of the model on the query region should follow the one observed during training. Conversely, in spatio-temporal regions specified by the query, that have not been used for training, the model would exhibit an error computed as a function of the distance between the query non seen data region and the model building data region.

1.3 Thesis Outline

2 Theoretical Foundations

In this work, we are interested in processes from Spatio–Temporal domains with high volumes of data. A quality of spatio–temporal data that distinguish it from other data is the presence of dependencies among measurements induced by the spatial and temporal dimensions. The data instances are structurally related to each other in the context of space and time and show varying properties in different spatial regions and time periods (ATLURI; KARPATNE; KUMAR, 2018).

Thus, space and time introduce a variety of spatio–temporal data types and representations. They have two generic properties: (i) auto-correlation, meaning that observations made at nearby locations and time stamps are not independent but are correlated with each other, and (ii) heterogeneity, (or non-stationarity) both in space and time in varying ways and levels. These properties imply that spatial observations have consistent values at nearby locations, and show a smooth variation in temporal observations; as for the heterogeneity in space and time, it requires the learning of different models for varying spatio-temporal regions (WIKLE; ZAMMIT-MANGION; CRESSIE, 2019)

In particular, we are interested in spatio–temporal data with the representation that involves treating spatial locations as objects and using the measurements collected from a spatial location over time to define the features. This type of data are time-series and in particular we are interested in univariate time-series:

Definition 2.1 (Time Series)**.** An ordered sequence of values of a variable at equally spaced time intervals. —

Definition 2.2. Univariate time-series, a univariate time series is the simplest form of temporal data and is a sequence of real numbers collected regularly in time, where each number represents a values. —

Several applications represented by this type of data exist, and depending of the solution proposed for the problem studied, it is necessary to use specific techniques and methods. We are interested in those to identify groups of time series that show similar temporal activity and are located nearby in space, recognize some temporal patterns that commonly repeat in a number of time series. At the same time, we are interested in techniques that use time series as input features to predict a target variable, in order to generate models that can predict the value of a time series at a future time stamp using its historical values.

In Section 2.1 and 2.2 we introduce methods and techniques to find groups of time-series with high similarity and 2.3

2.1 Clustering

Clustering is a data mining technique where similar data are placed into related or homogeneous groups without advanced knowledge of the groups' definitions. In detail, clusters are formed by grouping objects that have maximum similarity with other objects within the group, and minimum similarity with objects in other groups. It is a useful approach for exploratory data analysis as it identifies structure(s) in an unlabelled dataset by objectively organizing data into similar groups. According to (AGHABOZORGI; SHIRKHORSHIDI; WAH, 2015), time series clustering can be defined as follows:

Definition 2.3. Time-series clustering, given a dataset of n time-series data $\mathcal{D} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$; the process of unsupervised partitioning of $\mathcal{D}$ into $\mathcal{C} = \{C_1, C_2, \ldots C_k\}$, in such a way that homogenous time-series are grouped together based on a certain similarity measure, is called time-series clustering. Then, C_i is called a cluster, where $\mathcal{D} = \bigcup_{i=1}^{k} C_i$ and $C_i \cap C_j \neq \emptyset$ for $i \neq j$. —

We need to consider a similarity measure to calculate the similarity among the whole time-series. This is not a simple process, time-series data are naturally noisy and include outliers and shifts, in some cases the length of time-series varies and the distance among them needs to be calculated (PAL; PRAKASH, 2017). In the literature we found several approaches to find similar time-series and we are interested in a shape–based approach, shapes of two time-series are matched as well as possible, by a non-linear stretching and contracting of the time axes (considering raw time-series data).

2.1.1 Shape-based Distance Algorithms

Shape-based distance algorithms usually employ conventional clustering methods, which are compatible with static data while their distance/similarity measure has been modified with an appropriate one for time-series. A commonly used shape–based similarity measure is the Dynamic Time Warping (DTW) (SAKOE; CHIBA, 1978) and its variants. DTW is a generalization of classical algorithms for comparing discrete sequences to sequences of continuous values, and leverages dynamic programming to calculate an optimal match between two sequences of feature vectors by allowing for stretching and compression of sections of the sequences.

Given two time series, $\mathcal{S}_1 = \{s_{11}, s_{12}, \ldots, s_{1n}\}$ and $\mathcal{S}_2 = \{s_{21}, s_{22}, \ldots, s_{2m}\}$, DTW aligns the two series so that their difference is minimized. To this end, an $n \times m$ matrix

where the (i, j) element of the matrix contains the distance $d(s_{1i}, s_{2j})$ between two points s_{1i}, and s_{2j} . The Euclidean distance is normally used.

A warping path, $W = w_1, w_2, \ldots w_k, \ldots, w_K$ where $\max(m, n) \leq K \leq m + n - 1$, is a set of matrix elements that satisfies three constraints: boundary condition, continuity, and monotonicity.

The boundary condition constraint requires the warping path to start and finish in diagonally opposite corner cells of the matrix. That is $w_1 = (1, 1)$ and $w_K = (m, n)$. The continuity constraint restricts the allowable steps to adjacent cells. The monotonicity constraint forces the points in the warping path to be monotonically spaced in time. The warping path that has the minimum distance between the two series is of interest. Mathematically,

$$d_{DTW} = \min \frac{\sum_{k=1}^{K} w_k}{K} \tag{2.1}$$

Dynamic programming can be used to effectively find this path by evaluating the following recurrence, which defines the cumulative distance as the sum of the distance of the current element and the minimum of the cumulative distances of the adjacent elements:

$$d_{cum}(i, j) = d(s_{1i}, s_2 j) + \min d_{cum}(i - 1, j - 1), d_{cum}(i - 1, j), d_{cum}(i, j - 1). \tag{2.2}$$

2.1.2 Finding Cluster Representatives

Another aspect to consider in time-series clustering the most common way to approach optimal Steiner sequence is to use cluster medoid as the prototype or representative [150]. In this approach, the centre of a cluster is defined as a sequence which minimizes the sum of squared distances to other objects within the cluster. Given time-series in a cluster, the distance of all time-series pairs within the cluster is calculated using a distance measure such as Euclidean or DTW. Then, one of the time-series in the cluster, which has lower sum of square error is defined as medoid of the cluster [151]. Moreover, if the distance is a non-elastic approach such as Euclidean, or if the centroid of the cluster can be calculated, it can be said that medoid is the nearest time-series to centroid.

A partitioning clustering method makes k groups from n unlabelled objects in the way that each group contains at least one object. One of the most used algorithms of partitioning clustering is k-Means, where each cluster has a representative which is the mean value of its objects. The main idea behind k-Means clustering is the minimization of the total distance (typically Euclidian distance) between all objects in a cluster from their cluster center (representative).

Another member of partitioning family is k-Medoids (PAM) algorithm (KAUFMAN; ROUSSEEUW, 1990), where the representative of each cluster is one of the nearest objects

to the centre of the cluster. In k-Medoids clustering algorithm, the number of clusters k, has to be pre-assigned, which is not available or feasible to be determined for many applications, so it is impractical in obtaining natural clustering results and is known as one of their drawbacks in static objects [21] and also time-series data [15]. It is even worse in time-series because the datasets are very large and diagnostic checks for determining the number of clusters is not easy. k-Means and k-Medoids algorithms make clusters which are constructed in a 'crispy' manner and it means that an object is either a member of a cluster or not. They use raw univariate time-series of equal length.

In the next Section we describe some theoretical aspects for time-series classification, another

2.2 Time Series Classification

In general, classification consists of predicting the correct class of a given data point using a labeled training set. In the case of Time Series Classification, we consider the data points as the whole time series, and the task consists of predicting its correct label. Similar to unsupervised learning, traditional classification approaches can provide a basic baseline for solving this underlying time-series classification task. However over the past two decades, research has shown that designing algorithms that can exploit the temporal information is a need in order to achieve high classification accuracy (FAWAZ et al., 2019).

Definition 2.4. A dataset $D = \{(\mathcal{S}_1, Y_1), (\mathcal{S}_2, Y_2), \ldots, (\mathcal{S}_N, Y_N)\}$ is a collection of pairs $(\mathcal{S}_i, Y_i)$ where $\mathcal{S}_i$ is univariate time series with Y_i as its corresponding one-hot label vector. For a dataset containing k classes, the one-hot label vector Y_i is a vector of length k where each element $j \in [1, k]$ is equal to 1 if the class of X_i is j and 0 otherwise. —

Using a classification method we will be able to learn from data about the membership relation between elements of the domain and representatives, and then use this learning to classify new time-series.

Convolutional Neural Networks (CNNs) come under the umbrella of Deep Learning, a subset of machine learning that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks such as object detection, speech recognition, language translation and others (). Typically, a CNN maps 2D input (images) either to categorical outputs (in case of classification problems) or to a real value (in case of regression based problems). We describe the use of 1D CNNs for time-series data. Typically, a CNN model involves two kinds of layers: Convolution and Multilayer Feed Forward.

Convolution layer, involves the convolution operation that is performed as a window of weights slides across an image, where an output pixel produced at each position is a

weighted sum of the input pixels covered by the window. The weights that parameterize the window remain the same throughout the scanning process. Therefore, convolutional layers can capture the shift-invariance of visual patterns and learn robust features.

Multilayer Feed Forward or fully connected layer is a structure where neurons from a layer is connected to all the neurons in the next layer, for a multiclass classification the dense layer is commonly used. Considering the `softmax` function onto the last layer, it creates a probability distribution over K classes, and produces an output vector of length K. Each element of the vector is the probability that the input belongs to the corresponding class. The most likely class is chosen by selecting the index of that vector having the highest probability.

Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction. A CNN is typically composed of multiple alternating convolutional and pooling layers, followed by one or several fully connected layers. CNNs exploit local correlations by enforcing a local connectivity pattern between neurons of adjacent layers, namely, the inputs of neurons in the current layer come from a subset of neurons in the previous layer. This hierarchical architecture allows convolutional neural networks to extract more and more abstract representations from the lower layer to the higher layer.

2.3 Time Series Analysis and Forecast

The basic objective of predictive learning methods is to learn a mapping from the input features (also called as independent variables) to the output variables (also called as dependent variables) using a representative training set. In spatio-temporal applications, both the input and output variables can belong to different types of ST data instances, thus resulting in a variety of predictive learning problem formulations. In the following, we discuss some of the predictive learning problems that are commonly encountered in ST applications based on the type of ST data instance used as input variables

On the other hand, a successful utilization of time series data would lead to monitoring the evolution of a phenomenon over time. The field of time series analysis aims to utilize such data for several purposes that can be categorized as: to understand and interpret the underlying forces that produce the observed state of a system or process over time; to forecast the future state of the system or process in terms of observable characteristics.

The nature of time series data includes: large in data size, high dimensionality and update continuously. Moreover time series data, which is characterized by its numerical and continuous nature, is always considered as a whole instead of individual numerical field (PAL; PRAKASH, 2017). The

- Trend: A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes we will refer to a trend as "changing direction", when it might go from an increasing trend to a decreasing trend.
- Seasonal A seasonal pattern occurs when a time series is a effected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known frequency.
- Cyclic A cycle occurs when the data exhibit rises and falls that are not of a frequency.

One objective of time series analysis is to develop models that provide plausible descriptions for time series data. For this we consider the fact that time series data are usually not independent and the analysis must take into account the time order of the observations. As a consequence, when successive observations (time series) are dependent, future values may be predicted from past observations (MILLS, 2019). Many applications are represented by stochastic time series, meaning that the future is partly determined by past values. Thus the exact predictions are impossible to determine and must be replaced by the idea that future values have a probability distribution, which is conditioned on a knowledge of past values.

There are several possible objectives in analyzing a time series, in this Section we describe methods and techniques for prediction or forecast. In particular, time-series forecasting is a form of quantitative forecasting, it can be applied when three conditions exist: (1) Information about the past is available, (2) this information can be quantified in the form of numerical data, and (3) it can be assumed that some aspects of the past pattern will continue into the future. This last condition is known as the "assumption of continuity" (MAKRIDAKIS; WHEELWRIGHT; HYNDMAN, 2008); it is an underlying premise of all quantitative and many qualitative forecasting methods. Time series forecasting treats the system as a black box and makes no attempt to discover the factors affecting its behavior. Therefore, prediction of the future is based on past values of a variable and/or past errors, but not on explanatory variables which may affect the system. The objective of such time series forecasting methods is to discover the pattern in the historical data series and extrapolate that pattern into the future.

In this work we consider Auto Regressive Integrated Moving Average (ARIMA) as the time–series forecast method to generate predictive models. Also known as the Box-Jenkins model (CHATFIELD; XING, 2019), ARIMA are the most general class of models for forecasting a time series which can be made to be "stationary" by differencing (when necessary). A random variable, that is a time series, is stationary if its statistical properties (mean and covariance) are all constant over time. A stationary series has no trend, its variations around its mean have a constant amplitude, and its autocorrelations (correlations with its own prior deviations from the mean) remain constant over time. A

random variable of this form can be viewed (as usual) as a combination of signal and noise, and the signal (if one is apparent) could be a pattern of fast or slow mean reversion, or sinusoidal oscillation, or rapid alternation in sign, and it could also have a seasonal component. An ARIMA model can be viewed as a "filter" that tries to separate the signal from the noise, and the signal is then extrapolated into the future to obtain forecasts.

The ARIMA forecasting equation for a stationary time series is a linear (i.e., regression-type) equation in which the predictors consist of lags of the dependent variable and/or lags of the forecast errors. The integrated components are useful when data has non-stationarity, and the integrated part of ARIMA helps in reducing the non-stationarity. The ARIMA applies differencing on time series one or more times to remove non-stationarity effect. The ARIMA(p, d, q) represent the order for AR, MA, and differencing components. The d component aims to detrend the signal to make it stationary and ARMA model can be applied to the de-trended dataset .

Accuracy measures, error statistics or measures, and loss functions are alternative ways of conveying information about the ability of a certain forecasting method to predict actual data, either when a model is fitted to such data, or for future periods (post-sample) whose values have not been used to develop the forecasting model (MAKRIDAKIS, 1993). To measure the accuracy of forecast methods applied to univariate time series data, s_t denote the observation at time t and $\hat{s}_t$ denote the forecast of s_t.

We consider the following scale-dependent errors:

- Mean Squared Error (MSE): Measures the average squared error of the predictions. For each point, it calculates square difference between the predictions and the target, and then average those values:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(s_j - \hat{s}_j)^2 . \tag{2.3}$$

  The higher this value, the worse the model is. It is never negative, since we're squaring the individual prediction-wise errors before summing them, but would be zero for a perfect model .

- Root Mean Squared Error (RMSE): Represents the sample standard deviation of the differences between predicted and observed values (called residuals). Mathematically, it is calculated using this formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(s_j - \hat{s}_j)^2} \tag{2.4}$$

  Often, the RMSE is preferred to the MSE as it is on the same scale as the data. Historically, the RMSE and MSE have been popular, largely because of their theoretical relevance in statistical modelling. However, they are more sensitive to outliers.

Also we consider percentage errors, they have the advantage of being unit-free, and so are frequently used to compare forecast performances between data sets. Measures based on percentage errors have the disadvantage of being infinite or undefined if $s_j = 0$ for any j in the period of interest, and having extreme values if any s_j is close to zero. Another problem with percentage errors that is often overlooked is that they assume the unit of measurement has a meaningful zero (HYNDMAN; KOEHLER, 2006). This observation led to the use of the so-called "symmetric" MAPE (sMAPE):

- Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{100}{n} \sum_{j=1}^{n} \frac{|s_j - \hat{s}_j|}{|s_j|} \tag{2.5}$$

- Symmetric Mean Absolute Error (sMAPE):

$$MAPE = \frac{200}{n} \sum_{j=1}^{n} \frac{|s_j - \hat{s}_j|}{(s_j + \hat{s}_j)} \tag{2.6}$$

2.4 Problem Formalization

Let $\mathcal{D} = \{(\mathcal{S}, x, y) : \mathcal{S} = \{s_1, s_2, \ldots, s_t\}$ is a univariate time–series, and $(x, y) \in \mathbb{R}^2\}$, represents a spatio-temporal domain. Denote by $\mathcal{G} = \{g : \mathbf{S} \to \mathbf{S}$ such that $g(\mathcal{S}) = \mathcal{S}\}$, the set of temporal predictive models for $\mathbf{S} \subseteq \mathcal{D}$. Each predictive model $g \in \mathcal{G}$ is denoted as follows:

$$g = \langle \mathbf{S}, A, \mathbf{p}, E, \Sigma \rangle, \tag{2.7}$$

where:

- $\mathbf{S}$: input dataset included in the spatio–temporal domain $\mathcal{D}$,
- A: learner or hypothesis set,
- $\mathbf{p}$: model parameters (number of time-units to obtain the generalization error (tp), number of time-units to forecast (tf)),
- E: generalization error,
- Σ: implementation/execution quality metrics.

In this context, given a predictive spatio–temporal query Q denoted as:

$$Q = \langle R, t_p, t_f, Q_m \rangle, \tag{2.8}$$

where:

- R: represents the size/shape/type of interest region,
- t_p: $\{s_{t-t_p}, s_{t-t_p+1} \ldots, s_t\}$ number of steps used for prediction,
- t_f: $\{s_{t+1}, \ldots, s_{t+t_f}\}$ number of steps to predict ($n \geq 1$),
- Q_m: represents users qualitative measurements to evaluate the predictive output.

We formally define our problem: Let the spatio–temporal domain characterization $\mathcal{D} = \bigcup_{i=1}^{n} \mathbf{S_i}$ and $\mathcal{G}$ the set of temporal models for a single element of $\mathbf{S_i}$ that accounts for the generalization error bounds for the elements in $\mathbf{S_i}$. Given $Q = \langle R, t_p, t_f, Q_m \rangle$:

$$\forall \mathcal{S} \in R : \exists g \in \mathcal{G}' \text{ such that } \arg\min_{g \in \mathcal{G}'} E_g(R). \tag{2.9}$$

In Figure 2, we show the elements involved in the formalization of our problem:

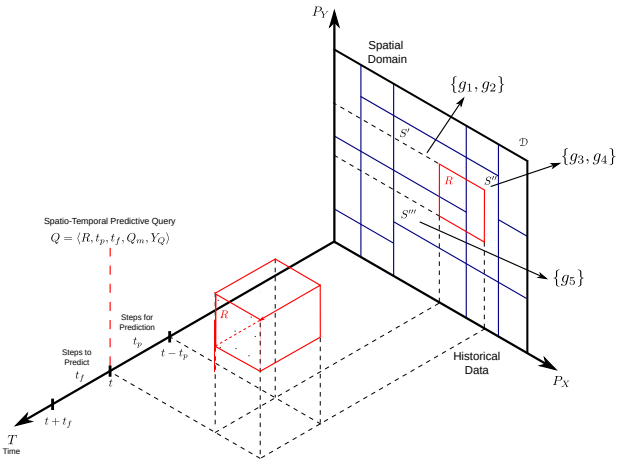


Figure 2: Predictive Spatio–Temporal Queries.

2.5 Final Considerations

3 Related Works

As described on the previous chapter, this work attends primarily to the study and analysis of model selection techniques implemented in the production of models for predictive serving systems, our emphasis are phenomena considering univariate time-series analysis.

Two main difficulties when modeling spatio–temporal data come from their size and from the complexity of the data generation process. ()

We reviewed related studies for three main topics that are spatio–temporal modeling considering univariate time–series, model selection approaches considered in predictive serving system, and how to these techniques solve spatio–temporal predictive queries.

The classification, clustering, and searching through time series have important applications in many domains. In medicine EEG and ECG readings translate to time-series data collections with billions (even trillions) of points. In fact many research hospitals have trillions of points of EEG data. Other domains where large time series data collections are routine include gesture recognition & user interface design, astronomy, robotics, geology, wildlife monitoring, security, and biometrics.

This chapter concludes by discussing works on which constitute the most related research areas regarding this work.

3.1 Spatio–Temporal Modeling – Uni-Variate Time Series Analysis

In several domains, including climate science, neuroscience, social sciences, epidemiology, transportation, criminology, and Earth sciences, space and time are the main characteristic to consider in its observations. These aspects along with the technical advantages of nowadays, allow to domain specialists collect a vast amount of data in order to analyze and study certain phenomenon with great detail ().

3.2 Processing Predictive Queries

A predictive spatio-temporal query, in particular a predictive range query has a query region R and a time t, and asks about the predictions expected to be inside R after time t based on historical data (or previous knowledge). In (AKDERE et al., 2011) the authors discusses the integration or extension of a RDBMS with a predictive component able to support data-driven predictive analytics. A predictive query and spatio-temporal

predictive query, is defined as a traditional query using a declarative language that also has a predictive capability (HENDAWI; MOKBEL, 2012).

They common uses for predictive queries: the support for predictive analytics to answer complex questions involving missing or future values, correlations, and trends, which can be used to identify opportunities or threats The predictive functionality can help build introspective services that assist in various data and resource management and optimization tasks (off-line or on-line predictive techniques). The scope of this production requires more efficient querying development to retrieve more accurate information (better results) within the shortest time frame.

3.3 Discussion

Time-Series Classification is a difficult task and the traditional ML techniques have limitations due to the time dependency in the observations. Exists traditional approaches using sequential models like LSTM or RNN, but recent studies shows that the use of Deep Learning models are good options to obtain a decent performance and accuracy (FAWAZ et al., 2019).

4 Methodology

In this work we propose a methodology focused on domain partitioning, that aims to reduce the computational workload and time that would be consumed if we were to train a model on each point of a spatio–temporal domain. We argue that, by considering a number of models generated over representative elements of the domain, which generalize the temporal dimension of other elements, it is possible to preserve a predictive quality comparable to the naive approach of using a model for every element. As a result of applying the proposed methodology, we could process a spatio–temporal predictive query effectively while maintaining a low error margin in the prediction.

The process described by the methodology is divided in two phases, off-line and on-line. The off-line phase comprises three steps: (1) the domain partitioning process, based on clustering partition techniques to find representative elements; (2) the generation of predictive models for the representatives; and (3) a model composition process to predict a subset of the domain region. The on-line phase consists of (4) a mechanism to compute a spatio–temporal predictive query. Figure 3 describes the methodology: the two main phases are represented in green, the steps for each phase in blue, the tasks performed in every step of the workflow in black and finally the corresponding expected products and outputs are shown in red.

In Sections 4.1 to 4.4, we detail the tasks performed in each step, describing the tools and methods used as well as techniques to validate their performance. The last section discusses the decisions adopted during the development of the methodology.

4.1 Step 1: Domain Partitioning

The purpose of the domain partitioning is to i) find groups with high similarity according to the temporal evolution on a spatio–temporal domain (in particular univariate time–series) and ii) find representatives for each of the aforementioned groups. We adopt clustering techniques to partition the domain by grouping elements based on a shape similarity measure. In general, a clustering algorithm can be considered as an optimization problem in which we try to minimize the total measure of dissimilarity between the elements in each group (LIAO, 2005).

We consider two methods for grouping elements that involve obtaining a representative element (which also exists on the domain) for each group. A representative is then said to generalize the temporal dimension of the other elements in the group. It is important to remember that in a clustering algorithm, the number of clusters, the size of clusters, the

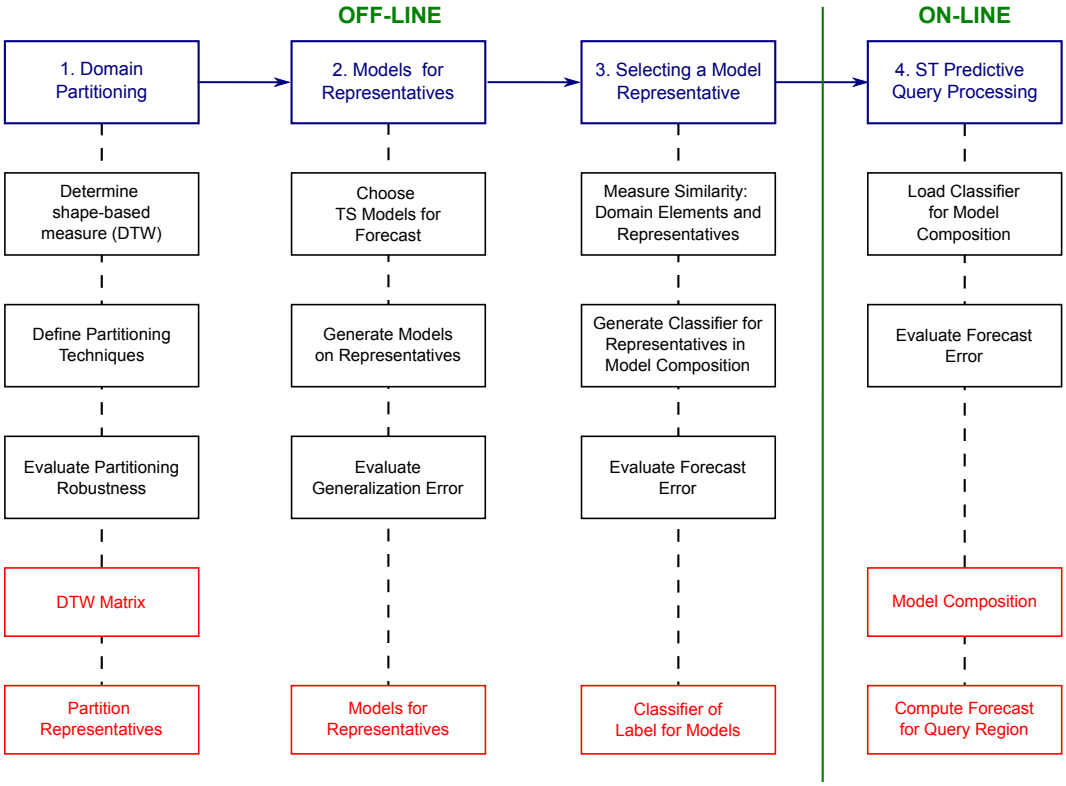


Figure 3: Workflow for the proposed methodology.

definition for outliers, and the definition of the similarity among the univariate time–series are all the concepts which depend on the problem (case study) at hand (AGHABOZORGI; SHIRKHORSHIDI; WAH, 2015).

For univariate time–series data, several clustering techniques exist (See Section 2.1). Since we are interested in leveraging the temporal behavior of the spatio–temporal domain, we adopt a shape–based similarity measure, i.e. we analyze the similarity of time–series based on the evolution of its shape over time. The similarity metric known as Dynamic Time Warping (DTW) (SAKOE; CHIBA, 1978) is based on dynamic programming, it calculates an optimal match between two sequences of feature vectors by allowing for stretching and compression of sections of the sequences.

As part of the off-line process we calculate the distance matrix that quantifies the DTW similarity between any two time series on the domain. This process can demand considerable computational power but it can be efficiently parallelized and it only needs to be executed once per domain dataset.

In this work we consider the following techniques for the domain partitioning:

- k–medoids clustering: the purpose of this technique is to group the domain elements that have a high degree of similarity. As mentioned before, we consider DTW as the similarity measure. This algorithm is a variation of the k–means algorithm and in contrast to this, k–medoids chooses existing objects within the domain instead of

computed centroids. The benefits of using k–medoids as a clustering algorithm are two-fold. In addition to finding the groups that help minimize local dissimilarity, the algorithm returns a medoid for each group. For our methodology, we can leverage the model trained on each medoid and measure the generalization error of this model when applied to other points in the corresponding cluster. A downside of k–medoids approach is that it requires the computation of the distances between many points, which can have order of time complexity $O(n^2)$. To mitigate this problem, a matrix of distances between all points in a region is calculated and saved once. This distance matrix is then reused for all k–medoids clusters for that region, thereby significantly reducing the time to compute each k–medoids partitioning. **[Fabio P.]: Talvez mereça um experimento mostrando o beneficio da matriz.**

- Regular or naive partitioning: The spatio–temporal domain is first divided into k rectangles of the same shape (allowing for irregularities when the dimensions cannot be exactly divided) and then, for each given group, we find the element (centroid) that minimizes the total sum of DTW distances to that element. This partitioning is used as a baseline in our methodology to better assess the adequacy of k–medoids. This partitioning has no parameters besides the desired number of groups.

Considering our formal representation in Section 2.4, we can express the domain partitioning for the spatio–temporal domain $\mathcal{D}$ as follows:

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathbf{S}_i, \tag{4.1}$$

where for any $i \neq j$ we have that $\mathbf{S}_i \cap \mathbf{S}_j = \emptyset$, due to the crisp clustering algorithm. And for $\mathbf{S}_i$, we admit that a $\mathcal{S}_i^* \in \mathbf{S}_i$ exists (the representative of the group) such that generalizes the temporal dependence of all the elements in $\mathbf{S}_i$.

Then we validate the performance of a time series clustering method, i.e. evaluate the goodness of clustering results. This has been recognized as one of the vital issues essential to the success of clustering applications (AGGARWAL; REDDY, 2013). Unfortunately, despite the vast amount of expert efforts spent on this issue, there is no consistent and conclusive solution to cluster validation.

4.1.1 Determining the number of groups

When using a clustering technique for high volumes of data and low variability of the data values throughout neighbor points, it is difficult to determine the optimal number of groups (k). This is particularly important for our problem, where there is low variation in the spatial data distribution of the different time-series. The k–medoids

algorithm requires the user to specify k, the number of clusters to be generated. Here, we explore some of the available methods for finding a suitable value for k.

To identify a cluster in a dataset, we try to minimize the distance between points in a cluster, and the Within-Sum-of-Squares (WSS) method, also know as the Elbow method, measures exactly that. The WSS score is the sum of the squares of the distances of all points within a cluster. Given that the WSS score converges towards zero as the number of groups k goes up to the number of points in the dataset, this method aims to choose a small value of k that still has a low SSE. This is achieved by finding the maximum numerical second derivative for each point, using its two neighboring points (we take into account that the k values are not equidistant when calculating the numerical second derivative). In practice, the elbow represents a point beyond which there are diminishing returns by increasing k (HAN; KAMBER; PEI, 2011).

The average silhouette is another method that assesses the quality of a clustering, by measuring how similar a member is to its own group compared to other groups. The silhouette index is a way to measure how close each point in a cluster is to the points in its neighboring clusters. Silhouette values lies in the range of $[-1, 1]$. A value close to $+1$ indicates that the sample is far away from its neighboring cluster and very close to the cluster to which it is assigned, meaning it is 'well-clustered'. In contrast, a value of -1 indicates that the sample is closer to a neighboring cluster than to the cluster to which it is assigned, meaning that it is 'misclassified'. A value of 0 means that it is not clear to which cluster the point should be assigned.

In this work we also consider a more refined procedure to find an optimal value for k by means of the inflection point (zero second derivative). But before calculating the second derivative, we apply a cubic smoothing spline to interpolate the values, to reduce the noise in the WSS scores. This operation will give us another suitable value for k.

Comparing partitions approaches, either with each other or with data, is a common requirement in cluster validation. For example, one might hope that different sub-samples of the same data set, or different methods applied to the data, would give similar results. This can also be considered as an aspect of robustness.

We compare these domain partitioning techniques against each other to assess the robustness of the k–medoids clustering over the regular partitioning. Since we try to optimize the cost function for k–medoids, we argue that for larger values of k the cost function for both partition techniques present a significant difference between them.

In the next Section we describe the methods adopted to built a predictive model for the representative element for every group.

4.2 Step 2: Identifying Models for Representatives

The main objective in this step is to generate predictive models for the representative elements in a domain partitioning; using these models we have the ability to compose and/or select those that correspond to a region of interest.

We generate predictive models when we apply a forecast method over a time-series in a spatio-temporal domain. In this work, we consider the following forecast methods: Auto-Regressive Integrated Moving Average (ARIMA) and k–Nearest Neighbor (kNN) models. Working with ARIMA model can offer a good trade-off between the predictive accuracy and computational resources to train and predict. The simpler k–Nearest Neighbor model is used as a baseline for comparing predictive accuracy.

In practice, two aspects to forecasting exist: the provision of a forecast for a future value (out-of-sample) of the series and the provision of a forecast error of known values of the series (in-sample). After the process of model tuning, it is possible to test the ability of the model to predict observations that the model didn't see before (as opposed to the fitted values that the model saw throughout the training process). The most common method for evaluating the forecast's success is to predict the actual values with the use of an error metric to quantify the forecast's overall accuracy (HYNDMAN; KOEHLER, 2006). The selection of a specific error metric depends on the forecast accuracy's goals. In this work we consider the following two:

- Root Mean Squared Error (RMSE): This is the root of the average squared distance of the actual and forecasted values:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\mathcal{S}_t - \hat{\mathcal{S}}_t\right)^2} \tag{4.2}$$

  Like MSE, the RMSE has a large error rate due to the squared effect and is therefore sensitive to outliers.

- Mean Absolute Percentage Error (MAPE): This measures the average percentage absolute error:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{\mathcal{S}_t - \hat{\mathcal{S}}_t}{\mathcal{S}_t}\right| \tag{4.3}$$

Recalling the formal representation in Section 2.4, once we obtain the representative element $\mathcal{S}^* \in \mathbf{S}_i$, we built $g : \mathbf{S}_i \to \mathbf{S}_i$ and we can represent $\mathbf{S}_i$ using $\mathcal{S}^*$ we denote:

$$g_i^* = \langle \mathcal{S}^*, A_p, \mathbf{p}, E, \Sigma \rangle \tag{4.4}$$

Thus, the initial approach of having a generic set of models for a domain has been refined, there can now be a reduced number of predictive models that depends on the

number of representatives considered. In the case of using a single domain partitioning scheme with k partitions to characterize a domain, there will be k instances of a particular model with $\langle A, p \rangle$ properties.

As an implementation detail of the off–line process, the predictive models are saved to persistent storage with the appropriate metadata (domain, similarity measure, domain partitioning scheme, group index). This will allow us to recover previously generated predictive models when computing a forecast value for a time-series. The result of applying a partitioning technique to a domain is also stored to speed up future computations.

4.3 Step 3: Selecting a Model Representative

For a given domain partitioning technique and its predictive models for the representatives, we define a "model composition" as the sub-set of representative models that has the ability to compute the prediction/forecast value of the elements in a region of interest on the domain. The model composition computes the RMSE of the forecast error E_{g^*} calculated by the models in the intersection $R \cap \bigcup_{i=1}^{k} \mathbf{S}_i$ on each element of R. In the following figure we represent the model composition for the region considering the medoid:

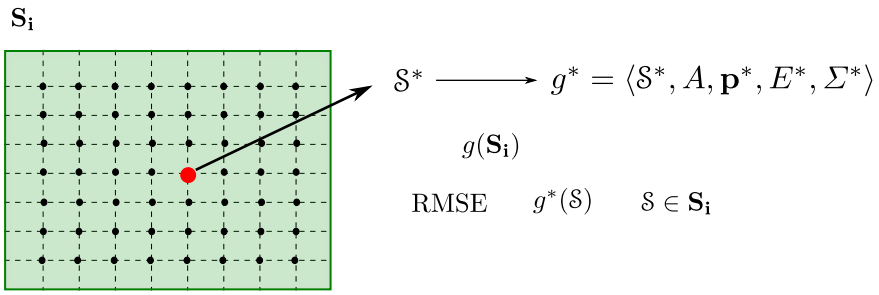


Figure 4: Model Composition using the Representative for a Region $\mathbf{S_i}$.

In order to assess the predictive quality of model for the representative element in a given domain partitioning technique, we compare the generalization error of three other model compositions, in addition to the composition that uses predictive model at the representatives. The four model compositions are described as follows:

- Model Composition with Minimal Error: for each group, we find the time series for which its predictive model, when applied to all elements of its corresponding group, minimizes the accumulated forecast error. This can be considered the 'best' choice for representatives that can only be found through an exhaustive approach.
- Model Composition with Minimal Local Error: for each time series ts_j in each group, we train its own model g_j and calculate the corresponding forecast error. We can

say that this would be the forecast error of a naive approach, because it generates as many models as there are time series in the region. This approach should offer the least accumulated forecast error but at a high computational cost, and is the scenario we want to improve on with our methodology.

- Model Composition at the Medoids: given the time series of the representatives in each group, we find the corresponding model for each representative, and use it to predict future values of the other time series in their corresponding groups. We then calculate, for each group, the accumulated forecast error when using the model in the representative for all elements in the group. This corresponds to the proposed methodology, and its forecast estimate and generalization error will be evaluated experimentally.

- Model Composition with Maximum Error: for each group, we find the time series for which its predictive model, when applied to all elements of its corresponding group, maximizes the accumulated forecast error. This can be considered the 'worst' choice for representatives. Together with the Minimal Error, it provides an interval that helps to quantify the appropriateness of the choice of representatives when using the Model Composition at the Medoids.

As discussed in the previous Section, in our problem we are not considering a unique k to analyze our methodology, because there is no specific way to optimize the value of k. In the following Section we describe the process of building a model composition that is able to leverage several given domain partitioning.

4.3.1 Preparing a Classifier for Representative Models

In the next two sections, we seek to implement a process that is able to select predictive models using some kind of intelligence, with the aim of finding a composition of models to predict a region of interest with increased accuracy. Here, it is necessary to recall that, when computing the future value of a variable for a time series, we have to consider a range of values in the immediate past. Thus, in order to create a process for selecting models, we need a mechanism that allows us to find and use relationships between the set of representatives and past temporal instances of every element in the region of interest.

We formulate this step as a univariate time series multi-class classification problem: Given an unlabeled univariate time series, assign it to one or more predefined classes. For our problem the classes are defined by the identifier of a predictive model on the representative, label composed with the value k and the group index `cluster_index` (c.i) where $0 \leq c.i < k$.

In order to show how we can leverage domain partitioning schemes and their respective representatives, we use the formal representation of the problem (see Section 2.4), with three different domain partitioning schemes as an example: Given three domain partitioning schemes for a spatio–temporal domain of sizes m, n and l and their respective representative sets, we have:

$$\begin{aligned}\mathcal{D} &= \bigcup_{i=1}^{m} \mathbf{P_i} \text{ and } \{\mathcal{P}_1^m, \ldots, \mathcal{P}_m^m\}, \\ \mathcal{D} &= \bigcup_{j=1}^{n} \mathbf{Q_j} \text{ and } \{\mathcal{Q}_1^n, \ldots, \mathcal{Q}_n^n\}, \\ \mathcal{D} &= \bigcup_{k=1}^{l} \mathbf{T_k} \text{ and } \left\{\mathcal{T}_1^l, \ldots, \mathcal{T}_l^l\right\},\end{aligned}$$

where $m+n+l < \text{card}(\mathcal{D})$. The set of representatives on the partitioning schemes considered is $\mathbf{R} = \left\{\mathcal{P}_i^m, \ldots, \mathcal{P}_m^m, \mathcal{Q}_1^n, \ldots, \mathcal{Q}_n^n, \mathcal{T}_1^l, \ldots \mathcal{T}_l^l\right\}$, and $\mathcal{G}_{(\mathbf{R})}$ is the set or their correspondent models. Now, for each element in the domain, we want to find some representative $\mathcal{S} \in \mathbf{R}$ that is the representative of some group in a partitioning scheme $\mathbf{P}$, so that the representative satisfies some optimization condition. Here, we choose the representative $\hat{S}$ whose model $g_{\hat{\mathcal{S}}} \in \mathcal{G}_{(\mathbf{R})}$ has the minimum forecast error for the corresponding time series among the other models:

$$\exists\, \hat{\mathcal{S}} \in \mathbf{R}, \text{ such that, } \underset{\substack{\hat{\mathcal{S}} \in \mathbf{R} \\ g_{\hat{\mathcal{S}}} \in \mathcal{G}_{(\mathbf{R})}}}{\arg\min}\, E(g_{\hat{\mathcal{S}}}(\mathcal{S})). \tag{4.5}$$

[Fabio P.]: na Eq. 4.5, $S' \in R$ ja aparece na antecedente da formula. In order to simplify the notation of the aforementioned relationship (4.5), we denote the model with a label `[number_of_clusters, cluster_index]`, that corresponds to the representative label of the partitioning schemes considered. Assuming that only the number of clusters is used as a parameter for partitioning schemes, this label can uniquely identify each of the representative models, so we can extract a dataset similar to Table 1.

Time–Series	Label for the Representative Model
$\mathcal{S}_1$	`[m, 1]`
$\mathcal{S}_2$	`[l, l-1]`
$\mathcal{S}_3$	`[n, n-2]`
$\mathcal{S}_4$	`[l, 2]`
$\mathcal{S}_5$	`[m, m-1]`
$\vdots$	$\vdots$

Table 1: Dataset of time-series and labels satisfying relationship (4.5) for domain partitioning sizes (m, n, l).

4.3.2 Training a Classifier for Representative Models

For the second part of the model selection process, we are interested in implementing a mechanism based on the knowledge acquired on an extracted dataset (Table 1). The objective is, given a previously unseen time-series, infer a suitable label (and thus a predictive model on a representative) for it by recognizing patterns in the time-series. This can be formulated as a time–series classification problem, which consists of training a classifier on a dataset TSD in order to map from the space of possible inputs to a probability distribution over the class variable values (labels).

With the proposed classification method, we will be able to learn from data about the membership relation between elements of the domain and representatives, and then use this learned knowledge to classify new time-series. A dataset $TSD = \{(\mathcal{S}_1, y_1), (\mathcal{S}_2, y_2), \ldots, (\mathcal{S}_N, y_N)\}$ is a collection of pairs $(\mathcal{S}_i, y_i)$ where $\mathcal{S}_i$ is univariate time series with y_i as its corresponding one-hot label vector of the labels. For a dataset containing $m + n + l$ classes, the one-hot label vector y_i is a vector of length $m + n + l$ where each element $j \in [1, (m + n + l)]$ is equal to 1 if the class of X_i is j and 0 otherwise (MITSA, 2010).

The task of time-series classification is not a simple one, considering the sequential aspect of time-series data requires the development of algorithms that are able to harness this temporal property select a class label from among a collection of candidates. While many neural network architectures can be used to solve this problem, some work better than others.

In this work we use Convolutional Neural Networks (CNNs), they typically fare very well on multiclass classification tasks, especially for images and text (FAWAZ et al., 2019). The powerful learning ability of deep CNN is primarily due to the use of multiple feature extraction stages that can automatically learn representations from the data. Research has shown that using CNNs for time series classification has several important advantages over other methods; they are highly noise-resistant models, and they are able to extract very informative, deep features, which are independent from time (SAINATH et al., 2015).

Another architecture are the Long-Short Term Memory Neural Networks, the benefit of this model is that the model can support very long input sequences. By implementing an hybrid CNN-LSTM Neural Network, we are able to integrate the capacity of process time-series as blocks or sub-sequences by the CNN model, then pieced together by the LSTM model. Then the model will further divide each sample into further sub-sequences. The CNN model will interpret each sub-sequence and the LSTM will piece together the interpretations form the sub-sequences.

4.4 Step 4: Spatio–Temporal Predictive Query Processing

In this Section we describe the on–line process implemented to compute a spatio–temporal predictive query, denoted in Equation 2.8, (see Section 2.4). We consider the following workflow in Figure 5:

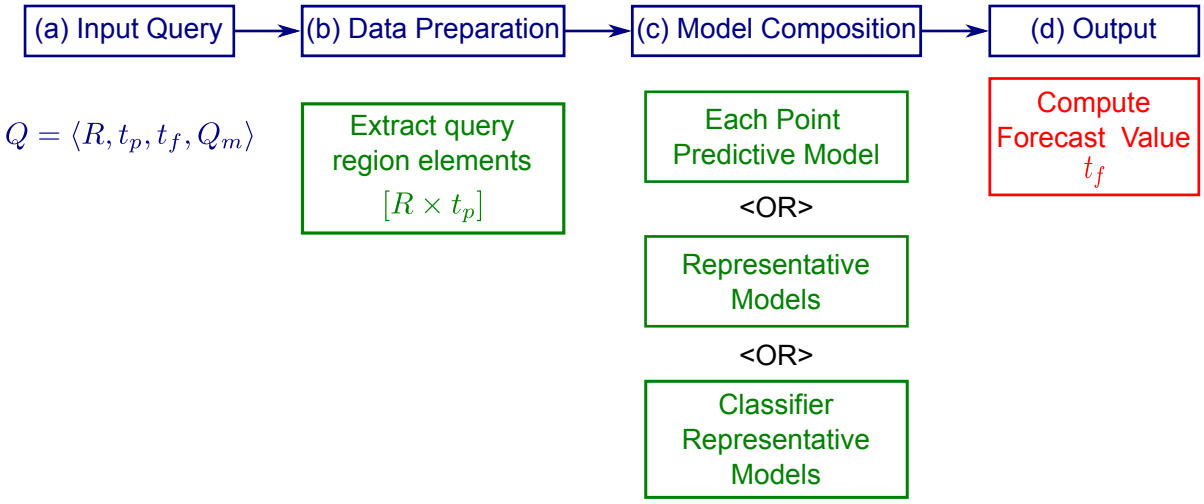


Figure 5: On-Line Processing Spatio-Temporal Predictive Query

(a) The input query is processed to extract the query region R and the number of steps t_p and t_f.

(b) A data validation and extraction from the original dataset are performed according to the input parameters. The data processing for the query consists of the extraction of instances for the query region R joint with past time units t_p, this means that for each point in R exists a univariate time–series with t_p values.

(c) When processing a spatio-temporal predictive query, initially we need to determine, for each point in the query region, which predictive models will be used. The model composition is then the resulting set of predictive models used to answer the predictive query with a forecast over the region of interest R. Here we implement three types of model composition for analysis:

- Composition of Point Predictive Models: For every point in the region of interest R we use a naive approach to train a predictive model on each point. This is based on the Model Composition with Minimal Local Error described in Section 4.3, but limited to the subset region R. To compute the forecast and associated forecast error, we simply use the model on each point.
- Composition of Representative Predictive Models: This approach uses the stored information corresponding to the domain partitioning schemes and their respective models. In this type of composition, we require that a domain

partitioning scheme (size k) is specified as input of the query. Then, we identify the representatives for every point in the query region to load the predictive models that have been previously trained at the representatives. For each point in R, the model associated to its representative will be used to compute the forecast and associated forecast error.

- Composition of Classifier for Predictive Models: A solver that calculates the result of a prediction query using the outcome of an external classifier (e.g. Neural Network). The classifier is a function that, given an input series of size t_p, a predetermined suite of partitioning schemes and a region, returns one of the representatives from any of the partitioning schemes that are available in the suite. Once the representative is determined, we follow a similar procedure as the previous model composition, also using the stored information from the domain partitioning scheme and their respective models. This is the main model composition that we wish to evaluate, while the other two are presented for comparisons.

(d) With the predictive models obtained from one of the model compositions of the previous step, the requested forecast for the t_f steps for each point in R is computed.

Additionally, if there is information available regarding the actual values of the predictive variable matching the forecast interval, the corresponding error is computed. Otherwise, the generalization error of the model is used. Finally, the errors of the points in R are combined using MSE to produce a single scalar metric for the prediction error over R.

4.5 Discussion

In this Chapter we describe tasks and techniques involved in the development for the proposed methodology. Each step uses techniques from different research areas to analyse, explore and extract knowledge from datasets, in particular for univariate time–series.

5 Implementation and Experimental Results

This Chapter is dedicated to describe the experimental validation of the methodology presented in the previous Chapter. Extensive results and analysis are presented to explore each of the steps in the proposed methodology. In particular, we focus on both the robustness of the domain characterization and in the accuracy of the results obtained from the spatio–temporal predictive queries. In each case, we employ alternative methods as baseline for comparing the performance of the proposed approach.

Several aspects of the methodology have been implemented in the form of a software called "Spatio-Temporal Tool for Time Series Analysis" or SPT-TSA. This software is the subject of the next section.

5.1 Spatio Temporal Framework for Time Series Analysis

We will describe the architectural design of SPT-TSA, a Python 3 package built to work with spatio-temporal data and predictive models. This package was developed as an implementation of a computational solution to execute the methodology described in the previous Chapter, but it can be extended to be used for other similar purposes in the field of spatio-temporal analysis.

5.1.1 SPTA-TSA Workflow

When dealing with spatio-temporal data and in particular time-series data, it is possible to find libraries and packages to perform operations for data manipulation, and for time-series analysis, particularly in the Python and R programming languages. Our software SPTA-TSA was designed to leverage widely known libraries and packages developed for Python 3, such as Pandas, Scikit-Learn, DTW, Keras, TensorFlow, Pickle, Numpy, MatplotLib, among others.

Figure 6 shows the tasks included on each step, as well as the libraries and packages responsible for the manipulation and analysis of spatio-temporal data.

5.1.2 Class Diagrams

To better describe the functionality and architectural design of SPT-TSA, we present two class diagrams, each showing a different, noteworthy aspect of the implementation. Figure 7 shows a class diagram concerning the implementation of spatio-temporal regions and operations that can be applied to them. The diagram indicates that spatio-temporal

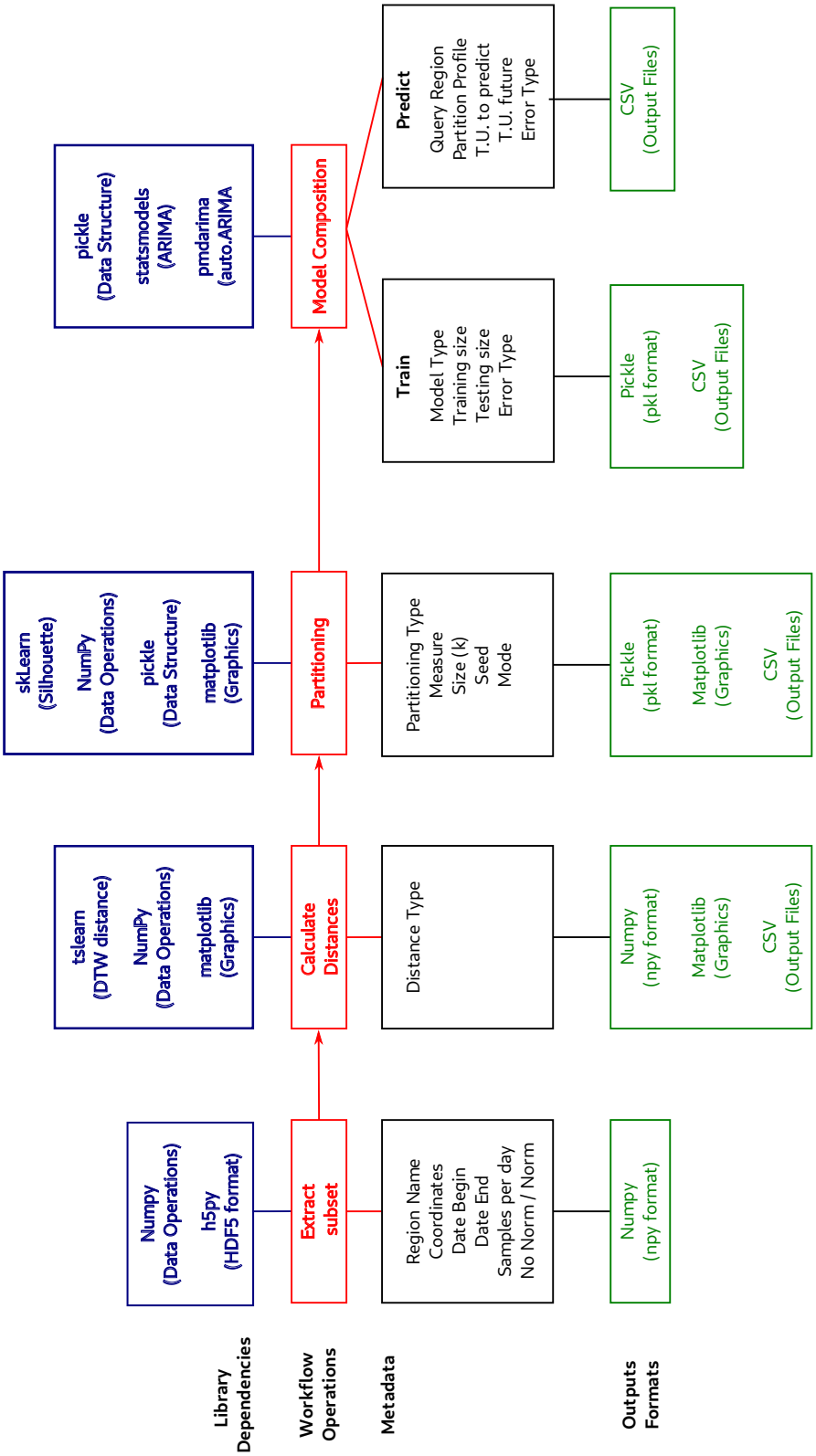


Figure 6: Steps for Dataset Transformation and Generated Products.

and spatial regions are built upon a common domain region, on which operations can be performed. The operations are special functions (hereby called function regions) that can be applied to either spatial or spatio-temporal regions, and can produce them as output. This means that, in SPT-TSA, these operations are transformations of domain regions. Additional properties of spatio-temporal regions, such as grouping from partitioning schemes and the scaling of temporal data, are achieved using decorators.

A brief description of each class is provided:

- `Point`: a simple implementation of a 2D position using a `namedtuple`.
- `BaseRegion`: The base class for all regions. It is a wrapper of a `numpy.ndarray` object, which constrains the dimensions of the region inside a 2D rectangle. In addition to providing basic functions such as saving a dataset to persistent storage, it allows the iteration of the domain points contained within the 2D boundary, using Point instances.
- `PartitionRegion`: Describes the application of a partitioning scheme to divide a spatio-temporal region into groups, by storing the membership of each point in the region to a group. Note that it is kept separate from `DomainRegion`, because it is not meant to accept function regions.
- `DomainRegion`: The parent class of both spatial and spatio-temporal regions, and also the operand and result of function regions. This abstract implementation has common functionalities for both 2D and 3D arrays, and can be used to create new spatial and spatio-temporal regions that are of a particular instance. Since function regions transform a domain region into a new region, we allow subclasses to determine which new instance to create by leveraging polymorphism. For example, a function region acting on an instance of `SpatialRegion` may produce a new `SpatialRegion` instance using the `new_sp_region` method declared in this class. However, that same function region acting on an instance of `SpatialCluster`, using the same `new_sp_region` method, will produce an instance of `SpatialCluster` instead, due to the polymorphic method call.
- `SpatialRegion`: The 2D implementation of `DomainRegion`, it stores a value in each point. Iterations of the points in the domain will yield the values for each point.
- `SpatioTemporalRegion`: The 3D implementation of `DomainRegion`, it stores an array in each point, representing a time series. Iterations of the points in the domain will yield the series for each point.
- `FunctionRegion`: A special type of `DomainRegion` used to transform `DomainRegion` instances, by accepting a `DomainRegion` instance as input and producing another

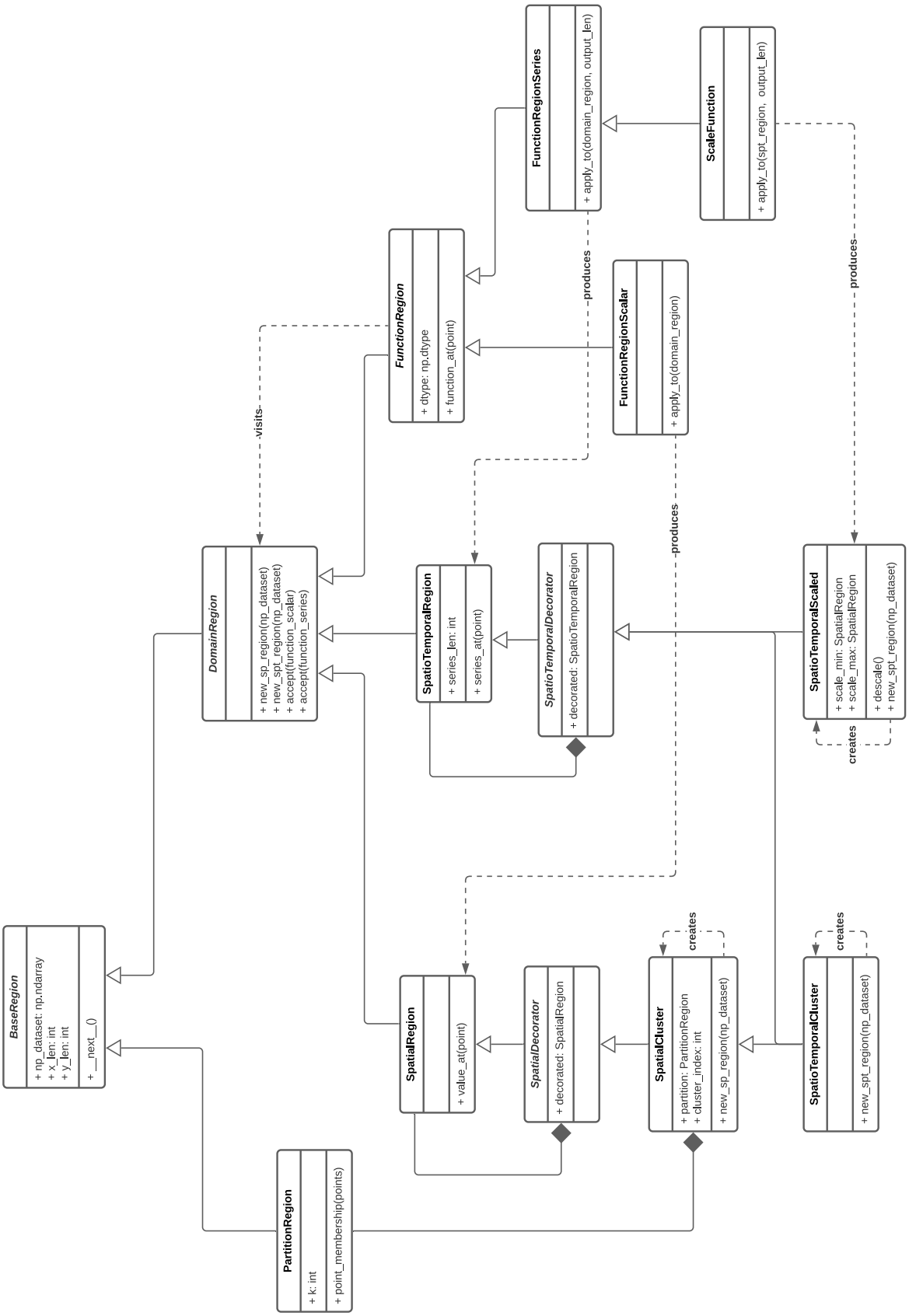


Figure 7: Class Diagram – Region Class.

`DomainRegion` instance as output, potentially of a different type. It serves as a common class for `FunctionRegionScalar` (produces a `SpatialRegion` instance as output) and `FunctionRegionSeries` (produces a `SpatioTemporalRegion` instance as output). The transformation of the input is achieved by storing a Python function at each point of the function region, and applying each function to the corresponding element in the operand region. To represent this application, consider a spatial region $S_{(m,n)}$ and a function region $F_{(m,n)}$, we then have $F(S) = \{f_{ij}(s_{ij}); (i,j) \in [0,m] \times [0,n]\}$. See `FunctionRegionScalar` for practical examples.

The implementation uses the Visitor pattern, where the `DomainRegion` is the visitor that 'visits' the function region to retrieve the corresponding function during an iteration of points. The benefit of the visitor approach is that the same `FunctionRegion` can be applied to different subclasses of `SpatialRegion` and `SpatioTemporalRegion` without knowledge of the particular subclasses, while still producing different and expected outputs. A limitation of this approach is that only one function region can be applied to a given region at the same time, otherwise the iterator of the region will be used twice and become corrupted.

- `FunctionRegionScalar`: A subclass of `FunctionRegion` that produces a subclass of `SpatialRegion` when applied to a `DomainRegion`, regardless of the input domain being spatial or spatio-temporal. The implementation will call the `new_sp_region` method of the operand polymorphically to produce the desired output.

  As a simple example, consider the problem of averaging each series in a spatio-temporal region, by using the function `numpy.mean` to the series at each point of the region. This can be achieved by creating an instance of `FunctionRegionScalar` so that each point of its internal 2D array contains the function `numpy.mean` itself, and applying this function region to the spatio-temporal region. A more complicated example involves having a special `FunctionRegionScalar` instance that applies training functions to spatio-temporal regions in order to produce, as output, a region with predictive models at each point.

- `FunctionRegionSeries`: Analogous to `FunctionRegionScalar` but will always produce a subclass of `SpatioTemporalRegion` by calling the `new_spt_region` method of the input. It requires to know the length of the output series in advance in order to allocate the output 3D array.

- `ScaleFunction`: Produces an instance of `SpatioTemporalScaled` (see below) when applied to a `SpatioTemporalRegion`.

- `SpatialDecorator`: Implementation of the Decorator pattern for a `SpatialRegion`.

- `SpatialCluster`: A decorated version of `SpatialRegion` that provides support for describing clusters obtained from a partitioning scheme. Each instance of

`SpatialCluster` represents one group only (a group index is used), so k instances would be required to represent the output of a partitioning scheme of k groups. Iterations of `SpatialCluster` will only yield the points that are members of the corresponding group, and attempts to read the value of a point outside the group result in error. This effect is achieved by keeping the same underlying data for all instances (no additional copies are made), while also using an instance of `PartitionRegion` and the corresponding group index to decorate some of the relevant methods.

- `SpatioTemporalDecorator`: Implementation of the Decorator pattern, but this time for a `SpatioTemporalRegion`.

- `SpatioTemporalCluster`: A decorated version of `SpatioTemporalRegion` that provides support for describing clusters obtained from a domain partitioning technique. This class uses multiple inheritance to also extend `SpatialCluster` to get cluster behavior, but at the same time work with a 3D array and provide a series at each point.

- `SpatioTemporalScaled`: A decorated version of `SpatioTemporalRegion` that represents the scaled version of a spatio-temporal region. The scaling process is performed by a `ScalingFunction`, which will iterate each point to scale independently the corresponding series. A series is scaled by finding its minimum and maximum values so that the new values are in the $[0, 1]$ interval. These minimum and maximum values are stored as `SpatialRegion` instances inside the scaled region, so that the original values can be restored at a later point. Since both `SpatioTemporalCluster` and `SpatioTemporalScaled` are decorators, it is possible to chain these decorators, arriving at the expected output (a cluster representing a group of scaled series at each point), even if the decorators are applied in different order. Also, restoring the scaling in that example will yield a `SpatioTemporalCluster` again.

The second class diagram is shown in Figure 8. The main focus is now the analysis of the forecast error. Here we find classes related to the training of predictive models, their forecasting of future values of the predictive variable and the subsequent calculation of the forecast error, when the actual values are available. The descriptions provided here are for the relevant ARIMA predictive models. Other predictive models used as baseline for comparison follow a similar class structure of classes.

- `ModelRegion`: Base class for the different predictive models supported by SPT-TSA. A model will produce, for each point in a spatio-temporal region, a series that represents the predicted values of the predictive variable. This is achieved by extending the behavior of `FunctionRegionSeries`, where the length of the series represents to the size of the forecast. If, for example, we pass an instance of

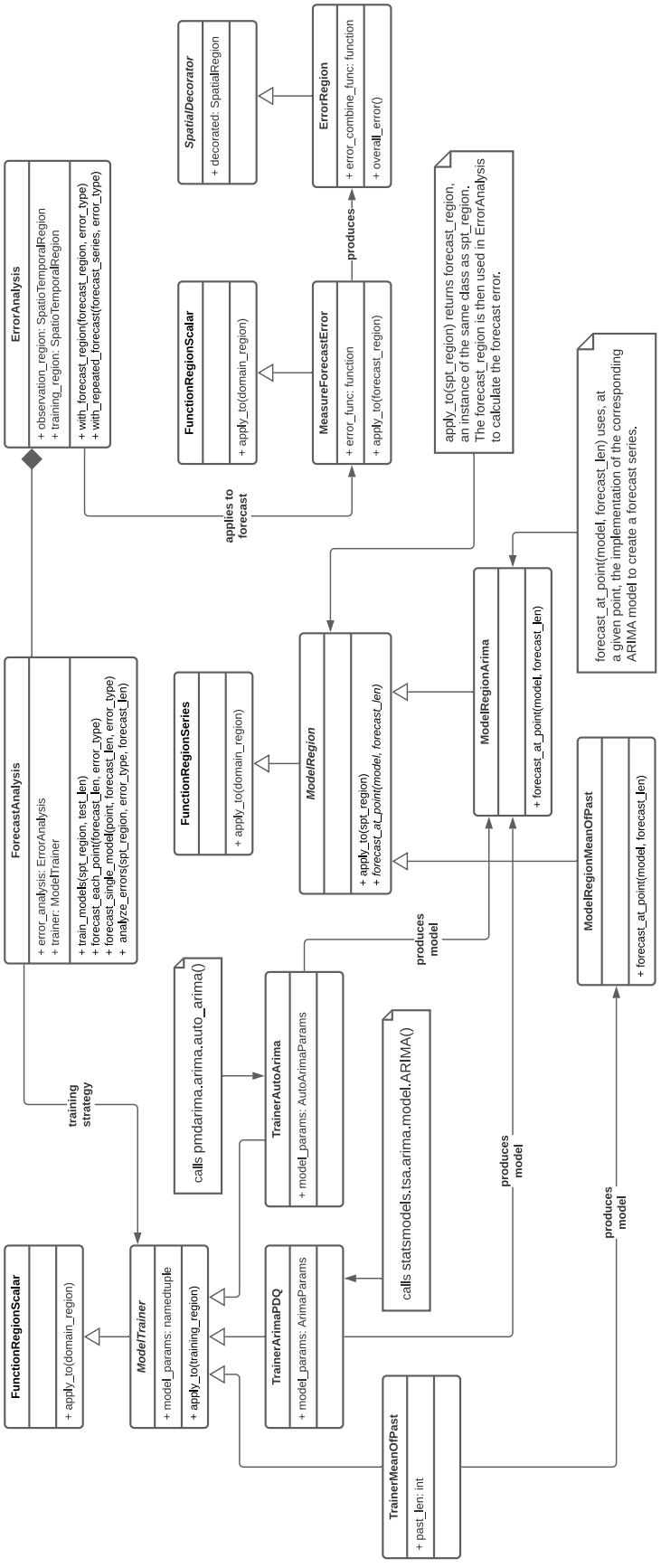


Figure 8: Class Diagram – Model Class.

`SpatioTemporalRegionScaled` as an operand, then the output is also scaled, and the original scaling information can be used to recover the desired forecast series.

- `ModelTrainer`: Base class for training predictive models supported by SPT-TSA. The models are trained by passing a training spatio-temporal region containing, for each point, a training series (same length for all points in the region). The implementation overrides the behavior of `FunctionRegionScalar` to produce an instance of `ModelRegion` instead of `SpatialRegion`. The exact class of the instance and its properties (resulting model parameters) will depend on the subclasses of `ModelTrainer` and how they process the `model_params` input.

- `TrainerArimaPDQ`: A model trainer that will use, for each point, the class `statsmodels.tsa.arima.model.ARIMA` to fit an ARIMA model using a training series and a (p, d, q) tuple. The result of applying this function to a training region is an instance of `ModelRegionArima`.

- `TrainerAutoArima`: A model trainer that will use, for each point, `pmdarima.arima.auto_arima` and a training series to determine the values for the (p, d, q) tuple that maximize the AIC metric. The trainer will then use the training function of `TrainerArimaPDQ` to fit the resulting ARIMA model.

- `ModelRegionArima`: Represents the application of ARIMA as predictive model. It is a region that has an instance of `statsmodels.tsa.arima.model.ARIMAResults` at each point (the same instance could be referenced by many points if needed). When applied to a spatio-temporal region, it will create a new spatio-temporal region that holds, for each point, its corresponding forecast series given by `statsmodels.tsa.arima.model.ARIMAResults.forecast()`.

- `ErrorRegion`: A simple decoration of `SpatialRegion` that is meant to hold, for each point, a value representing the forecast error of some model. The decoration provides some additional functions for combining the errors in each point into a single metric, for example using the RMSE.

- `MeasureForecastError`: Calculates the associated forecast errors between an observation region (an observed series in each point) and a forecast region (a forecast series in each point). It leverages the implementation of `FunctionRegionScalar` to operate in each point with a given error metric, for example RMSE. The result of applying this function region to a forecast region is an instance of `ErrorRegion`.

- `ErrorAnalysis`: A class that provides methods to calculate forecast errors using `MeasureForecastError` as a helper class. It can either appply the helper implementation directly given an entire forecast region, or it can create a forecast region by repeating a single forecast series over each point of the observation region.

This is useful when using the predictive model of a representative to create the same forecast for an entire region (`SpatioTemporalRegion`) or an entire group (`SpatioTemporalCluster`).

- `ForecastAnalysis`: Provides several strategies for training predictive models and evaluating their associated forecasts. It supports the training strategies described in the methodology, such as training a model at every point (naive approach), or training a model only at the representatives of each group. The second approach is designed so that each group is treated independently using `SpatioTemporalCluster`, so that the actual implementation remains unaware of clusters: it uses the model of a single given point (meant to be the representative) and repeats it over the entire region using point iteration (will iterate only over the members of the group). This class also provides additional methods to calculate several associated errors, including the errors obtained when the forecast of every single model is repeated independently over the entire region. This is what allows the calculation of the 'Model Composition with Minimal Local Error' and 'Model Composition with Maximum Error' described in Section 4.2, they are found by exhausting every point in the region. Since the models at every point can be analyzed independently, this calculation has been parallelized to improve efficiency.

5.2 Experiments and Results

In this section we describe the experiments and discuss the results obtained.

5.2.1 Case Study: Temperature Forecasting

The case study considered is the forecast of surface temperature, and for that we consider the CFSR (Climate Forecast System Reanalysis) dataset. It is described extensively in (SAHA et al., 01 Aug. 2010), here we provide a summary. It is a third generation reanalysis product; it consists of a high-resolution, global surface system, of the coupled ocean-atmosphere-terrestrial surface-sea ice system, covering the period from January 1979 to November 2015. The CFSR includes: (1) coupling of atmosphere and ocean during the generation of kick-off fields every 6 hours; (2) an interactive model of sea ice and; (3) assimilation of satellite radiance from the statistical interpolation scheme by grid points throughout the period. The resolution of the CFSR's global atmosphere is ~ 38 km. The global ocean is 0.25°at the equator, extending to 0.5°, in the tropics.

In the next subsection, we describe the pre-processing of the dataset and the overall preparation for our proposed methodology. Then, we show how each of the steps are applied to the use case of temperature forecasting, with the corresponding presentation and analysis of the results of each step.

5.2.2 Dataset Preparation and Computational Environment

To validate the proposed methodology, we are interested in the information about the temperature T from the original dataset, for this variable the data acquired are time-series with 4 readings per day (every 6 hours) for each geographical coordinate $(x, y) \in ([278, 348], [10, -60])$, with a resolution of ~ 38 km or $0.5°$. This spatial subset comprises the Brazilian territory, and has been studied before in (SOUTO et al., 2018). Additionally, we restrict our use case to the last year collected. Thus, we can denote the dataset for our region of interest as $\mathcal{D} = [365 \times 4, 285.5 : 330, 7.5 : -37]$ (there are 4 daily measurements with 6 hour interval during one year, the spatial coordinates denote the Brazilian territory).

In total there are 8100 time-series representing the temperature variation for one year. Each time-series in the dataset consists of 1260 values, four readings per day. To reduce daily noise, we calculate the mean of each four daily values to produce a single average daily value; this results in 365 time-units.

To further pre-process the data, all the time-series are scaled so the values are bounded in the $[0, 1]$ interval. This scaling process is done independently for each point: we find the minimum and maximum values, then apply a simple scaling function $t_{scaled} = \frac{t - t_{min}}{t_{max} - t_{min}}$. There are many reasons to apply this scaling function, most importantly is to allow for offsets when comparing similar temporal trends. This will be specially useful when using a predictor to estimate the future values of a predictive variable: this way the same predictor can produce different predicted values for each point in the region after applying the inverse scaling function corresponding to each point, even though all the predictions come from the same scaled output.

5.2.3 Calculating the DTW matrix

In Section 4.1, we argue for the choice of DTW as the shape-based similarity measure for the partitioning schemes. Here, we describe the computation of the DTW matrix for our entire region of interest, as part of the off-line process. This compute-intensive process needs to be performed only once for each region of interest (bounded not only by the Brazilian territory but also by the chosen temporal slice). It can be performed in parallel using a multi-core machine with satisfactory speedup gain. To achieve parallelism, we leveraged the *multiprocessing* library from Python to split the tasks of calculating the DTW for each pair points. To optimize speedup, the dataset was loaded into shared readable memory between processes, while allowing a shared writeable memory for writing the DTW distance. The time used to compute the DTW matrix for 8100 univariate time-series with 365 temperature readings was approximately 90000 seconds (25 hours).

Once the DTW matrix is computed, it is saved to persistent storage for later retrieval.

Afterwards, we are ready to apply a clustering algorithm to find groups based on the measure similarity. The retrieval of distance values between points is simplified to finding rows or columns in the DTW matrix, thereby significantly reducing the computational cost of algorithms such as k-medoids.

Here, we provide a couple of examples that highlight the need for computing the DTW matrix for efficiency, by comparing execution times on the same reference hardware. Running our implementation of k–medoids without the DTW matrix involves calculating pair-wise DTW distances between candidate medoids and other points until the algorithm converges. Let's first consider a subset of the spatio-temporal region described in the previous section, with only 70 time-series (7×10 spatial region). If the DTW matrix is used, running k–medoids with $k = 8$ on this smaller region takes less than one second to execute three iterations with a total of 25 swaps in the candidate medoids; as opposed to 292 seconds when the DTW matrix is not used. The difference is more dramatic when running k–medoids with $k = 24$ over the entire region of interest with 8100 time-series: with the DTW matrix, it takes 72.5 seconds to execute six iterations with a total of 271 swaps; without the DTW matrix only the first swap took over 4400 seconds. We need not only to execute $k-$-medoids once but many times with different partitioning schemes, this would be unfeasible without the DTW matrix calculated beforehand.

In Figure 9, we can observe the variations in the DTW distances for the Brazilian region, using two plots. The DTW distances are actually calculated as a $[x\_len * y\_len, x\_len * y\_len]$ symmetric matrix. Here we use two plots to highlight the distances between the point $(0, 0)$ and all other points (left image), and between the center of the domain and all other points (right image). For the first plot, we retrieve the first row of the DTW matrix, and arrange the data to fit into a 2D grid that represents the region of interest using indices that match the spatial location of the time series. The distances are represented using colors obtained by applying the default `viridis` colormap from the `matplotlib` Python package. With this colormap, darker colors represent lower values of the DTW distance (black means 0, which is the distance of a point with itself), and the maximum value of the DTW distance is displayed as a bright yellow. These maximum values were 3.19 (left) and 3.80 (right) for our dataset with the scaling of the time-series.

5.2.4 Analyzing the Domain Partitioning

In this Section we describe the experiments performed to validate the strategies adopted for the domain partitioning. We consider two partitioning techniques: k–medoids based and regular partitioning; the first groups elements of the domain according to the DTW similarity measure, whereas the second is a partitioning technique based on the geometry of the domain (see Section 4.1 for details). In both approaches, we consider the existence of representative elements for each group in the partition, medoid and centroid

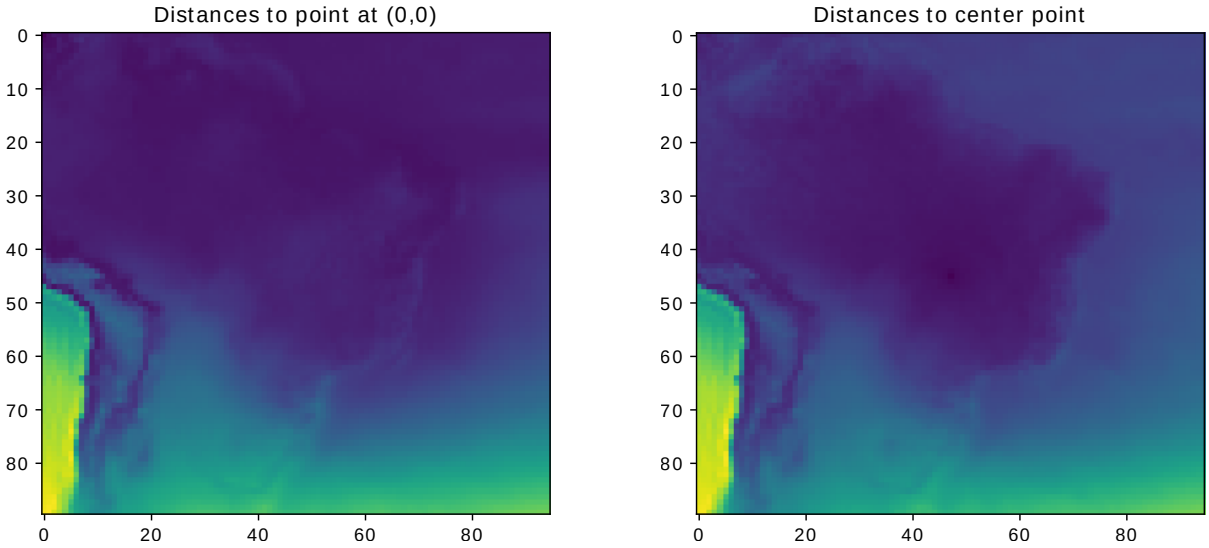


Figure 9: DTW Distance of the time-series in the Brazilian region: axes represent spatial indices of the time series, color represents the value of the distance.

respectively.

A hypothesis we want to highlight is that it should be preferable to use a partitioning technique that considers grouping domain elements based on the similarity of their temporal evolution, rather than creating groups just according to a regular division of the domain geometry. To verify this, we compare the intra-cluster sum of both partitioning techniques for several values of k.

Another important aspect to evaluate in k–medoids is the choice of the number of partitions (k), for this we use three approaches: elbow method, silhouette index and the *point with minimum curvature* inflection point using a smooth spline (see Section 4.1.1 for a description of these methods).

5.2.4.1 Evaluating Partitioning Quality

For the two partition techniques, k–medoids based and regular partitioning, we vary the number of groups from 2 to 150. As mentioned in Section 4.1, the clustering algorithm k–medoids involves a similarity measure. We are using the DTW distance, due to its much improved alignment based on shape. And the regular partitioning is a method based on the domain geometry, i.e. divide in regions of the same size and shape.

The k–medoids approach is a seed based algorithm, this means that the performance is dependent on initial cluster center selection and the optimal number of clusters. The convergence of this algorithm primarily depends on the initialization phase. It produces different clustering results for different values of k. In addition, increasing k without penalty will always reduce the error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster (i.e., when k equals the number of

data points) (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

In Table 2, we show the Total Sum of Intra-Cluster distances for both domain partitioning techniques, with k vaying from 2 to 150, graphically represented in Figure 10.

k	k–Medoids	Regular	k	k–Medoids	Regular	k	k–Medoids	Regular
2	12468.405	13190.136	52	8645.736	10032.124	102	7796.861	8855.428
4	11677.377	13115.736	54	8606.070	9485.886	104	7772.689	8706.568
6	11218.553	12546.427	56	8566.783	9480.166	106	7752.124	12067.876
8	10861.431	12517.237	58	8518.216	11466.201	108	7720.876	8404.685
10	10556.589	12144.754	60	8470.036	9360.978	110	7705.371	8359.722
12	10320.748	11537.237	62	8432.220	11922.456	112	7676.569	8470.927
14	10162.890	12038.902	64	8391.178	9238.163	114	7657.374	9012.854
16	10016.352	11185.784	66	8352.620	9222.670	116	7631.154	9513.241
18	9874.266	11011.460	68	8315.007	9769.658	118	7612.637	11943.777
20	9755.024	10864.368	70	8278.471	9148.631	120	7587.633	8276.233
22	9623.499	11663.244	72	8232.783	9011.589	122	7570.840	11900.903
24	9539.343	10613.759	74	8203.346	11645.734	124	7547.208	10152.428
26	9445.446	11741.562	76	8169.848	9864.922	126	7526.430	8218.706
28	9358.812	10467.305	78	8130.028	9243.270	128	7510.721	8428.220
30	9279.477	10250.783	80	8101.370	8879.863	130	7485.228	8341.508
32	9197.485	10305.462	82	8069.011	11478.049	132	7465.314	8149.479
34	9135.674	11595.505	84	8039.939	8947.657	134	7447.742	11768.667
36	9083.476	10081.895	86	8014.447	11439.713	136	7431.451	8247.420
38	9018.006	11685.706	88	7985.764	8719.332	138	7406.728	9136.191
40	8969.207	9903.558	90	7955.133	8654.838	140	7390.761	8083.724
42	8902.609	9865.422	92	7922.941	10004.969	142	7366.944	11675.846
44	8856.496	9990.224	94	7906.157	12251.765	144	7352.245	8126.600
46	8795.638	11808.071	96	7884.088	8642.817	146	7333.966	11621.432
48	8740.765	9653.982	98	7855.217	8773.346	148	7311.080	9715.298
50	8698.159	9605.287	100	7822.593	8525.302	150	7293.661	7932.553

Table 2: Total Sum of Intra Cluster Distances of k–Medoids and Regular Partitioning Techniques for $k = \{2, \ldots, 150\}$.

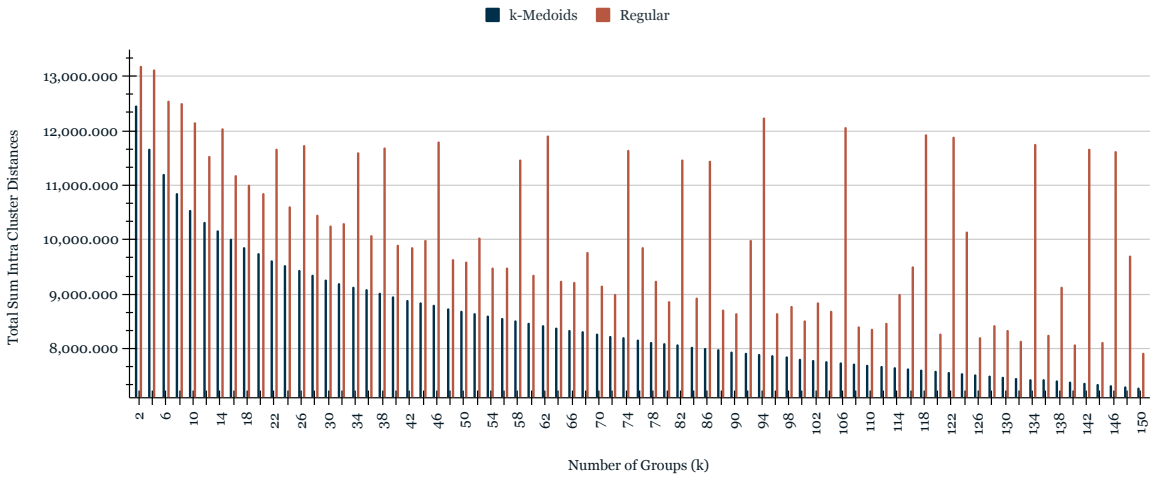


Figure 10: Total Sum of Intra Cluster Distances of k–Medoids and Regular Partitioning Techniques.

5.2.4.2 Selecting k

The performance of the k–Medoids algorithm is dependent on the optimal value of k. The automatic detection of k is very challenging for time-series data set. The choice

of k is often ambiguous, with interpretations depending on the shape and scale of the distribution of the time–series in a dataset (LIAO, 2005). An optimal choice of k should strike a balance between maximizing the compression of the data using a single cluster, while maximizing the accuracy when assigning each data point to its own cluster.

One common method of choosing the appropriate cluster solution is to compare the Within-Sum-of-Squares (WSS) for a number of cluster solutions. Thus, WSS can be seen as a global measure of error. In general, as the number of clusters increases, the WSS should decrease because clusters are, by definition, smaller. A plot of the WSS against a series of sequential cluster levels can provide a useful graphical way to choose an appropriate cluster level. An appropriate cluster solution could be defined as the solution at which the reduction in WSS slows dramatically. This produces an "elbow" in the plot of WSS against cluster solutions. For our dataset we find an "elbow" at the 4 cluster solution suggesting that solutions > 4 do not have a substantial impact on the total SSE, Figure 11 shows this result.

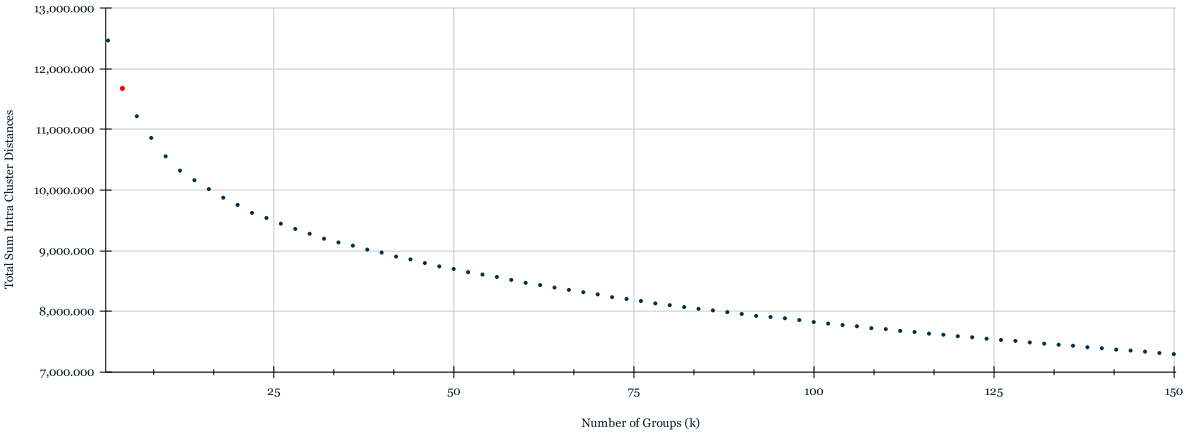


Figure 11: Elbow Method to find the optimal k for the k–Medoids Approach.

When performing a Silhouette Analysis, a way to measure how close each point in a cluster is to the points in its neighboring clusters. We expect to find the optimum value for k during k-means clustering. Silhouette values lies in the range of $[-1, 1]$. A value of $+1$ indicates that the sample is far away from its neighboring cluster and very close to the cluster its assigned. Similarly, value of -1 indicates that the point is close to its neighboring cluster than to the cluster its assigned. And, a value of 0 means its at the boundary of the distance between the cluster. Value of $+1$ is ideal and -1 is least preferred. Hence, higher the value better is the cluster configuration. For our dataset we perform the silhouette analysis giving as a result the index value 0.145 for $k = 8$.

An additional method to find the optimal value for k, based on the inflection point, consists in fitting the values of total sum of intra cluster distances using a cubic smooth spline. Splines are a smooth and flexible way of fitting Non linear Models and learning the Non linear interactions from the data. We are interested in finding the point where

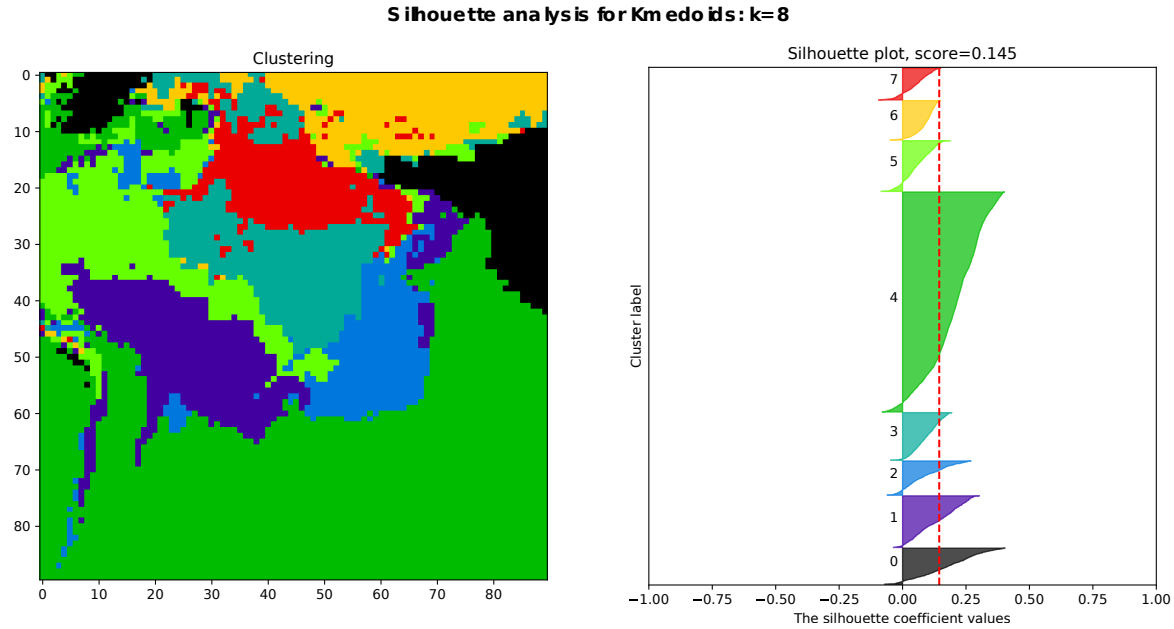


Figure 12: Silhouette Index for $k = 8$.

the curvature of the fitted model is the maximum, for our dataset the computed value is $k = 66$.(See Figure 13)

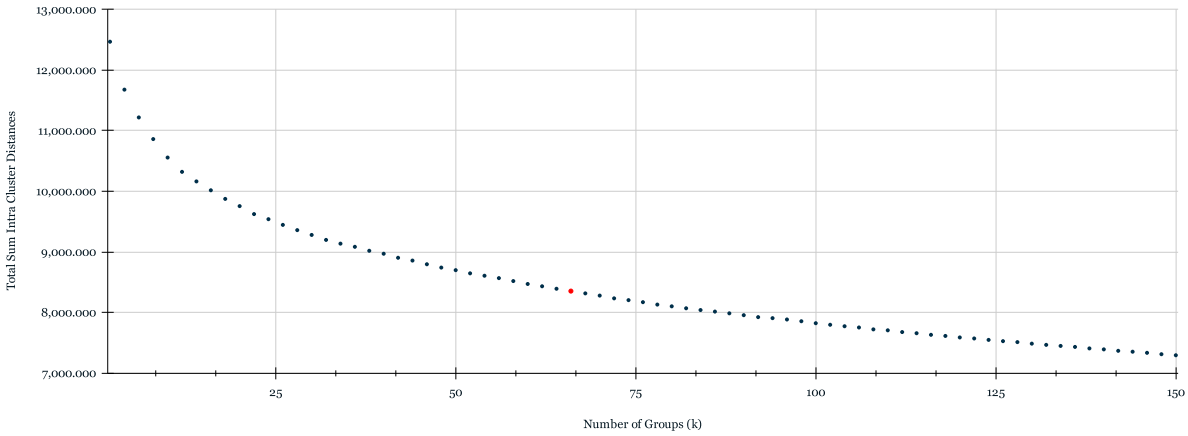


Figure 13: Smooth Spline Fitting Method to find the optimal k for the k–Medoids Approach.

We can observe that several methods to find an optimal value for k give us different values. Figure 14 shows the resulting groups when applying two of these partitioning techniques; the marks indicate the representative points of each group. The following table is a summary of the optimal values for k found in each method:

Method	Optimal k
Elbow	4
Silhouette	8
Smooth Spline for SSE	66

Table 3: Methods to find the optimal value for k.

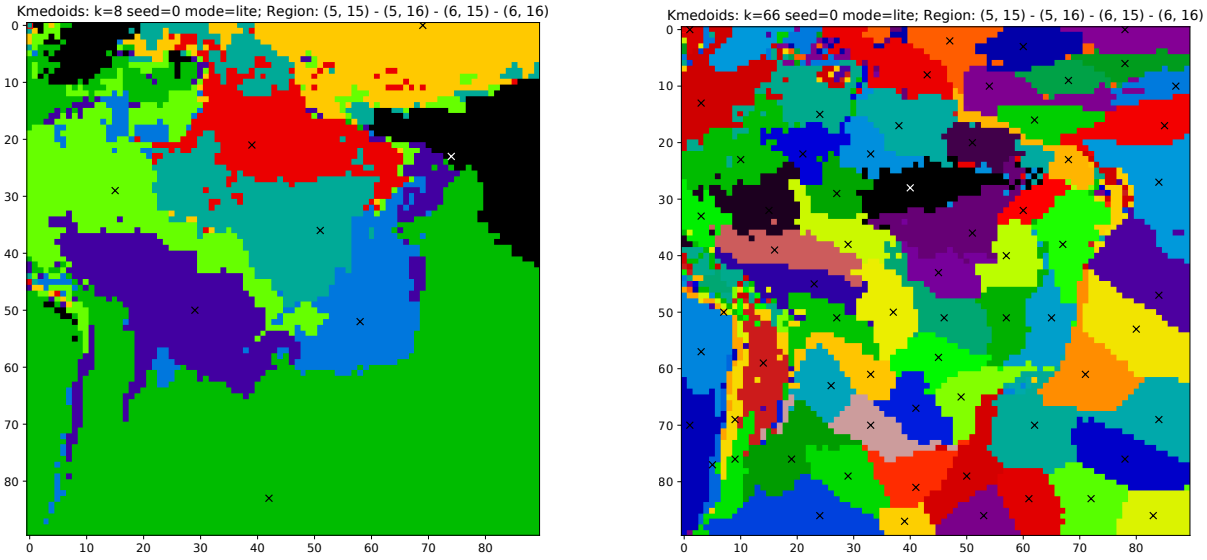


Figure 14: Groups obtained with k–Medoids using $k = 8$ (left) and $k = 66$ (right).

5.2.4.3 Discussion on Domain Partitioning

Spatial-temporal data are heterogeneous and autocorrelated, consequently the data are consistent and smoothly variable. The division of the domain into k parts by a method that considers the similarity in the temporal evolution of its elements is indeed superior to a method that does not consider any similarity. Furthermore, there is more than one k that, based on temporal similarity, expresses better different regions of the domain. So, we can consider different sizes k for the domain partitioning, for example those that are targeted by the validation methods of k in the k– medoids algorithm.

In addition to k–Medoids, we also explored DBSCAN, another clustering method that can also use DTW as the similarity measure. DBSCAN is a procedure to find 'optimal' number of groups based on the spatial density of the data, and uses two parameters: the maximum ray to agglomerate points and the minimum number of elements per group. Its disadvantage is that being an algorithm that only finds convex groups, if the elements are not similar they are considered outliers. When DBSCAN was applied to our dataset, it would either find only two groups containing most of the data points, or few groups that failed to contain most of the points, because DBSCAN would mark other points as outliers. After this exploration of DBSCAN, further analysis with it was discontinued.

In the following section, we describe the process to generate predictive models on the representative elements and the experiments to evaluate its predictive quality.

5.2.5 Forecast Error Analysis of Predictive Models on Representatives

In this section we perform extensive experiments to evaluate the predictive power of models trained on the representative elements of each group.

Following the Step 2 of our proposed methodology, (Section 4.2, we generate an ARIMA model for each representative obtained from the domain partitioning. We denote the subset of representative elements for different values of k by $\mathcal{R} \subset \mathcal{D}$, and the generated models g as per equation 2.7, $g \in \mathcal{G}_{(\mathcal{R})}$. In this context, the model parameters $\mathbf{p}$ comprise both the ARIMA (p, d, q) parameters, and the default hyper-parameters used in an implementation `auto.ARIMA` that finds a suitable (p, d, q). When using `auto.ARIMA`, the route performs a grid search that selects the (p, d, q) model parameters based on the minimization of the AIC (Akaike Information Criteria) (HYNDMAN; KHANDAKAR, 2008).

5.2.5.1 Obtaining In-sample and Forecast Errors

Figure 15 shows the division of time units on the time-series in order to train and test an ARIMA model. Recalling the preparation of the dataset described in Section 5.2.2, we work with 365 values for each representative point, denoted by the series $s[0:364]$.

Let's consider a particular representative $r \in \mathcal{R}$ and its associated temperature series s_r. To evaluate an ARIMA model g_{s_r} and its predictive power, the series is then split into three parts. The first part is used for training the model ($s_r^t[0:349]$ as an example in Figure 15), here we determine the tuple $(p_{s_r}, d_{s_r}, q_{s_r})$ that corresponds to the model g_{s_r}. Then, we take the next t_p data points ($t_p = 8$ in our example) to create the validation series $s_t^v[349:357]$. The purpose of this subset is to obtain the in-sample error E^* of the model: this is achieved by using the model g_{s_r} to make a prediction of t_p steps, and then calculating an associated error using one of the error expressions presented in Section 4.2.

To enable forecasting, the model is retrained using the same (p, d, q) model parameters found with s_r^t, but now using a longer series $s[0:357]$. In order to evaluate the model's predictive power, we use it to forecast t_f steps into the future ($t_f = 8$ in our example) that the model has not seen. But, since the subset $s[357:364]$ is actually available from the dataset, we are evaluate to calculate the forecast error, allowing us to perform forecast error analysis on different representatives from different domain partitioning schemes, as well as other predictive models used as baseline (kNN).

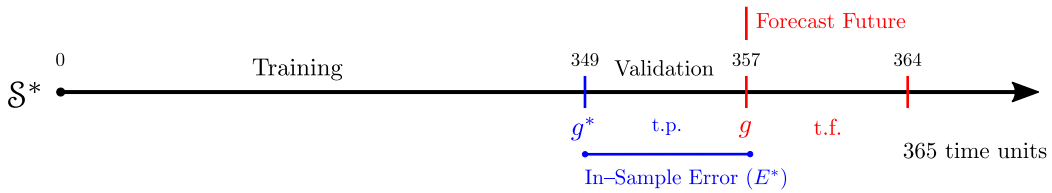


Figure 15: Dataset Split to Generate Time-Series Predictors.

5.2.5.2 Analyzing Forecast Errors

Given that the confidence intervals of the generalization error for the model built for the representative element is a varying value between the forecast error of the model composition listed, we need to analyze the intra cluster variability. When applying k–medoids to the domain, the number of elements on each cluster are different and it is almost impossible to find a correlation between the model composition forecast error and the DTW distance from the medoid.

Analyze the generalization error for the predictor in each representative, considering that the

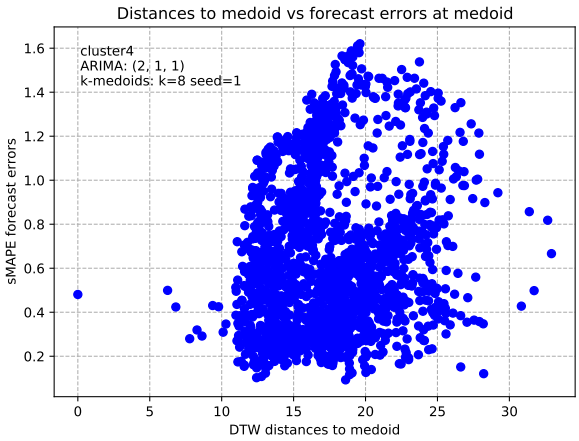


Figure 16: Forecast Error vs. DTW distance in a group k–medoids with $k = 8$

5.2.5.3 Predictive Quality of Model Composition

As explained in the methodology we are interested in evaluate the generalization error for a model composition, a model composition is formed by one or more predictors which will be used to perform a prediction on a domain region. Recalling from Section 4.2, the model composition of interest are:

- Model Composition with Minimal Error: for each group, we find the time series for which its predictive model, when applied to all elements of its corresponding group, minimizes the accumulated forecast error. This can be considered the 'best' choice for representatives that can only be found with an exhaustive approach.
- Model Composition with Minimal Local Error: for each time series in each group, we find its own predictor and calculate the corresponding forecast error. We can say that this would be the forecast error of a naive approach, because it generates as many predictors as there are time series in the region.

- Model Composition formed by the Medoids: For the time series representatives in each group, we find the corresponding predictors and calculate the forecast error in all elements of each group. This corresponds to the proposed methodology.
- Model Composition with Maximum Error: for each group, we find the time series for which its predictive model, when applied to all elements of its correspond

By comparing qualitatively the in-sample error for these model compositions we guarantee the validity of using the model for the representative as a good generalization of the group. At the same time we reduce the number of predictive models generated for a spatio–temporal domain. As a part of the off–line process we store the resultant predictive models, this will allow us to speed–up the task of compute a forecast for a time-series.

In the previous step we consider maintain several domain partitioning schemes. Considering the values for $k = \{4, 8, 66\}$, we compute the forecast error of the model composition listed above, applied over its group generated in the partitioning technique. In the following tables we

cid	size	(p, d, q)	AIC	T. F. (seg)	T. T. (seg)	Each	Min.	Min. Local	Medoid	Max.
0	990	(2, 1, 2)	-727.287	1.642	3601.959	0.333	0.305	0.379	0.331	0.826
1	990	(0, 1, 2)	-655.858	1.665	3221.489	0.192	0.180	0.198	0.268	0.517
2	990	(1, 1, 2)	-769.059	1.621	4063.693	0.384	0.460	0.888	0.587	1.608
3	990	(2, 1, 1)	-896.605	1.610	3424.456	0.341	0.387	0.634	0.394	1.028
4	990	(2, 1, 2)	-744.805	1.651	4305.636	0.782	0.571	1.182	0.769	3.058
5	990	(1, 1, 2)	-688.749	1.713	3818.287	0.470	0.489	0.580	0.609	2.062
6	1080	(0, 1, 0)	-659.159	1.744	4674.208	1.190	0.564	0.862	2.454	2.935
7	1080	(2, 1, 1)	-681.813	1.795	4847.003	0.546	0.463	0.563	0.583	1.188

Table 4: Model Composition Forecast Error for Regular Partitioning with $k = 10$.

In the table 4, the parameters (p, d, q) and AIC corresponds to the ARIMA predictor for the time-series representative. T.F is the elapsed forecast time for 8 time units and T.T is the total time used for training and forecast.

For the k–medoids partitioning we compute the same values:

cid	size	(p, d, q)	AIC	T. F. (seg)	T. T. (seg)	Each	Min.	Min. Local	Medoid	Max.
0	574	(0, 1, 2)	-782.717	1.069	2041.469	0.170	0.161	0.289	0.185	0.438
1	817	(2, 1, 2)	-814.564	1.299	3447.608	0.689	0.460	0.531	0.926	1.566
2	542	(1, 1, 2)	-782.089	0.880	2011.441	0.581	0.514	0.577	0.678	1.420
3	755	(1, 1, 1)	-767.516	1.238	2685.912	0.413	0.289	0.364	0.492	0.878
4	3479	(1, 1, 2)	-759.429	5.727	14542.318	0.785	0.475	0.548	0.838	1.983
5	803	(1, 1, 1)	-737.956	1.375	3231.718	0.407	0.294	0.378	0.437	1.194
6	625	(3, 1, 1)	-650.722	0.957	1930.740	0.157	0.168	0.175	0.203	0.478
7	505	(3, 1, 1)	-660.001	0.853	1811.335	0.388	0.375	0.422	0.551	1.015

Table 5: Model Composition In-Sample Error for Partitioning Technique k–Medoids with $k = 8$ and `seed 0`.

Experimentally we can see that when considering the prediction of the medoid as the prediction for the points of its group, there is a balance in the final prediction value

in the analyzed region. This ensures that we can use the medoids and their respective predictive models to predict the time evolution of the entire domain.

For each considered k, it is possible to observe that the predictions made by the models in the medoids are lower, this is due to the fact that the points in each group are increasingly similar. There are exceptions since the models considered are built on historical data from a point in the region, and the variations may be the product of an unexpected change in the behavior of the temperature at that point.

The advantage of an off-line process is the ability to maintain various domain division schemes (consequently various predictive models), in such a way that their representatives are shown as generalizations of various regions in the domain. The next step is to develop a mechanism that considers the composition of models for the prediction of temperature in a region of interest, considering the total set of representatives generated by more than one k.

5.2.6 Building a Classifier

In the Methodology, Sections 4.3.1 and 4.3.2, we describe the implementation of a Neural Network Model for the Time-Series Multiclass Classification task. First we extract the dataset, each sample is formed by a time-series $\mathcal{S}$, and a label y. The label is a tuple that represents domain partitioning scheme and the group in a partitioning technique (Section 4.3). Considering maintaining domain partitioning schemes for $k = \{8, 66, 132\}$, and its corresponding predictive models on representatives. The following table represent the dataset extracted (described on Table 1) to generate a NN Model:

	s0	s1	s2	s3	s4	s5	...	s364	label
0	0.484	0.563	0.443	0.386	0.326	0.391	...	0.359	8-3
1	0.482	0.508	0.520	0.477	0.354	0.444	...	0.361	132-69
2	0.639	0.618	0.549	0.609	0.433	0.476	...	0.421	66-26
⋮									⋮

Table 6: Dataset with 3000 instances for Time Series Classification.

We consider two independent architectures, Long-Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) and third hybrid CNN-LSTM. The architecture hyperparameters and optimization parameters are listed in tables 7 and 8. In the first table we can see the number and type of layers and their respective activation function, Rectified Linear Unit (ReLU), for the CNN we consider the MaxPooling option and we regularization dropout (0.4).

All models were optimized using a variant of SGD such as Adam ((KINGMA; BA, 2015)), and the batch size that defines the number of samples to work through before updating the internal model parameters. For the fourth model we consider a lower learning

rate which require more training epochs given the smaller changes made to the weights each update ().

Methods	Architecture						
	#layers	#conv	#lstm	norm	pooling	activate	regularize
LSTM	2	0	2	none	none	ReLU	dropout
1DCNN	2	2	0	none	max	ReLU	dropout
1DCNN–LSTM	4	2	2	batch	max	ReLU	dropout

Table 7: Architecture's hyperparameters.

Methods	Optimization					
	alg.	valid.	loss	epochs	batch	learn
LSTM	Adam	Split 20%	Cross Entropy	150	64	0.001
1DCNN	Adam	Split 20%	Cross Entropy	100	64	0.001
1DCNN-LSTM	Adam	Split 20%	Cross Entropy	200	128	0.001
1DCNN-LSTM	Adam	Split 20%	Cross Entropy	300	128	0.0001

Table 8: Optimization's hyperparameters.

For both architectures we need to reshape the dataset which originally is `(3000, 366)`, representing number of samples and dataset (365 instances of an univariate time series and the label). A way to look at it is, interpret a time-series as 2 dimensional data, where the first dimension is time-steps and other is the values of the temperature in 1 axe. For the model `input_shape = (None, 15, 1)`, represents 15 time-steps with 1 data points in each time step. These 1 data points are the temperature in 1 axe. The transformation of the dataset is the following:

Listing 5.1: Input Shape Sizes for CNN1D-LSTM Model

```
# First Shape:
num_features = 1 (Temperature)
n_length =  15
input_shape = (None, n_length, num_features)
```

We implement the architectures in `keras` ((CHOLLET, 2015)),

Listing 5.2: CNN1D-LSMT Model for Multiclass Classification.

```
# CNN1D-LSTM Model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu'),
                          input_shape=(None, n_length, num_features)))
,model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu')))
model.add(TimeDistributed(Dropout(0.4)))
model.add(TimeDistributed(MaxPooling1D(1)))
model.add(TimeDistributed(Flatten()))
model.add(BatchNormalization())
model.add(LSTM(1024))
model.add(Dropout(0.4))
model.add(BatchNormalization())
```

```
model.add(Dense(1024, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(number_labels, activation='softmax'))

optimizer = keras.optimizers.Adam(lr=0.0001)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])

print(model.summary())
```

The resultant accuracy of the first two models was very low, which means that the model was not able to predict the label for a new time-series. As a result both models learnt only one class.

Model	Layers	Batch Size	Learning Rate	Accuracy	Validation Loss
LSTM	2	64	0.001	2.610	4.630
CNN1D	2	128	0.001	17.600	3.380
CNN1D–LSTM	2-2	128	0.001	57.740	2.673
CNN1D–LSTM	2-2	128	0.0001	64.759	1.865

Table 9: NN Models Training Metrics for TSC.

5.2.6.1 Discussion

In the third model, a CNN1D-LSTM, we obtain higher accuracy. Performing a set of parameters tuning (considering several values for the learning rate and batch size). The variation of these parameters affects the training time and how fast we achieve convergence in the validation loss function, the following table shows the results obtained:

The key element in these experiments is to find a sweet spot between the window size, learning rate and the batch size. By lowering the learning rate we take smaller steps in order to find the best possible weights for the model and considering a greater batch size we speed-up the training of the model and reduce the noise added to the dataset, which is desirable for time-series data.

Having a sufficiently large time delay window is important for a time series predictor - if the window is too small then the attractor of the system is being projected onto a space of insufficient dimension, in which proximity is not a reliable guide to actual proximity on the original attractor. Thus, two similar time delay vectors y1 and y2, might represent points in the state space of the system which are actually quite far apart. Moreover, a window of too large a size may also produce problems: since all necessary information is populated in a subset of the window, the remaining fields will represent noise or contamination. Various heuristics can be used to estimate the embedding dimension, and here we use the false nearest neighbor method and the singular-value analysis (FAWAZ et al., 2019).

5.2.7 Spatio–Temporal Predictive Query Processing

In this Section we show the results and discussion for the on–line procedure implemented to execute a query in order to obtain a prediction (Section 4.4). A spatio-temporal predictive query is defined in Section 2.4 as the tuple:

$$Q = \langle R, t_p, t_f, Q_m \rangle,$$

where:

- R: represents the size/shape/type of interest region,
- t_p: $\{s_{t-t_p}, s_{t-t_p+1} \ldots, s_t\}$ number of steps used for prediction,
- t_f: $\{s_{t+1}, \ldots, s_{t+t_f}\}$ number of steps to predict ($n \geq 1$),
- Q_m: represents users qualitative measurements to evaluate the predictive output.

The data processing step consist in extract elements from the domain corresponding to the query region with the t_p instances, we denote this as $[R \times t_p]$. Next to compute the forecast value for the query, we use one of the following approaches to perform the model composition:

- Composition of Point Predictive Models (ARIMA and kNN).
- Composition of Representative Predictive Models.
- Composition of Classifier for Predictive Models.

In the following section we describe the results of the experiments considering queries with different sizes and over different regions of the domain.

5.2.7.1 Evaluating Spatio–Temporal Predictive Queries

We define a spatio–temporal predictive query as the tuple $Q = \langle R, t_p, t_f, Q_m \rangle$, for the experiments performed here we have considered $t_p = 8$ and $t_f = 8$. For the model composition we consider the following type of models:

1. Representatives from k–Medoids Domain Partitioning.
2. Representatives from Regular Domain Partitioning.
3. Selected by the Classifier for several domain partitioning schemes.

We compare the results of the previous predictions with a point model composition strategy, which are computationally more expensive, the considered forecast methods are:

- k Nearest Neighbor (kNN).

- AutoRegressive Integrated Moving Average (ARIMA).

To analyze the variation in the value of the prediction model in each composition composed consider a mesh size 10×10 squares over the entire domain. Consider spatio–temporal predictive queries where the region of interest corresponds to each square (Figure 17). We also consider queries with a region of interest of size 20×20, 30×30, 15×20 and 30×15.

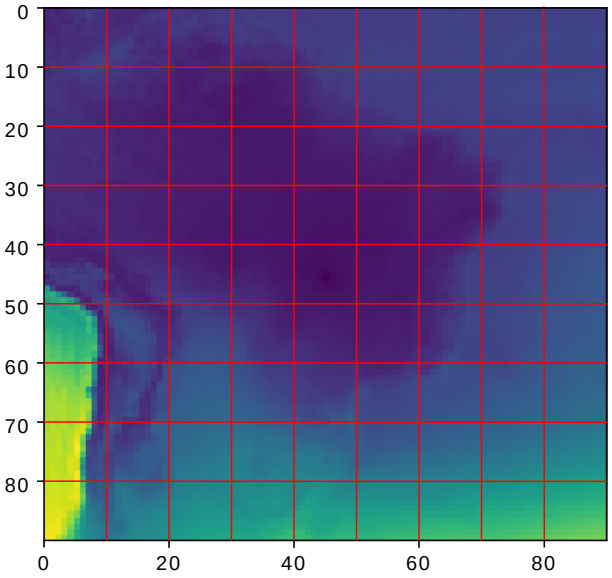


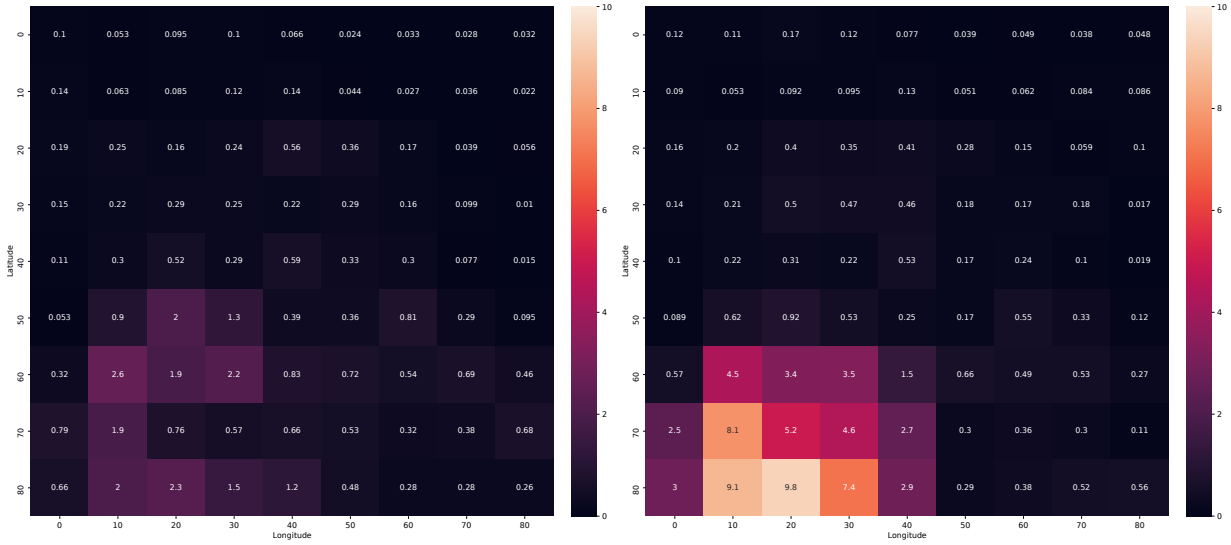
Figure 17: Mesh to consider Spatio–Temporal Predictive Queries with Region of size 10×10.

The results for 81 spatio–temporal predictive queries, product of considering the mesh of squares (10×10) over the domain, are described in the following table. For each domain partitioning scheme with $k = \{8, 66, 132\}$, we present a descriptive statistics summary for the prediction query value computed. We can see the variation of mean and its respective standard deviation for the prediction value when considering the model composition formed by the representative models in each domain partitioning technique: k–Medoids and Regular.

Dom. Partitioning Technique	k–Medoids			Regular		
	$k = 8.$	$k = 66$	$k = 132$	$k = 8$	$k = 66$	$k = 132$
Forecast Error ($t_f = 8$)	0.48 ± 0.59	0.47 ± 0.86	0.39 ± 0.62	1.05 ± 2.07	1.17 ± 2.59	0.55 ± 0.68

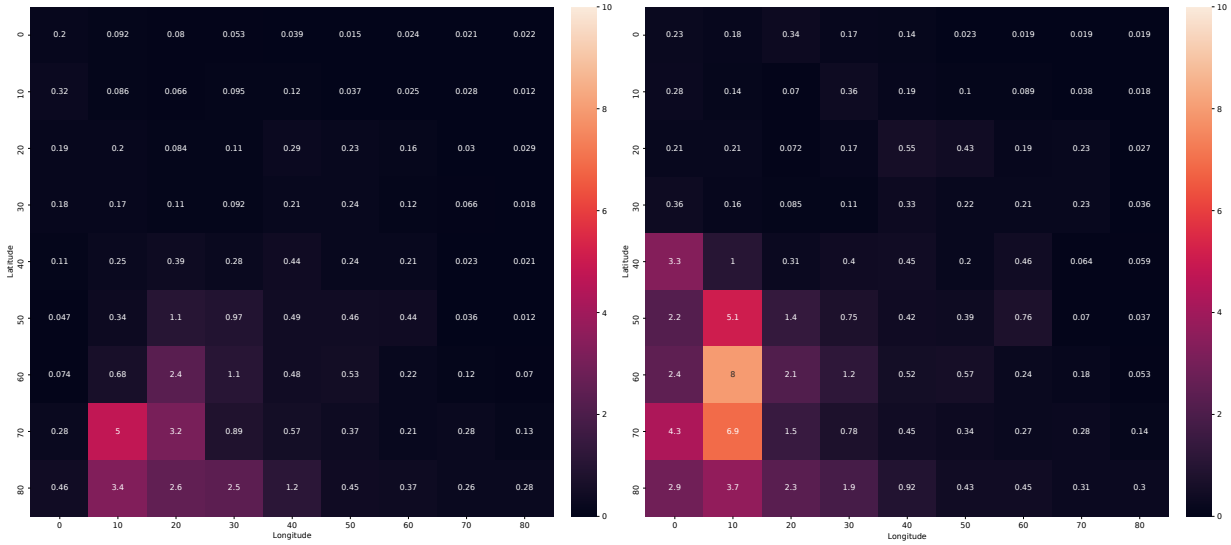
Table 10: Forecast Error Summary.

According to Table 10, the model composition formed by representative models of the k–Medoids representatives predict better than the regular approach. Analisis experimental que no es tan riguroso, pues solo muestra una variacion del error para regiones The following figures are heat maps of the computed forecast errors for the considered spatio–temporal predictive queries:



(a) k–Medoids. (b) Regular.

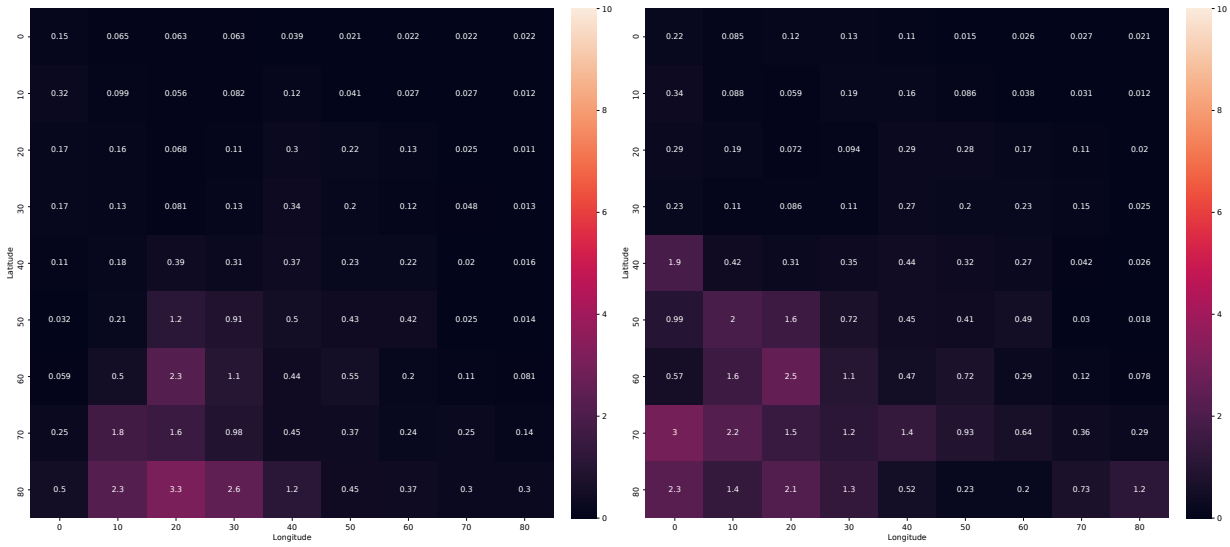
Figure 18: Model Composition formed by Models Representatives in a Domain Partitioning Technique ($k = 8$)



(a) k–Medoids. (b) Regular.

Figure 19: Model Composition formed by Models Representatives in a Domain Partitioning Technique ($k = 66$).

The experiments carried out in Sections 5.2.4 and 5.2.5 enable the evaluation and analysis of the grouping quality for each partitioning technique, as well as the predictive quality of the models generated in the representative points, respectively. With these results we evaluate the predictive quality of the models generated in the representative points



(a) k–Medoids. (b) Regular.

Figure 20: Model Composition formed by Models Representatives in a Domain Partitioning Technique ($k = 132$.)

when they are used in the composition of models for the execution of a spatio-temporal predictive query. This representative generalizes the temporal evolution and its predictive model is within a prediction limit for future values.

Experimentally we observe that for different values of k, the composition of models also presents variations in the prediction values, this is due to the properties of spatio–temporal data. When we consider the presence of several domain partitioning schemes, we use the model classifier for time series, which corresponds to our proposal.

Next we show the results of model composition using the classifier of predictive models for univariate time–series, and also the results when the model composition is the point model generation.

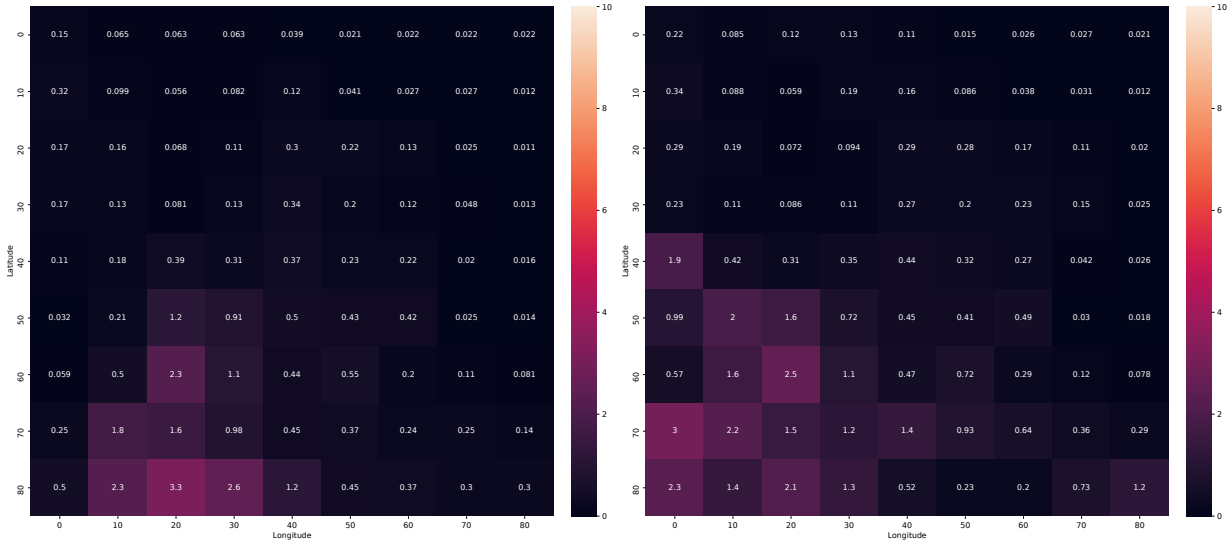
Model Composition	Model Classifier	Model Each Point (ARIMA)
Forecast Error ($t_f = 8$)	$1.07(s = 1.59)$	0.38 ± 0.61

Table 11: Forecast Error Summary.

In the following Figure we show the heat map for the prediction values computed during the query execution.

Next we consider different sizes for the region of interest, and the next table shows the region size and the total number of queries considered to compute its predictions:

The latitude and longitude for the queries are:



(a) k–Medoids. (b) Regular.

Figure 21: Model Composition by Models Representatives in a Domain Partitioning Technique ($k = 132$.)

Size	Query Region	Model Composition								
		k–Medoids			Regular			Classifier	kNN	ARIMA
		$k = 8$	$k = 66$	$k = 132$	$k = 8$	$k = 66$	$k = 132$			
20×20	$[0, 20] \times [0, 20]$	0.089	0.174	0.160	0.094	0.169	0.205	0.190	0.209	0.158
20×20	$[20, 40] \times [35, 55]$	0.335	0.199	0.230	0.395	0.234	0.225	0.330	0.258	0.203
20×20	$[50, 70] \times [60, 80]$	0.584	0.203	0.188	0.475	0.165	0.186	0.274	0.145	0.170
20×20	$[15, 35] \times [65, 85]$	0.063	0.045	0.038	0.106	0.074	0.057	0.093	0.067	0.034
30×30	$[20, 50] \times [50, 80]$	0.203	0.147	0.135	0.170	0.180	0.158	0.202	0.210	0.122
30×30	$[15, 45] \times [20, 50]$	0.262	0.155	0.168	0.367	0.199	0.184	0.281	0.198	0.156
15×20	$[40, 55] \times [20, 40]$	0.707	0.530	0.541	0.368	0.682	0.581	0.618	0.707	0.483
15×20	$[65, 80] \times [50, 70]$	0.470	0.302	0.308	0.353	0.266	0.300	0.343	0.190	0.248
30×15	$[30, 60] \times [5, 20]$	0.353	0.205	0.147	0.273	0.318	0.293	0.391	0.208	0.137
30×15	$[10, 40] \times [55, 70]$	0.139	0.111	0.098	0.127	0.158	0.115	0.135	0.226	0.095

Table 12: MSE Forecast Error for Spatio-Temporal Queries in the domain $\mathcal{D}$.

5.3 Results and Discussion

The first part of this Chapter we describe the SPTA-TSA, an application developed to implement and analyze the proposed methodology. Next, we describe the outline of experiments and analysis performed to validate our hypothesis and decisions.

5.4 Final Comments

6 Conclusions and Future Works

This final chapter contains the closing remarks of this work. Here, the most important contributions and findings are highlighted, with some additional discussion about the methodology, the analysis and the experiments. The chapter concludes with possible research lines that this work can open in the future.

6.1 Results Summary

The main objective of the experiments is to validate the proposed methodology, considering the case study of temperature prediction. According to the proposed experiments we can assert that the domain can be grouped according to some measure of similarity between the elements, and that these groups may be represented by a main element that generalizes the behavior of the group.

Ese elemento generico a su vez, nos sirve para encontrar modelos predictivos. Los analisis realizados para asegurar la que pueden ser usados para predicciones I

Considerando el caso de estudio prediccion de temperatura, fue posible validar la metodologia propuesta. Los experiementos

6.2 Main Contributions

6.3 Future Works

Para el dominio de dados espacio-temporal considerado, la dimension temporal es la base de nuestra metodologia. Habiendo demostrado la utilidad del proceso completo, admitimos que existe espacio para mejorias considerables en el analisis presentado.

- Dado que hemos considerado la division del dominio mediante tecnicas de aprendizaje supervisionado, es posible ver que segun la naturaleza de los datos (autocorrelacionados y no estaticos) existe una dificultad en encontrar un numero ideal o un metodo para verificar si existe un tamano que presente un balance entre el costo de computar las particiones y la calidad de generalizacion de los elementos en cada grupo.
- Los modelos predictivos considerados son modelos simples, que presentan , considerar modelos secuenciales
-

Bibliography

AGGARWAL, C. C.; REDDY, C. K. *Data Clustering: Algorithms and Applications*. 1st. ed. [S.l.]: Chapman and Hall/CRC, 2013. ISBN 1466558210. Cited in page 30.

AGHABOZORGI, S.; SHIRKHORSHIDI, A. S.; WAH, T. Y. Time-series clustering - a decade review. *Inf. Syst.*, Elsevier Science Ltd., GBR, v. 53, n. C, p. 16–38, out. 2015. ISSN 0306-4379. Cited 2 times in 18 and 29.

AKDERE, M. et al. The case for predictive database systems: Opportunities and challenges. In: *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*. [S.l.: s.n.], 2011. p. 167–174. Cited in page 26.

ATLURI, G.; KARPATNE, A.; KUMAR, V. Spatio-temporal data mining: A survey of problems and methods. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 4, ago. 2018. ISSN 0360-0300. Cited in page 17.

CHATFIELD, C.; XING, H. *The analysis of time series: an introduction with R*. 7th. ed. Florida, US: CRC Press, 2019. Cited in page 22.

CHOLLET, F. *keras*. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>. Cited in page 59.

CRANKSHAW, D. et al. Clipper: A low-latency online prediction serving system. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017. p. 613–627. ISBN 978-1-931971-37-9. Cited in page 15.

FAWAZ, H. I. et al. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.*, Kluwer Academic Publishers, USA, v. 33, n. 4, p. 917–963, jul 2019. ISSN 1384-5810. Cited 4 times in 20, 27, 36, and 60.

GHANTA, S. et al. ML health: Fitness tracking for production models. *CoRR*, abs/1902.02808, 2019. Cited in page 15.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123814790. Cited in page 31.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. H. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. [S.l.]: Springer, 2009. (Springer series in statistics). ISBN 9780387848570. Cited in page 51.

HENDAWI, A. M.; MOKBEL, M. F. Predictive spatio-temporal queries: A comprehensive survey and future directions. In: *Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*. New York, NY, USA: Association for Computing Machinery, 2012. (MobiGIS '12), p. 97–104. ISBN 9781450316996. Cited in page 27.

HYNDMAN, R. J.; KHANDAKAR, Y. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software, Articles*, v. 27, n. 3, p. 1–22, 2008. ISSN 1548-7660. Cited in page 55.

HYNDMAN, R. J.; KOEHLER, A. B. Another look at measures of forecast accuracy. *International Journal of Forecasting*, v. 22, n. 4, p. 679 – 688, 2006. ISSN 0169-2070. Cited 2 times in 24 and 32.

KAUFMAN, L.; ROUSSEEUW, P. Partitioning around medoids (program pam). In: _____. *Finding Groups in Data: Introduction to Cluster Analysis*. [S.l.]: John Wiley and Sons, Ltd, 1990. cap. 2, p. 68–125. ISBN 9780470316801. Cited in page 19.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: *ICLR (Poster)*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1412.6980>. Cited in page 58.

LIAO, T. W. Clustering of time series data: A survey. *Pattern Recognition*, v. 38, n. 11, p. 1857 – 1874, 2005. ISSN 0031-3203. Cited 2 times in 28 and 52.

MAKRIDAKIS, S. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, v. 9, n. 4, p. 527–529, 1993. Cited in page 23.

MAKRIDAKIS, S.; WHEELWRIGHT, S.; HYNDMAN, R. *Forecasting Methods and Applications*. 3rd. ed. [S.l.]: Wiley India Pvt. Limited, 2008. ISBN 9788126518524. Cited in page 22.

MILLS, T. C. *Applied time series analysis : a practical guide to modeling and forecasting*. London, England: Elsevier, 2019. ISBN 0-12-813118-7. Cited in page 22.

MITSA, T. *Temporal Data Mining*. 1st. ed. [S.l.]: Chapman and Hall/CRC, 2010. ISBN 1420089765. Cited in page 36.

PAL, A.; PRAKASH, P. *Practical Time Series Analysis*. [S.l.]: Packt Publishing, 2017. ISBN 9781788290227. Cited 2 times in 18 and 21.

POLYZOTIS, N. et al. Data lifecycle challenges in production machine learning: A survey. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 47, n. 2, p. 17–28, dec 2018. ISSN 0163-5808. Cited in page 15.

SAHA, S. et al. The ncep climate forecast system reanalysis. *Bulletin of the American Meteorological Society*, American Meteorological Society, Boston MA, USA, v. 91, n. 8, p. 1015 – 1058, 01 Aug. 2010. Cited in page 47.

SAINATH, T. N. et al. Convolutional, long short-term memory, fully connected deep neural networks. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2015. p. 4580–4584. Cited in page 36.

SAKOE, H.; CHIBA, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 26, n. 1, p. 43–49, 1978. Cited 2 times in 18 and 29.

SOUTO, Y. M. et al. A spatiotemporal ensemble approach to rainfall forecasting. In: *2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8-13, 2018*. [S.l.: s.n.], 2018. p. 1–8. Cited in page 48.

WIKLE, C.; ZAMMIT-MANGION, A.; CRESSIE, N. *Spatio-temporal Statistics with R*. [S.l.]: CRC Press, Taylor & Francis Group, 2019. (Chapman & Hall/CRC the R series). ISBN 9781138711136. Cited in page 17.

Appendix

APPENDIX A – Using SPTA-TSA for Spatio-Temporal Temperature Dataset

A.1 Dataset Extraction

A.2 Domain Characterization

A.3 Generating Predictors on Representatives

A.4 Solvers Execution

A.5 Query Processing

A.6 Título da seção

Figure 22: Legenda para a figura.