Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

8-13-2019 11:00 AM

Spatiotemporal Forecasting At Scale

Rafael Felipe Nascimento de Aguiar, *The University of Western Ontario*

Supervisor: Capretz, Miriam A. M., *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering
© Rafael Felipe Nascimento de Aguiar 2019

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Software Engineering Commons

Recommended Citation

Nascimento de Aguiar, Rafael Felipe, "Spatiotemporal Forecasting At Scale" (2019). *Electronic Thesis and Dissertation Repository*. 6316.
https://ir.lib.uwo.ca/etd/6316

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

Spatiotemporal forecasting can be described as predicting the future value of a variable given when and where it will happen. This type of forecasting task has the potential to aid many institutions and businesses in asking questions, such as how many people will visit a given hospital in the next hour. Answers to these questions have the potential to spur significant socioeconomic impact, providing privacy-friendly short-term forecasts about geolocated events, which in turn can help entities to plan and operate more efficiently.

These seemingly simple questions, however, present complex challenges to forecasting systems. With more GPS-enabled devices connected every year, from smartphones to wearables to IoT devices, the volume of collected spatiotemporal data that accompanies these questions has exploded, following the Big Data trend. This thesis proposes a forecasting framework that employs distributed computing in order to scale its internal components and overcome this high data volume scenario. It also designs discretization components that allow for flexibility in the framing of the forecasting questions. Furthermore, it devises a Geographically Global Model (GGM) backed by an ensemble of Stochastic Gradient Boosted Trees, a collection of Geographically Local Models (GLMs) backed by ARIMA models, and a non-linear blending of those as part of its multistage machine learning pipeline in order to boost its performance and stability.

The merit of the proposed research is evaluated in three experiments, each of which comprises millions of records, namely forecasting hourly taxi demand in the city of New York, forecasting daily crime density in the city of Chicago, and forecasting hourly visits to places of interest across Canada. The experimental results show the effectiveness of the proposed *Spatiotemporal Forecasting Framework* in forecasting stable results across the three domains, while also outperforming the naive baseline by at least 49.8% with respect to the SMAPE residuals.

Keywords: Spatiotemporal, Forecasting, Big Data, Distributed Computing, Ensemble Learning, Blending, Local Learning

Summary for Lay Audience

Spatiotemporal forecasting can be described as predicting the future value of a variable given when and where it will happen. This type of forecasting task has the potential to aid many institutions and businesses in asking questions, such as how many people will visit a given hospital in the next hour. Answers to these questions have the potential to spur significant socioeconomic impact, providing privacy-friendly short-term forecasts about geolocated events, which in turn can help entities to plan and operate more efficiently.

These seemingly simple questions, however, present complex challenges to forecasting systems. With more GPS-enabled devices connected every year, from smartphones to wearables to IoT devices, the volume of collected spatiotemporal data that accompanies these questions has exploded, following the Big Data trend. This thesis proposes a forecasting framework that employs distributed computing in order to scale its internal components and overcome this high data volume scenario. It also designs discretization components that allow for flexibility in the framing of the forecasting questions.

The merit of the proposed research is evaluated in three experiments, each of which comprises millions of records, namely forecasting hourly taxi demand in the city of New York, forecasting daily crime density in the city of Chicago, and forecasting hourly visits to places of interest across Canada. The experimental results show the effectiveness of the proposed *Spatiotemporal Forecasting Framework* in forecasting stable results across the three domains, while also outperforming the naive baseline by at least 49.8%.

Acknowlegements

I thank God for every person you sent my way. The person that I am today is, without a doubt, much better because of everything that I learned from them and all the moments I shared with them.

I want to thank my supervisor Dr. Miriam Capretz, for seeing potential in me and opening the doors to Western University. Also, I would like to say that I sincerely appreciate your guidance and the effort you put in helping me with this research.

To my colleagues, fellow TAs, students, and faculty members at Western University a big thanks. My graduate school years were fantastic because of all of you.

To Dr. Abdelkader Ouda, I extend my heartfelt gratitude. I learned a lot while I was a TA under your supervision.

I would also like to extend my appreciation to the incredibly talented and helpful people from Western University's Writing Centre, and Map and Data Centre. Your support was of exceptional value to me.

To my cousin Lilo, Thamiris and Donatello, I'll be forever grateful for your support here in Canada. Thank you for always being there for me.

To my father, mother, brother, sister, family and friends, you are the joy of my life. Thank you all so much for your kindness and encouragement.

To my wife, I could not have done any of this without you. Thank you for making my life a whole lot better.

Contents

Abstract ii

Summary for Lay Audience iii

Acknowledgements iv

List of Figures viii

List of Tables x

1 Introduction 1

- 1.1 Motivation 3
- 1.2 Contributions 5
- 1.3 Thesis Organization 6

2 Background and Literature Review 8

- 2.1 Feature Discretization 8
  - 2.1.1 Equal Width Discretization 9
  - 2.1.2 Grid-based Discretization 9
- 2.2 Time Series Forecasting Models 14
- 2.3 Machine Learning 15
  - 2.3.1 Tree-based Ensembles 16
  - 2.3.2 Blending Models 21
  - 2.3.3 Local Learning 22

2.3.4 Time-sliding Cross-validation 22

2.4 Distributed Computing And Scalability 23

2.5 Literature Review 24

2.5.1 Linear Models 26

2.5.2 Neural Networks 27

2.5.3 Ensemble Models 29

2.6 Summary 30

3 Spatiotemporal Forecasting Framework 32

3.1 Data Preprocessing 34

3.1.1 Temporal Discretization 34

3.1.2 Spatial Discretization 35

3.1.3 Data Cleaning 37

3.1.4 Large Scale Aggregation 40

3.2 Multistage Machine Learning Pipeline 42

3.2.1 Geographically Global Model 43

3.2.2 Geographically Local Models 44

3.2.3 Blending Model 48

3.3 Summary 54

4 Evaluation 56

4.1 Implementation 56

4.2 Experiments 62

4.2.1 New York City Taxi Trips 64

4.2.2 Crimes in The City of Chicago 70

4.2.3 Visits to Places of Interest Across Canada 76

4.2.4 Discussion 82

4.3 Summary 85

5 Conclusions and Future Work **86**

5.1 Conclusions 86

5.2 Future Work 87

Bibliography **90**

Curriculum Vitae **98**

List of Figures

2.1 Grid-based discretization of the Earth 11
2.2 Illustration of two levels of subdivision of the unit square 11
2.3 Six iterations of the Hilbert Curve 13
2.4 Unfolded Earth cube indexed by a global Hilbert Curve 14
2.5 Decision Tree region boundaries in a 2-dimensional (latitude, longitude) feature space 18
2.6 Decision Tree nodes with their predicates in boldface, leaves with their underlying records (described in a set), and predictions at the lower bottom 19
2.7 Time-sliding window for cross-validation 23
2.8 Distributed Computing vs. Parallel Computing 25

3.1 Spatiotemporal Forecasting Framework Overview 33
3.2 Earthquakes in the state of California from 1967 to 2018 38
3.3 Earthquakes in the state of California from 1967 to 2018 after discretization and aggregation processes 39
3.4 A geo-located polygon (in red) representing an area of interest 40
3.5 Three spatial grids covering the same geographical region (not to scale) at different resolutions 46
3.6 A discretized map of Manhattan 47
3.7 Supporting Class Diagram 49
3.8 Stage 0 - Training Of The Base Models. 50
3.9 Stage 1 - Training Of The Blending Model. 53

3.10 Stage 2 - Predicting With the Blending Model. 54

4.1 Docker Containers vs. Virtual Machines 58
4.2 Spark Cluster Architecture Overview 59
4.3 Polygons in grey representing the spatial filter over the Manhattan region. . . . 66
4.4 NYC Taxi Trips – Average SMAPE per hour on a small sample of the testing data 68
4.5 NYC Taxi Trips – Average RMSE per hour on a small sample of the testing data 69
4.6 NYC Taxi Trips – SMAPE box plot 70
4.7 Polygons in grey representing the spatial filter over the Chicago region. 72
4.8 Chicago Crimes – Average SMAPE per day on a small sample of the testing data. 74
4.9 Chicago Crimes – Average RMSE per day on a small sample of the testing data. 75
4.10 Chicago Crimes – SMAPE box plot 76
4.11 Polygons in grey representing the spatial filter across the Canadian landscape. . 78
4.12 Visits to Places of Interest Across Canada – Average SMAPE per hour on a small sample of the testing data 80
4.13 Visits to Places of Interest Across Canada – Average RMSE per hour on a small sample of the testing data 81
4.14 Visits to Places of Interest Across Canada – SMAPE box plot 82
4.15 Average SMAPE per model across experiments. 84

List of Tables

3.1 Examples of timestamp truncations at arbitrary resolutions. 35
3.2 Examples of the discretization of spatial coordinates at arbitrary resolutions. . . 36
3.3 Spatiotemporal records before aggregation 42
3.4 Records after density-based spatiotemporal aggregation. 42
3.5 Geographically Global Model Input Features 44
3.6 Sample spatiotemporal data 45
3.7 Data from Table 3.6 after the spatial partitioning step 45

4.1 NYC Taxi Trips – Dataset Statistics 64
4.2 NYC Taxi Trips – Dataset Statistics After Preprocessing. 65
4.3 GGM – Stochastic Gradient Boosted Trees Hyperparameters. 67
4.4 NYC Taxi Trips – Model Metrics Across All Testing Windows 69
4.5 Chicago Crimes – Dataset Statistics. 70
4.6 Chicago Crimes – Dataset Statistics After Preprocessing. 71
4.7 Chicago Crimes – Model Metrics Across All Testing Windows 76
4.8 Visits to Places of Interest Across Canada – Dataset Statistics. 77
4.9 Visits to Places of Interest Across Canada – Dataset Statistics After Preprocessing. 78
4.10 Visits to Places of Interest Across Canada – Model Metrics Across All Testing Windows 82

Chapter 1

Introduction

Spatiotemporal forecasting has been applied to a diverse number of domains, including crime, influenza, taxi demand, and visits forecasting. When used for crime forecasts, this technique can alert police officers to developing crime patterns in a city, helping them to coordinate patrols in order to better fight crime, for example [20]. In influenza forecasting, this technique can provide insights into which hospitals will see more demand in the next influenza season, helping health-care practitioners to prepare accordingly [52]. In the taxi demand prediction, spatiotemporal forecasting can assist with the placement of drivers so that they can quickly reach passengers, saving time and reducing overall costs [66]. Spatiotemporal forecasting can also be utilized to estimate the anticipated number of visits that a site will receive, providing the manager with foot-traffic information that they could leverage to optimize their staff schedule and better serve customers [27].

A spatiotemporal dataset is a collection of records with labels showing where and when they were collected [23]. More formally, it is a dataset sampled from a process that has both spatial and temporal components. Even more specifically, in this research, a spatiotemporal dataset is sampled from temporal and geographical spaces. That is, it contains a timestamp denoting when it happened and a pair of latitude and longitude coordinates that identify where it happened on Earth. Frequently this “when” label comes from the device’s system clock and the "where" label comes from a satellite-based radio-navigation system like the Global Positioning System (GPS) that tracks the positions of receivers on Earth. Because of the ubiquity of GPS, it is common to refer to latitude and longitude as GPS coordinates, and it is also accepted to refer to location-aware devices as "GPS-aware" or "GPS-enabled"; this research uses these terms interchangeably henceforth.

A concrete example of a spatiotemporal dataset could be a dataset that describes the occurrence of earthquakes while containing the timestamp of when they happened, the GPS coordinates of where the center of the earthquake was, and potentially some associated measurements such as the magnitude of the earthquake and the depth from the epicentre to the surface.

The task of predicting out-of-sample values of a variable based on historical data is denoted as a prediction task in the research field of Machine Learning [31]. For researchers in the field of Time Series, however, the term "forecasting" carries the same meaning, with the added connotation of predicting the future values of a random variable [65]. Henceforth, this thesis utilizes the terms "prediction" and "forecasting" interchangeably to mean the prediction of future values. What this work sets out to do, then, is to provide a domain agnostic, scalable framework that can answer a predictive question of the form: given an input feature vector x, an arbitrary timestamp t, and a location s, what is the forecasted value of a target variable $\hat{y}(x)$ at $t + 1$?

A fundamental facet of that question is the spatial (S) and temporal resolution (T) of the study. In other words, assuming sufficient resolution in the initial underlying data, one could ask forecasting questions based on longer time intervals or wider regions of interest. Taking the previously mentioned earthquake dataset as a basis, and assuming an initial temporal resolution of seconds and a spatial resolution at the city block level, one could conceivably ask questions similar to the following:

1. $\hat{y}(x)_{t+1,s} \mid t \sim T(\text{hourly}) \wedge s \sim S(\text{city block})$

2. $\hat{y}(x)_{t+1,s} \mid t \sim T(\text{daily}) \wedge s \sim S(\text{city})$

3. $\hat{y}(x)_{t+1,s} \mid t \sim T(\text{monthly}) \land s \sim S(1\ km^2 \text{ area})$

The first of these questions models a one-hour-ahead forecast at the city block level, the second models a one-day-ahead forecast at the city level, and the third models a next-month forecast for spatial regions that comprise a $1km^2$ area. The appropriate resolution for the spatial and temporal variables ultimately depends on which forecasting questions a user wants to answer. For example, it might be sensible to forecast a macro event, such as an earthquake, at the daily resolution and city level because of how its aftermath can wreak chaos in large regions for an entire day. On the other hand, for other types of events, such as taxi demand, an hourly forecast at the city block level might be more appropriate as drivers would have predictions that would be more in line with the transit patterns of a car. Another essential facet is the volume of historical data. In terms of time frame, volume relates to how many months, years or decades of data are available, for example. In terms of geographical extension, volume relates to how many neighbourhoods, cities, or states the data cover.

1.1 Motivation

The gathering of space and time contextualized information about people's offline behaviour, has made it possible for businesses and institutions to use forecasting techniques to ask questions such as: how many visits will this coffee shop experience in the next hour? A significant challenge in this type of forecasting comes from the fact that the underlying datasets are increasingly growing, following the Big Data trend [52]. With more GPS-enabled devices being connected every year, from smartphones to wearables to IoT devices, the amount of collected geolocated data has been exploding [68].

This Big Data volume that increasingly accompanies spatiotemporal datasets poses difficulties for the preprocessing algorithms that run in a single pass through the data and even more so for the training of learning algorithms, which often iterate a few times on the input dataset. For instance, the NYC Taxi Trips dataset utilized in this work's experiments has over one billion records before preprocessing, making it laborious to process it in a reasonable time frame even in a powerful machine. Furthermore, the choices of resolution, considered time frame, and geographical extension also profoundly impact this matter, making it more challenging to support arbitrary choices for these parameters without constraints. For example, forecasting over vast regions and long time frames at fine resolutions can be a remarkably challenging task for a single machine.

Difficulty also emerges from the fact that not all learning algorithms can deal with datasets at the millions-of-records scale, with many such algorithms requiring all data to fit in memory, and even when they can, their learning times can be prohibitive for datasets of such magnitude [42][30].

Another front on which obstacles arise is the stability of the forecasts. It is generally desirable that forecasts be stable, even at some loss of accuracy. For example, in the health-care domain, a forecasting model that consistently performs at 25% error in every influenza season might be favoured over another forecasting model that averages 20% error across seasons, but that at times has an error of 40%.

Few researchers have designed spatiotemporal forecasting methods with Big Data as a foundation, and to the best of our knowledge, a framework that enables forecasts that are less constrained by resolution choices and volume of historical data has not yet been proposed. Furthermore, most studies have focussed on training the single best forecasting model and have not taken model stability into consideration, except for Reich *et al.* [52] and Ray *et al.* [51]. Moreover, most studies have considered spatiotemporal forecasting problems with a high degree of attachment to the specific domain area of the underlying data source, making it difficult to gauge whether they would have similar efficacy in other domains.

1.2 Contributions

State of the art in spatiotemporal forecasting impose constraints in terms of either temporal resolution, spatial resolution, time span or geographical extension in order to cope with Big Data. For instance, Xu *et al.* [66] limits the time span to 3.5 years and fixes the spatial resolution at an 80 x 80 grid with cells averaging 23000 m^2 in order to scale to millions of records, and Liao *et al.* [41] fixes the spatial resolution at a 32 x 32 grid with cells averaging 80000 m^2 and limits the geographical extension to the Manhattan region in order to achieve similar scale. These limitations would restrain the user from running experiments similar to the Chicago Crimes and Visits Canada experiments reported in this work.

Thus, the contribution of this research is to provide a framework that is accurate, stable, and free of these constraints, enabling the user to choose the configuration that is adequate to the spatiotemporal forecasting question that they are trying to answer despite the high volume of records that accompanies its underlying dataset.

Each facet of this research's contribution is addressed as follows:

Temporal and spatial resolution. Flexibility in the choice of temporal and spatial resolution is gained by using fast *feature discretization* methods based on an equal width discretization of time and a grid-based discretization of space.

Geographical extension and time span. By utilizing training strategies that balance the memory requirements and CPU load of the learning components, the framework is able to cover wider geographical regions and longer time spans than previous work. The proposed *Geographically Global Model* (GGM) backed by tree-based ensembles can learn many decision trees over the entire geographical space in a distributed fashion, covering regions as vast as Canada. The proposed collection of *Geographically Local Models* (GLMs) backed by time series models, can learn independent models at every location, capturing local patterns in time series spanning as long as 18 years.

Prediction accuracy and stability. A multistage machine learning pipeline, introduces a *Blending Model* that learns how to weight the predictions of the GGM and GLMs, and a subset of the input features to come up with the final prediction of the framework. This blending scheme is employed in order to boost prediction accuracy, by fostering diversity, and to confer more stability than a single model solution perhaps could. Diversity is achieved by combining a non-linear model that is better at capturing spatial patterns (GGM) with time series models that are more capable of modelling sequential data (GLMs). Stability is achieved by learning how to weight the contributions of these base models with respect to their historical errors so that less-performing models have a weaker impact on the framework predictions.

Scale. Supporting all the components in the framework in scaling to high-volume datasets is a distributed computing foundation that enables the spreading of load across a cluster of machines.

Furthermore, experimental results on diverse domain datasets, yet taking only features derived from temporal and spatial components, shed light onto how the structure of the data in terms of temporal and spatial sparsity can affect forecasting capabilities, and demonstrate the flexibility of the framework in handling a variety of configurations to support arbitrary spatiotemporal forecasting questions.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 introduces background on the most important techniques utilized throughout this work. It starts by describing feature discretization methods that can be applied to temporal and spatial variables. Then it points out the beneficial properties of time series forecasting models for spatiotemporal forecasting tasks and gives an overview of one of the most popular such models. Then it proceeds to a section that describes in depth a few machine learning models, and techniques that also offer advantages for handling large volumes of spatiotemporal data. Moreover, a section on distributed computing lays out the concepts that support scaling the components of the proposed solution in the face of

Big Data. Finally, it presents a literature review of recent work done by other researchers in diverse domains.

- Chapter 3 details the proposed research, diving deep into each block of the proposed framework. First, the components that support the preprocessing of spatiotemporal data are presented, namely temporal discretization, spatial discretization, data cleaning, and large scale aggregation. Finally, the machine learning core of this research is described: a multistage machine learning pipeline composed of a Geographically Global Model (GGM), a collection of Geographically Local Models (GLMs), and a Blending Model.
- Chapter 4 describes details that are pertinent to the implementation of the proposed research. Then it moves on to an experiments section where the capabilities of the proposed approach are evaluated on different datasets, namely a dataset containing information on taxi trips in the city of New York, another dataset containing crime density information in the city of Chicago, and finally a dataset containing visits to places of interest across Canada.
- Chapter 5 concludes this work and offers an overview of its challenges and achievements. Finally, it provides some context into areas that could be explored in future work.

Chapter 2

Background and Literature Review

The goal of this chapter is to introduce important techniques that are utilized throughout this work. It starts by describing feature discretization methods. Then, it presents the Time Series Forecasting and Machine Learning methods employed in this work. Next, it moves into relevant aspects of distributed computing. Finally, it presents a literature review.

2.1 Feature Discretization

This section describes discretization processes that take a variable defined in a continuous interval and transforms it into a nominal feature space. For example, the time from the day a person was born until today can be transformed into yearly buckets representing their age in the time continuum. Moreover, Catlett [21] notes that many algorithms possess the capacity to handle continuous attributes, but to do so, they require substantially more CPU and memory than a corresponding task based on discrete features. Catlett also mentions experimental results where feature discretization resulted in reductions in learning time accompanied by no significant loss, or sometimes even significant improvements, in accuracy.

Hence, feature discretization is used in this work with two primary purposes: to produce summarized information that is meaningful to human beings, and to achieve performance and accuracy gains in Machine Learning.

Furthermore, discretization techniques are categorized according to three aspects: global versus local, supervised versus unsupervised, and static versus dynamic [26].

- Global methods take as input the entire interval that the subject variable defines. Local methods, in contrast, take as input subsets of the original feature space.
- Supervised methods take into consideration the record label while computing the partitions (also referred as intervals, bins, or buckets). Unsupervised methods, on the other hand, do not consider the label.
- Static methods require a parameter (k) indicating the maximum number of discrete intervals to produce, whereas dynamic methods conduct a search over the space of possible values of k.

2.1.1 Equal Width Discretization

Equal Width Discretization (EWD) is perhaps one of the simplest and fastest discretization methods and can be characterized as global, unsupervised, and static [26]. EWD transforms a continuous space into a discrete one by dividing it into equally sized bins. Equation (2.1) expresses the relationship between the bin width (δ), the minimum (X_{min}) and maximum (X_{max}) values of a subject variable X, and the number of resulting bins (k).

$$\delta = \frac{X_{max} - X_{min}}{k} \tag{2.1}$$

This work utilizes EWD in order to summarize temporal information into culturally acceptable bins such as hourly and daily intervals.

2.1.2 Grid-based Discretization

Grid-based discretization techniques can be seen as an extension of the 1-dimensional EWD to a 2-dimensional space. Hence, the objective of a grid-based discretization is to transform a continuous surface of spatial events akin to a point cloud into a discrete surface constituted of disjoint regions. In the context of this work, grid-based discretization is used to condense continuous geospatial points into discrete tiles on the Earth's surface.

A geospatial grid-based system is defined by cells of fixed shape and approximately equal area. One of the fastest processes, with potentially the least distortion, to obtain a geospatial grid over the Earth, involves projecting the six faces of a circumscribing cube (shown in Figure 2.1) onto the Globe, then recursively subdividing each face projection into four smaller quadrilaterals with geodesic edges (i.e., lines projected onto a sphere) [53]. For simplicity, these quadrilaterals are referred to as squared cells henceforth.

The hierarchical partitioning of each cell is done via a Quadtree [57] data structure that recursively decomposes the originating squared cell into four children cells, where the number of recursive subdivisions equates to the depth of the Quadtree and is denoted as the level (or resolution) of the spatial partitioning. Consequently, changes in the partitioning level affect the total number of cells and also their areas. This means that, increases in resolution grow the number of cells and reduce their area, whereas decreases in resolution diminish the number of cells and enlarge their area. An example of this hierarchical partitioning is illustrated in Figure 2.2, where the highlighted square is recursively divided into top-left, top-right, bottom-left, and bottom-right children.

Space-filling Curves

Representing the Earth as six square grids, one per face of the cube, is enough to be able to identify any square cell on the planet. However, that would require an index with four attributes: the resolution level, the face number, the row, and the column number. Determining whether two cells were neighbors would require a match on all four attributes, for example. The addition of a space-filling curve as an index to the Earth-cube model aims to provide a faster, more compact, 1-dimensional indexing of spatial operations. This sub-section describes the use of a space-filling curve to index each cell across the faces of the Earth-cube at all possible levels

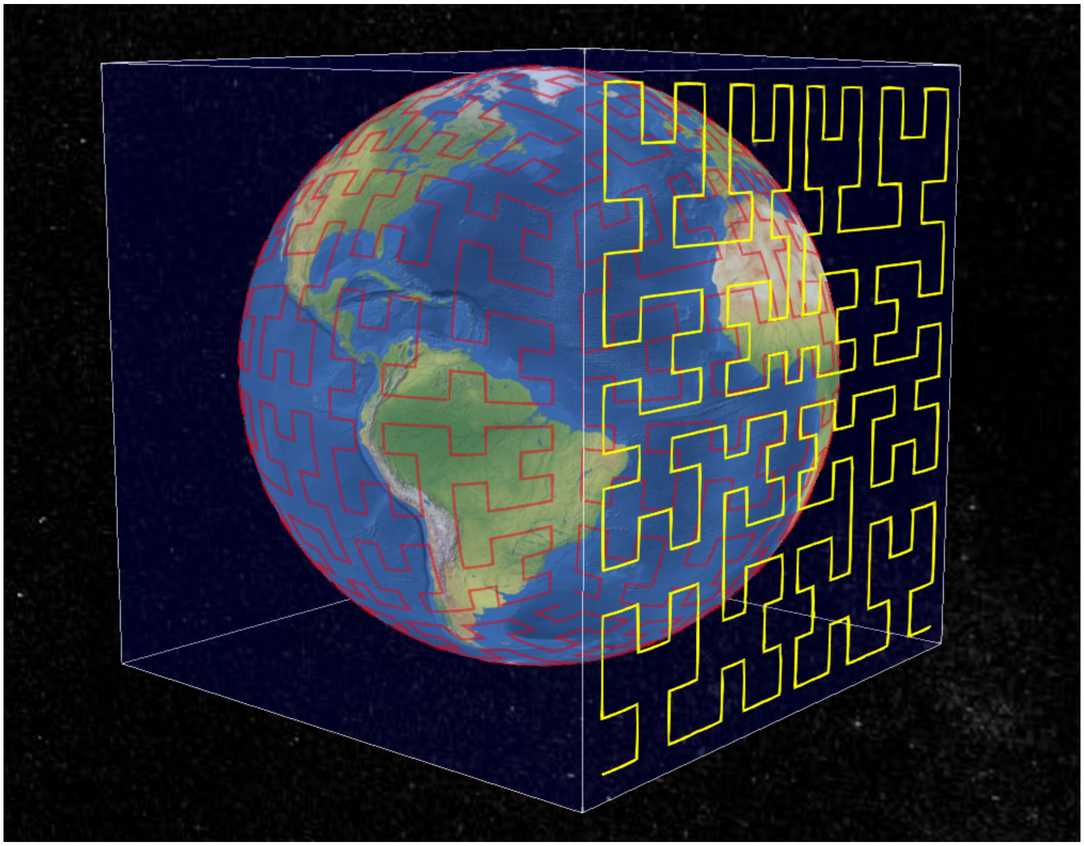

Figure 2.1: Grid-based discretization of the Earth [58]. The Globe is inscribed in a cube where each face is partitioned into square cells that are later indexed with a space-filling curve.

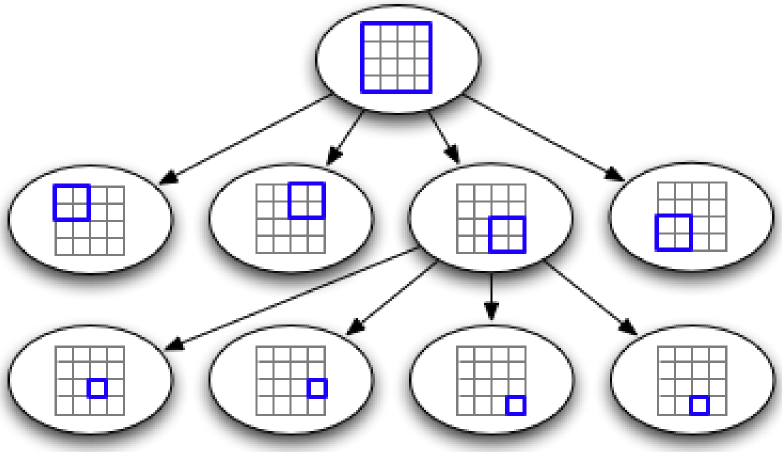

Figure 2.2: Illustration of two levels of subdivision of the unit square [35]. Each highlighted square is recursively divided into four child squares.

of subdivision of the grid down to the cm^2 level. With regard to this work, space-filling curves are harnessed to provide an efficient encoding for spatial features at arbitrary resolutions.

A space-filling curve is a curve that contains in its domain all the possible values in the 2-dimensional unit square. The Hilbert Curve is a spatial-filling curve that is both fast to compute and has high locality of reference (i.e., elements that are close in the curve are also close in the square) [32].

The initial state of the Hilbert Curve is the curve shown in Figure 2.3(a). From there, an iterative procedure is employed to generate higher-resolution versions of the curve. For example, one iteration goes from the initial curve, which indexes only four squares, to the curve in Figure 2.3(b), which indexes sixteen. A subsequent iteration goes from this past curve to the one in Figure 2.3(c), which now indexes sixty-four squares, and so on. Thus, from an initial state, with a segment size ($s = 1$), and assuming the upwards orientation of copies to be the equivalent to the starting state, the following process is repeated to generate the next iteration of the curve [32]:

1. In the first iteration, place the initial state of the curve.
2. Move to the right of the current state, leaving a gap of size s. Place a mirrored copy of the current state there.
3. Copy the curve segment on the left, rotate it 90 degrees counter-clockwise, move a distance s to the bottom, and place the copy there.
4. Copy the curve segment on the right, rotate it 90 degrees clockwise, move a distance s to the left, and place the copy there.
5. Go to the lower-left corner of the state. Start traversing the curve and connecting its segments through their closest points.

The last step in the process to construct the Hilbert Curve involves a traversal through all the cells in the square, uniquely mapping each to one value in the curve, as depicted in the top

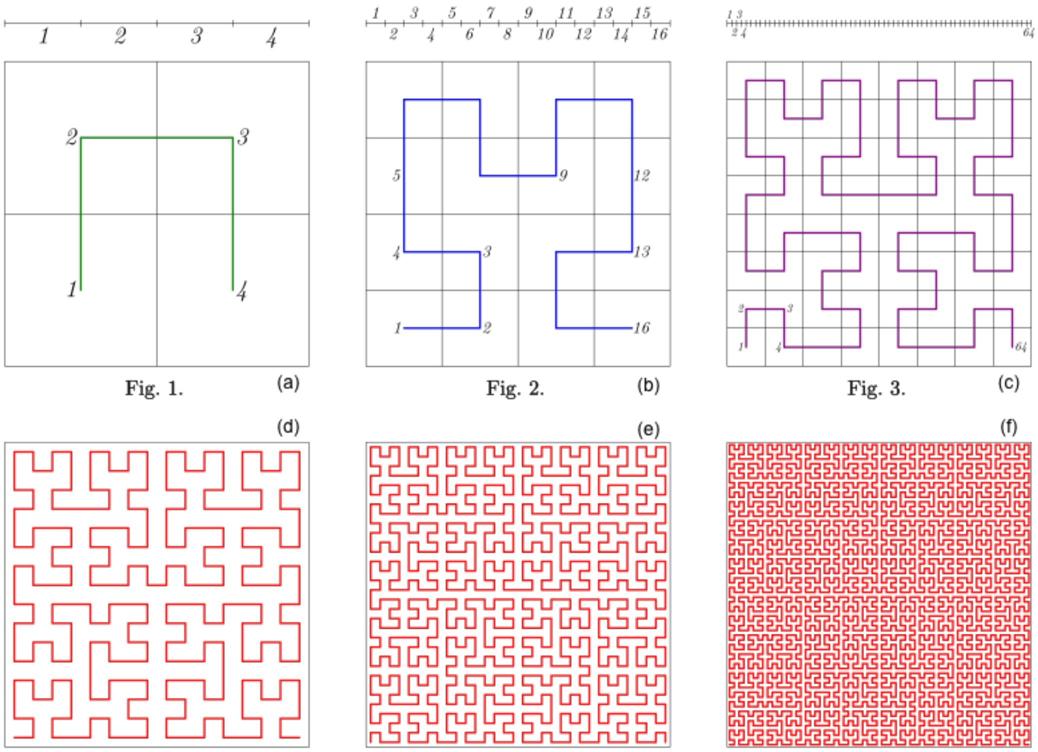

Figure 2.3: Six iterations of the Hilbert Curve [15]. Each iteration builds a better approximation of the square.

axes of Figure 2.3. This step effectively generates an index from a 2-dimensional space into a 1-dimensional space.

From Figure 2.3, it is also possible to observe the aforementioned locality of reference property. For example, in the second iteration of the curve in Figure 2.3(b), cells that are close in the curve are always close in space (e.g., cells 2 and 3); however, the converse is not always true (e.g., cells 3 and 9). Although the converse property would be desirable, it is unfortunately not possible [32].

Figure 2.4 closely (i.e., before projection onto the sphere) summarizes the effect of the grid-based discretization followed by the Hilbert curve indexing. In the figure, the six faces of the cube are unfolded, and a single Hilbert curve that encompasses the entire Globe is revealed. This research will rely on this one-dimensional coordinate system, made possible by the Hilbert indexing, to represent the spatial component of a spatiotemporal dataset.

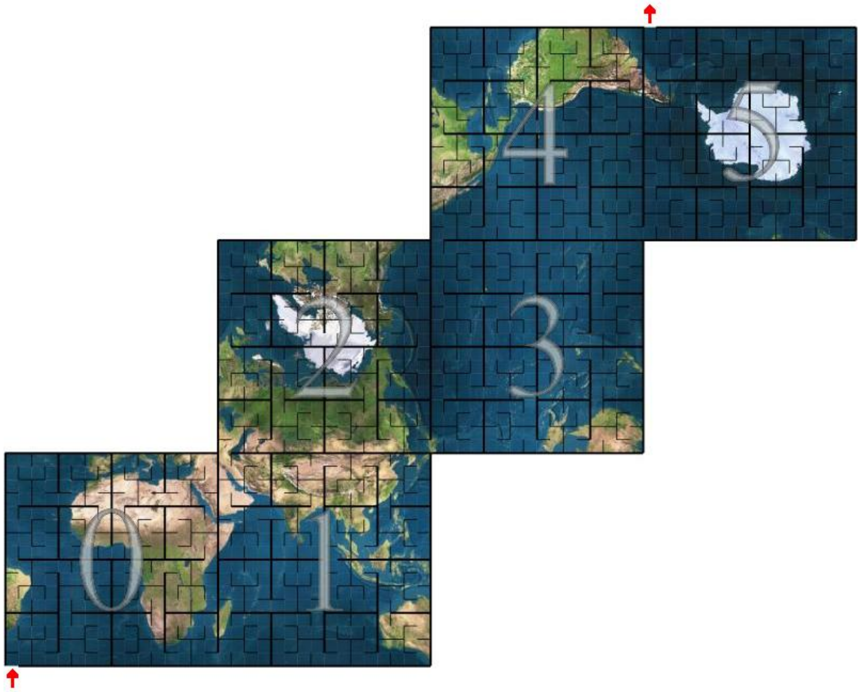

Figure 2.4: Unfolded Earth cube indexed by a global Hilbert Curve [54]. Start and end points of the curve are marked with arrows.

2.2 Time Series Forecasting Models

The core value of this research work is the use of learning algorithms to approximate the future value of arbitrary spatiotemporal measurements. One example is the number of visits that a given place will experience in the next hour. This section presents a formal definition of time series and describes the time series forecasting model utilized in this work.

Längkvist *et al.* defined time series data as a stream of sampled data points taken from a continuous, real-valued process over time [39]. Under this definition, a univariate time series represents a single variable sampled through time, whereas a multivariate time series represents multiple variables. A time series can also be classified as stationary or non-stationary, depending on whether its variance, mean, and frequency are invariant to shifts in time. A stationary time series maintains these properties invariant to shifts in time, whereas a non-stationary one does not. Moreover, the time series dependency on time brings a fundamental property to the problem: given the same input at different times, a time series governed system will likely present different responses [39].

Autoregressive integrated moving average (ARIMA) models are well-known time series forecasting models, and are generally denoted by ARIMA(p, d, q). Parameters p, d, and q are non-negative integers, in which p is the order (number of time lags) of the autoregressive model, d is the degree of differentiation (the number of times that the data have had past values subtracted), and q is the order of the moving-average model [13].

Given a time series of data $X_t \in \mathbb{R}$, where t is an integer index, an ARIMA(p, d, q) model is given by Equation (2.2). In the equation, L is the lag operator, α and θ are parameters, and ϵ is the error term, which is assumed to be an independent and identically distributed (I.I.D) variable sampled from a Gaussian distribution with zero mean and variance σ^2.

$$\left(1 - \sum_{i=1}^{p} \alpha_i L^i\right)(1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right)\varepsilon_t \tag{2.2}$$

The ARIMA model can be seen as three composable parts (AR, I, MA) [13]:

1. AR(p): Forecasts the variable of interest using a linear combination of past values of the variable.
2. I(d): Applies differentiation (i.e., the difference between consecutive observations) to make the series stationary.
3. MA(q): Uses past forecast errors in a regression-like model.

With regards to this work, time series forecasting models are used in a univariate fashion and address purely the time component of the spatiotemporal data.

2.3 Machine Learning

This section introduces the Machine Learning techniques utilized throughout this work. First, it details the inner workings of tree-based ensembles and their advantages with respect to Big

Data. Then it describes a method for blending Machine Learning models to boost their accuracy. Next, it outlines a process for partitioning the input space and delegating it to multiple independent learners. Finally, it presents a method for cross-validation on temporal data.

2.3.1 Tree-based Ensembles

The Decision Tree ensemble techniques that follow are less equipped to deal with sequential data than the earlier mentioned time series forecasting methods, but they are still capable of producing competitive forecasts, especially for multivariate time series [2][24]. This work explores tree-based ensembles in order to capture non-linear patterns in the spatial and temporal variables and assumes the potential loss of neglecting sequentiality. This difference in capabilities will be explored later to build an even stronger predictor.

A Decision Tree is a non-linear learning algorithm that induces a binary tree representing relationships in the data [50]. Each node in the tree splits the data according to an attribute until only leaves (i.e., terminal nodes) are left. Thus, the traversal from root to leaf is characterized as the prediction path of this model.

Algorithm 1 Decision Tree Induction

```
Require:
  node to be the root node at the first iteration
  $D$ to be the $D^*$ at the first iteration
  function INDUCTION(node, $D$)
      if REACHEDSTOPPINGCRITERIA(D) then
          $node.prediction \leftarrow$ PREDICT($D$)
          return
      predicate, $D_L, D_R \leftarrow$ FINDBESTSPLIT($D$)
      node.splittingAttribute ← predicate.attribute
      node.splittingValue ← predicate.value
      INDUCTION(node.left, $D_L$)
      INDUCTION(node.right, $D_R$)
```

The induction process of a Decision Tree is described in Algorithm 1. Its initial state is a root node containing the original data (D^*). The stopping condition (*ReachedStoppingCriteria*) comprises the hyper-parameters of the model that control whether a node should still be split, such as the maximum depth of the tree, and the minimum number of records in a leaf. The *Predict* function computes the model prediction based on the data available at the leaf (in regression tasks, it is the average of the values). The *FindBestSplit* function is the most computationally expensive part of the induction process and has the objective of reducing the data impurity (a measure of dissimilarity between the record labels; typically variance in regression problems). *FindBestSplit* works by selecting the feature split that results in the highest reduction in impurity. Hence, for regression problems, *FindBestSplit* aims to maximize Equation (2.3).

$$I = |D| * Var(D) - (|D_L| * Var(D_L) + |D_R| * Var(D_R)) \tag{2.3}$$

$$\begin{aligned} \text{Var}(X) &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \\ \mu &= \frac{1}{n}\sum_{i=1}^{n} x_i \end{aligned} \tag{2.4}$$

In the equation, I is the impurity reduction, *Var* is the variance as defined in Eq. (2.4), $D_L \subset D$ and $D_R \subset D$ are the datasets that result from splitting D on the given predicate.

This greedy, top-down, recursive induction process is depicted in Figure 2.5. In the figure, a dataset is split based on two features (latitude, longitude), with a minimum number of records per leaf equal to 5, generating a partitioned space with less impurity after each iteration. First, the dataset starts with a single region $A \cup B \cup C$ with variance 53.2; then it is split at latitude equals 20.0 into two regions, top (A) with variance 38.0, and bottom ($B \cup C$) with variance 10.12, leading to a variance of 48.12 after the split. Second, the bottom region is split at longitude equals 70.0 into another two regions, left (B) with variance 4.66, and right (C) with variance 1.25, leading to a total variance of 43.91 after the split. Because there are no nodes left with more than five records, the induction process stops. This spatial partitioning results in three distinct regions (A, B, C) for which the decision tree will predict constant values based

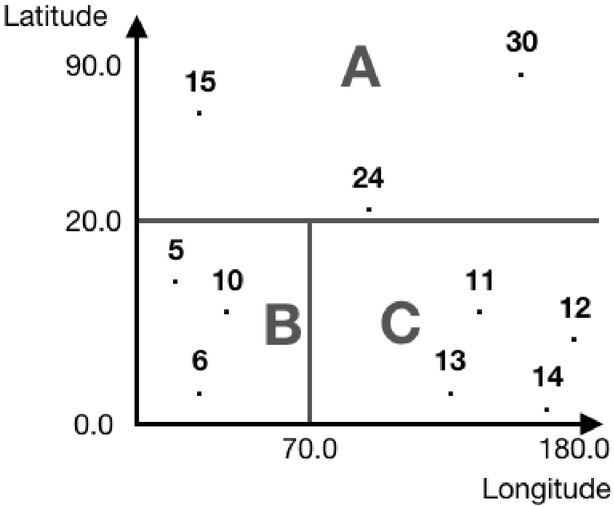

Figure 2.5: Decision Tree region boundaries in a 2-dimensional (latitude, longitude) feature space. Each node in the tree bisects the space into distinct regions.

on the average record values in the leaves. Figure 2.6 shows a view of the resulting tree, with the node predicates in boldface, node elements in curly brackets, and predictions (A=23, B=7, C=12.5) at the lower bottom of each leaf.

Random Forests

In order to improve accuracy and reduce overfitting, Ho [33] proposed an ensemble learning method that utilizes a combination of decision trees, trained on random subsets of the original feature space, to solve classification, regression, and other tasks. To reach a consensus over the individual predictions, this ensemble outputs the mode of the classes of the individual trees for a classification task or the mean of the predictions of the individual trees for a regression task [33].

Breiman [18] proposed the Random Forest as a method that uses the random subsampling of the input features (i.e., columns) in tandem with random subsampling of the input data (i.e., rows), a process known as bagging [16], in order to grow an ensemble of decision trees. Breiman also proved that bagging these weak learners would result in a more complex model that was not hurt by overfitting in the same way as traditional learners, mainly because of the

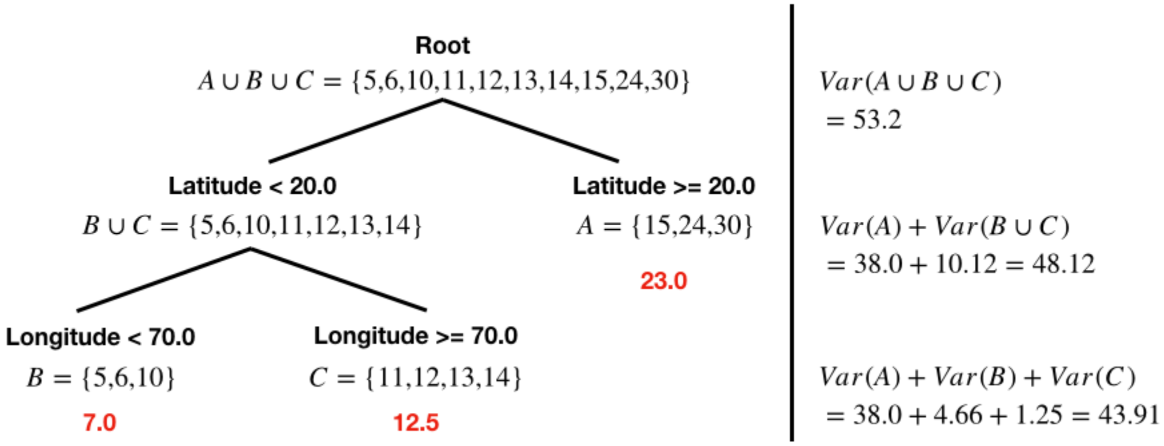
Root
$A \cup B \cup C = \{5,6,10,11,12,13,14,15,24,30\}$
Latitude < 20.0
$B \cup C = \{5,6,10,11,12,13,14\}$
Latitude >= 20.0
$A = \{15,24,30\}$
23.0
Longitude < 70.0
$B = \{5,6,10\}$
7.0
Longitude >= 70.0
$C = \{11,12,13,14\}$
12.5
$Var(A \cup B \cup C) = 53.2$
$Var(A) + Var(B \cup C) = 38.0 + 10.12 = 48.12$
$Var(A) + Var(B) + Var(C) = 38.0 + 4.66 + 1.25 = 43.91$

Figure 2.6: Decision Tree nodes with their predicates in boldface, leaves with their underlying records (described in a set), and predictions at the lower bottom. A summary of the accumulated variance after each split is also shown on the right.

way they were trained on random subsets and the way each node had a random set of features on which it could be split [18].

$$F(x) = \frac{1}{M} \sum_{i=1}^{M} h_i(x) \tag{2.5}$$

Equation (2.5) relates the output of the bagged ensemble to that from the base learners [18], where $F(x)$ is the ensemble output, M is the number of base learners, and $h_i(x)$ is the prediction of the i-th model given input x. Furthermore, each hypothesis $h_i(x)$ is formed by training a base learner on a random sample $\tilde{N} \subset N$ of the original dataset N.

Gradient Boosted Trees

Motivated by Breiman [17], Friedman proposed the Stochastic Gradient Boosting method for constructing ensembles via a boosting procedure. Stochastic Gradient Boosted Trees (GBT) [29] differ from Random Forests in that they are sequentially grown by fitting a new tree to the residuals of the previous one as opposed to grown independently. This sequential dependency poses difficulties for the training of many decision trees in parallel; however, GBTs can still be parallelized at the level of a single Decision Tree. Hence, the *FindBestSplit* function can be computed in parallel because the variance can be calculated in subsamples of the data and later aggregated [49].

$$F_0(x) = \arg\min_{\gamma} \sum_{i-1}^{N} \Psi(y_i, \gamma) \tag{2.6}$$

$$\tilde{y}_{im} = -\left[\frac{\partial\Psi(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right], i = 1...\widetilde{N} \tag{2.7}$$

$$\{R_{lm}\}_1^L = \text{terminal node } tree(\{\tilde{y}_{im}, x_i\}_1^{\widetilde{N}}) \tag{2.8}$$

$$\gamma_{lm} = \arg\min_{\gamma} \sum\nolimits_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma) \tag{2.9}$$

$$F_m(x) = F_{m-1}(x) + v * \gamma_{lm}\mathbf{1}(x \in R_{lm}) \tag{2.10}$$

Equations (2.6) through (2.10) formally express the GBT boosting process [29], where m is the index of the current model, and $\{y_i, x_i\}_1^{\widetilde{N}}$ is a random subsample of the training set ($\widetilde{N} < N$), with y_i representing the i-th record label and x_i representing the i-th input vector. $F_0(x)$ in (2.6) is the base case of the sequential process, and Ψ is the loss function, typically least squares in regression problems. $\tilde{y}_{im}$ in (2.7) are the residuals that the current model inherits from the previous one. R_{lm} in (2.8) is a set of residuals at disjoint regions $(1, \ldots, L)$ represented by the tree leaves at iteration m. γ_{lm} in (2.9) is the value that minimizes the residuals at leaf l. Finally, in (2.10), v is the learning rate, $\mathbf{1}$ is an indicator function that outputs one if $x \in R_{lm}$ and zero otherwise, and $F_m(x)$ is the final output of the ensemble.

The following are some properties of decision tree ensembles taken from Breiman [18].

1. Relatively robust to outliers and noise.

2. Useful internal estimates of error, strength, correlation, and variable importance.

3. Simple and easily parallelized.

These properties summarize the most important characteristics of the models presented in this section and point to a means of overcoming the challenges of conducting Machine Learning over big datasets: parallelization.

2.3.2 Blending Models

Ensemble methods (e.g., bagging, boosting, stacking) train multiple learners in order to solve a problem. Whereas ordinal learning approaches will train a single model on the dataset, ensemble methods will train multiple models and weight their contributions to come up with a final answer [71]. Although there are many different ways of constructing ensembles, (e.g., Random Forests are built via bagging), the strategy employed in this work is a special form of stacking, called blending.

Sill *et al.* describe stacked models as methods designed to boost predictive accuracy by blending the predictions of multiple machine learning models [59]. A stacked model is in itself a meta-model that learns how to weight the contributions of its members based on their predictions.

Blending is a derived concept introduced by the Netflix prize winners that is very close to stacked generalization, but more straightforward to implement and safer to guard against information leaks between the stacker and the generalizers because it keeps a small, completely separate holdout set to train the stacker [59].

The key to a successful blending is the inclusion of diverse base-learners [59]; where the diversity in a set of learners can be quantified as a low correlation in their predictions. Hence, in the context of this work, blending is utilized to enhance the prediction accuracy by combining tree-based ensembles with time series forecasting models.

2.3.3 Local Learning

Bottou and Vapnik devised what might seem a slow and ineffective way to scale machine learning algorithms, but one that, in reality, scored, and more importantly scaled, incredibly well [12]. Noticing that training data are rarely evenly distributed in the input space, Bottou and Vapnik suggested a change in the typical training process. Instead of relying on a single, global learner over an entire dataset, they proposed that the training process should rely on multiple learners trained independently on closely related chunks of data. A simple example of a local learning algorithm would be as follows [12]. For each testing pattern:

1. Select a few training examples located in the vicinity of the testing pattern.
2. Train any learning model over these few examples.
3. Use that model to predict over the testing pattern.

In particular, local learning provides a construct to scale learning algorithms in terms of memory, enabling otherwise memory-constrained techniques such as ARIMA to be applied on large datasets [42][30]. Hence, in the context of this work, local learning is applied with the objective of scaling time series forecasting models over large geographical regions.

2.3.4 Time-sliding Cross-validation

The aforementioned machine learning models require several hyperparameters in order to be properly fitted [10]. For example, the maximum depth of the decision tree impacts the complexity of the model, and its value can make the difference between an under-fitted tree and an over-fitted one. Thus, a fair amount of time is devoted to the tuning of these hyperparameters. Typically, methods such as grid-search and random-search are used in practice [10]. The former performs a train-and-evaluate cycle for all possible combinations of hyperparameters values given as input. The latter conducts a search on the same space randomly and non-exhaustively.

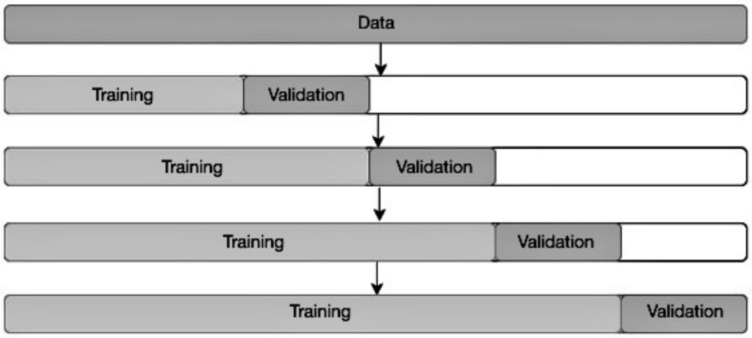

Figure 2.7: Time-sliding window for cross-validation. The Training fold chronologically precedes the validation fold.

Popular methods for evaluating the performance of a machine learning model are train-validation splits, and cross-validation, both of which are often done by picking records at random and including them into disjoint training and validation sets. In the Time Series domain, however, this randomness would wrongly allow the model to be trained on future data, and as a consequence, to be asked to predict the past [9]. Thus, traditionally in the Time Series literature out-of-sample methods such as a time-sliding window are utilized to provide a more accurate evaluation [24]. The process illustrated in Figure 2.7 splits the original training set into a training set that increases in size for each consecutive fold and a fixed-size validation set that is slid forward in time to preserve sequentiality.

Forcing the evaluation process to respect the time dimension will likely hurt the model's score and running-time performance [9]. However, as a result, the trained model will report an evaluation metric that is more semantically sound, because the model was never allowed to peek into the future.

2.4 Distributed Computing And Scalability

Sections 2.3.1 and 2.3.3 offered insights into how the training of tree-based ensembles and time series forecasting models respectively can be parallelized. This research makes use of distributed computing to explore that parallelism even further and achieve reasonable training times for machine learning models despite the challenges of Big Data [45][43][14][38].

Distributed computing is a computing paradigm that relies on the communication and coordination of computing systems (i.e., nodes) across a computer network in order to achieve a common goal [44]. As in parallel computing problems, it is best used when the global goal can be broken down into independent tasks. For example, the mergesort algorithm has a partition step that creates independent chunks of data that can later be pair-wise merged into a globally sorted solution.

Furthermore, the distribution of nodes in a computer network brings additional challenges in comparison to parallel computing. The abscence of shared memory between nodes means that they need to communicate by message passing [5]. Figure 2.8 illustrates this memory-wise difference between distributed computing systems and thread-based parallel systems. Moreover, because messaging over the network can fail, and nodes can also fail, distributed systems need to be engineered with fault-tolerance in mind [40].

A computing system is deemed horizontally scalable if its computation can be sped up when distributed across a cluster of machines (i.e., adding a new node reduces the load on the system). A vertically scalable computing system, however, explores speedups via improvements in the hardware of a single node (i.e., via more expensive hardware with more memory and processors). Although both techniques have the effect of improving the performance of a computing system, Michael *et al.* [46] have demonstrated that scale-out (i.e., horizontal) solutions offer better *price*/*performance* ratio than scale-up (i.e., vertical) solutions.

Hence, this research work makes use of distributed computing to offer a cost-effective scale-out solution that enables unconstrained spatiotemporal forecasting at scale.

2.5 Literature Review

This section reviews the relevant work done by other researchers. It describes successful approaches to solving spatiotemporal problems found in the literature and comments on the

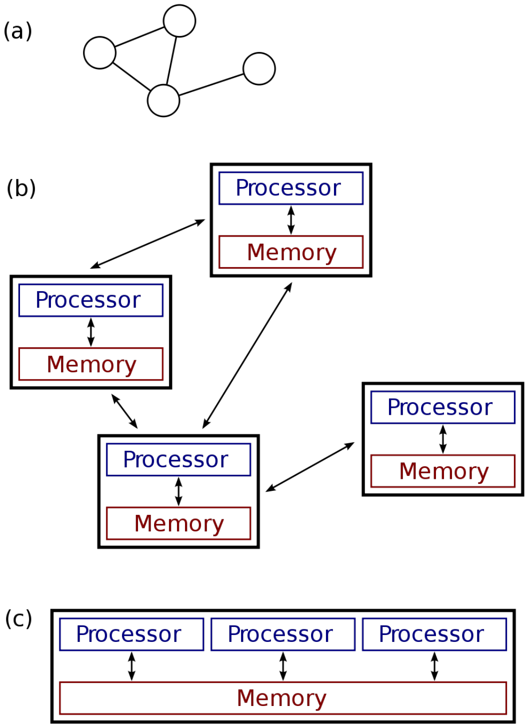

Figure 2.8: Distributed Computing vs. Parallel Computing [47]. Item (a) displays an abstract view of a distributed computing cluster. Item (b) shows a detailed view of the nodes with their own memory and message passing (outer arrows). Item (c) displays a thread-based parallel computing model with shared memory.

shared and diverging aspects concerning this work.

2.5.1 Linear Models

Catlett *et al.* [20] presented a spatiotemporal forecasting method for predicting crime density in hot spot regions. The authors pointed out that such information could enable police departments to allocate their limited resources better and develop insightful strategies for crime prevention. Their proposed approach was made up of two main components: a spatial clustering partitioner that identifies high-density crime areas, and a collection of multi-variate auto-regressive models (one per region) that account for the regions discovered in the previous step. Furthermore, in their implementation, the authors utilize a density-based partitioner (i.e., DBSCAN) for the first component, which resulted in hot spot regions that were diverse in size and shape. Catlett *et al.* [20] also demonstrated by means of experiments with public data from the city of Chicago and New York that their locally learned ARIMA models can outperform other approaches in the crime forecasting literature. In contrast to their work, this research partitions the geographical space into square cells of an approximately equal area to facilitate the forecasting at arbitrary resolutions (e.g., cells with 1 m^2 or 1 km^2 area).

Tran *et al.* [61] proposed a distributed implementation of the spatial regression model devised by Fotheringham *et al.* [28]. They argued that building a global model over data that exhibit a spatial relationship could result in misleading global coefficients. For example, a naive ordinary least-squares regression model trained to predict house prices over an entire city could learn global coefficients that imply that the older the house, the lower the price. In fact, there are significant local effects that could indicate that old country houses are more valuable, possibly because they are reminiscent of an acclaimed architectural style, and old urban houses tend to be cheaper possibly because they were built with lower construction standards [28]. The authors then proposed a distributed linear regression method that embedded the spatial variation of the data in the models' coefficients [61]. The Geographically Weighted Regression (GWR) model operates by weighting the influence of the points in the neighborhood of the point in question via a kernel function (e.g., Gaussian) so that points closer to the point in question have more influence on its value than points that are further away. In synergy with their work, this research acknowledges the difference between local and global patterns in the data by training separate models for each. Likewise, it makes use of distributed computing to train learning models over massive geographical datasets. However, unlike Tran *et al.* [61], it explores the use of a feature discretization process before the training of the learning algorithms, and even more importantly, it models both spatial and temporal features as opposed to only spatial.

Agoua *et al.* [1] proposed a spatially aware linear regression to predict photovoltaic power production. Similarly, Cabral *et al.* [19], and Yadav and Sheoran [67] proposed modifications to the inner workings of the ARIMA model to deal with spatiotemporal data. In contrast to their work, this study goes in a different direction, handing off the spatial responsibilities to another model, and utilizing pure, locally independent ARIMA models to account for the temporal component. This separation of concerns was also employed by André *et al.* [4], who built a linear blend, with fixed weights, of Spatiotemporal Vector Autoregressive (STVAR) and Cloud Motion Vector (CMV) models. In contrast to André *et al.*, this research utilizes a non-linear meta-model to learn the weights of the blend as opposed to fixing them.

2.5.2 Neural Networks

Xu *et al.* [66] proposed a one-step-ahead prediction for taxi demand based on Recurrent Neural Networks (RNN). The authors claim that many taxi companies still use a naive seasonal average as their demand forecasting method and argue that sounder methods would enable shorter waiting times and increased fuel efficiency, translating into cost savings for both drivers and passengers. Their work leaned on the long-term learning capabilities of the Long-Short Term Memory (LSTM) networks to model the taxi demand prediction as a sequence forecasting problem. LSTM networks, which are a type of RNN, contain a memory gating mechanism that is tuned to allow past observations to contribute to the current network input, and therefore to influence its final output. The length of that time window from which past observations are drawn is called the sequence length and had a fixed value of one week in their experiments to constraint the computing power required to train the model. Similarly, Huang *et al.* [34] proposed an LSTM-based approach for crime prediction while also overcoming the limitations of the fixed sequence length by using an attention mechanism that enabled the network to pay more attention to parts of the input sequence. Wang *et al.* [63] demonstrated another LSTM-based technique that accounted for spatial sparsity by reducing the geographic plane into a geographical graph, where nodes were disjoint regions determined by zip-code and edges were placed according to a statistical process. In contrast to their work, this research relies on time series models that learn the optimal sequence length during the tuning process and deals with sparsity by facilitating the use of arbitrary spatial and temporal resolutions.

Liao *et al.* [41] investigated the use of Deep Neural Networks (DNN), in particular, ST-ResNet and FLC-Net, to forecast taxi demand. The authors reckoned that the challenges in making spatiotemporal predictions are caused by strong dependencies in three different realms: time, space, and external factors (e.g., weather, and holidays). Hence, the DNNs they examined were selected because their architectures were explicitly designed to deal with both space and time simultaneously: ST-ResNet [64], which is a weighted blend of Convolution Neural Networks (CNN) trained on near-term, mid-term, and long-term data; and, FLC-Net [37] which is a fused Convolutional-LSTM [22] approach, also trained with near, intermediate, and distant time horizons. Their [41] experimental results, however, surprisingly revealed that the fusion of models in different temporal horizons had more impact in the results than the specific choice of DNN, with Feed-Forward networks outperforming all models when such fusion was not in effect. Moreover, the dependence on convolutional operations resulted in a computationally expensive architecture when dealing with large geographical regions, even after several optimizations to compact the representation of the data used in the convolution window [41]. The framework proposed in this thesis, however, does not rely on a spatial window to capture spatial patterns, but instead uses the spatial position as an input feature and never instantiates a spatial grid in memory. This representation of the geographical space, akin to an adjacency list as opposed to a matrix, has lower memory requirements and potentially works better with sparse data.

Furthermore, to achieve reasonable training times, all the aforementioned neural network techniques were tested in machines with highly capable GPUs. Due to the high cost of such hardware, this work favoured techniques that have reasonable training time on CPUs, enabling the deployment of the proposed framework on cheap commodity hardware, even for considerably large datasets.

2.5.3 Ensemble Models

Reich *et al.* [52] compiled a seven-year benchmark of influenza forecasts across the United States. Initially, they commented on how near-real-time forecasts of influenza, fuelled by the promise of Big Data, could positively impact the public health response to disease outbreaks. Then they analyzed the performance of twenty-two different models, showing that they consistently outperformed the historical baseline in week-ahead forecasts over multiple influenza seasons. One important shortcoming that arose in their study was the degradation of the reliability of the forecasts as the prediction horizon became more distant. In other words, in their experiments, most models became less reliable than the baseline after the forecasting horizon expanded further than four weeks, leaving public health officials with little time to prepare. Currently, the proposed research does not contemplate multi-step forecasting and instead focusses only on the next step, for example, the next hour or the next day, depending on the required prediction resolution.

Another notable aspect of their work [52] is the performance and stability gains they achieved by combining several base models into weighted ensembles. Likewise, Ray *et al.* [51] also argued that forecasting stability is crucial in any application in the health domain, so much so that public health officials would favour a trustworthy stable ensemble model over another model that frequently had better performance, but that at times gave extremely diverging results [51]. In accordance with their work, this research absorbs this concern for stability and also relies on a pool of base models as opposed to conceding the forecast responsibility to a single one. Furthermore, unlike Vanichrujee *et al.* [62], it uses a blending model to weight the base learners' predictions as opposed to a fixed weighting scheme based on the individual ranked performance of each base model.

Alves *et al.* [3] demonstrated that tree-based ensembles are highly capable of predicting crime density at the city level based on socio-economic indicators. Unlike Alves *et al.* [3], this research does not directly use such features in its experiments section to keep the results agnostic of domain features, so that only the merits of spatiotemporal modelling can be compared. However, it leaves the option for users to do so easily in their own experiments. In fact, the addition of up-to-date domain specific features is expected to lift performance substantially [3].

2.6 Summary

This chapter has presented a background view of the techniques that support the proposed framework. It started by describing feature discretization methods pertinent to spatiotemporal datasets. These methods are used by the *Spatiotemporal Forecasting Framework* to build its representation of temporal and spatial features. It also introduced one of the most popular time series forecasting models, ARIMA. It also dived into machine learning techniques that support spatiotemporal forecasting at scale: ensembles of randomly sampled trees; blending models that promote diversity and stability; a training strategy that allows for the training of local models; and an evaluation strategy that is suitable for temporal data. These techniques are what power the multistage machine learning pipeline employed by the *Spatiotemporal Forecasting Framework*. In addition, this chapter portrayed ways in which distributed computing can help to scale preprocessing and machine learning tasks on top of Big Data. This use of distributed computing is what enables the *Spatiotemporal Forecasting Framework* to scale to large volumes of data. Finally, this chapter reviewed the available literature on spatiotemporal forecasting, inspecting techniques based on linear models, neural networks, and ensemble models in diverse domains. To enable accurate and stable spatiotemporal forecasting at arbitrary resolutions despite the challenges posed by Big Data and without the shortcomings of the reviewed techniques, this research proposes a framework based on a distributed multistage machine learning pipeline: the *Spatiotemporal Forecasting Framework*.

Chapter 3

Spatiotemporal Forecasting Framework

The popularity of smartphones, wearables, IoT, and many other GPS-enabled devices, points to a future of endless streams of spatiotemporal data arising from diverse knowledge domains.

Machine Learning and Time Series Forecasting emerged as disciplines that could approximate the future value of a given measurement in a dataset and turn it into actionable information. The forecasts, however, require the training of learning models on historical data, and since that amount of data is expected to keep growing following the Big Data trends, it is fitting to devise a framework that can scale that training.

Hence, the primary objective of this research is to support spatiotemporal forecasts at the scale of millions of records. To do so, the proposed framework mounts each of its components on top of a distributed computing backbone, represented in the outer layer of Figure 3.1, which enables on-demand addition of new machines to handle heavier workloads. In turn, its internal components also promote parallelism whenever possible, resulting in a system that is highly horizontally scalable (Section 2.4). Therefore, throughout this chapter, where other viable techniques existed, this work leaned towards options that conferred more scalability, adhering to end-to-end scalability by design.

Figure 3.1 illustrates the top-level components of the proposed framework. The goal of this chapter is to present in detail the inner workings of each component and how they inte-

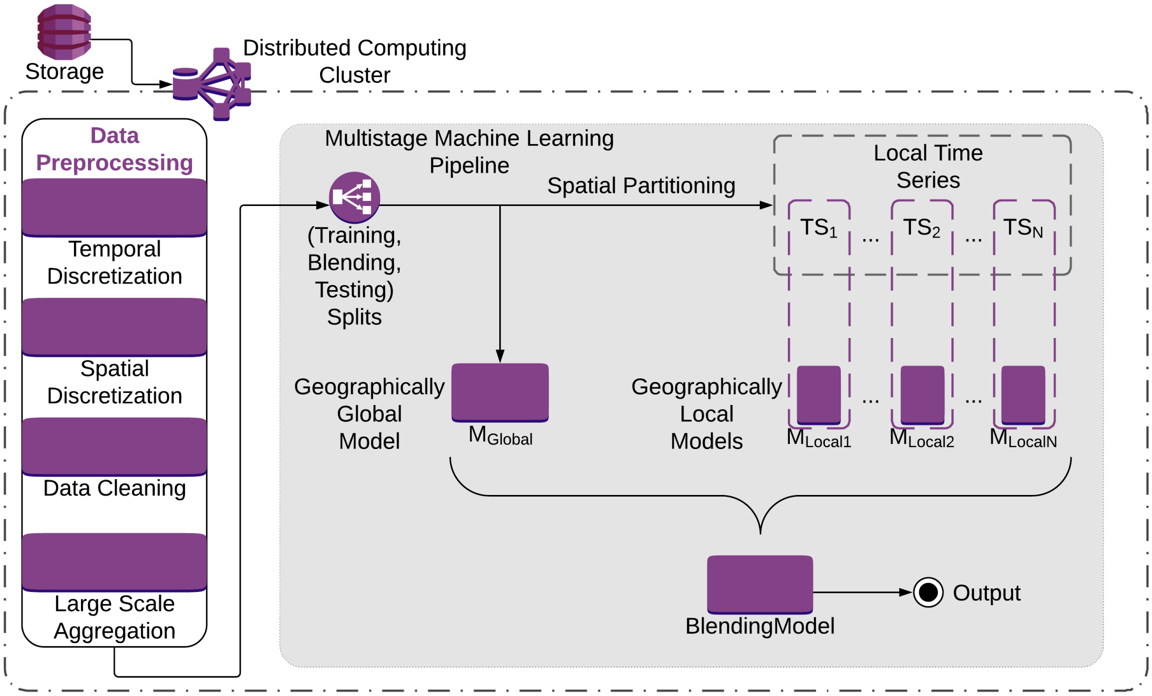

Figure 3.1: Spatiotemporal Forecasting Framework Overview.

grate. This chapter is divided as follows: *Data Preprocessing*, including the discretization, cleaning, and aggregation processes, and *Multistage Machine Learning Pipeline*, comprising the geographically global and local models as well as the blending model.

3.1 Data Preprocessing

This section describes the preprocessing steps that are performed before the data enter the machine learning pipeline, namely: temporal discretization, spatial discretization, data cleaning, and large scale aggregation.

Since the essence of this work is to produce forecasts of summaries of events in space and time, and ultimately to report that information to humans, a discretization of both space and time is required in order to map a continuous range of values into bins (aggregates or buckets).

The aggregation of values into buckets has the effect of reducing the overall amount of data and summarizing information into more meaningful chunks [36]. The former reduces the computing time, and the latter diminishes the cognitive load on the user of the application.

Furthermore, it is worth noting that every preprocessing transformation described in this section was carefully designed down to the bit level. This low-level optimization is important since these operations are literally executed millions of times in the preprocessing block.

3.1.1 Temporal Discretization

This work utilizes an Equal Width Discretization (EWD) of the temporal component, a technique that has been extensively explored in the literature [26] and is directly related to common temporal aggregates (e.g., day, hour, minute). Discretizing over such fixed width intervals allows the framework to answer forecasting questions that are closer to the cultural norm: for example, what is the forecasted value for the next hour?

Furthermore, this research work utilizes the Unix Epoch, also known as POSIX Time, to represent timestamps. The POSIX Time is a count of the number of seconds elapsed since

Original Timestamp	Grain	String Representation	POSIX Time Result
2019/08/08 09:40:40	Minute	2019/08/08 09:40:**00**	1565257200
	Hour	2019/08/08 09:**00:00**	1565254800
	Day	2019/08/08 **00:00:00**	1565222400

Table 3.1: Examples of timestamp truncations at arbitrary resolutions.

midnight, January 1st, 1970 (also known as Epoch Zero) and is encoded as a 64-bit integer. With this encoding, the discretization operation becomes fast to compute since it can be done via an integer truncation: one integer division followed by one multiplication.

The discretization of a timestamp involves choosing its adequate grain size (also referred to as resolution), for example, one hour, and then zeroing all finer grains below it: minutes and seconds. Table 3.1 illustrates the process of truncating an arbitrary timestamp (2019/08/08 09:40:40) to different grain sizes, which can be seen as a transformation that squashes a continuous time interval into a fixed discrete interval. In the table, the 64-bit integer result is obtained by starting with 1565257240 (the POSIX Time equivalent of the original timestamp) and performing an integer division (i.e., ignores the remainder) by the number of seconds in the grain, followed by a multiplication by the same factor.

In addition to a fast truncation, the choice of the Unix Time encoding also enables fast integer-based comparison operations such as filtering and ordering, both of which are of compelling relevance throughout this research.

3.1.2 Spatial Discretization

Whereas an equal width discretization, that squashed a continuous time interval into discrete temporal bins ordinarily used by humans was possible, a spatial EWD with the same characteristics was not. This impossibility stems from the fact that humans often conquered new territories through a series of challenging endeavours (e.g., wars, rough terrains), resulting in geographical regions that broadly vary in shape and size.

Relying on a spatial discretization process based on human-based knowledge would be preferable in terms of communicating the results of forecasts, because it would be closer to the

Original Location	Grain	2-D Coordinates	1-D Index Result
42.9959039,-81.2778887	$19m^2$	42.9958849,-81.2778856	**-8633697**493758771200
	$79000m^2$	42.9957268,-81.2780361	**-8633697**494468919296
	$1.2km^2$	42.9922132,-81.2794123	**-8633697**508427563008

Table 3.2: Examples of the discretization of spatial coordinates at arbitrary resolutions.

traditional norm, but it would dramatically hinder the performance of one critical preprocessing operation: hierarchical indexing, which is heavily utilized in the large scale aggregation.

In other words, the spatial part of the aggregation process is responsible for reducing a cloud of nearby location points with their individual measurements into a single point with a summary measurement. Thus, with human-based discretization and its arbitrarily shaped and sized regions, the finding of the right bucket to place a point would require a search for nearby regions' polygons centers, followed by another operation to find which of these polygons contained the point; both of these are computationally expensive, making the operation impractical at scale. Another dangerous effect that arises from utilizing regions with widely different sizes is the potential to increase data skewness in the aggregation phase. That is, one vast region could end up holding most of the points, while many other small regions would end up containing almost none, risking memory bottlenecks in the aggregation phase.

Hence, in order to benefit from faster hierarchical indexing, and easiness to operate at different grains (e.g., province, city, city block), this work utilizes a grid-based discretization. More specifically, this work favours a squared kind of grid, which when indexed with a space-filling Hilbert Curve, has higher locality of reference than triangular or hexagonal grids. This means that, cells that are close on the Hilbert curve are also close in space, although the converse does not always hold. The reason behind this is that this work aims to utilize the spatial index beyond the aggregation process, taking advantage of the increased locality of reference to power a spatially compact machine learning feature [21].

Table 3.2 exemplifies the translation from a pair of latitude and longitude coordinates in a continuous 2-dimensional space, to a unique 1-dimensional 64-bits identifier on the Hilbert Curve over the Earth-cube at arbitrary grains (i.e., cell areas). Table 3.2 also depicts the hierarchical properties of the index: a location has a common index prefix, highlighted in boldface, even at different resolutions.

A concrete visual example of the discretization process described in this section, followed by an aggregation of the event counts and magnitude, can be seen in the transformation from Figure 3.2 to Figure 3.3. Figure 3.2 illustrates an example of spatial events (earthquakes in the state of California) distributed over a continuous space, forming a cloud of location points, whereas Figure 3.3 illustrates the resulting grid of disjoint squared cells with their compiled measurements. Visualizations were built on Kepler.gl[1] with publicly available data from the Northern California Earthquake Data Center (NCEDC) portal[2].

3.1.3 Data Cleaning

Spatiotemporal datasets are prone to two primary sources of input error: drifting system clocks and imprecise GPS measurements. To avoid passing incorrect data down the pipeline, this framework implements a data cleaning layer that allows the filtering of out-of-range timestamps (i.e., too far back in the past or in any form of future) as well as the filtering of out-of-region points via a bounding multi-polygon.

Clocks can often drift (i.e., differ from the reference clock) depending on the precision of the mechanism they use to count time (e.g., quartz crystals, pendulum, atom). Although this drift can result in small shifts in the discretized timestamp, this research is more concerned with imprecisions that are large enough (e.g., years before or after the true timeline of events) to create outliers that can compromise the learning stage in the machine learning pipeline [26]. Therefore, the temporal filtering component allows the user to input a start and end timestamp that will be used to crop the dataset timeline and discard temporal outliers.

The Global Positioning System (GPS) is a satellite-based navigation system that can identify the position of a receiver on Earth and express it as a pair of latitude and longitude co-

---

[1] https://kepler.gl/

[2] http://service.ncedc.org/

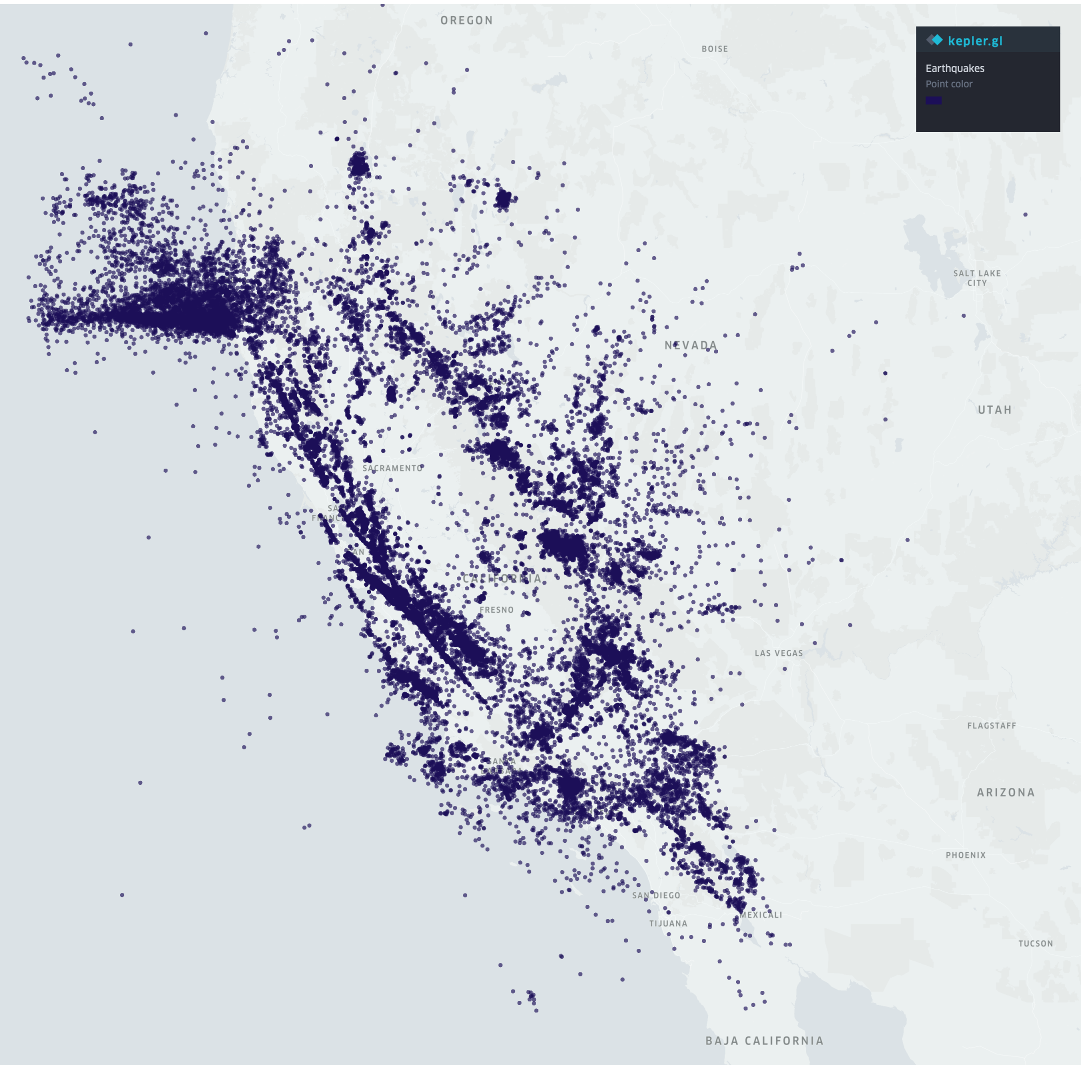

Figure 3.2: Earthquakes in the state of California from 1967 to 2018. Each point represents the source of an earthquake event.

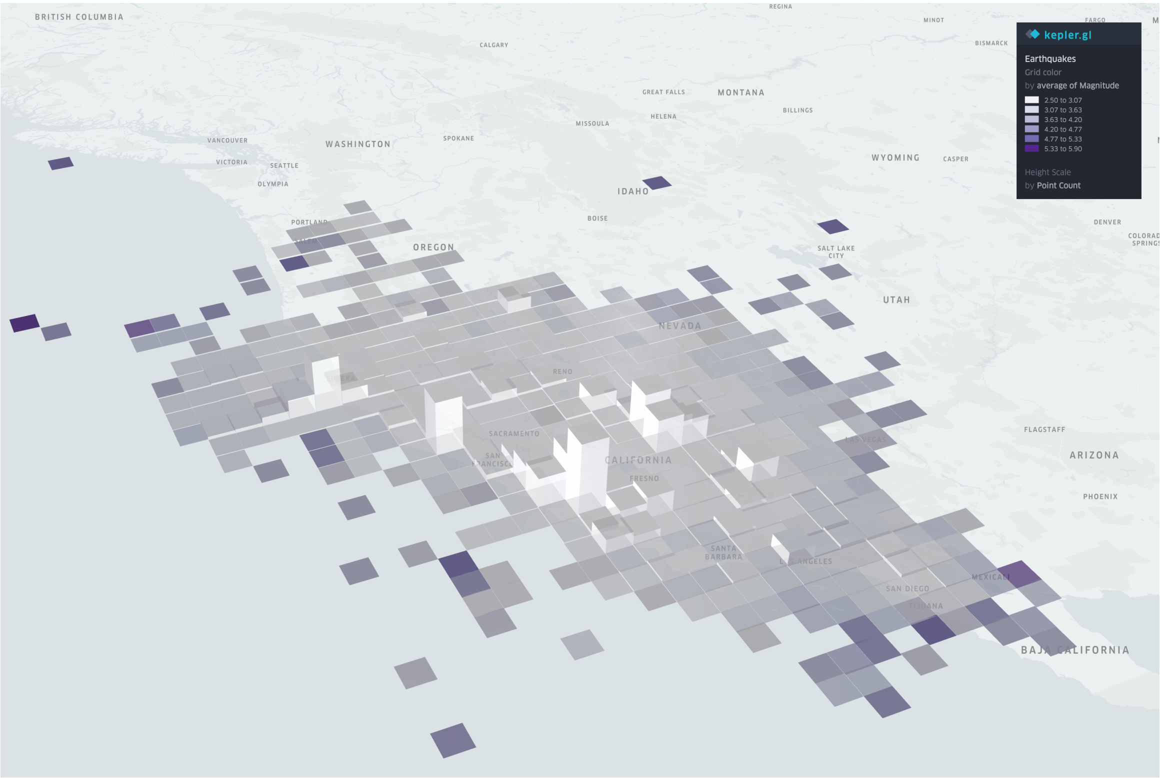

Figure 3.3: Earthquakes in the state of California from 1967 to 2018 after discretization and aggregation processes. Each 2500 km^2 squared cell summarizes the earthquake events that happened within that region. Taller cells indicate a larger number of occurring events and darker colors indicate a larger average magnitude.

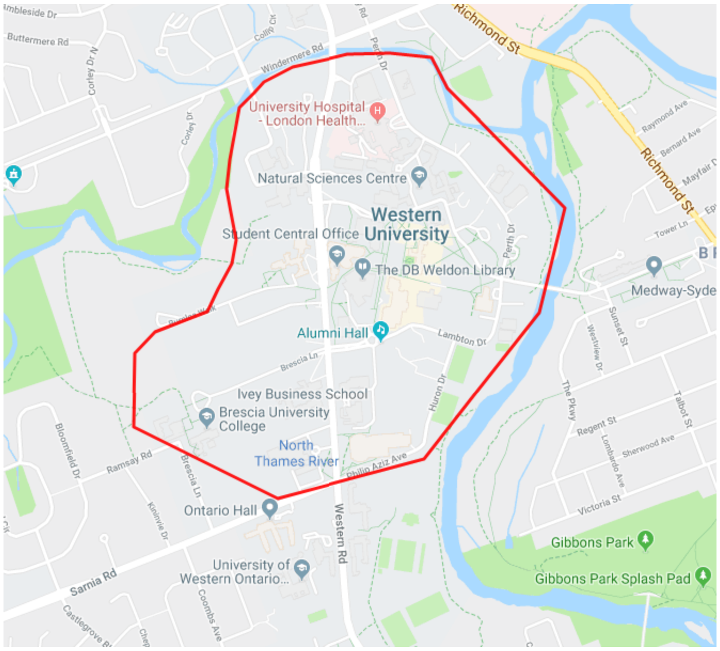

Figure 3.4: A geo-located polygon (in red) representing an area of interest.

ordinates. However, land features such as buildings and mountains can pose obstacles to the transmission path between satellite and receiver, resulting in an approximated position that can be far away from its real value. Because of this, the proposed framework allows the user to specify a collection of geo-located polygons that will be utilized to crop the desired regions of the map. An example of this type of spatial filter is shown in Figure 3.4, where only points inside the region contained in the polygon are kept.

3.1.4 Large Scale Aggregation

After the data have been discretized and cleaned, many events that happened at different times and places when determined at finer grain size will collide, generating multiple measurements for a given place and time. Although most of the domain-specific applications analyzed in this study could be supported by a pure summation of measurements, this work supports different summarization procedures through a User Defined Aggregate Function (UDAF) that follows the signature of Function 3.1.

$$mergeFunction(initialValue{:}Double, x{:}Double, y{:}Double) \longrightarrow Double \tag{3.1}$$

The merge function takes an initial value, two double-precision numbers, each from one colliding event, and returns a single double-precision that expresses the desired behaviour (e.g., sum, min, max, count). For example, the number of taxi pickups at any given hour in a given city block is equal to the sum of all pickups that happened within that hour and city block, independent of the more precise minute or street at which they actually happened. To support this summary behaviour, the *mergeFunction* would have an *initialValue* of zero and would be recursively called to count each pair of colliding events and increment its count until it ran out of pairs.

This stage is the most resource intensive in the preprocessing pipeline. Therefore, to support the aggregation of millions of spatiotemporal events into meaningful measurements, this research makes use of a distributed computing cluster, where the merging of colliding event pairs can happen in parallel and later be reduced across machines.

Moreover, both the aforementioned discretization processes refer to a decision regarding the appropriate grain. Although it is very likely that the decision is going to be application driven (i.e., based on the desired outcome of the target application), it is of utmost importance to consider its impacts in the rest of the pipeline. In fact, choosing finer resolutions for time (e.g., minutes) or space (e.g., street level) over large temporal intervals and vast regions will lead to an enormous number of records overall, substantially degrading the performance of any data-driven application that runs on top of it.

Table 3.3 puts together a few spatiotemporal records along with their truncated representations in space ($5188km^2$ cell area resolution) and time (day resolution). Table 3.4 shows the aggregated results considering a density-based count as the *mergeFunction*. The truncated timestamps (TS) of the first three records collides in the same day, and their truncated locations also lead to the same spatial cell. Therefore they get merged, resulting in a count of three. Fi-

Original TS	Truncated TS	Original Location	Truncated Location
1566691200	1566691200	-8633697493758909797	-8633752329390129152
1566723600	1566691200	-8633701486735168543	-8633752329390129152
1566766800	1566691200	-8633701643754327097	-8633752329390129152
1566777600	1566777600	5535525082222034515	5535557260735938560
1566810000	1566777600	5534362237210301769	5534431360829095936
1566853200	1566777600	5534365754237559225	5534431360829095936

Table 3.3: Spatio-temporal records before aggregation. Timestamp (TS) and location are represented by the POSIX Time and the index in the Hilbert Curve respectively.

Truncated Timestamp	Truncated Location	Density Count
1566691200	-8633752329390129152	3
1566777600	5535557260735938560	1
1566777600	5534431360829095936	2

Table 3.4: Records after density-based spatiotemporal aggregation.

nally, the truncated TS of the last three records also resolve to the same day, but their locations only collide for the last two. Therefore only the last two get merged, resulting in a count of two and a separate count of one for the left-out location.

3.2 Multistage Machine Learning Pipeline

In order to provide more freedom in the choosing of spatial and temporal resolutions for forecasting tasks over big data sets, while maintaining acceptable performance, this work proposes a multistage machine learning pipeline that can be easily distributed across commodity hardware.

More specifically, this pipeline is composed of two families of base models that work at different hierarchical geographical levels: a geographically global model that accounts for the entire observed region, and a collection of geographically local models that individually account for the single spatial cell that they observe. This nomenclature is taken from Cressie and Wikle [23] which note how models trained on local vs global scenarios can learn distinct patterns, some times even contradicting each other. These models also differ in the type of data they are more capable of modelling, with the former capturing spatiotemporal patterns without concern for sequentially in the data, and the latter capturing the sequentiality in the data while being unable to model space varying patterns.

Finally, this pipeline is augmented with a blending model that is responsible for deciding on what the final prediction should be, given the original inputs and the predictions that were made by the base models. The following sections explain these components and how they interact.

3.2.1 Geographically Global Model

Although time series (e.g., ARIMA) and geostatistics (e.g., GWR) based approaches have had their share of success in spatiotemporal forecasting problems [72][55][52][20], one of their major drawbacks is that they require the entire training data set to fit into memory [42][30], making them unfit to handle Big Data scale problems.

In order to scale the training of a global model over a vast geographical region, at arbitrary resolutions, this work employs as one of its learning algorithms, a non-linear tree-based ensemble such as the Stochastic Gradient Boosted Trees. The advantages of this approach are two-fold: ensemble models can be trivially parallelized and do not require that all data fit in memory [18], and a non-linear model can hierarchically partition the space and effectively capture global and local relationships [8].

Furthermore, the fact that each individual model in the ensemble is given a sample of the original data, and that the models' computation can be broken down into independent tasks, enables the training of the ensemble in a distributed fashion. In other words, computing steps can operate in parallel across different machines in a cluster, resulting in a faster training process.

The Geographically Global Model takes as input the features listed in Table 3.5. It is worth noting that although most of the time-based features are standard in time series problems, the spatial feature is not. This work moves away from the typical spatial features (i.e., latitude and longitude) and explores the use of the position in the Hilbert space-filling curve described in Section 2.1.2. It does so to provide the model with a more compact representation of the

Feature Name	Description	Example
CellId	Index on Hilbert curve over the earth cube	-8633697494468919296
Year	Year	2019
Month	Month	8
Day	Day	16
Hour	24h hours	4
Timestamp	POSIX Time	1565971200
WeekDay	Day of week (1 to 7; from Sun to Sat)	1
IsWeekend	1 if Saturday or Sunday, 0 otherwise	1

Table 3.5: Geographically Global Model Input Features

geographical space (one dimension as opposed to two).

This tree-based ensemble is then tuned via grid-search with respect to the number of trees, maximum depth, and the maximum number of bins while being evaluated on folds generated by the time-sliding cross-validation process described in Section 2.3.4. After this, this model is then ready to predict on new records and thereby to assist the blending model in computing the final prediction.

3.2.2 Geographically Local Models

To take advantage of the success of time series models in the modelling of spatiotemporal problems while mitigating their memory constraints, this work leans on a Local Learning strategy backed by a spatial partitioning. These models are built using the following procedure:

1. **Data Reduction**. The input data are projected to contain only the space component at a chosen resolution, the time component, and the target value (i.e., label). This transformation reduces the amount of data intake and avoids the higher computational costs of a multivariate model.

2. **Spatial Partitioning**. The input data are spatially partitioned by *cellId* at the original resolution. The effect of this spatial partitioning is demonstrated in the transformation from Table 3.6 to Table 3.7, where records with the same *cellId* are grouped in the same local dataset. Optionally, the input data can be partitioned at lower spatial resolutions

cellId	timestamp	label
$cellId_1$	ts_2	$label_1$
$cellId_3$	ts_1	$label_2$
$cellId_3$	ts_2	$label_3$
$cellId_1$	ts_1	$label_4$

Table 3.6: Sample spatiotemporal data. The label is the measured value at the given *cellId* and *timestamp* (e.g., number of taxi pickups).

cellId	Local Dataset
$cellId_1$	$[(ts_2, label_1), (ts_1, label_4)]$
$cellId_3$	$[(ts_1, label_2), (ts_2, label_3)]$

Table 3.7: Data from Table 3.6 after the spatial partitioning step. Records that belonged to the same spatial cell were grouped together, forming an independent dataset at each location.

and aggregated to reduce the number of resulting spatial cells and fit smaller computational budgets at some loss of accuracy. Figure 3.5 illustrates a spatial partitioning at the following three resolutions: 79000 m^2, 0.32 km^2, and 1.27 km^2. In the figure, it can be seen how altering the spatial resolution affects the area and number of cells.

3. **Local Time Series Construction**. The records in each local dataset are chronologically ordered by *timestamp*, resulting in one local time series per *cellId*. Then, each local time series is subjected to linear interpolation to fill the missing values.

4. **Local Learning**. With data clustered in these local time series at arbitrary spatial dimensions (e.g., neighbourhood, city, state level), Local Learning is employed to build multiple independent models that represent the space as a whole. Under this creation strategy, one time series forecasting model is built to address the patterns in each particular cluster (hence the name, geographically local model). For example, Figure 3.6 represents a squared grid over Manhattan in which one time series model would be trained for each marked squared cell.

5. **Tuning**. These models undergo an optimization of hyper-parameters via the Akaike Information Criterion (AIC) [56] and become ready to issue predictions.

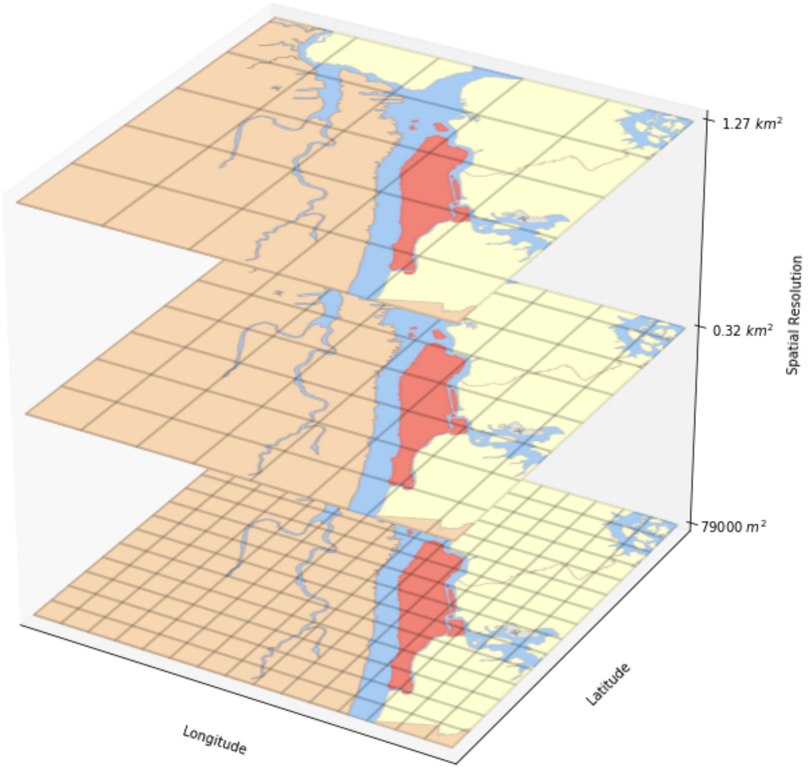

Figure 3.5: Three spatial grids covering the same geographical region (not to scale) at different resolutions. The higher-resolution grid at the bottom has cells of an approximate area of 79000 m^2, the intermediate-resolution grid in the middle has cells of an approximate area of 0.32 km^2, and the lower-resolution at the top has cells of an approximate area of 1.27 km^2.

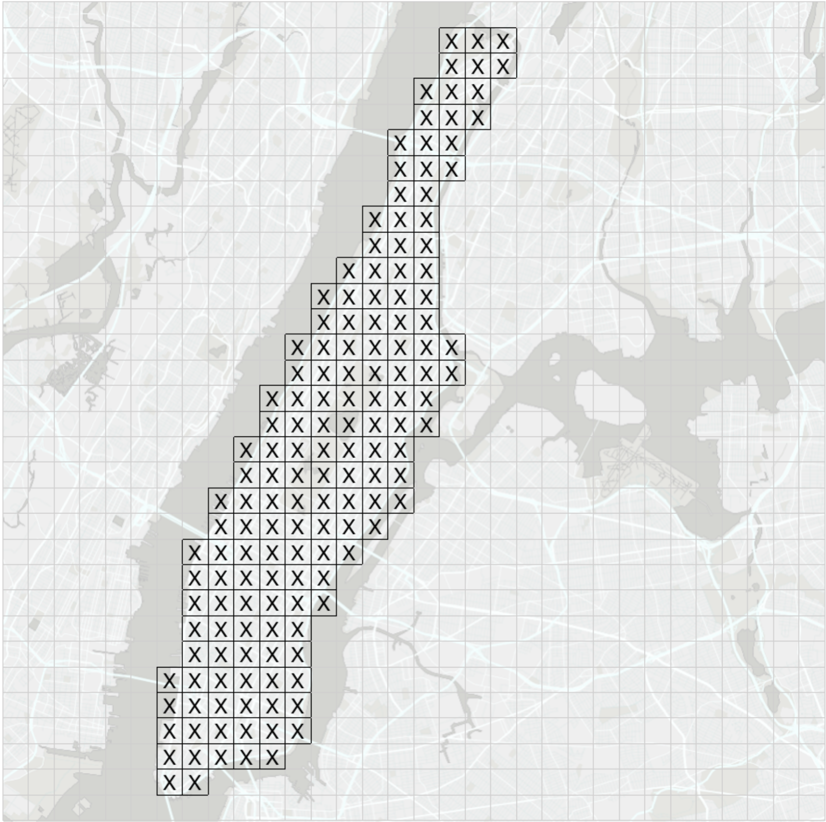

Figure 3.6: A discretized map of Manhattan. Each marked cell holds its own time series.

In summary, the benefit of employing a lightweight, independent, univariate time series model per disjoint location is that it becomes trivial to scale to large regions and fine resolutions. In other words, an increase in the number of mapped locations or in the resolution grain can be linearly compensated for by adding more machines to the computing cluster.

3.2.3 Blending Model

A natural consequence that emerges from the choosing of the tree-based models for the Geographically Global Model presented in Section 3.2.1 is that data will be modelled as static. In other words, the GGM assumes independence between consecutive records, and therefore will lose the sequential patterns in the data. However, because each node of the tree can bipartition the space represented by the *cellId*, it is expected that spatial relationships (e.g., distances from location to location) are adequately captured. On the other hand, the times-series based models chosen for the Geographically Local Models presented in Section 3.2.2 will shine in the sequence forecasting, but will not capture spatial patterns because they assume independence between spatial cells.

Hence, these models are preferred as base learners for the problem of spatiotemporal forecasting at scale: each family of model addresses one part of the problem while keeping up with the parallelism necessary to handle Big Data, given that the latter model is trained through Local Learning. Moreover, although other families of models, such as Neural Networks, could have been included, doing so would have incurred higher computational costs and added more complexity to managing the pipeline in a production environment. In other words, there is a trade-off between accuracy in predictions and the number of models included in the base models layer.

Figure 3.7 displays a class diagram with the high-level classes and interfaces that support the *Spatiotemporal Forecasting Framework*. In the figure, a GGMView portrays a Dataset containing records that are compatible with the schema that the GGM accepts. The same notion is valid for the GLMsView and BlendingView. Moreover, the schemas for all such

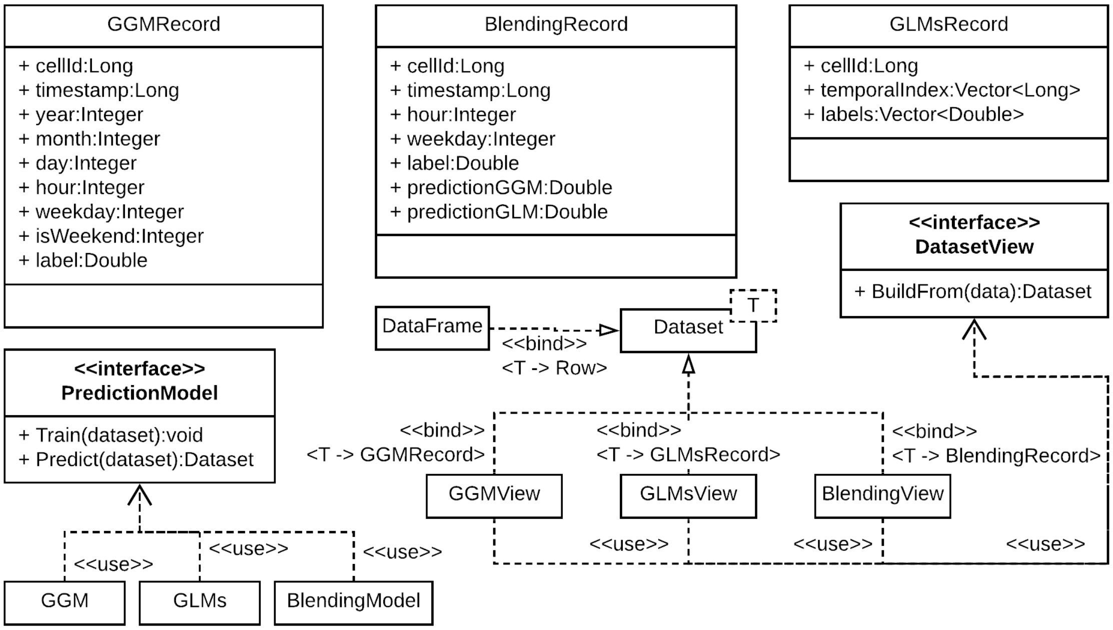

Figure 3.7: Supporting Class Diagram.

records are shown, as well as the signatures of relevant methods.

Algorithms 2 through 4 and the accompanying interaction diagrams 3.8 through 3.10 describe the strategy employed to build a geographically global model, a collection of geographically local models, and a blend of these two while respecting the time series constraints and avoiding data leakage.

After preprocessing, the dataset is split into $train_{BaseModels}$, $train_{BlendingModel}$, and $test$, without shuffling, so that those slices are consecutive in time. Then the pipeline proceeds as follows:

- **Stage 0 — Training Of The Base Models**. Both the geographically global model and the geographically local models are trained and tuned over the $train_{BaseModels}$ dataset. They build their views of the training data and run in a distributed fashion. As mentioned before, the GGM is trained globally, whereas each GLM is trained only on the records that pertain to the spatial cell that it observes. This training procedure is formalized in Algorithm 2 and illustrated in its accompanying diagram depicted in Figure 3.8.
- **Stage 1 — Training Of The Blending Model**. All base models make predictions

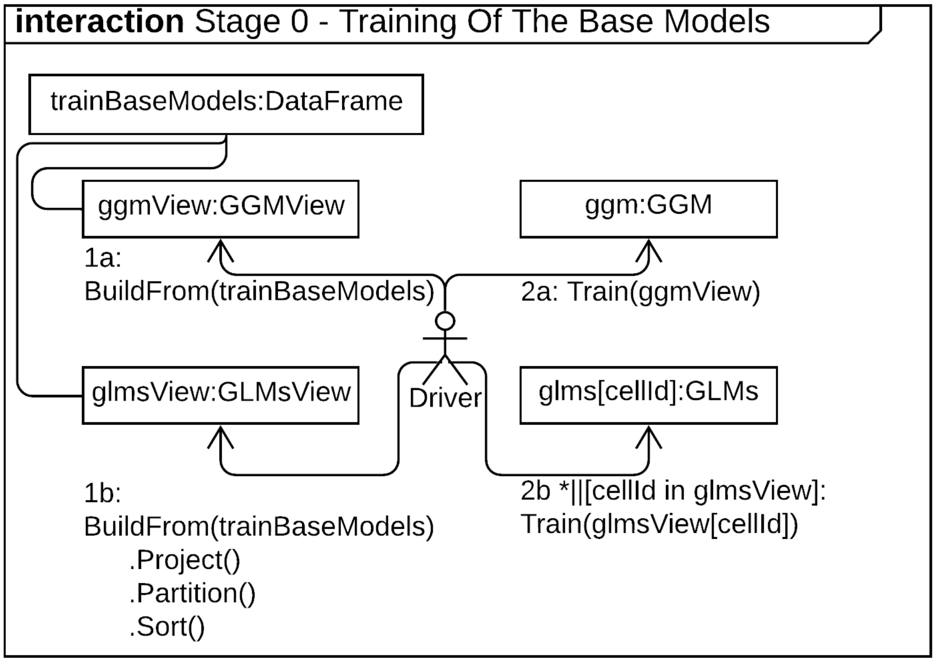
interaction Stage 0 - Training Of The Base Models
trainBaseModels:DataFrame
ggmView:GGMView
ggm:GGM
1a: BuildFrom(trainBaseModels)
2a: Train(ggmView)
glmsView:GLMsView
Driver
glms[cellId]:GLMs
1b: BuildFrom(trainBaseModels)
.Project()
.Partition()
.Sort()
2b *||[cellId in glmsView]: Train(glmsView[cellId])

Figure 3.8: Stage 0 - Training Of The Base Models.

on the $train_{BlendingModel}$, and then the blending model is trained and tuned over a subset of the original features augmented with predictions of the base models, denoted as $train^{*}_{BlendingModel}$. The augmentation originates from the join operations in Algorithm 3 where the predictions of the base models are introduced into the input dataset that the blending model receives. Furthermore, the reserved $train_{BlendingModel}$, with no intersection with $train_{BaseModels}$, is necessary to guarantee that the blending model never sees the same data as the base models in the ensemble, and its size is required to be small to avoid massively overfitting the data [59]. Because of this smaller size, the choice of what machine learning model to use when blending does not have the same scalability requirements as the base models do. A more important requirement for it is non-linearity because the best representation of the relationship between the contributions of the base models and the subset of original features can be non-linear. For this reason, a stochastic gradient boosted trees ensemble is also utilized as the learning algorithm in the imple-

Algorithm 2 Stage 0 - Training Of The Base Models

```
Require:
  dataset to be a spatiotemporal dataset with a timestamp and a cellId
  TRAIN to be an operation that optimizes a models' hyperparameters and fits it to the input data
  proportions to be an array of percentages by which to split the dataset
  SPLITON to be a function that chronologically slices a dataset
  GGM to be a Geographically Global Model
  GLMs to be a collection of Geographically Local Models

  temporalIndex ← Chronological index of dataset based on timestamp
  [$train_{BaseModels}$, $train_{BlendingModel}$, $test$] ← dataset.SPLITON(temporalIndex, $proportions$)
  baseModels ← TRAINBASEMODELS($train_{BaseModels}$)

  function TRAINBASEMODELS(dataset)
                        ▷ Execution of GGM and GLMs code is independent and distributed
    ggm ← $GGM()$
    glms ← $GLMs()$
    baseModels ← [ggm, glms]
    $ggmView$ ← $GGMView$.BUILDFROM(dataset)
    $glmsView$ ← $GLMsView$.BUILDFROM(dataset)
      .PROJECT()
      .PARTITION()
      .SORT()
    ggm.TRAIN($ggmView$)
    for cellId in glmsView do                                ▷ parallel loop
      glms[cellId].TRAIN($glmsView$[cellId])
    return baseModels
```

mentation of the blending model in the *Spatiotemporal Forecasting Framework*. This procedure, where the predictions of the base models are utilized to influence the training of the blending model, is formalized in Algorithm 3 and illustrated in its accompanying diagram depicted in Figure 3.9.

- **Stage 2 — Predicting With The Blending Model**. Once the blending model has learned how to weight the inputs and the contributions of the base models in Stage 1, these intermediate models are then used to predict on the test set ($test$), generating a new test set ($test^*$) on which the blending model can make predictions. Then the predictions of the blending model are returned as the final output of the ensemble. This prediction proce-

Algorithm 3 Stage 1 - Training Of The Blending Model

```
Require:
  TRAIN to be an operation that optimizes a models' hyperparameters and fits it to the input
  data
  JOIN to be a function that joins datasets
  baseModels to be the fitted base models defined in Algorithm 2
  $train_{BlendingModel}$ to be the dataset split defined in Algorithm 2

  $blendingModel \leftarrow$ TRAINBLENDINGMODEL($train_{BlendingModel}$)

  function TRAINBLENDINGMODEL($train_{BlendingModel}$)
      $newFeatures \leftarrow$ PREDICTWITHBASEMODELS($baseModels, train_{BlendingModel}$)
      $train^*_{BlendingModel} \leftarrow$ JOIN($train_{BlendingModel}, newFeatures$).ON($cellId, timestamp$)
      $blendingView \leftarrow BlendingView$.BUILDFROM($train^*_{BlendingModel}$)
      $blendingModel \leftarrow$ BlendingModel()
      $blendingModel$.TRAIN($blendingView$)
      return $blendingModel$
  function PREDICTWITHBASEMODELS($baseModels$, $ds$)
                              ▷ Execution of GGM and GLMs code is independent and distributed
      $(ggm, glms) \leftarrow baseModels$
      $ggmView \leftarrow GGMView$.BUILDFROM($ds$)
      $glmsView \leftarrow GLMsView$.BUILDFROM($ds$)
          .PROJECT()
          .PARTITION()
          .SORT()
      $ggmPredictions \leftarrow ggm$.PREDICT($ggmView$)
      $glmsPredictions \leftarrow Dataset()$
      for $cellId$ in $glmsView$ do                                        ▷ parallel loop
          $glmsPredictions$.append($glms[cellId]$.PREDICT($glmsView[cellId]$))
      return JOIN($ggmPredictions, glmsPredictions$).ON($cellId, timestamp$)
```

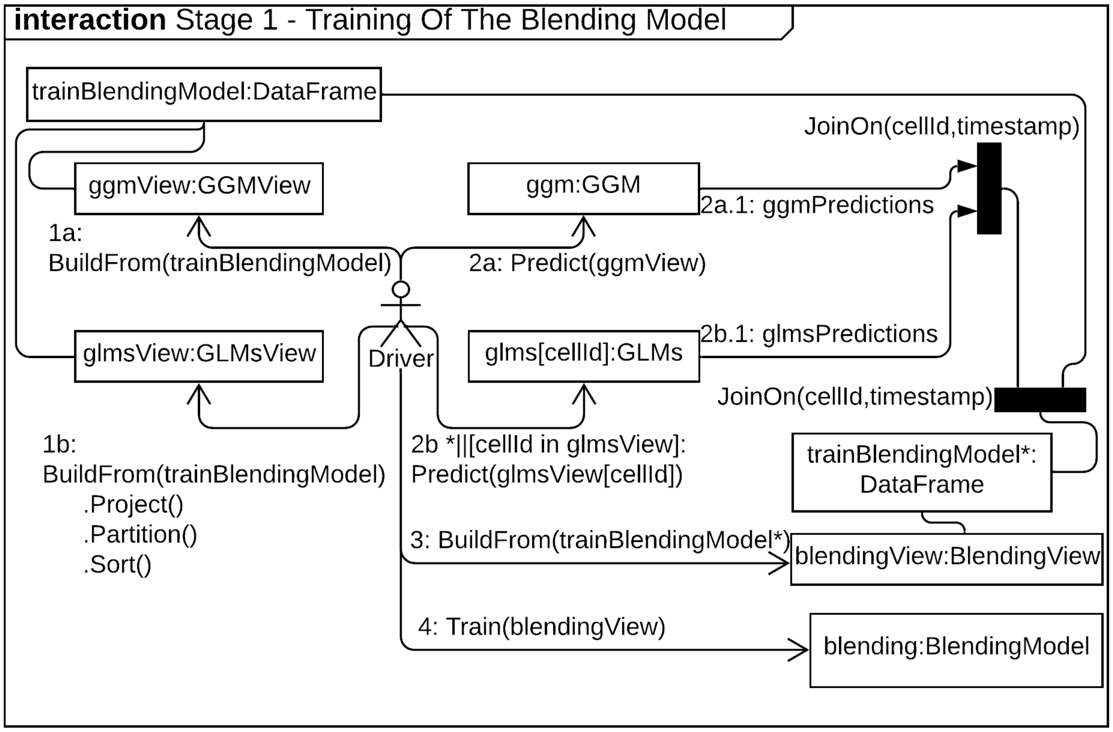

Figure 3.9: Stage 1 - Training Of The Blending Model.

dure is formalized in Algorithm 4 and illustrated in its accompanying diagram depicted in Figure 3.10.

In essence, each base model predicts a number, and the blending model considers these well-thought-out guesses in addition to the original features to predict the outcome of the framework. A concrete example of this would be forecasting visits to a coffee shop at a given *cellId* and *timestamp*. In this scenario, a GGM backed by a Stochastic Gradient Boosted Trees (GBT) ensemble could predict a value of 10 visits. A GLM backed by an ARIMA model at that *cellId* could predict 20. A blending model backed by another GBT could predict 12 visits as the final output of the framework, based on how much importance it has learned that the input and the base models have in predicting the outcome (i.e., the models' coefficients learned during the training).

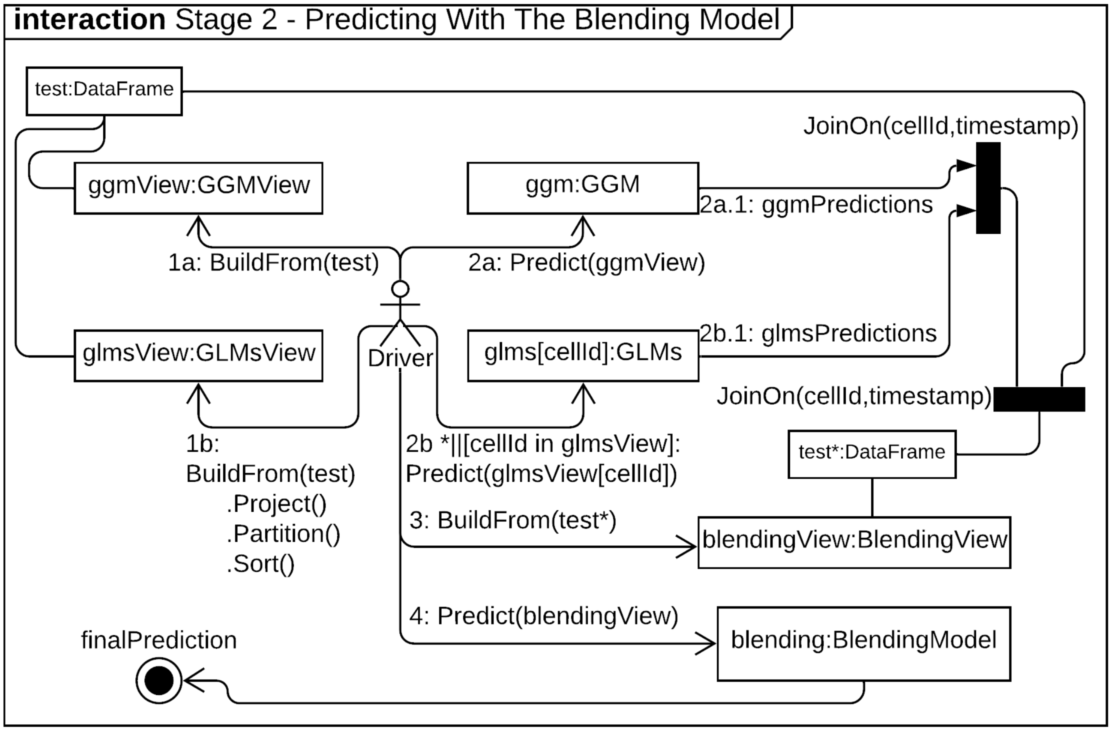

Figure 3.10: Stage 2 - Predicting With the Blending Model.

3.3 Summary

This chapter has presented the data preprocessing components of this work, as well as its multistage machine learning pipeline blocks. The data preprocessing contemplated the discretization of temporal and spatial variables via equal width discretization methods, the data cleaning via customizable temporal and spatial filters, and the large scale aggregation via a user-defined aggregation function that made it possible to choose arbitrary spatial and temporal resolutions. The multistage machine learning pipeline envisioned a geographically global model that took the entire training data as input, a collection of geographically local models each trained on its local time series, and a blending model that was responsible for issuing the final prediction of the framework based on the response of the other models.

Algorithm 4 Stage 2 - Predicting With the Blending Model

```
Require:
  baseModels to be the fitted base models defined in Algorithm 2
  blendingModel to be the fitted blending model defined in Algorithm 3
  test to be the dataset split defined in Algorithm 2
  PREDICTWITHBASEMODELS to be the function defined in Algorithm 3
  JOIN to be a function that joins datasets

  finalPrediction ← PREDICTWITHTHEBLENDINGMODEL(test)

  function PREDICTWITHTHEBLENDINGMODEL(test)
      newFeatures ← PREDICTWITHBASEMODELS(baseModels, test)
      test* ← JOIN(test, newFeatures).ON(cellId, timestamp)
      blendingView ← BlendingView.BUILDFROM(test*)
      finalPrediction ← blendingModel.PREDICT(blendingView)
      return finalPrediction
```

Chapter 4

Evaluation

This chapter covers the evaluation of the proposed framework. It starts by describing the implementation details of each component in the framework and discussing the supporting technologies. Finally, it presents three experiments, backed by different datasets, which shed light on the expected performance of the proposed research in diverse scenarios.

4.1 Implementation

This section presents implementation details that are pertinent to the development of the proposed research. The implementation of the proposed framework is guided by valuable software engineering principles: single-responsibility, maintainability, simplicity, and scalability.

The majority of its components are written in Scala[1], a statically typed, high-level language that runs on the Java Virtual Machine (JVM). Furthermore, most components are accompanied by an extensive test suite written with the ScalaTest[2] framework that verifies the desired behaviour, serves as documentation, and enables less error-prone modification of the codebase. Moreover, because components are designed with single responsibility and dependency inversion in mind, they are easier to maintain and evolve.

---

[1] https://www.scala-lang.org/

[2] http://www.scalatest.org/

The rest of this section is organized as follows: first, the packaging and deployment of the software is described; then a brief introduction to the underlying distributed computing framework is given; next, a high-level description of the implementation of the data preprocessing components is provided; finally, an overview of the implementation of the multistage machine learning pipeline is presented.

Software Packaging and Deployment

This research used Docker[3] container images in order to provide a unit of software that can be easily packaged and deployed into commodity hardware, providing a seamless deployment across machines in a cluster.

A Docker image is a lightweight software package that contains the application code and all its dependencies, enabling applications to run reliably even when transferred from one computing environment to another [11]. Docker containers, like virtual machines, allow for resource isolation (e.g., file system, network), but are much more lightweight than virtual machines [11]. This reduced resource footprint stems from the fact that containers run on top of a host Operating System (OS), as opposed to running on its own guest OS, as it is the case with virtual machines. Figure 4.1 illustrates this resource management strategy difference between Docker containers and virtual machines.

Distributed Computing Framework

The distributed computing foundation of the proposed research was implemented on top of Apache Spark, allowing the inner components of the framework to scale horizontally to meet the demanding resource requirements of Big Data [6].

Apache Spark[4] is an open-source distributed computing framework that employs a cluster of worker nodes to process tasks in parallel [7]. In Spark's programming model, a driver program acquires executors from a cluster manager and implements the control-flow of an ap-

---

[3] https://www.docker.com/

[4] https://spark.apache.org/

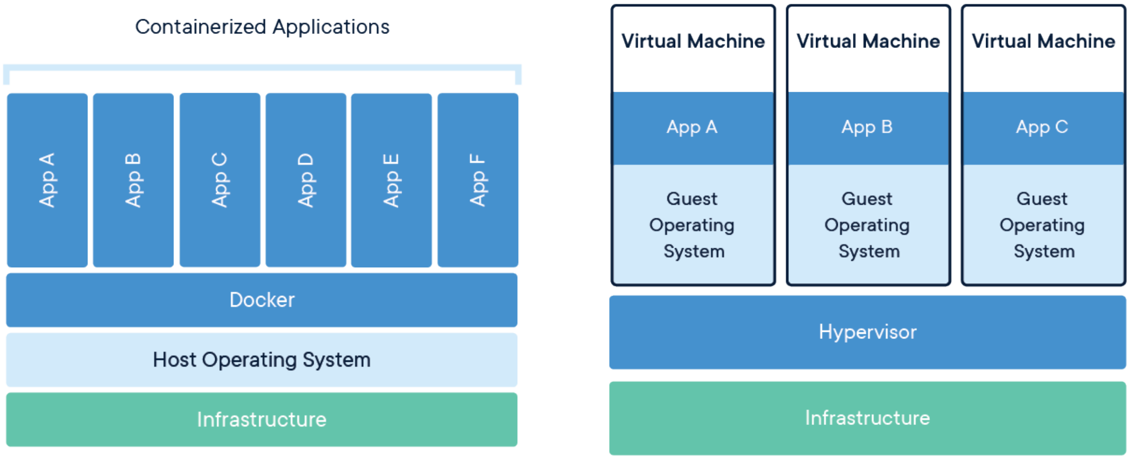

Figure 4.1: Docker Containers vs. Virtual Machines [25].

plication by handing off tasks to those executors. Architecturally, worker nodes are computing machines, executors are workers' processes that run tasks, and a task is a unit of work that is sent by the driver to one executor [70]. Figure 4.2 illustrates this architecture.

The Resilient Distributed Dataset (RDD) is Spark's abstraction of an immutable collection of records partitioned across machines [69]. RDDs also provide abstractions for a wide range of high-level, coarse-grained parallel operations, such as Map, Filter and Group-by, which can accommodate a diverse set of data workflows while providing scalability and fault tolerance. Together, these abstractions yield a remarkably powerful data-processing framework because they enable developers to execute high-level operations while abstracting distribution and fault tolerance concerns.

The sample code in Listing 4.1 illustrates a simplistic example where Apache Spark is utilized to read a spatiotemporal dataset and count only records that happened in Manhattan in 2019. In the code, the dataset itself is transparent to the user, who, independently of the original dataset size or record placement in each worker's partitions, can filter and request a count of the desired records. In the same code snippet, one filter accepts an equality-based comparison over the timestamp, while the other accepts a user-defined function that can verify whether a cell identifier lies within the Manhattan region. Spark's execution engine chains these filter statements and defers their execution until the ”count” action is issued. This lazy

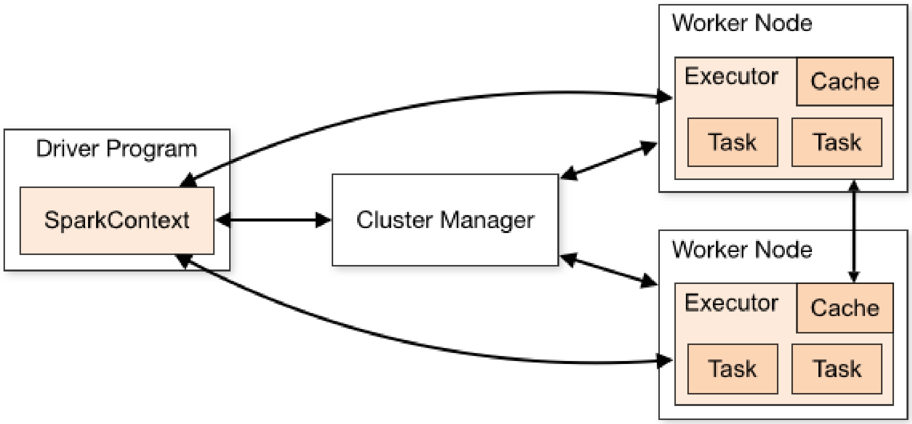

Figure 4.2: Spark Cluster Architecture Overview [60].

evaluation strategy allows for operations to be coupled and executed with a single pass through the dataset, saving substantial amounts of time in preprocessing tasks.

Listing 4.1: Sample of a short Spark program that filters spatiotemporal records that happened in Manhattan in 2019.

```
val dataset = spark.read.load(inputPath)
  .filter(year($"timestamp") === 2019)
  .filter(isInsideManhattan($"cell_id"))
  .count()
```

Data Preprocessing

At a high level, the data preprocessing components were implemented as follows:

Temporal Discretization. The temporal discretization component was implemented via a Map function over an RDD representing the input dataset. The mapping allowed for the framework to apply a temporal truncation across all records in the input.

Spatial Discretization. The spatial discretization component was implemented via a subsequent map on the same dataset, and because RDD operations are lazy, this did not incur in another pass through the dataset. This time, however, the map took a function based on the

S2Geometry[5] library that knows how to convert GPS coordinates into a cell_id at arbitrary resolutions.

Data Cleaning. The temporal filter block was implemented via a Filter operation that took a user-supplied JSON[6] document, like the one shown in Listing 4.2, with a start and end timestamps and kept only the pertinent records from the RDD. The spatial filter block was implemented via another filter operation that verified whether records' cell_ids lay inside a polygon of interest. That polygon was backed by an S2Geometry Polygon constructed from a user-supplied GeoJSON[7]. Listing 4.3 presents a fragment of such user input, where each pair of coordinates represents one corner of a geospatial polygon and the last pair is just a repetition of the first, which serves the purpose of connecting all segments into a triangular region.

Large Scale Aggregation. The large scale aggregation block was implemented via a Group-by operation that reduced the original RDD into an RDD of density counts at the specified temporal and spatial resolution. This aggregated dataset was then consolidated in persistent storage using a columnar format known as Parquet[8].

Listing 4.2: Example of a JSON document specifying a temporal filter.

```
{"start": "2019-08-01T00:00:00", "end":"2019-08-31T23:59:59"}
```

Listing 4.3: Example of a GeoJSON document describing a spatial filter made up of a triangular geographical region.

```
{
 "type": "FeatureCollection",
 "features": [
   {
     "type": "Feature",
     "properties": {},
```

---

[5] https://s2geometry.io/

[6] https://json.org/

[7] https://geojson.org/

[8] https://parquet.apache.org/

```
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [-81.2764048576355, 43.01475191944554],
            [-81.28818511962889,43.00210497752506],
            [-81.26443147659302,43.002152055337895],
            [-81.2764048576355, 43.01475191944554]
          ]
        ]
      }
    }
  ]
}
```

Multistage Machine Learning Pipeline

The following blocks portray an overview of the implementation of the machine learning core of this research:

Geographically Global Model (GGM). The GGM was implemented as a Gradient Boosted Tree Regressor from the Apache Spark machine learning library (MLLib[9]). MLLib's implementation of decision trees provided a distributed computation of variance that massively sped up the node splitting process.

Geographically Local Models (GLMs). The GLMs were implemented on top of ARIMA models from a third-party time series library, named SparkTS[10], open-sourced by Cloudera. The GLMs are themselves a persisted RDD of ARIMA models partitioned by cell_id, which

---

[9] https://spark.apache.org/mllib/

[10] https://github.com/sryza/spark-timeseries

are spatially joined with test records in order to issue predictions.

Blending Model. The blending model, like the GGM, is also backed by the MLLib, but it takes its input features from an intermediate RDD generated by the base models, as opposed to the original input.

Others. Dataset metrics gathering was implemented via SparkSQL[11] operations, the SMAPE evaluation metric was implemented via a custom Spark regression evaluator, and the time-sliding cross-validation was implemented via a SparkSQL window function.

4.2 Experiments

This section reports on experimental results that the proposed framework achieved in three diverse domains: taxi trips in New York City, crimes in the city of Chicago, and visits to places of interest across Canada. These domains are backed by spatiotemporal datasets, which in turn are also diverse in several relevant statistics, such as the number of records, time span, geographical region, and temporal and spatial sparsity. This variability in the characteristics of input datasets is welcomed and serves the purpose of demonstrating the merit of the proposed research in different scenarios.

Before the experiments can be presented, a few relevant terms need to be defined. With regards to sparsity statistics, this study utilizes the definitions of temporal sparsity and spatial sparsity formalized in equations (4.1) and (4.2), respectively. In Equation (4.1), the temporal sparsity of a given spatial cell is measured as one minus a fraction between the number of time intervals that the cell contained and the total number of time intervals in the dataset. In Equation (4.2), the spatial sparsity at a given time is measured as a fraction between the number of spatial cells with no neighbours and the total number of spatial cells in the dataset. In this study, these metrics are reported in their aggregate form (e.g., average, standard deviation), summarizing data across all available spatial cells for Equation (4.1), and across all temporal

---

[11] https://spark.apache.org/sql/

intervals for Equation (4.2).

$$\mathit{temporalSparsity(cell)} = 1 - \frac{\mathit{\#local\ temporal\ intervals}}{\mathit{\#global\ temporal\ intervals}} \tag{4.1}$$

$$\mathit{spatialSparsity(time)} = \frac{\mathit{\#spatial\ cells\ with\ no\ neighbours}}{\mathit{\#total\ spatial\ cells}} \tag{4.2}$$

As for the evaluation metrics, this study makes use of the standard regression metrics, Root Mean Squared Error (RMSE) and Symmetric Mean Absolute Percentage Error (SMAPE), formulated in equations (4.3) and (4.4), respectively. In those equations, n is the number of records, Y is the true value of the variable being predicted, and $\hat{Y}$ is the predicted value:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \tag{4.3}$$

$$\text{SMAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|Y_i - \hat{Y}_i|}{|Y_i| + |\hat{Y}_i|} \tag{4.4}$$

The last definition necessary for understanding the experiments is the definition of the Naive Model. In this study, the Naive Model is an overly simplistic heuristic commonly used as a baseline in one-step time series forecasts, which predicts the future value of a variable to be equal to its current value. For example, under this naive baseline model, a forecast for the next hour $t + 1$ would output the variable value at t. This naive model tends to be successful if there is not enough variability between consecutive values in a time series, and exposing its performance in an experiment helps in evaluating how much the proposed learning technique actually learned.

Following are self-contained reports of experiments that ran in a three-machine cluster with 24-cores and 94GB of memory each.

Number of Records	Over 1.3 billion
Time Span	2009 to 2018
Geographical Region	New York City
Provider	New York City Taxi and Limousine Commission

Table 4.1: NYC Taxi Trips – Dataset Statistics

4.2.1 New York City Taxi Trips

The New York City Taxi and Limousine Commission (TLC) provides an open dataset containing taxi trips in the city of New York since 2009. The more than 1.3 billion records available have information on the time and GPS coordinates of where pickups and drop-offs happened, making it one of the most massive spatiotemporal datasets that are openly available to the public. This dataset and accompanying schemas can be downloaded from the NYC-TLC data portal[12]. A summary of a few characteristics of this dataset in its raw form, with no alteration in comparison to the original source, are reported in Table 4.1.

Data Preprocessing

For this experiment, the Data Preprocessing block was implemented as follows:

Temporal Discretization. The temporal component was discretized into hourly bins to enable one-step forecasts of one hour, giving enough time for drivers to position themselves in order to better serve customers.

Spatial Discretization. The spatial component was discretized via a square grid with cells having an average area of 79000 m^2 to account for the mobility range of a car.

Data Cleaning. Data were filtered to be contained in the interval from June 2014 to June 2016 and within the polygon representing the Manhattan region, as shown in Figure 4.3. The cut-off point at 2016 was necessary because the data provider stopped publishing fine location data to avoid privacy issues, only reporting on coarse locations since then.

Large Scale Aggregation. After the discretization process, over 400 million records were aggregated into hourly counts for each spatial cell, resulting in a much smaller dataset with

---

[12] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Number of Records	Over 9.6 million	
Time Span	June 2014 to June 2016	
Temporal Resolution	Hour	
Number of Temporal Intervals	Over 17.5k	
Temporal Sparsity	Avg = 36.8%	Std = 37.8%
Geographical Region	Manhattan	
Spatial Resolution	Cells with an average area of 79000 m^2	
Number of Spatial Cells	Over 800	
Spatial Sparsity	Avg = 0.8%	Std = 3.0%

Table 4.2: NYC Taxi Trips – Dataset Statistics After Preprocessing.

just over 9.6 million records. Furthermore, the discretized dataset is characterized by a low temporal sparsity averaging 36.8% and an extremely low spatial sparsity averaging 0.8%. In other words, on average, each spatial cell is missing 36.8% of the timestamps contained in the global timeline, and only 0.8% of the spatial cells do not have at least one neighbour at any given time. Summary statistics on the final dataset after preprocessing are reported in Table 4.2.

Multistage Machine Learning Pipeline

After preprocessing, the dataset was split into 81% training, 9% blending training, and 10% testing. The larger training set potentially allows for the learning algorithms to be exposed to more patterns, whereas the smaller blending training set avoids overfitting the base models contributions. The remaining records, which are numbered in the high hundreds of thousands, were reserved for testing. Next, the machine learning models were trained according to the following scheme:

Geographically Global Model (GGM). One stochastic gradient boosted tree ensemble was trained over the entire training set. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the following model hyperparameters: maximum depth, number of iterations, and maximum number of bins. The considered values for these parameters are reported in Table 4.3.

Geographically Local Models (GLMs). The training dataset was partitioned into one time

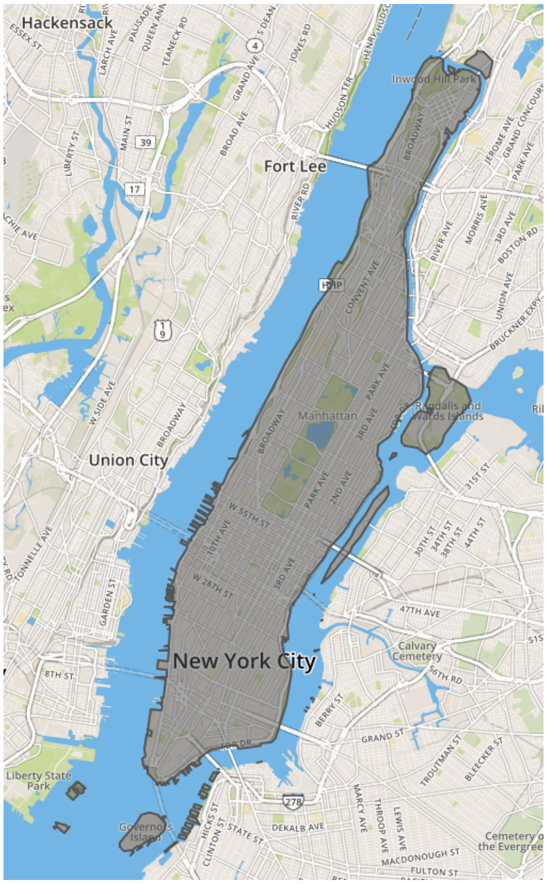

Figure 4.3: Polygons in grey representing the spatial filter over the Manhattan region.

Hyperparameter	Considered Values
Maximum Depth	[3, 5, 6, 10, 15, 20, 25]
Number of Iterations	[10, 20, 50, 100, 150, 200, 250]
Maximum Number of Bins	[32, 64, 128, 1024, 2048]

Table 4.3: GGM – Stochastic Gradient Boosted Trees Hyperparameters.

series per spatial cell at the 79000 m^2 resolution. Then one ARIMA model was trained on each of these cells and tuned according to the Akaike Information Criterion (AIC), considering maximum values of p, d, and q to be 5, 2, and 5 respectively. Such thresholds were based on Nau [48], who stated that in most cases, the optimal values for these parameters are small integers where $p + q \leq k$ for some small integer k, and that most of the time, either p or q is zero.

Blending Model. One stochastic gradient boosted tree ensemble was trained over the blending training set, taking as input features the base-learners' predictions, the cell id, the hour, and the day of the week. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the same hyperparameters as the GGM. The considered values for these parameters were taken from the lower half of the ranges utilized for the GGM.

Results

The metrics reported in this section result from applying a one-hour time-sliding window to a test set (i.e., 10% of the preprocessed data) comprising over 1800 hours from April to June 2016 and more than 800 different spatial cells. The measured RMSE and SMAPE values are reported for the naive model, the base-learners (GGM, GLMs), and the blending model.

Figure 4.4 illustrates the SMAPE for all models in question. Noticeably, the blending model sits at the bottom with the lowest error and slightly close to the GGM. Also, it can be seen that the GLMs clearly performed much better than the naive model, but seemed to struggle during off-peak hours (7 pm to 3 am). In fact, all the evaluated models struggled to some extent because fewer taxi trips tend to happen at off-peak hours, leaving relatively fewer training records available for the models to learn.

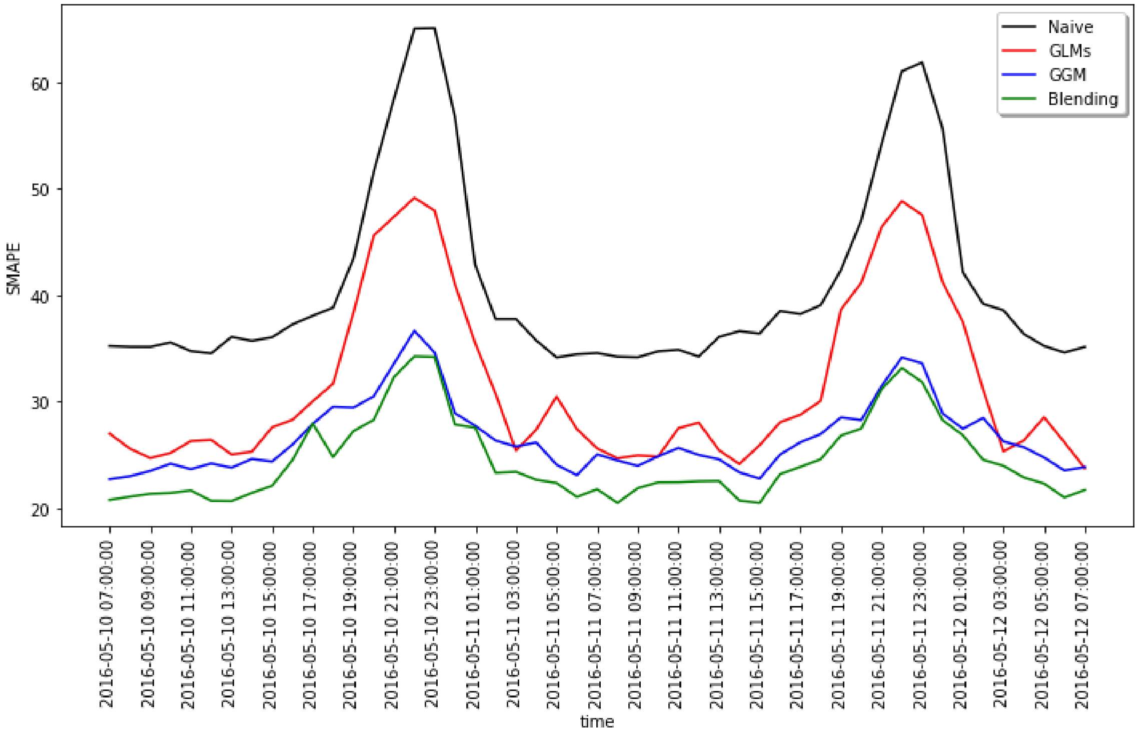

Figure 4.4: NYC Taxi Trips – Average SMAPE per hour on a small sample of the testing data. Timestamps are in the local timezone (EST).

Figure 4.5 depicts the RMSE for all models in question and offers more insight into the off-peak behaviour. The ranked performance of the models remains the same, with the blending model still having the lowest error, followed by the GGM, GLMs, and naive models. However, the RMSE, unlike the SMAPE, decreases during off-peak hours. This pattern might seem counter-intuitive, but it can be explained by the fact that the overall scale of the label is weaker in most places during the evening and the RMSE captures that reduction in scale, whereas the SMAPE, because it is scale-invariant, does not.

Table 4.4 summarizes the average and standard deviation of both RMSE and SMAPE for all models across the iterations of the sliding window. Thus, for the entirety of the test set, and according to both metrics: the naive model performs the worst; the GLMs are worse than the GGM by a small amount, and yet with lower standard deviation; and the proposed blending model performs the best on both metrics while also maintaining the lowest standard deviations. Figure 4.6 also illustrates the performance of the models and point to higher stability of the

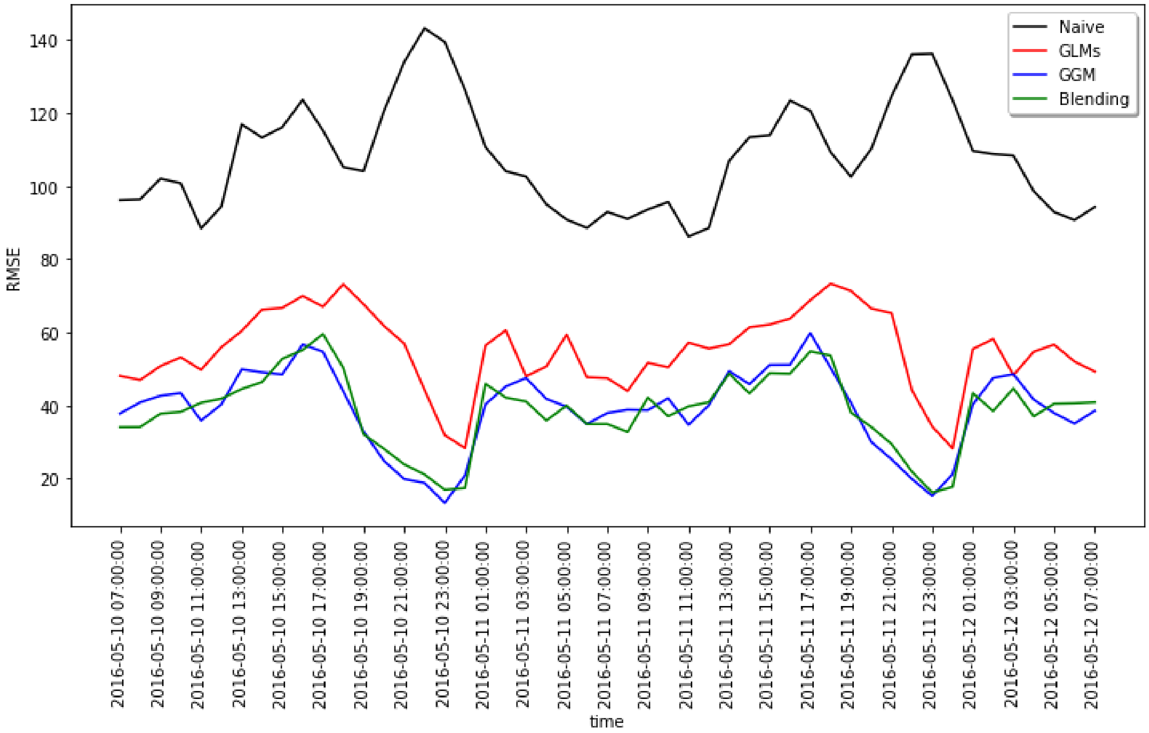

Figure 4.5: NYC Taxi Trips – Average RMSE per hour on a small sample of the testing data. Timestamps are in the local timezone (EST).

Model	Avg(RMSE)	Std(RMSE)	Avg(SMAPE)	Std(SMAPE)
Naive	106.18	20.75	40.02%	9.89%
GLMs	56.50	14.77	32.11%	7.56%
GGM	46.60	16.20	31.00%	10.53%
Blending	42.78	13.57	26.70%	4.02%

Table 4.4: NYC Taxi Trips – Model Metrics Across All Testing Windows

blending model as captured by the range between the whiskers.The lower performance of the naive model points to significant variability between consecutive labels. In other words, merely using the labels from the last hour to make predictions yielded weak results, suggesting a more complex relationship between the label and the time. The performance of the base models in comparison to the baseline (the naive model) shows that they were more successful in capturing patterns in the data. Furthermore, the performance of the blending model shows that enough diversity existed among the base models so that the blend could achieve far surpassing results.

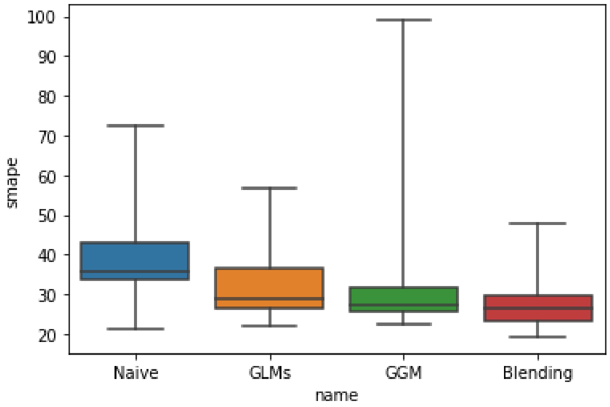

Figure 4.6: NYC Taxi Trips – SMAPE box plot. Blending model has the lowest error and the highest stability (smallest range between whiskers).

Number of Records	Almost 7 million
Time Span	2001 to present
Geographical Region	Chicago
Provider	Chicago Police Department

Table 4.5: Chicago Crimes – Dataset Statistics.

4.2.2 Crimes in The City of Chicago

The Chicago Police Department provides an open dataset, originating from its CLEAR (Citizen Law Enforcement Analysis and Reporting) system, that contains crime incident records in the city of Chicago since 2001. The almost 7 million records available have information on the time and GPS coordinates (at the city block level) of where incidents happened, making it one of the largest geolocated crime datasets available to the public. This dataset and accompanying schemas can be downloaded from the city of Chicago open data portal [13]. A summary of a few characteristics of this dataset in its raw form, with no alteration in comparison to the original source, are reported in Table 4.5.

Data Preprocessing

For this experiment, the Data Preprocessing block was implemented as follows:

Temporal Discretization. The temporal component was discretized into daily bins to enable

---

[13] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

Number of Records	Over 4.5 million	
Time Span	January 2001 to May 2019	
Temporal Resolution	Day	
Number of Temporal Intervals	Over 6.7k	
Temporal Sparsity	Avg = 72.82%	Std = 20.91%
Geographical Region	City of Chicago	
Spatial Resolution	Cells with an average area of 0.32 km^2	
Number of Spatial Cells	Over 2.4k	
Spatial Sparsity	Avg = 8.29%	Std = 4.69%

Table 4.6: Chicago Crimes – Dataset Statistics After Preprocessing.

one-step forecasts of one day, giving a reasonable time for police officers to plan patrol routes.

Spatial Discretization. The spatial component was discretized via a square grid, with cells having an average area of 0.32 km^2 to account for areas that are large enough to exhibit recurrent crime patterns.

Data Cleaning. Data were filtered to be contained in the interval from January 2001 to May 2019 and within the polygon representing the Chicago region, as seen in Figure 4.7.

Large Scale Aggregation. After the discretization process, more than 6.8 million records were aggregated into daily counts for each spatial cell, resulting in a substantially smaller dataset with just over 4.5 million records. Furthermore, the discretized dataset was characterized by an extremely high temporal sparsity averaging 72.82%, and a low spatial sparsity averaging 8.29%. In other words, on average, each spatial cell misses 72.82% of the timestamps contained in the global timeline, and only 8.29% of the spatial cells do not have at least one neighbour at any given time. Summary statistics on the final dataset after preprocessing are reported in Table 4.6.

Multistage Machine Learning Pipeline

After preprocessing, the dataset was split into 81% training, 9% blending training, and 10% testing. The larger training set potentially allows for the learning algorithms to be exposed to more patterns, whereas the smaller blending training set avoids overfitting the base models contributions. The remaining records, which numbered several hundreds of thousands, were

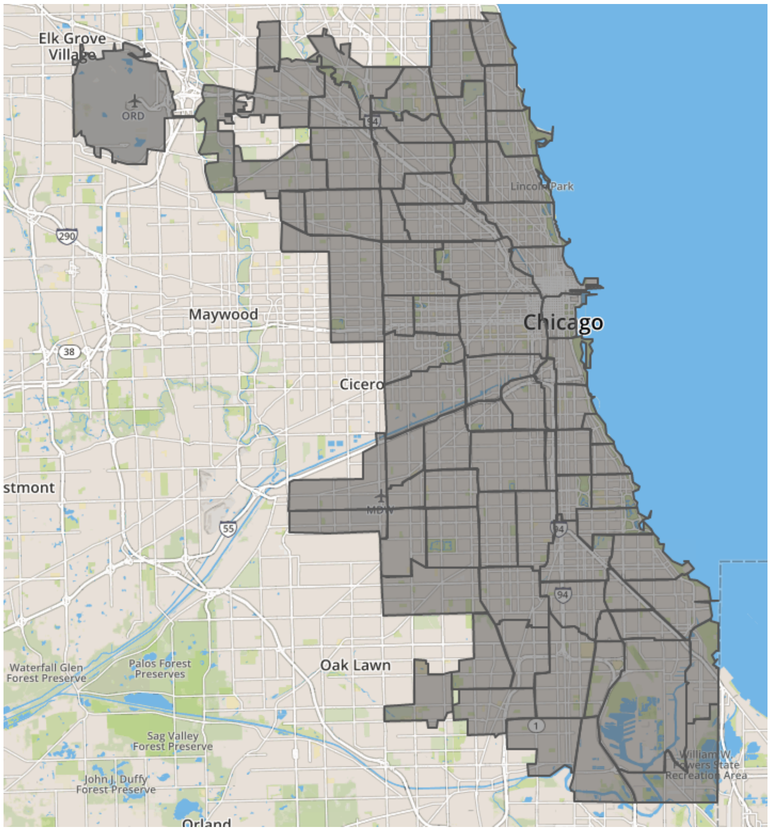
Elk Grove Village
ORD
290
Maywood
38
Cicero
stmont
55
MDW
Chicago
Lincoln Park
94
Waterfall Glen Forest Preserve
Palos Forest Preserves
Sag Valley Forest Preserve
John J. Duffy Forest Preserve
Oak Lawn
Orland
William W. Powers State Recreation Area

Figure 4.7: Polygons in grey representing the spatial filter over the Chicago region.

reserved for testing. Next, the machine learning models were trained according to the following scheme:

Geographically Global Model (GGM). One stochastic gradient boosted tree ensemble was trained over the entire training set. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the following model hyperparameters: maximum depth, number of iterations, and maximum number of bins. The considered values for those parameters are reported in Table 4.3. For clarity, this was done in the exact same way as the correspondent step in the NYC Taxi Trips experiment.

Geographically Local Models (GLMs). The training dataset was partitioned into one time series per spatial cell at the 1.27 km^2 resolution. Then one ARIMA model was trained on each of those cells and tuned according to the Akaike Information Criterion (AIC), considering maximum values of p, d, and q to be 5, 2, and 5 respectively. These thresholds were based on Nau [48], who stated that in most cases, the optimal values for these parameters are small integers where $p + q \leq k$ for some small integer k, and that most of the time, either p or q is zero.

Blending Model. One stochastic gradient boosted tree ensemble was trained over the blending training set, taking as input features the base-learners' predictions, the cell id, the day, and the day of the week. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the same hyperparameters as the GGM. The considered values for these parameters were taken from the lower half of the ranges utilized for the GGM. For clarity, this was done in almost the same way as the corresponding step in the NYC Taxi Trips experiment, except for not including the "hour" as an input feature because this dataset was discretized at the "day" resolution.

Results

The metrics reported in this section are the result of applying a one-day time-sliding window to a test set (i.e., 10% of the preprocessed data) comprising over 670 days from July 2017 to May 2019 and over 2.4k different spatial cells. The measured RMSE and SMAPE values are

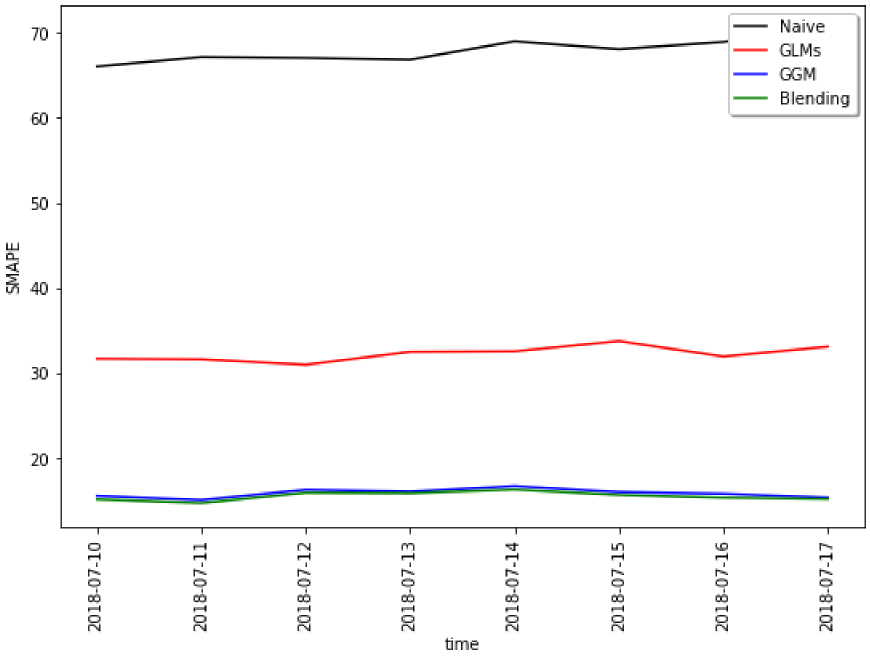

Figure 4.8: Chicago Crimes – Average SMAPE per day on a small sample of the testing data.

reported for the naive model, the base learners (GGM, GLMs), and the blending model.

Figure 4.8 illustrates the SMAPE for all models in question. Noticeably, the blending model sits at the bottom with the lowest error and significantly close to the GGM. Also, the GLMs performed much better than the naive model, but that they still underperformed with respect to the others by a significant margin. This impact on the prediction capabilities of the GLMs can be explained by the increased difficulty of forecasting over a highly sparse time series.

Figure 4.9 depicts the RMSE for all models in question. The ranked performance of the models remains about the same, with the naive model and the GLMs with the highest error, and the GGM with slightly less error than the blending model for that particular window. Furthermore, the higher standard deviation of labels on the dataset for days of the week from Thursday through Saturday resulted in more challenges for the GLMs, which rely purely on time series forecasting. The GGM and blending models, however, were not as profoundly influenced by these changes in distribution because they likely chose spatial patterns that outweighed them.

Table 4.7 summarizes the average and standard deviation of both RMSE and SMAPE, for all models, across the iterations of the sliding window. For the entirety of the test set, and

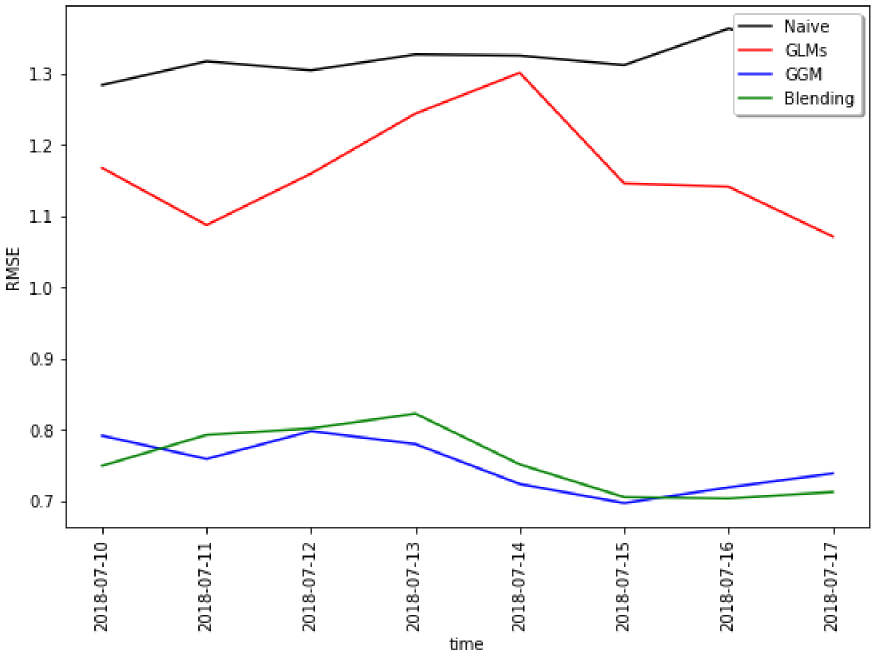

Figure 4.9: Chicago Crimes – Average RMSE per day on a small sample of the testing data.

according to both metrics: the naive model performed the worst; the GLMs were worse than the GGM by a significant amount; the proposed blending model performed slightly better than the best base model on both metrics while also maintaining the lowest standard deviations. Figure 4.10 also illustrates the performance of the models and point to higher stability of the blending model as captured by the range between the whiskers. The lower performance of the naive model points to significant variability between consecutive labels. In other words, merely using the labels from the last day to make predictions yielded weak results, suggesting a more complex relationship between the label and the time. The performance of the base models in comparison to the baseline (the naive model) shows that they were more successful in capturing patterns in the data; however, the GLMs were worse than the GGM by a fair margin. One possible reason for this gap in performance is that crime events may be more correlated with space than they are with time, weakening the temporal patterns that the GLMs could leverage, and therefore impairing its performance. Furthermore, the performance of the blending model shows that even when one of the base models was worse than the other by a fair margin, the blend itself still maintained a better performance, serving as a guard to the

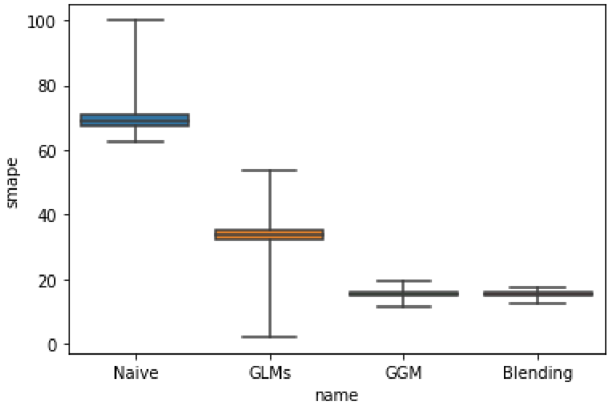

Figure 4.10: Chicago Crimes – SMAPE box plot. Blending model has the lowest error and the highest stability (smallest range between whiskers).

Model	Avg(RMSE)	Std(RMSE)	Avg(SMAPE)	Std(SMAPE)
Naive	1.33	0.11	69.50%	3.77%
GLMs	1.17	0.16	33.88%	3.07%
GGM	0.76	0.18	15.60%	0.86%
Blending	0.74	0.18	15.32%	0.77%

Table 4.7: Chicago Crimes – Model Metrics Across All Testing Windows

overall performance of the *Spatiotemporal Forecasting Framework*.

4.2.3 Visits to Places of Interest Across Canada

An industry partner in the mobile application space provided a dataset containing visits to places of interest across Canada in the period from June to December 2018. The more than 175 million records available include information on the number of visits, the time, and the GPS coordinates of the place where they happened, making this the dataset with the largest spatial coverage that the authors could find. With respect to privacy issues, because the dataset only reports aggregated visit counts from the point of view of an arbitrary location, there is no personal identifiable information that could trace a visit back to a user. Hence, users are completely anonymous and only places can be analyzed. A summary of a few characteristics of this dataset in its raw form, with no alteration in comparison to the original source, is given in Table 4.8.

Number of Records	Over 175 million
Time Span	June to December 2018
Geographical Region	Canada
Provider	Industry Partner

Table 4.8: Visits to Places of Interest Across Canada – Dataset Statistics.

Data Preprocessing

For this experiment, the Data Preprocessing block was implemented as follows:

Temporal Discretization. The temporal component was discretized into hourly bins to enable one-step-ahead forecasts of one hour, giving enough time to support a variety of short-term optimization tasks, such as anticipating the number of staff needed to serve customers better during peak hours in a coffee shop.

Spatial Discretization. The spatial component was discretized via a square grid with cells having an average area of 79000 m^2 to account for areas that are large enough to exhibit recurrent visit patterns.

Data Cleaning. Data were filtered to be contained in the interval from June to December 2018 and within the polygon representing the Canadian borders, as shown in Figure 4.11. The shorter timeframe in this experiment is due only to less recent data not being available. This shortage of historical information is caused by the relatively recent start of our industry partner's effort to collect such data; a shared reality with many other mobile companies that recently understood the value in spatiotemporal data.

Large Scale Aggregation. After the discretization process, more than 175 million records were aggregated into hourly counts for each spatial cell, resulting in a much smaller dataset with just over 33 million records. Furthermore, the discretized dataset is characterized by an extremely high temporal sparsity averaging 79.45% and a high spatial sparsity averaging 51.64%. In other words, on average, each spatial cell misses 79.45% of the timestamps contained in the global timeline, and 51.64% of the spatial cells do not have at least one neighbour at any given time. Summary statistics on the final dataset after preprocessing are reported in Table 4.9.

Number of Records	Over 33 million	
Time Span	June to December 2018	
Temporal Resolution	Hour	
Number of Temporal Intervals	Over 5k	
Temporal Sparsity	Avg = 79.45%	Std = 19.71%
Geographical Region	Canada	
Spatial Resolution	Cells with an average area of 79000 m^2	
Number of Spatial Cells	Over 31.5k	
Spatial Sparsity	Avg = 51.64%	Std = 19.57%

Table 4.9: Visits to Places of Interest Across Canada – Dataset Statistics After Preprocessing.

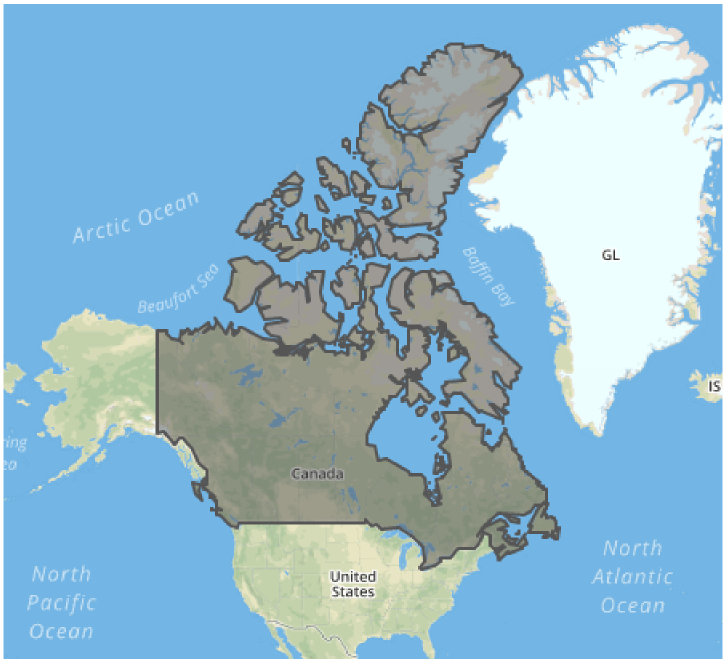

Figure 4.11: Polygons in grey representing the spatial filter across the Canadian landscape.

Multistage Machine Learning Pipeline

After preprocessing, the dataset was split into 81% training, 9% blending training, and 10% testing. The larger training set potentially allowed for the learning algorithms to be exposed to more patterns, whereas the smaller blending training set avoided overfitting the base models contributions. The remaining records, which numbered in the millions, were reserved for testing. Next, the machine learning models were trained according to the following scheme:

Geographically Global Model (GGM). One stochastic gradient boosted tree was trained over the entire training set. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the following model hyperparameters: maximum depth, number of iterations, and maximum number of bins. The considered values for these parameters are reported in Table 4.3. For clarity, this was done in the exact same way as the correspondent step in the NYC Taxi Trips, and Chicago Crimes experiments.

Geographically Local Models (GLMs). The training dataset was partitioned into one time series per spatial cell at 81 km^2 resolution. Then one ARIMA model was trained on each of these cells and tuned according to the Akaike Information Criterion (AIC), considering maximum values of p, d, and q to be 5, 2, and 5 respectively. These thresholds were based on Nau [48], who stated that in most cases the optimal value for these parameters are small integers where $p + q \leq k$ for some small integer k, and that most of the time, either p or q is zero.

Blending Model. One stochastic gradient boosted tree was trained over the blending training set, taking as input features the base-learners' predictions, the cell id, the hour, and the day of the week. A grid-search with 3-fold time-sliding cross-validation was utilized to tune the same hyperparameters as the GGM. The considered values for these parameters were taken from the lower half of the ranges utilized for the GGM. For clarity, this was done in the exact same way as the corresponding step in the NYC Taxi Trips experiment.

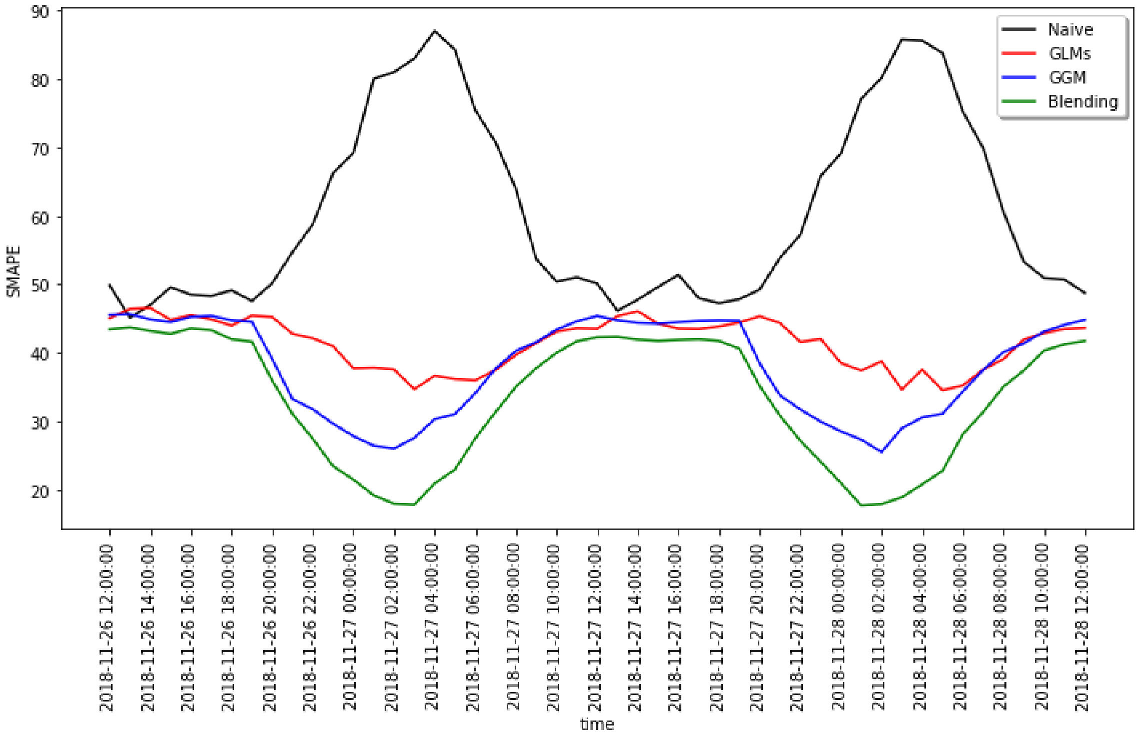

Figure 4.12: Visits to Places of Interest Across Canada – Average SMAPE per hour on a small sample of the testing data. Timestamps are in the associated local timezone.

Results

The metrics reported in this section are the result of applying a one-hour time-sliding window to a test set (i.e., 10% of the preprocessed data) covering over 630 hours from November to December 2018 and over 27k different spatial cells. The measured RMSE and SMAPE values are reported for the naive model, the base-learners (GGM, GLMs), and the blending model.

Figure 4.12 illustrates the SMAPE for all models in question. Noticeably, the blending model sits at the bottom with the lowest error and slightly close to the GGM. Also, the GLMs performed much better than the naive model and had only a small variance across hours. Furthermore, except for the naive model, all others showed an increased error during working hours (9 am to 6 pm), which correlated with an increased volume and variability of test records.

Figure 4.13 depicts the RMSE for all models in question. The ranked performance of the models remained the same, with the blending model still having the lowest error, followed by

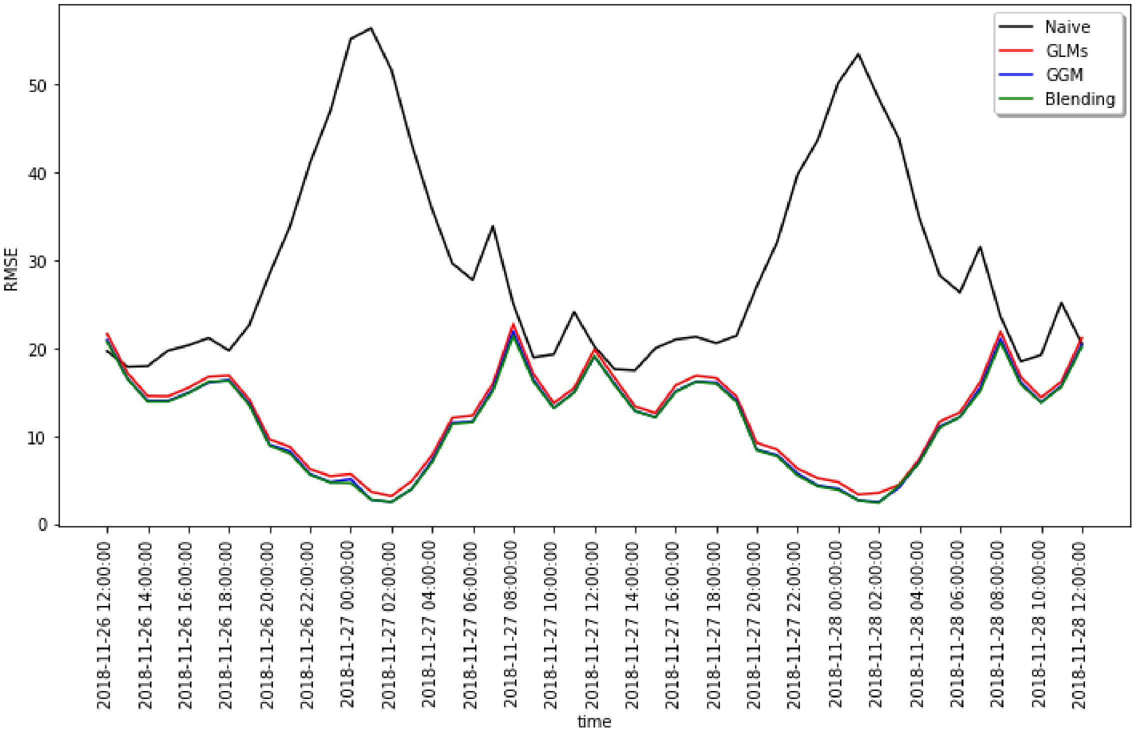

Figure 4.13: Visits to Places of Interest Across Canada – Average RMSE per hour on a small sample of the testing data. Timestamps are in the associated local timezone.

the GGM, GLMs, and naive models. However, this time, the residual errors of the blending, GGM, and GLMs models are much closer, indicating a greater agreement in the scale of their predictions. Nevertheless, their predictions are not precisely the same and are similar only when some margin is considered. This marginal difference is better captured by the earlier mentioned SMAPE residuals depicted in figure 4.12.

Table 4.10 summarizes the average and standard deviation of both RMSE and SMAPE, for all models, across the iterations of the sliding window. For the entirety of the test set, and according to both metrics: the naive model performed the worst; the GGM and blending models were close in terms of RMSE, and the GLMs were also somewhat close; the blending model outperformed all others in terms of SMAPE, but held higher standard deviations than its base models. Despite the mildly increased standard deviation in its SMAPE, the blending model was reasonably stable in comparison to others, with a 95% confidence interval bound between 32.74 and 35.12. For reference, the corresponding confidence intervals for the naive,

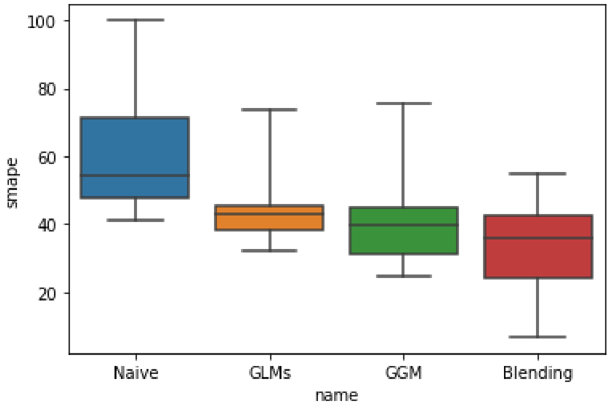

Figure 4.14: Visits to Places of Interest Across Canada – SMAPE box plot. Blending model has lower errors at stability comparable to the GGM.

Model	Avg(RMSE)	Std(RMSE)	Avg(SMAPE)	Std(SMAPE)
Naive	28.61	13.04	59.93%	14.24%
GLMs	12.84	6.12	42.06%	4.47%
GGM	12.28	6.16	38.50%	7.82%
Blending	12.18	6.18	33.93%	9.93%

Table 4.10: Visits to Places of Interest Across Canada – Model Metrics Across All Testing Windows

GLMs, and GGM models were [58.63, 61.22], [41.66, 42.47], and [37.79, 39.21] respectively. This stability aspect is also illustrated in Figure 4.14, where the blending model demonstrates lower errors at stability comparable to the GGM.

4.2.4 Discussion

This section, has presented three diverse experiments. They varied greatly with respect to the domain, the number of records, time span, geographical region, temporal resolution, spatial resolution, and even more eminently, with respect to temporal and spatial sparsity.

The experiments explored the domains of taxi trips, crime, and visits to places of interest, showing different emerging spatiotemporal patterns in each, while still reporting comparable performance, demonstrating the domain-agnostic capacities of the *Spatiotemporal Forecasting Framework*.

Furthermore, the three experiments demonstrated the capabilities of the *Spatiotemporal*

Forecasting Framework in scaling to as many as 1.3 billion records in the Data Preprocessing component and 33 million records in the Multistage Machine Learning Pipeline.

They also portrayed the framework's ability to support time spans of tens of years and geographical regions as large as one of the world's largest countries, while enabling unconstrained flexibility in the choice of temporal (e.g., day, hour) and spatial resolutions (e.g., $79000m^2$, $0.32km^2$). To the best of our knowledge, this work is the first to allow such freedom in framing the forecasting question at the Big Data scale.

Figure 4.15 gathers the reported SMAPE averages from tables 4.2, 4.6, 4.9 for a direct comparison across experiments. Clearly, the naive model generally has poor performance, and especially in the Chicago Crimes experiment, it produced the worst results, possibly because the label had low correlation with the last measured value. It is also evident that the blending model improved upon the performance of the GGM and GLMs across all experiments, although less so in the Chicago Crimes experiment because, like the naive model, the GLMs also faced difficulties in extracting temporal patterns from highly sparse time series. Furthermore, considering the SMAPE residuals and the same order as the presented experiments, the blending model improved performance by about: 14%, 2%, and 12% with respect to the GGM; and about 17%, 55%, and 20% with respect to the GLMs. Hence, the blending model successfully boosted the overall performance of the framework by improving upon the performance of the best base model and also added robustness to the framework by lifting performance even when one of the base models produced less accurate forecasts. Moreover, even though the GGM outperformed the GLMs across experiments, the reverse could have happened in different datasets, and because the *Spatiotemporal Forecasting Framework* makes use of a blending model and does not rely on a single type of learning algorithm, stability could still be expected.

With respect to the temporal sparsity, it can be seen how its rise (36.8%, 72.82%, 79.45%) across the experiments hindered the performance of the GLMs (SMAPE: 32.11%, 33.88%, and 42.8%) because there were fewer consistent temporal patterns from which to learn. With respect to the spatial sparsity, lower values (0.8%, 8.29%) led to better performance for the

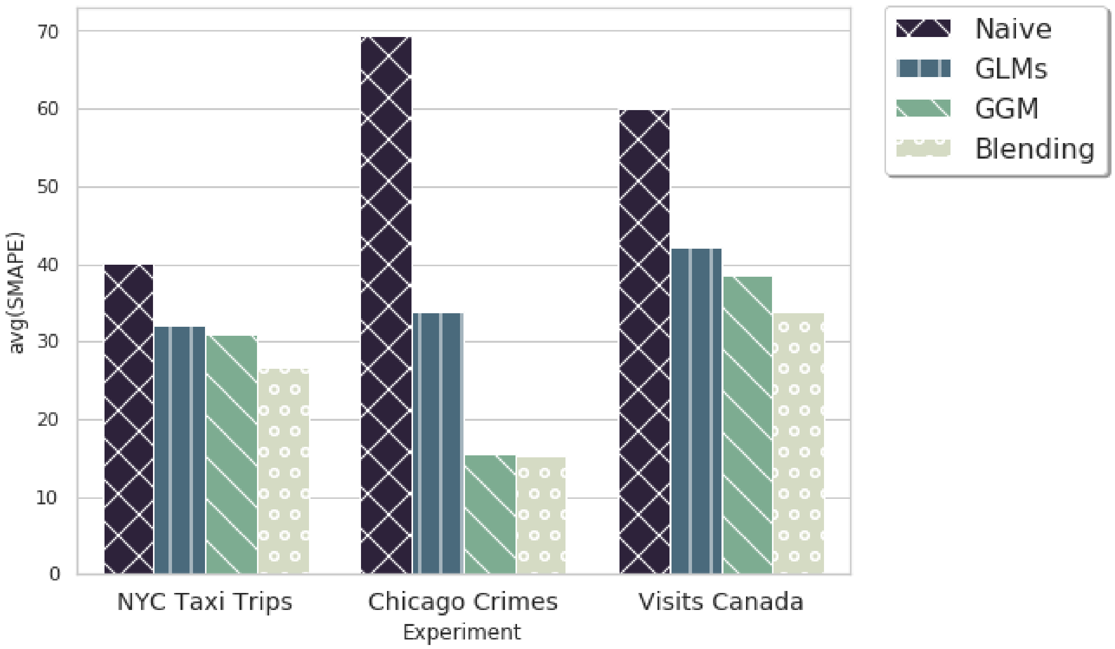

Figure 4.15: Average SMAPE per model across experiments.

GGM and GLM models in the NYC Taxi Trips and Chicago Crimes experiments, but a higher value (51.64%) led to increased errors for these in the Visits Canada experiment. Because this sparsity is a consequence of the type of reported events, more so than the data collection methods utilized to collect them, it is possible that by using more data, although probably not drastically changing sparsity, could lead to improved performance. For example, an extended time span in the training set could have improved the results from the NYC Taxi Trips and Visits Canada experiments by improving the chance of repeating patterns appearing. Therefore, it is advised that users of the framework acknowledge the challenges that spatial and temporal sparsity pose to spatiotemporal forecasting tasks and inspect for these characteristics in their gathered datasets beforehand to know what can be expected in terms of forecasting performance.

4.3 Summary

This chapter delved into the implementation of the *Spatiotemporal Forecasting Framework*. It described how the framework was packaged and deployed to commodity hardware, how a distributed computing foundation was leveraged to scale its internal components, and at a high level, how each component was implemented. Finally, this chapter presented three experiments that demonstrated the merits of the *Spatiotemporal Forecasting Framework* in diverse forecasting scenarios.

Chapter 5

Conclusions and Future Work

This chapter closes this thesis with conclusions about the proposed research and its achievements. In addition, it points to a few areas that can be explored in future work.

5.1 Conclusions

This research has proposed a framework built on top of a distributed computing cluster to tackle the high-volume datasets that often accompany spatiotemporal forecasting tasks. The architectural choice of using Apache Spark as the backbone of the framework enabled it to scale its internal components to process as many as 1.3 billion records during the *Data Preprocessing* stage and 33 million records during the *Multistage Machine Learning Pipeline* stage.

To put the user in complete control of the forecasting question, this research designed several components that enable unconstrained configuration of temporal and spatial resolution, time span, geographical extension, and data cleaning routines, all this by accepting user-provided JSON and GeoJSON documents without necessary changes to the code base. Notably, changes in temporal resolutions are supported via fast POSIX time truncations, and changes in spatial resolutions are supported via a Hilbert curve that indexes the Earth. The authors believe that providing such flexibility is the right path to take despite its potential impact on the system load. As a counterbalance, this approach enables seamless scaling-out of the cluster via the addition of Apache Spark workers. The training strategies presented in the multistage machine learning pipeline enabled the learning components to be scaled to millions of records. The Geographically Global Model (GGM) took advantage of a distributed implementation of the Stochastic Gradient Boosted Trees (GBT) ensemble, and of Hilbert indexing over the globe, to train several decision trees comprising the entire observed geographical space. The Geographically Local Models (GLMs) took advantage of Local Learning to train independent ARIMA models that otherwise could not even have fitted the data because of their memory constraints. The blending model employed in the final stage of the machine learning pipeline, and implemented as another GBT, enabled a non-linear combination of the GGM and GLMs predictions, resulting in higher performance and stability for the framework.

Three experiments gave credibility to this research: New York City Taxi Trips, Crimes in The City of Chicago, and Visits to Places of Interest Across Canada. They showed comparable performances of the *Spatiotemporal Forecasting Framework* across domains while dealing with challenges, such as high variation in the number of records, and temporal and spatial sparsity. Furthermore, they point to the effectiveness of the proposed model in modelling spatiotemporal patterns by demonstrating that it outperforms the naive baseline by at least 49.8% in terms of Symmetric Mean Absolute Percentage Error (SMAPE).

It can be expected that spatiotemporal forecasting will gain in popularity, with GPS sensors being on-boarded in even more categories of electronic devices in the future. This research is a testament to that future and the positive impact that the authors believe that such predictions could have in the cities of the future.

5.2 Future Work

This section presents areas where future work can be explored:

- **Multi-step forecasts**. The proposed research addresses only single-step forecasts like the next hour or the next day, and currently does not allow the user to predict multiple steps in the future. Single-step predictions already enable a wide range of forecasting questions to be answered, but in some scenarios such as influenza outbreaks, accurately predicting more steps ahead might give health-care practitioners adequate time to put preventive measures in place.

- **Multi-resolution GLMs**. The proposed GLMs account for the observed area at a single arbitrary resolution. Adding multi-resolution GLMs could help the framework capture temporal patterns that are more prominent at some other macro or micro scale. For example, the three layers of cells from Figure 3.5 could be used to build three-level GLMs. On a different note, multi-resolution GLMs could also be utilized to model dense areas at a high resolution and sparse areas at a lower resolution. For example, city centers could be modelled at $79000m^2$ and rural regions at $1.27km^2$, saving computational costs and reducing the sparsity perceived by the time series models.
- **Multi-time-horizon GGM**. The proposed GGM accounts for all the training data without having a strong capability to model the sequentiality of such records. Decoupling the GGM into short-term, mid-term, and long-term horizon models could potentially help the GGM to capture such patterns.
- **Extreme Events**. The proposed research does not specifically address extreme events such as holidays and sports events, resulting in higher residual errors in these cases. The introduction of an Anomaly Detection component could help the framework in automatically detecting extreme events and responding accordingly.
- **Integration with Geospatial Analysis Tools**. The proposed research provides ways for a user to interact and configure the framework to their needs, but the authors believe that the *Spatiotemporal Forecasting Framework* would be more useful if it were integrated with commercial tools that researchers already use for geospatial analysis. From the rich user interface of a tool such as ArcGIS, researchers can already use machine learning algorithms to aid their analysis, but such tools currently lack support for handling datasets

at the Big Data scale. The *Spatiotemporal Forecasting Framework* could make use of an integration channel available on ArcGIS[1] to receive tasks and hand back the results to the user interface, enabling the researcher to do spatiotemporal forecasting at scale from the tool that for them is most comfortable.

---

[1] https://www.esri.com/en-us/arcgis/products/arcgis-pro/overview

Bibliography

[1] Xwegnon Ghislain Agoua, Robin Girard, and George Kariniotakis. Short-Term Spatio-Temporal Forecasting of Photovoltaic Power Production. *IEEE Transactions on Sustainable Energy*, 9(2):538–546, 2018.

[2] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621, 2010.

[3] Luiz G.A. Alves, Haroldo V. Ribeiro, and Francisco A. Rodrigues. Crime prediction through urban metrics and statistical learning. *Physica A: Statistical Mechanics and its Applications*, 505:435–443, 2018.

[4] Maïna André, Richard Perez, Ted Soubdhan, James Schlemmer, Rudy Calif, and Stéphanie Monjoly. Preliminary assessment of two spatio-temporal forecasting technics for hourly satellite-derived irradiance in a complex meteorological context. *Solar Energy*, 177(April 2018):703–712, 2019.

[5] Gregory R Andrews. *Foundations of multithreaded, parallel, and distributed programming*, volume 11. Addison-Wesley Reading, 2000.

[6] Michael Armbrust, Tathagata Das, Aaron Davidson, Ali Ghodsi, Andrew Or, Josh Rosen, Ion Stoica, Patrick Wendell, Reynold Xin, and Matei Zaharia. Scaling Spark in the Real World: Performance and Usability. *Proc. VLDB Endow.*, 8(12):1840–1843, aug 2015.

[7] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1383–1394, New York, NY, USA, 2015. ACM.

[8] L. Bel, D. Allard, J. M. Laurent, R. Cheddadi, and A. Bar-Hen. CART algorithm for spatial data: Application to environmental and ecological data. *Computational Statistics and Data Analysis*, 53(8):3082–3093, 2009.

[9] Christoph Bergmeir and José M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[11] Carl Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, January 2015.

[12] Léon Bottou and Vladimir Vapnik. Local learning algorithms. *Neural computation*, 4(6):888–900, 1992.

[13] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[14] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[15] Braindrain0000. Six iterations of the Hilbert Curve. `https://commons.wikimedia.org/wiki/File:Hilbert_curve.svg` / CC BY-SA 3.0, 2007.

[16] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[17] Leo Breiman. Using adaptive bagging to debias regressions. Technical report, Technical Report 547, Statistics Dept. UCB, 1999.

[18] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] Joilson de Assis Cabral, Luiz Fernando Loureiro Legey, and Maria Viviana de Freitas Cabral. Electricity consumption forecasting in Brazil: A spatial econometrics approach. *Energy*, 126:124–131, 2017.

[20] Charlie Catlett, Eugenio Cesario, Domenico Talia, and Andrea Vinci. Spatio-temporal crime predictions in smart cities: A data-driven approach and experiments. *Pervasive and Mobile Computing*, 53:62–74, 2019.

[21] Jason Catlett. On changing continuous attributes into ordered discrete attributes. In Yves Kodratoff, editor, *Machine Learning — EWSL-91*, pages 164–178, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[22] Yong Chen, Shuai Zhang, Wenyu Zhang, Juanjuan Peng, and Yishuai Cai. Multifactor spatio-temporal correlation model based on a combination of convolutional neural network and long short-term memory neural network for wind speed forecasting. *Energy Conversion and Management*, 185(November 2018):783–799, 2019.

[23] Noel Cressie and Christopher K Wikle. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015.

[24] Thomas G Dietterich. Machine learning for sequential data: A review. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–30. Springer, 2002.

[25] Docker. Docker Containers vs. Virtual Machines. `https://www.docker.com/sites/default/files/d8/2018-11/docker-containerized-and-vm-transparent-bg.png`, 2019.

[26] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. *Machine Learning Proceedings 1995*, pages 194–202, 1995.

[27] Alireza Ermagun, Greg Lindsey, and Tracy Hadden Loh. Bicycle, pedestrian, and mixed-mode trail traffic: A performance assessment of demand models. *Landscape and Urban Planning*, 177:92–102, sep 2018.

[28] A Stewart Fotheringham, Chris Brunsdon, and Martin Charlton. *Geographically weighted regression*. John Wiley & Sons, Limited West Atrium, 2003.

[29] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[30] Hossein Hassani and Emmanuel Sirimal Silva. Forecasting with Big Data: A Review. *Annals of Data Science*, 2(1):5–19, 2015.

[31] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[32] David Hilbert. *Über die stetige Abbildung einer Linie auf ein Flächenstück*, pages 1–2. Springer Berlin Heidelberg, Berlin, Heidelberg, 1935.

[33] Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE, 1995.

[34] Chao Huang, Junbo Zhang, Yu Zheng, and Nitesh V. Chawla. DeepCrime. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management - CIKM '18*, pages 1423–1432, New York, New York, USA, 2018. ACM Press.

[35] Nick Johnson. A representation of how a quadtree is structured internally. http://static.notdot.net/uploads/quadtree.png, 2009.

[36] Juyoung Kang and Hwan-Seung Yong. Spatio-temporal discretization for sequential pattern mining. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, page 218, 2008.

[37] Jintao Ke, Hongyu Zheng, Hai Yang, and Xiqun (Michael) Chen. Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transportation Research Part C: Emerging Technologies*, 85(October):591–608, 2017.

[38] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *Cidr*, volume 1, pages 2–1, 2013.

[39] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[40] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th \${\$USENIX\$}\$ Symposium on Operating Systems Design and Implementation (\${\$OSDI\$}\$ 14)*, pages 583–598, 2014.

[41] Siyu Liao, Liutong Zhou, Xuan Di, Bo Yuan, and Jinjun Xiong. Large-scale short-term urban taxi demand forecasting using deep learning. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 428–433. IEEE Press, 2018.

[42] Chenghao Liu, Steven C H Hoi, Peilin Zhao, and Jianling Sun. Online ARIMA Algorithms for Time Series Prediction. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI2016)*, pages 1867–1873, 2016.

[43] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *Proc. VLDB Endow.*, 5(8):716–727, apr 2012.

[44] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[45] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D B Tsai, Manish Amde, Sean Owen, and Others. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[46] Maged Michael, José E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x scale-out: A case study using nutch/Lucene. *Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM*, 2007.

[47] Miym. Distributed Computing vs. Parallel Computing. `https://commons.wikimedia.org/wiki/File:Distributed-parallel.svg` / CC BY-SA 3.0, 2009.

[48] Robert Nau. Notes on nonseasonal ARIMA models. `http://people.duke.edu/~rnau/Notes_on_nonseasonal_ARIMA_models--Robert_Nau.pdf`, 2014.

[49] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009)*, 2009.

[50] J R Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, mar 1986.

[51] Evan L. Ray and Nicholas G. Reich. Prediction of infectious disease epidemics via weighted density ensembles. *PLoS Computational Biology*, 14(2):1–23, 2018.

[52] Nicholas G. Reich, Logan C. Brooks, Spencer J. Fox, Sasikiran Kandula, Craig J. McGowan, Evan Moore, Dave Osthus, Evan L. Ray, Abhinav Tushar, Teresa K. Yamana, Matthew Biggerstaff, Michael A. Johansson, Roni Rosenfeld, and Jeffrey Shaman. A

collaborative multiyear, multimodel assessment of seasonal influenza forecasting in the United States. *Proceedings of the National Academy of Sciences*, 116(8):201812594, 2019.

[53] C. Ronchi, R. Iacono, and P. S. Paolucci. The ”Cubed sphere”: A new method for the solution of partial differential equations in spherical geometry. *Journal of Computational Physics*, 124(1):93–114, 1996.

[54] S2Geometry. Earth Cube. `https://s2geometry.io/devguide/img/s2cell_global.jpg`, 2019.

[55] Abolfazl Safikhani, Camille Kamga, Sandeep Mudigonda, Sabiheh Sadat Faghih, and Bahman Moghimi. Spatio-temporal modeling of yellow taxi demands in New York City using generalized STAR models. *International Journal of Forecasting*, nov 2018.

[56] Yosiyuki Sakamoto, Makio Ishiguro, and Genshiro Kitagawa. Akaike information criterion statistics. *Dordrecht, The Netherlands: D. Reidel*, 81, 1986.

[57] Hanan Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[58] SideWalkLabs. Planetary View of s2sphere. `https://s2.sidewalklabs.com/planetaryview/`, 2009.

[59] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*, 2009.

[60] Apache Spark. Cluster Overview. `https://spark.apache.org/docs/latest/img/cluster-overview.png`, 2019.

[61] Hung Tien Tran, Hiep Tuan Nguyen, and Viet Trung Tran. Large-scale geographically weighted regression on Spark. *Proceedings - 2016 8th International Conference on Knowledge and Systems Engineering, KSE 2016*, pages 127–132, 2016.

[62] Ukrish Vanichrujee, Teerayut Horanont, Wasan Pattara-atikom, Thanaruk Theeramunkong, and Takahiro Shinozaki. Taxi Demand Prediction using Ensemble Model Based on RNNs and XGBOOST. In *2018 International Conference on Embedded Systems and Intelligent Technology & International Conference on Information and Communication Technology for Embedded Systems (ICESIT-ICICTES)*, pages 1–6. IEEE, may 2018.

[63] Bao Wang, Xiyang Luo, Fangbo Zhang, Baichuan Yuan, Andrea L Bertozzi, and PJ Brantingham. Graph-based deep modeling and real time forecasting of sparse spatio-temporal data. *MiLeTS18, London, United Kingdom*, 2018.

[64] Bao Wang, Penghang Yin, Andrea L Bertozzi, P Jeffrey Brantingham, Stanley J Osher, and Jack Xin. Deep learning for real-time crime forecasting and its ternarization. *arXiv preprint arXiv:1711.08833*, 2017.

[65] Andreas S Weigend. *Time series prediction: forecasting the future and understanding the past*. Routledge, 2018.

[66] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Boloni, and Damla Turgut. Real-Time prediction of taxi demand using recurrent neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2572–2581, 2018.

[67] R Yadav and S Kumari Sheoran. Modified ARIMA Model for Improving Certainty in Spatio-Temporal Crime Event Prediction. In *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–4, nov 2018.

[68] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial data management in apache spark: the GeoSpark perspective and beyond. *GeoInformatica*, 2018.

[69] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed

Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 2, Berkeley, CA, USA, 2012. USENIX Association.

[70] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM*, 59(11):56–65, oct 2016.

[71] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[72] Shanjiang Zhu, Mark L. Franz, Arefeh Nasri, Lei Zhang, Zhuo Yang, and Jina Mahmoudi. Analysis of Washington, DC taxi demand using GPS and land-use data. *Journal of Transport Geography*, 66(February 2017):35–44, 2018.

Curriculum Vitae

Name:	Rafael Nascimento de Aguiar
Post-Secondary Education and Degrees:	The University of Western Ontario London, ON 2017 - 2019 MESc
	Federal University of Pernambuco Recife, PE - Brazil 2009 - 2015 BSc Computer Science
Honours and Awards:	CNPq Science Without Borders 2013-2014
Related Work Experience:	Teaching Assistant The University of Western Ontario 2017 - 2019
	Data Engineer In Loco Media 2015 - 2017
	Co-founder and Software Engineer CriticalLab 2011 - 2013