11-13-2020

Geographic Data Mining and Knowledge Discovery

Liangdong Deng
liadeng@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

GEOGRAPHIC DATA MINING AND KNOWLEDGE DISCOVERY

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Liangdong Deng

2020

To: Dean John Volakis
College of Engineering and Computing

This dissertation, written by Liangdong Deng, and entitled Geographic Data Mining and Knowledge Discovery, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Dong Chen

Masoud Sadjadi

Armando Barreto

Malek Adjouadi

Naphtali Rishe, Major Professor

Date of Defense: November 13, 2020

The dissertation of Liangdong Deng is approved.

Dean John Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of University Graduate School

Florida International University, 2020

DEDICATION

I dedicate this dissertation work to my beloved family and friends. A special gratitude to my wife. This work wouldn't have been possible without her selfless support, understanding and love.

ACKNOWLEDGMENTS

During my Ph.D. study, I received tremendous help and support from various people. Without them, it was nearly impossible for me to complete my Ph.D. research and dissertation.

First and foremost, I would like to give my most profound gratitude to Dr. Naphtali Rishe for his tireless guidance, insightful direction, and priceless academic advice. My past 8 years' working experience with him in the High Performance Database Research Center has been a breeze. With his continuous support, I managed to handle all the difficulties and finally accompanish my academic goal.

Second, I want to extend my appreciation to Dr. Dong Chen and Dr. Malek Adjouadi. Their knowledge and expertise in the field of system security and machine learning is truly remarkable. Many of my researches wouldn't have been as successful without his valuable suggestion and help.

Third, I would like to thank my dissertation committee members Dr. Armando Barreto and Dr. Masoud Sadjadi for their professional support and assistance throughout my dissertation stage. I must be the luckiest person to have them as my committee members.

Additionally, I'm very grateful to the scholarships and travel funds provided by my department, the School of Computing and Information Sciences.

I also acknowledge the support provided by the National Science Foundation under grants: Nos. I/UCRC IP-1338922, III-Large IIS-1213026, MRI CNS-1429345, MRI CNS-1532061, MRI CNS-1920182.

ABSTRACT OF THE DISSERTATION

GEOGRAPHIC DATA MINING AND KNOWLEDGE DISCOVERY

by

Liangdong Deng

Florida International University, 2020

Miami, Florida

Professor Naphtali Rishe, Major Professor

Geographic data are information associated with a location on the surface of the Earth. They comprise spatial attributes (latitude, longitude, and altitude) and non-spatial attributes (facts related to a location). Traditionally, Physical Geography datasets were considered to be more valuable, thus attracted most research interest. But with the advancements in remote sensing technologies and widespread use of GPS enabled cellphones and IoT (Internet of Things) devices, recent years witnessed explosive growth in the amount of available Human Geography datasets. However, methods and tools that are capable of analyzing and modeling these datasets are very limited. This is because Human Geography data are inherently difficult to model due to its characteristics (non-stationarity, uneven distribution, etc.).

Many algorithms were invented to solve these challenges – especially non-stationarity – in the past few years, like Geographically Weighted Regression, Multiscale GWR, Geographical Random Forest, etc. They were proven to be much more efficient than the general machine learning algorithms that are not specifically designed to deal with non-stationarity. However, such algorithms are far from perfect and have a lot of room for improvement.

This dissertation proposed multiple algorithms for modeling non-stationary geographic data. The main contributions are: (1) designed a novel method to evaluate non-stationarity and its impact on regression models; (2) proposed the Geographic

R-Partition tree for modeling non-stationary data; (3) proposed the IDW-RF algorithm, which uses the advantages of Random Forests to deal with extremely unevenly distributed geographic datasets; (4) proposed the LVRF algorithm, which models geographic data using a latent variable based method. Experiments show that these algorithms are very efficient and outperform other state-of-the-art algorithms in certain scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Geographic data are information associated with a location on the surface of the earth. They are comprised of spatial attributes (latitude, longitude, and sometimes altitude), non-spatial attributes (features or characteristics of an object, or observations associated with the location), and sometimes also temporal attributes [SG16].

Typically, scientists and researchers categorize geographic datasets by the source where they come from. Data collected from the natural processes of the Earth are called Physical Geography datasets. Data generated by activities of people are categorized as Human Geography datasets. For example, mineral resources, hydrology, weather, and climate all belong to Physical Geography [Mas99]. Whereas housing, culture, traffic, disease, war, and crime are in the category of Human Geography. This categorization method is widely used because datasets in different categories usually have distinct characteristics which makes methods or theories that work with one category less efficient or don't work at all for the other category.

Traditionally, Physical Geography datasets were considered to contain more value and thus attracted more interest from researchers. Unarguably, these research are important and crucial to the survival and development of the human race, such as learning how to avoid or dampen damage brought by natural hazards [AMlBI17], efficiently locate and extract mineral resources [PW16] and so on.

On the other hand, Human Geography datasets have been growing at a fast pace in recent years due to the adoption of GPS enabled cellphones and other IoT (Internet of Things) devices [BBFRS12]. Values in these datasets are gaining increased interest every day. However, some methods and algorithms that worked well on Physical Geography data cannot be applied to Human Geography data directly without any change, as some of the underlying assumptions have changed. Thus, new theories and tools need to be invented accordingly.

1.2 Problem Statement

Comparing with Physical Geography, Human Geography datasets have the following traits that make them more difficult to deal with:

- Non-stationary
  - A non-stationary dataset means the underlying rules that determine the target value (the feature in the dataset that we are interested in or want to predict) would change with the location. Conversely, in stationary data, this rule is the same everywhere.
  - Usually, data collected during the natural processes of the Earth are completely or mostly stationary. For example, the presence of a certain mineral deposit is always decided by a few factors no matter where it is. Different locations will affect the values of these factors but won't change the formula between these factors and the presence of the mineral.
  - But for Human Geography datasets, a formula that works in one city doesn't necessarily work in another city. There are always hidden factors that cannot be collected or precisely measured that influences the target value. For example, crime data collected from different cities may not fit the same models because a lot of social, political, or historical factors are not included in the dataset.

  - However, this doesn't mean non-stationary datasets are impossible to learn. Spatial autocorrelation is still effective in these data and makes it possible to create multiple local models instead of one global model.

- High dimensionality

  - Human Geography datasets usually tend to have a much higher dimension than Physical ones because of the number of features available. This is especially true when the datasets are enlarged by deriving additional information from other sources.

  - For example, when predicting the sale price of houses, one can easily find attributes of the property itself (dozens of features), plus countless derived location data (like nearby schools, supermarkets, tax info, and so on) which can easily expand to hundreds of features, in which case many of the traditional algorithms will be much less efficient or doesn't applicable at all.

- Large data size

  - Many of the Physical Geography datasets require researchers to actually visit the sample location to retrieve critical data. This is generally very expensive thus greatly limits the amount of data available.

  - Human Geography datasets, however, do not have this limitation and can be retrieved from various ways at a very low cost like crow sourcing, Internet questionnaire, widely available government data sources, and so on. Thus, datasets that contain millions of observations are more and more common.

- More data usually means more accurate models can be trained. But with Human Geography datasets, this can hardly be true due to the existence of non-stationary. For any location, if nearby observations are the only reliable data source that can be used to build the model, it would not help no matter how many observations are available at other places.

These challenges make it difficult to learn accurate models from Human Geography datasets. Traditional geographic spatial regression methods are almost useless in this case. So, approaches specifically designed to solve these challenges were proposed in the last few years, like Geographically Weighted Regression (GWR) [BFC96], Semiparametric GWR (SGWR) [FBC02], Multiscale GWR (MGWR) [FYK17], Geographical Random Forest (GRF) [GGG$^+$19] and etc. However, state-of-the-art algorithms are essentially building multiple local models instead of one global model. While these types of algorithms work relatively well under normal situations, they're much less accurate when the dataset is unevenly distributed. Thus, this paper provides a different understanding of non-stationarity and proposes a better way to build models for non-stationary geographic datasets.

1.3 Contribution

My dissertation is centered around Geographic datasets and non-stationarity. It addresses challenges outlined in the previous section, by proposing and implementing approaches that are specially designed to handle non-stationarity.

The main contributions of this paper are: (1) designed a novel method to evaluate non-stationarity and its impact on regression models; (2) proposed the Geographic R-Partition tree for modeling non-stationary data; (3) proposed the IDW-RF algorithm, which uses the advantages of Random Forests to deal with extremely unevenly distributed geographic datasets; (4) proposed the LVRF algorithm, which models geographic data using a latent variable based method.

1.4 Organization

This dissertation is organized as follows. First, related work on the same topic is discussed in Chapter 2. Then, Chapter 3 suggests novel approaches to understand and evaluate non-stationarity. The rest of the chapters focus on building improved models for non-stationary geographic data. Chapter 4 proposes the GRP-tree to improve modeling accuracy by aggregating spatially similar data into the same partitions. Chapter 5 proposes the IDW-RF algorithm to solve the challenge of modeling extremely unevenly distributed geographic datasets. And Chapter 6 proposes a latent variable based approach to explain and model non-stationarity as a hidden factor. Experiments show that these algorithms outperform other state-of-the-art algorithms in certain scenarios.

CHAPTER 2

PRELIMINARIES AND RELATED WORK

The previous chapter listed challenges in modeling geographic data. Over the years, many researchers studied these challenges and proposed different modeling algorithms. In this chapter, I will highlight some of the most influential and state-of-art literature on this topic.

2.1 Spatial Interpolation Methods

There is a long history of study on Physical Geography data. A few decades ago, before machine learning algorithms are well developed, the modeling of geographic datasets was mostly done in the form of spatial interpolation.

Spatial interpolation means use known observations to predict the values of unknown observations. This type of methods are widely adopted in environmental science studies, especially in the field of geosciences, water resources, and agriculture or soil sciences [ZGHW07]. For these disciplines, spatially continuous data is typically required or important for the decision-making process. But obtaining spatially continuous data that cover the entire region of interest is sometimes difficult or impossible, as these disciplines often involve studies of remote, mountainous, inaccessible areas. For these cases, a spatial interpolation process will be first performed to fill in the blank areas. The accuracy of the interpolated data is often critical to the success of the entire study, thus, many spatial interpolation theories and methods were invented in order to obtain better interpolation results.

Inverse Distance Weighting (IDW)

Of these methods, the most basic and frequently used one is Inverse Distance Weighting (IDW) [She68]. This method is relatively simple comparing with the others. But in applications where results are required to be highly intuitive and interpretable, IDW has its unique advantage.

IDW works by predicting unknown observations' values from a weighted average from the known observations. One advantage of IDW is that it's flexible on the weighing function. The default weighing function, used in [She68], is the multiplicative inverse of the distance (any form of distance function can be accepted here). But it's highly customizable and often replaced by a more appropriate one depending on the actual dataset. Generally speaking, when the weighing function is properly chosen (which requires prior-knowledge of the dataset), IDW will produce results that are accurate enough, even when comparing with the most complicated spatial interpolation algorithms.

Kriging

The Kriging method, which was invented in the year 1952 by D. G. Krige [Kri52], is another widely used spatial interpolation method. It was first used in mining geology [ZGHW07], then discovered to be also a powerful tool in many other earth related research fields.

Since then, the development of Kriging based methods had thrived. Multiple variants were invented like ordinary kriging (OK), universal kriging (UK) and ordinary co-kriging (OCK). Comparative studies were done on spatial interpolation methods (like [LH11]) and showed that although many methods have their own advantages at certain circumstances, kriging and kriging based methods are the most competitive for generating accurate spatial interpolation results.

However, Kriging is an unbiased methods and assumes data are stationary, which make it unsuitable for human geography datasets.

2.2 Non-stationary Spatial Data Modeling

Geographically Weighted Regression (GWR)

Even with Physical Geography data, it is not an unusual thing to observe non-stationarity in the data. Applying method designed to handle stationary data on such datasets would generally mean less accurate or completely unusable interpolation results are expected. To solve this problem, Brunsdon, Fotheringham and Charlton proposed the Geographically Weighted Regression (GWR) in 1996 [BFC96], which later become a very important method and had various extensions.

Instead of producing an average global model, GWR tries to interpret the relationship between features and target variables differently across the space. The way how it works is to learn a regression model for every single feature in the dataset. During this process, nearby observations will be examined to include dependent and explanatory components in the model. The distance between the nearby observation and the observation being processed determines how much influence will be applied, which is why the word "weighted" is included in the name of the method.

Semiparametric GWR (SGWR)

After the success of GWR, Brunsdon proposed an enhanced version of GWR in 2002 [FBC02]. This method is called Semiparametric GWR (SGWR). In this variant, users of the method can specify a list of features to be stationary across the space, whereas others remain non-stationary.

Multiscale GWR (MGWR)

This method was introduced by Fotheringham, Yang and Kang in 2017. It "is similar in intent to Bayesian nonseparable spatially varying coefficients (SVC) models, although potentially providing a more flexible and scalable framework in which to examine multiscale processes" [FYK17]. By doing so, it is able to model datasets on different scales (hence the name multiscale).

GRF (Geographical Random Forest)

The Geographical Random Forest (GRF) is relatively new. It was proposed by Stefanos, Tais, et al. in 2019. The method adopts the famous Random Forests algorithm [Bre01] as an underlying modeling method to create many local models, instead of a global average model. When predicting target values, these local models will work together to produce a weighted calculation result.

CHAPTER 3

GEOGRAPHIC REGRESSION MODELS REVISITED: PROGRESS AND CHALLENGES

Geographic datasets are usually discrete observations collected from one or multiple areas, as continuous data are oftentimes difficult or even impossible to obtain (for example, environmental data that requires retrieving physical samples from remote, inaccessible places). But when put into practical use, such data is frequently required to be spatially continuous (at least for the region of interest) for researchers and engineers to make efficient use of them. Thus, in order to produce spatially continuous data, one must give his best estimation of values at unsampled points. And, to make these predictions as accurate as possible, geographic regression models are created to better accomplish the task.

This chapter first gives a general idea of the main challenges in learning accurate geographic regression models. Then, existing algorithms are reviewed and discussed. After that, several testing methods are proposed and implemented to help better evaluate and understand these challenges. Finally, conclusions are made and potentially better ways for producing more accurate models are discussed.

3.1 Challenges in Geographic Datasets

The challenges in modeling geographic datasets, especially datasets from Human Geography, mainly come from these factors:

- Non-stationarity
  - Geographic datasets are usually generated from either Physical Geography or Human Geography. Physical Geography datasets come from

  natural processes of the Earth, such as natural disasters, mineral resources, hydrology, land and ocean boundaries, elevation, weather and climate ([Mas99]). Whereas Human Geography datasets are generated by activities of people, like land use, traffic, population, crime and real estates.

  - When learning models from geographic datasets, non-stationarity is an unignorable factor. Physical geography datasets tend to be more stationary as they are decided by certain rules of the nature. But things are totally different with human geography datasets, which is produced by human activities thus rules can be very different from place to place. For these datasets, non-stationarity is normal and stationarity is abnormal. For example, crime activities will have different patterns in different areas; house prices will be influenced by different factors depending on location of the real estate; traffic congestions at different places are decided by different conditions.

- Spatial autocorrelation

  - According to the first law of geography [Tob70], "everything is related to everything else, but near things are more related than distant things". In most datasets, the spatial features are unavoidably correlated with non-spatial features to a certain degree. In fact, this type of correlation can be observed easily because these datasets are typically joined by matching the coordinates.

  - Another source of autocorrelation comes from the fact that objects with similar characteristics tend to cluster together. For example, real estate properties in the same community tend to share similar characteristics like layouts, market prices and household income. Also, people from the same background tend to cluster together. When analyzing geographic data, ignoring such correlation would cause inaccuracy or inconsistency in the result [SSV+02].

- High dimensionality
  - Besides the original geographic features (latitude, longitude, altitude and time), there're typically a lot of additional features which are derived from other sources of information. For example, when predicting the market price of real estate properties, the dataset will usually contain the property's latitude, longitude, year built, number of stories, number of bedrooms/bathrooms, distance to shopping centers and schools, and so on.
  - To make the situation worse, the dimensionality can be further increased by including statistical features such as average/lowest/highest market price nearby, average year built nearby, etc. In this example, one can easily expand the dimensionality to over one hundred, in which case most of the traditional algorithms will lose efficacy.
- Irregularity in spatial features
  - Most of related researches focus on datasets in Euclidean space. But lots of geographic datasets are actually based on network space, despite of the fact that original coordinates are given in Euclidean space.
  - For example, it would be more effective to use road network [OBK+10] rather than latitudes and longitudes to analyze and predict people's daily activities, travel habits and points of interest. In cases when these kind

of information is missing, or inaccurate, or relatively difficult to obtain, it will be more challenging to learn a good model from the data.

Over the years, researchers and scientists had been studying these challenges and invented many algorithms to solve them. [KHS99] proposed an efficient method for building decision trees for the classification of spatial data. [SSV$^+$02] invented two classification approaches to create spatial data models using probabilistic framework. [SYA11] extended ID3 decision tree algorithm to includes spatial information as additional features. [JSZ$^+$14] proposed local and focal test-based decision tree algorithm.

Many of these researches use a common method to handle the spatial attributes, which is transforming the relationships implied by spatial attributes to features on which classical algorithms can be applied [KBL95]. These features are typically statistical features generated from existing spatial and non-spatial attributes of the training dataset. It is worth noting that generating spatial statistics is a well-researched area. For example, spatial continuity and weak stationarity can be easily extracted from the dataset [Cre93]. Basing on these statistics, various common algorithms have been created to perform different types of tasks like outlier detection [SZK14], prediction [GA89], coupling [Gü17] and clustering [JMF99].

However, during the transformation, information may be lost [SZHV03]. Thus, a better method is needed to handle the spatial attributes directly with fewer or no loss of information, whereas being able to deal with the implicit spatial relationships among the data points.

3.2 Literature Review

By definition, geographic data are comprised of spatial and non-spatial attributes, sometimes also temporal attributes. But due to the fact that research of datasets with temporal attributes are still relatively preliminary [SY18] and very few algorithms can actually derive accurate models from them, in this chapter, discussion is limited to geographic data with spatial and non-spatial attributes only.

If we remove non-spatial attributes from geographic datasets and keep only the spatial attributes, they simply become spatial datasets. And the regression task for the hybrid data will be simplified to a pure spatial regression analysis task. But even spatial regression analysis is not a trivial task. So, to understand how to build regression models for geographic datasets, one must first learn how to model spatial datasets.

Luckily, spatial regression methods had been studied by researchers for decades and widely used in many disciplines. A research [ZGHW07] found that they're most adopted in environmental sciences, geosciences, water resources and agriculture or soil sciences. As previously explained, the reason why they play such an important role in these disciplines are because spatially continuous data are essential to perform the research but usually very difficult to obtain, thus requires additional procedure for generating predicted values for unsampled places.

For such purpose, spatial regression is often referred to as spatial interpolation and many methods were invented in order to obtain the best interpolation results. Of these methods, the most frequently used ones are Inverse distance weighting (IDW) and various kriging methods like ordinary kriging (OK), universal kriging (UK) and ordinary co-kriging (OCK). Many researches were made to evaluate these methods [LH11] and although different methods have their best application scenarios, it's widely agreed that for general datasets, kriging methods are the most promising and tend to produce the best interpolation results.

Kriging, which originated in mining geology [ZGHW07], is a regression method that calculates the least squares estimation for data points. D. G. Krige first invented this method in the year 1952 [Kri52] and applied it in ore reserve estimation for a region located in South Africa. Later, in 1963, Georges Matheron further extended this research and established a generic method which can be used in geostatistics to model the spatial autocorrelation within spatial features. From then on, many variances of the Kriging method had been invented and widely applied in the field of geoscience, environmental science, and many other earth related researches as a powerful tool.

However, Kriging and other spatial modeling methods mentioned above are unbiased methods and work under the assumption of stationary data, which means the features must behave universally the same across the region of interest. While this is true in earth science and environmental science, it might not be the case for other fields like human geography and social science, for which most of the time studying of non-stationary spatial data is involved.

For example, one cannot expect the relationship between house prices and house features to remain the same in different regions. Features such as number of bedrooms and total square footage have much obvious impact on the house price in urban areas than in rural areas. Whereas other features like distance to nearby supermarkets and schools usually matters more in rural areas. This non-stationarity imposes additional challenges to spatial data modeling. Thus, methods designed to process stationary data such as Kriging will perform much worse, or doesn't work at all.

3.3 Evaluating Impact of Non-Stationarity

To further understand non-stationarity and how it impacts the accuracy of models learned from spatial data by various machine learning algorithms, we designed a simulated spatial dataset and ran different machine learning algorithms on the dataset.

Although there are plenty of real world datasets available, a simulated one is preferred here because in such a dataset, non-stationarity can be easily controlled and changed, which makes it much easier to observe how non-stationarity impacts the performance of various machine learning algorithms. It would be otherwise difficult to do so with a real world dataset for which most of the time it's hard to say how much non-stationarity is embedded within the dataset.

The simulated dataset is comprised of spatial attributes (x and y) and labels (ground truth) only. Non-spatial attributes are not included as some of the Kriging algorithms were designed to handle spatial attributes only. To test their performance and compare it horizontally with other algorithms, the dataset is required to contain spatial attributes only, in order to create a level playing field.

Adding Non-Stationarity

Then, the tricky part comes to how to add non-stationarity into the data while allowing the amount of non-stationarity can be somehow manipulated. Because non-stationarity is merely a virtual concept and not something can be precisely measured, we can only say one dataset has more non-stationarity in it than the other one, but impossible to quantify how much more it has.

For this reason, the concept of Point of Influence (POI) is introduced in order to solve the problem mentioned above. A POI is defined as a hidden factor that

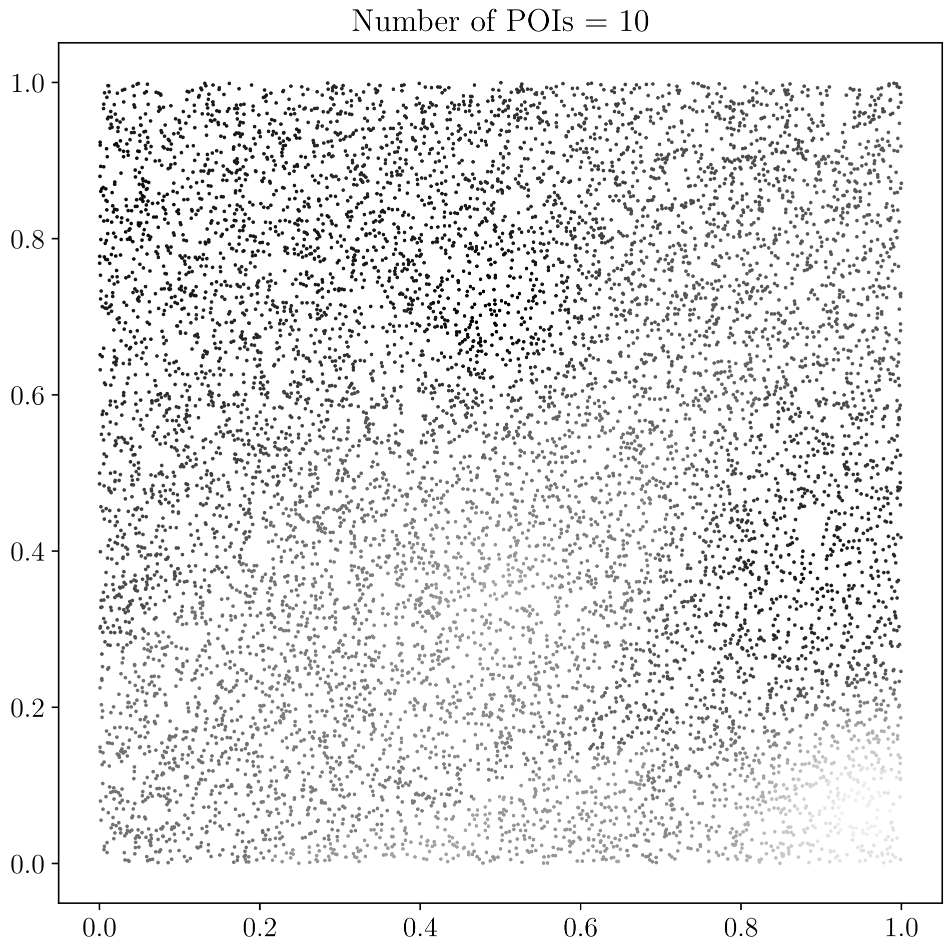


Figure 3.1: Nonstationarity spatial dataset generated by simulator (POI=10)

influences the surrounding data points and their label values. It is a part of the ground truth, but cannot be observed directly. One can only infer the existence of POIs from the labels values of the data points.

Then, the label values of data points can be calculated by checking their nearby POIs. To produce the best results, a data point should be affected by neither too few POIs (in which case non-stationarity will be too strong), nor too many POIs (otherwise non-stationarity can be too weaker to be observed). A number that falls within the range of $[3, 7]$ will be the best.

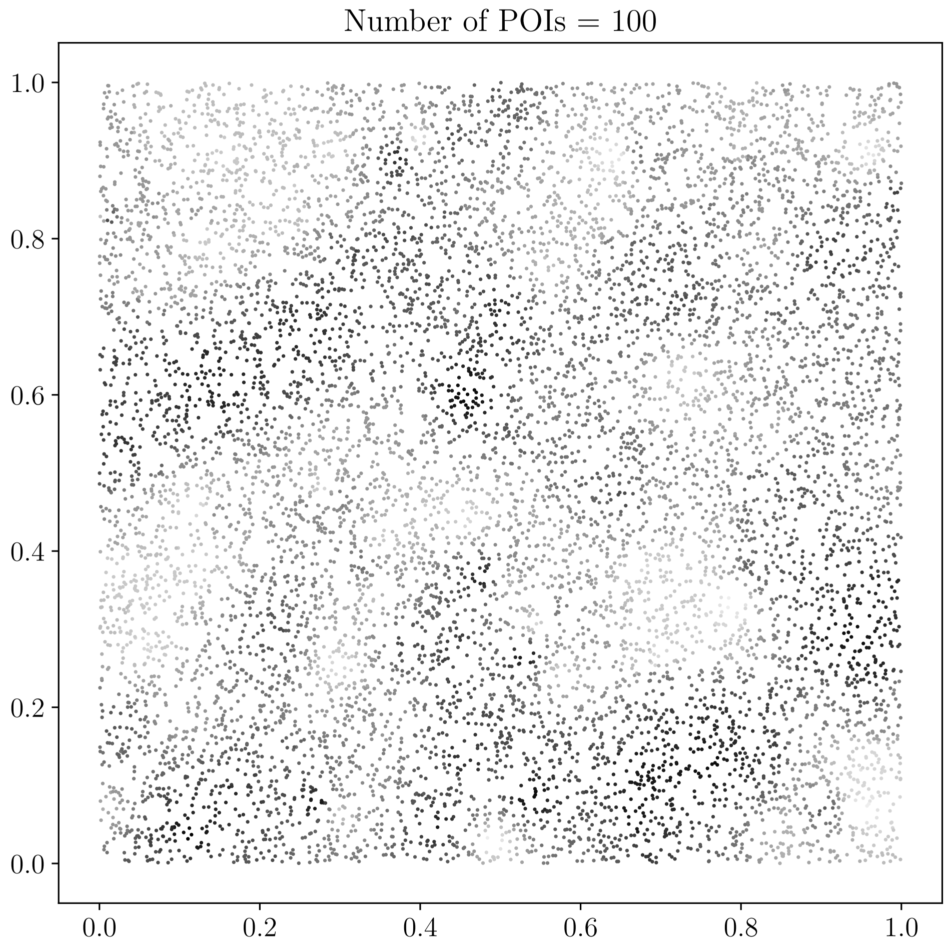


Figure 3.2: Nonstationarity spatial dataset generated by simulator (POI=100)

After deciding the number of POIs influencing a data point, the formula for calculating the label values also need to be decided. Here, Inverse distance weighting [She68] (IDW) is adopted to calculate label values from nearby POIs. The IDW method is chosen because it's relatively simple but sufficient to our needs here. It predicts values of unknown points by calculating a weighted average from the values of known points. The original form of IDW looks like below. Here, we don't want a data point to be calculated from all of the other known points so the value of N will be a predetermined number as previously discussed.

$$u(\mathbf{x}) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(\mathbf{x}) u_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}, & \text{if } d(\mathbf{x}, \mathbf{x}_i) \neq 0 \text{ for all } i \\ u_i, & \text{if } d(\mathbf{x}, \mathbf{x}_i) = 0 \text{ for some } i \end{cases}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

Manipulating Non-Stationarity

By adding random POIs into the data, non-stationarity can be added. Figure 3.1 and 3.2 show how the generated datasets look like. Every dot in the figure represents a data point. The grayscale indicates the label value of the data point. Black data points have a value close to 1 and white data points have a value close to 0. As observed, dataset with 100 POIs is way more localized than dataset with only 10 POIs. In fact, non-stationarity can be controlled by two ways:

Method 1. Increase or decrease the number of POIs.

- When the number of POIs is increased, the non-stationarity will also increase. This is because the more POIs there are, two data points with the same distance tend to be less correlated to each other because they're now more likely to be influenced by different POIs, rather than the same POIs.

Method 2. Increase or decrease the influence radius of POIs.

- When the influence radius of POIs is increased, the non-stationarity will decrease. This is because two data points with the same distance tend to be more correlated to each other because they're now more likely to be influenced by the same POIs.

Here, method 1 is adopted rather than method 2 because it's easier to implement and requires the changing of only a single number, which makes it more intuitive and the test results more straightforward and understandable.

Table 3.1: Statistics of the generated dataset (Number of POI = 10)

	X	Y	Label
count	10000.000	10000.000	10000.000
mean	0.502	0.500	0.522
std	0.288	0.291	0.155
min	0.000	0.000	0.012
25%	0.254	0.245	0.409
50%	0.504	0.498	0.534
75%	0.753	0.756	0.652
max	1.000	1.000	0.819

Table 3.1 shows the statistics of 10000 data points generated by the simulator program. As we can see, Label values are designed to fall in the range of $[0, 1]$ and have a mean value of around 0.5. The purpose of this design is to create an unbiased dataset to make it as fair as possible for all machine learning algorithms.

Performance Evaluation

Now that the datasets are ready, it's time to pick some algorithms. A total of 9 regression algorithms are chosen as test candidates, as listed below.

- Baseline
- Linear Regression

- Stochastic gradient descent (SGD)
- Support Vector Regression (SVR)
- K-nearest Neighbors (KNN)
- Decision Tree
- Random Forest
- Multi-layer Perceptron (MLP)
- Ordinary Kriging

The Baseline algorithm always predict the mean of the training set, no matter what the input is. Its purpose is to provide an important reference point for the comparison of other algorithms. Since it's the simplest possible algorithm, any other algorithm that performs worse or close to this one can be considered as worthless.

The Linear Regression, SGD and SVR algorithms are all regressors that assume the dataset follows a linear distribution, which is obviously not true in our case. The purpose of introducing them in the comparison is to provide some insights on how a model performs when its assumption doesn't match the underlying dataset.

KNN, Decision Tree and Random Forest are traditional machine learning algorithms that can deal with non-stationary datasets. They're not specifically designed for spatial datasets but the way how they work sounds promising to our test dataset. So they're also included in the comparison.

MLP is also included in this study. Although it's not made for situations like this (very few features with strong correlations), how it performs comparing with the other algorithms is still an interesting topic and may provide additional findings.

Then, finally the Ordinary Kriging is added to see how it performs under non-stationarity.

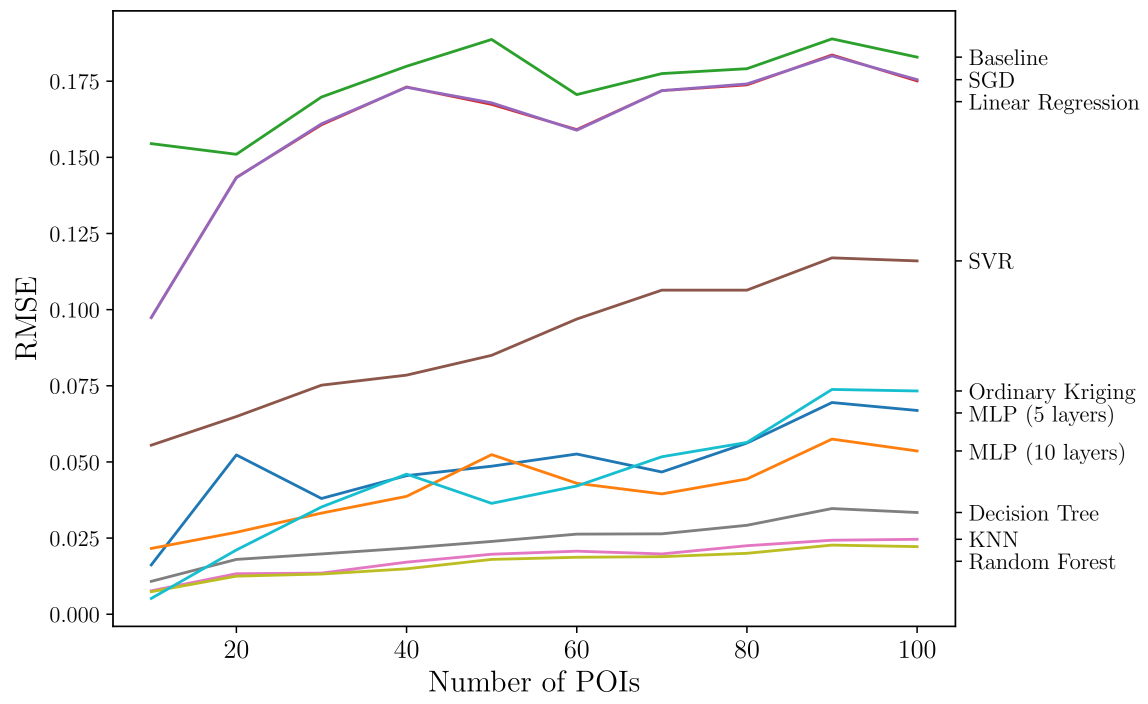


Figure 3.3: Performance comparison of different algorithms

Figure 3.3 shows the result of how the aforementioned algorithms perform under the test dataset with different number of POIs.

Here, Root Mean Squre Error (RMSE) is used as the performance indicator because all the algorithms are executed on the same dataset thus RMSE is directly related to the performance. Usually when algorithms run on different datasets, indicators like R^2 score (coefficient of determination) is used to measure the performance of regressors. But here we don't have this concern thus RMSE is obviously the best choice of all indicators.

As shown in the figure, the Baseline algorithm has a RMSE that ranges from 0.15 to 0.18. Given the fact that the label values range from 0 to 1, have an average of 0.5, and a standard deviation of 0.15, this is a pretty reasonable result. And unsurprisingly, the linear regressors (SGD, Linear Regression and SVR) didn't do well on this dataset which is not a linear dataset in any way. But it's worth noting that SVR (with a non-linear kernel) performs better than the other linear regressors, which is an indication that non-linear kernels are capable of capturing some of the characteristics of the dataset, but not all of them.

The most surprising result is that Decision Tree and KNN do much better than Ordinary Kriging and MLP on this dataset. This observation sort of proves the previous hypothesis that when there is a lot of non-stationarity in the data, Ordinary Kriging and Neural Network based algorithms won't adapt very well with the non-stationary thus produce inaccurate predictions. An algorithm specifically designed to handle non-stationarity, even as simple as Decision Tree and KNN, can easily outperform complicated algorithms that didn't take non-stationarity into consideration.

On the other hand, it seems as the number of POIs increases, all algorithms tend to do worse except the baseline algorithm, which does nothing but outputting mean value of training dataset no matter what the input is. From this figure, one cannot decide which algorithms is more non-stationarity resistance. So we converted all algorithms' absolute performance (as a RMSE value) to relative performance (as a percentage of its base performance when number of POI = 10).

The result is shown in figure 3.4. From the picture, it can be easily told that the Ordinary Kriging algorithm's performance decreases much faster than the others, which is another proof that algorithms not designed to handle non-stationarity will quickly become unusable when non-stationarity increases in the dataset.

Conclusion

Here are a few conclusions that could be made basing on the tests performed in this section:

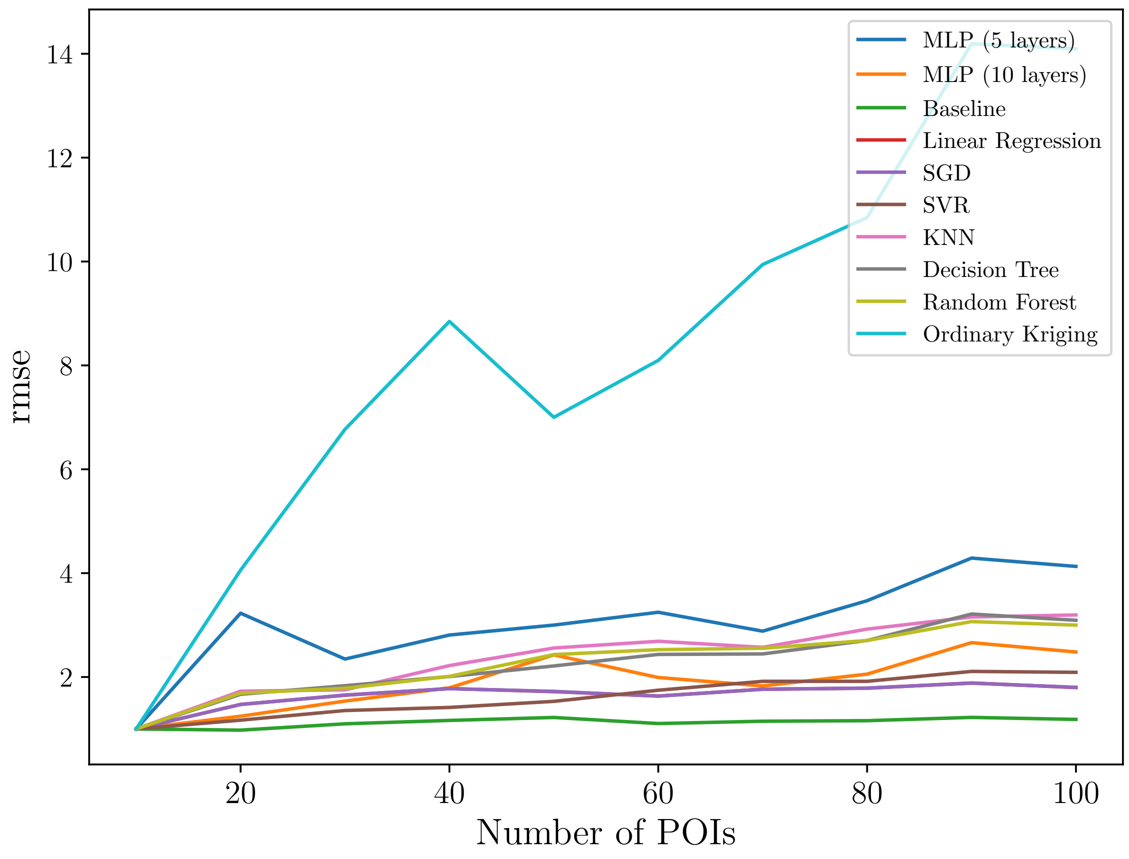


Figure 3.4: Comparison of relative performance

- Non-stationarity can greatly affect algorithm's performances.
- The more non-stationarity there are, the more difficulty it is to learn an accurate model from the dataset.
- On datasets with a lot of non-stationarity, Algorithms that are designed to handle non-stationarity, even the simplest ones, will easily outperform those who don't.
- No algorithm performs universally well. Different algorithms have different assumptions thus only perform well when the underlying dataset matches their corresponding assumptions.

3.4 Evaluating Impact of Correlation

In geographic datasets, there usually exist multiple types of correlations. The first type of correlation is spatial autocorrelation.

As stated previously, spatial autocorrelation comes from the fact that objects with similar characteristics tend to cluster together. It is almost impossible to find a dataset without any spatial autocorrelation. This type of correlation is actually both our friend and enemy. It is our friend in a way that it help predict label values by examine the neighboring data points. And it will also become an enemy when it is inconsistent and hard to capture.

As spatial autocorrelation is such a useful tool for the studying of geographic datasets, many methods were invented to quantify spatial autocorrelation. The most widely used one is Moran's I [Mor50] and its many variants, which is defined as below:

$$I = \frac{N}{W} \frac{\sum_i \sum_j w_{ij} \left(x_i - \bar{x}\right) \left(x_j - \bar{x}\right)}{\sum_i \left(x_i - \bar{x}\right)^2}$$

In the formula, x is the spatial variable that we're concerned of. w_{ij} is a spatial weights matrix. N is the number of spatial data points. W is the sum of all w_{ij}.

But this only provides a general idea of how much spatial autocorrelation is there in the dataset. Spatial autocorrelation can and will vary from place to place. A low spatial autocorrelation calculated from the formula above doesn't necessarily mean that autocorrelation is low everywhere. [Ans95] noticed this issue and proposed a local Moran's I method to alleviate this problem. But it does not completely solve the problem since it only gives a relative measurement of z-score and p-value which answers what region has relatively high or low autocorrelation comparing with the rest of the area.

And to make things even more challenging, non-spatial features haven't been added into the discussion yet. They are highly likely to be both spatial autocorrelated and cross-correlated. [Hol04] provided some insights on multivariate autocorrelation but leave the majority part of the problem unsolved.

Adding Correlation

Thus, due to the fact that it is not easy to measure correlation in a dataset, the idea of using a real world dataset to evaluate how correlation affects the performance of different algorithms seems to be unrealistic. Hence we need to come up with a way to inject correlation into a simulated dataset and use that to test performance of different modeling methods. And it would be the best if the amount of correlation can be controlled to make it easier to observe how algorithms perform when correlation increases or decreases.

The simulated dataset will be comprised of spatial attributes (x and y), non-spatial attributes (other features of the data point), and label values(ground truth, the value that needs to be learned and predicted).

In order to add correlation to the data, 3 sets of features needs to be created:

- One set of spatial features (Spatial Component)
    - These features are highly spatial autocorrelated. They're responsible for creating correlation in the dataset.
- Two sets of non-spatial features (Structural Component)

- These features are used to simulate the structural component in the dataset. Each of the feature is randomly generated and independent of each other. Thus there is no correlation implied by them.
- The purpose of adding them to the dataset is to create a mixture of correlated and uncorrelated features thus the amount of correlation in the dataset can be somehow manipulated. To reduce the correlation, one simply need to assign more weight to the structural components, or less weight to the spatial components. And the opposite operations can also be performed to increase the correlation.

Then, two tests can be performed. $Test_1$ is to mix SPC (spatial component) with STC_2 (one set of structural component). $Test_2$ is to mix STC_1 (the other set of structural component) with STC_2.

For $Test_1$, the label value can be calculated by the formula below, where L is the label value, W_1 and W_2 are the weights for SPC and STC. $L(SPC)$ stands for the generated label value of SPC.

$$L_f = W_1 * L(SPC) + W_2 * L(STC_2)$$

Then, $Test_2$ can be performed similarly using the following formula with everything stays the same except that SPC is replaced by STC_1:

$$L_f = W_1 * L(STC_1) + W_2 * L(STC_2)$$

By changing the values of W_1 and W_2, the amount of correlation in the dataset will also change correspondingly. But if an algorithm performs worse when W_1 increases, it's hard to say whether it's caused by the increase of correlation, or the fact that more features in the input contributes to the output (with more features, it is inherently harder to learn a model as accurate).

Thus a control group of $Test_2$ is introduced, in which correlation is removed (STC_1 and STC_2 are generated independently thus has no correlation with each other) but the difficulty of learning the models remains more or less the same.

Here, with $Test_2$, SPC is replaced by STC_1 so that correlation is gone but the number of features stay the same. If an algorithm performs worse on $Test_1$ than $Test_2$, it can be inferred that the performance loss is caused by correlation in the spatial component SPC.

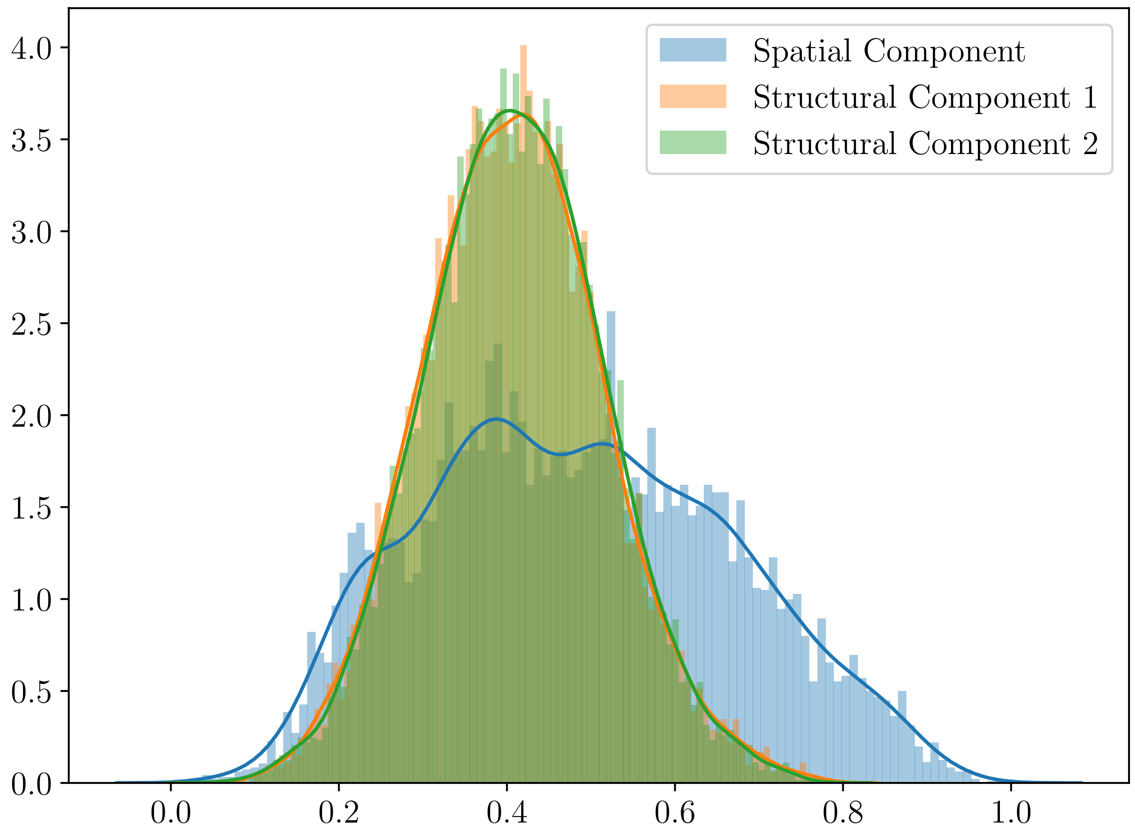


Figure 3.5: Distribution of different components' Label Values

However, there is one more problem to be dealt with. The label values of all three components should have more the less the same distribution. Otherwise they need to be normalized and standardized before added together, which creates additional hassle as the normalization and standardization process may have different impacts to the component data thus introduces unwanted uncertainly to the entire test.

Thus, the formula used to calculate label value from features (this is also the ground truth that needs to be learned by algorithms) is carefully adjusted to make sure the three components' label value distributions are as close to each other as possible. For a final result, see figure 3.5 for the distribution of label values.

As shown in the figure, the two structural components have almost the identical distribution, whereas the spatial component's distribution is more spread out but still more or less comparable with the others. Ideally we'd like to see the spatial component also has the same distribution but that would mean too much manipulation in the construction of datasets, which we would like to avoid if possible. This distribution serves the purpose very well and doesn't involve deliberate fabrication of the formula, thus is the final one that being adopted.

Performance Evaluation

Now that the datasets are ready, we can move forward and test the algorithms. Random forest [LW01] is the first to be tested as it outperformed other candidates from the previous non-stationarity test.

Figure 3.6 shows the RMSE result of the Random Forest algorithm under different alpha values. The last section discussed that the formula used to generate label values for the hybrid dataset is:

$$L_f = W_1 * L(SPC) + W_2 * L(STC_2)$$

In the formula, W_1 and W_2 are the weight values that control how much of the two components being mixed influences the final label value (ground truth). However, we don't truly care what values they are, but the relative relationship

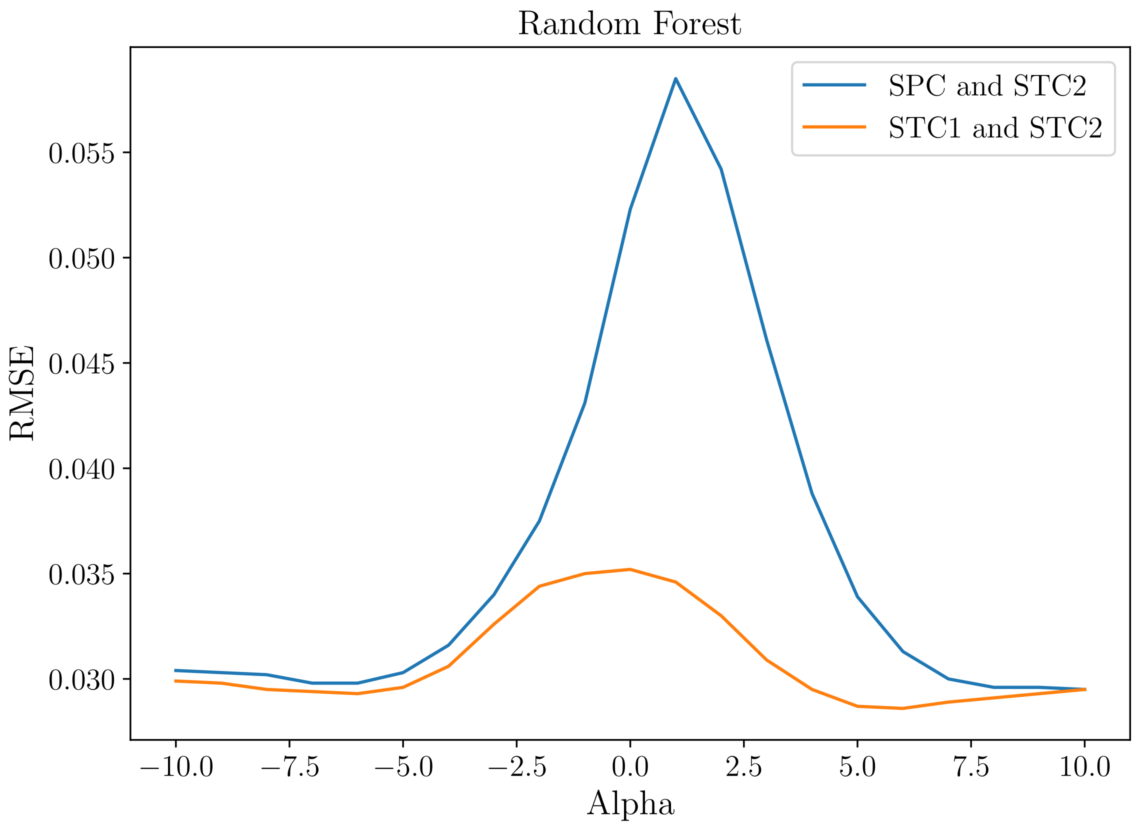


Figure 3.6: RMSE of Random Forest under different alpha values

between them. Thus, the formula can be evolved into the following one:

$$L_f = \beta * L(SPC) + (1 - \beta) * L(STC_2)$$

where β is in the range of $[0, 1]$. With this new formula, only one variable need to be changed to generate various datasets with different mix of spatial and structural components.

The value of β for different test datasets shouldn't be changing linearly, but rather exponentially. This is because we need to observe how algorithms behave under zero, fifty percent, and one hundred percent of correlation, and also have steps denser towards both ends (where changes are expected to happen fast) but sparser towards the middle (where performance doesn't fluctuate as much). Thus,

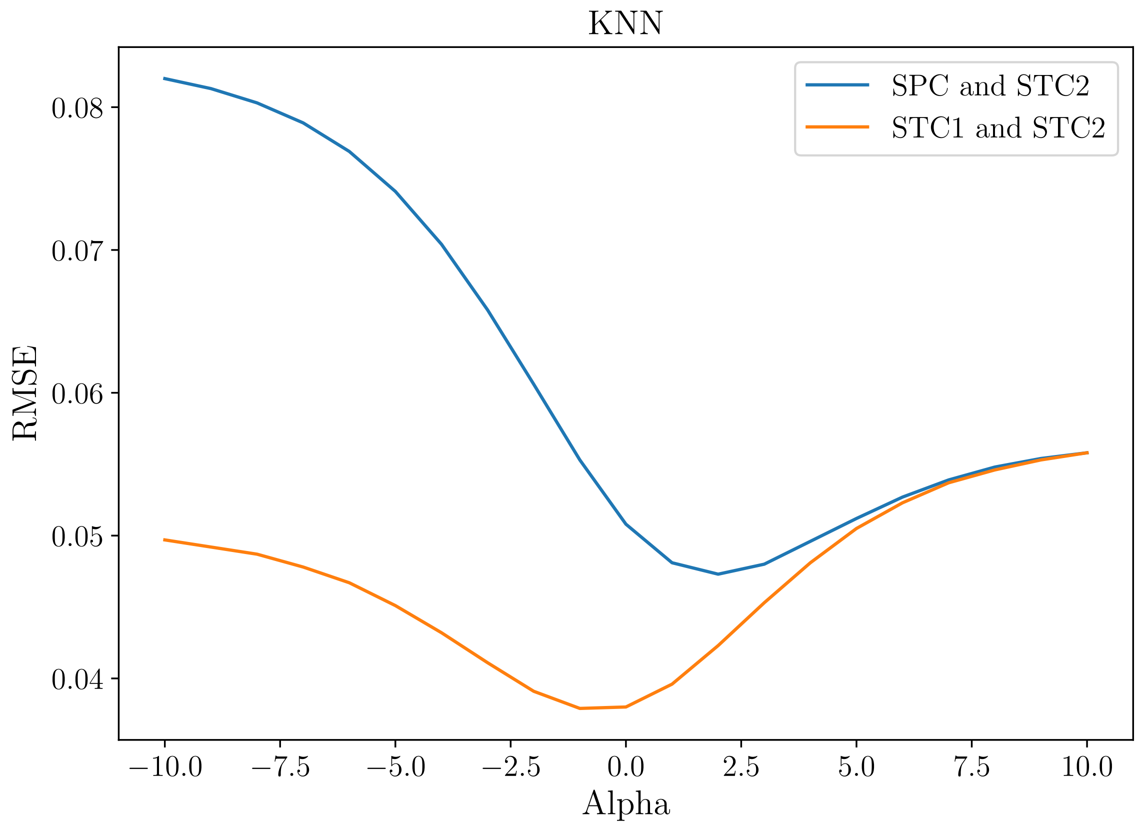


Figure 3.7: RMSE of KNN Regressor under different alpha values

α is introduced as:

$$\beta = 1/(1 + x^{\alpha})$$

Here, x determines how fast β changes exponentially and the value should not be too large or too small, otherwise it would be difficult to capture the changes in performance. After experimenting with different values, $x = 1.5$ seems to be the best balanced values. So, the final formula becomes:

$$L_f = \frac{L(SPC) + 1.5^{\alpha} * L(STC_2)}{1 + 1.5^{\alpha}}$$

So, in figure 3.6, when $\alpha = -10$, it's a mixture of mostly SPC and very few $STC2$. When $\alpha = 10$, the ratio is reversed. And when $\alpha = 0$, it has the same amount of SPC and $STC2$. As we can see, the Random Forest algorithm performs relatively well when there is either a lot of correlation or very few correlation, but poorly when the ground truth is a mixture of both. As a control group, the mixture of $STC1$ and $STC2$, which doesn't have any correlation in them at all, proves that although it's harder to learn the ground truth when mixing two components half and half, due to the fact that more features are involved in the calculation thus implies a rise of dimensionality, the additional challenge brought by this doesn't impose a serious impact to the performance of the Random Forest algorithm. The performance difference under the two situations can be only explained as it substantially difficult to learn an accurate model with a hybrid dataset in which some components are correlated but others don't.

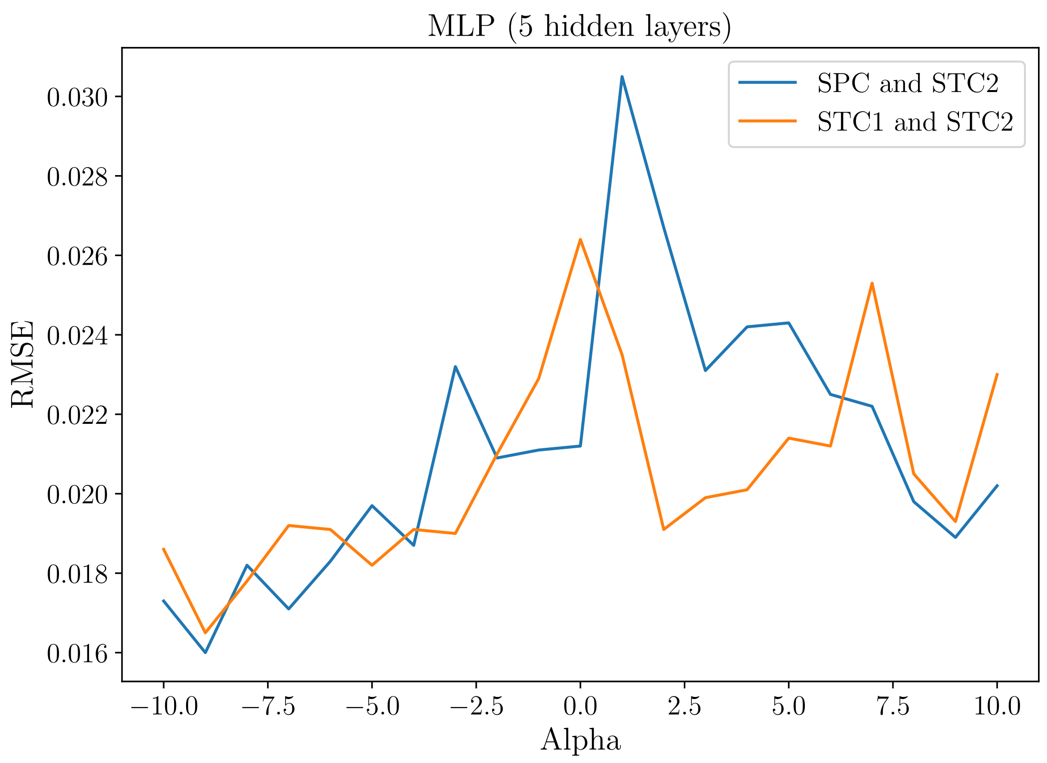
However, with KNN regressors, such behavior is not observed, as shown in figure 3.7. It does not mean the discovery is not true any more, but rather due to the fact that KNN works fundamentally different. The crucial difference is that KNN treats all features as having the same weight, thus performs extremely bad when some features weigh much more than the others. In fact, the more imbalanced the weights are, the worse KNN performs. From the figure, we can tell that this factor has the most impact thus when features are evenly weighted when α is close to 0, KNN's perform gets more gains than the loss brought by mixing correlated features with uncorrelated features. The test is further extended to the MLP algorithm. The results are shown in figure 3.8. As one can see, the performance of MLP is not as smooth as the other algorithms, possibly due to the fact that MLP can not capture the relationship among deeply correlated features very well, thus suffers performance a lot even if the data changes a little. Despite of this, it can be still observed that the mixture of correlated data with uncorrelated data will also cause performance loss for the MLP algorithm.

3.5 Discussion

In this chapter, we discussed the two main challenges in modeling geographic datasets: non-stationarity and correlation. Then proposed and proved it's possible to use simulated datasets to test traditional machine learning algorithms' performance under these challenges.

As a result of the tests, it was found that all machine learning algorithms' performances would be impacted by non-stationarity to a certain degree. Those not specifically designed to handle non-stationarity will suffer from a severe performance loss when used to process datasets with non-stationarity. Algorithms that are designed with the consideration of non-stationarity, even the simplest ones, will handle the situation much better.

On the other hand, correlation will also make models less accurate, no matter what methods were used to learn them. And the impacts will be even greater when some features in the dataset are correlated, whereas the others are not. These impacts are more obvious when the underlying algorithms are inherently good at handling datasets with weighted features but less observable for those who don't.



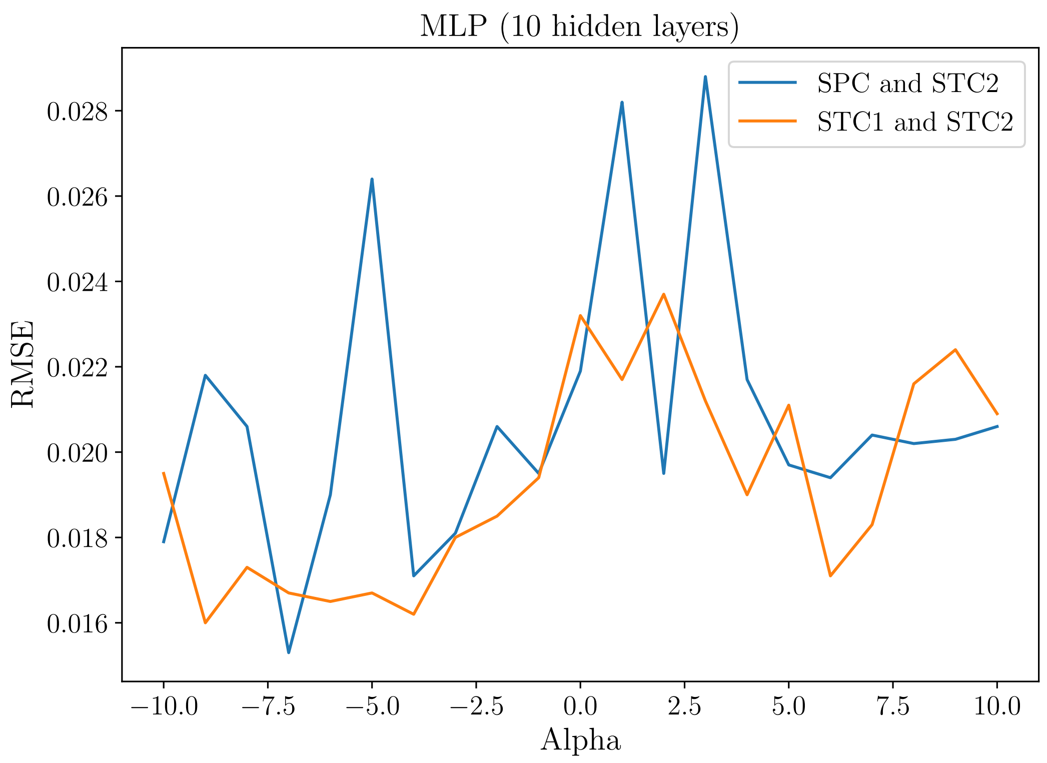


Figure 3.8: RMSE of MLP Regressor under different alpha values

CHAPTER 4

GEOGRAPHIC R-PARTITION TREE: MODELING NON-STATIONARY SPATIAL DATA

* The content of this chapter is an extended version of an in-press paper ([DARssa]) which is accepted by the 2020 International Conference on Machine Learning and Applications (ICMLA).

The previous chapter discussed two main challenges in modeling geographic datasets (especially those from Human Geography): non-stationarity and spatial autocorrelation.

It was shown that specialized algorithms are needed to train strong models for geographic datasets because non-stationarity can cause trouble for algorithms that assume stationarity of data. And, a good amount of information will be lost if spatial autocorrelation is not well taken care of because nearby objects are inherently more similar to each other than remote objects.

Thus, in this chapter, the Geographic R-Partition Tree is proposed to solve these challenges. It first groups spatial objects into blocks utilizing a modified version of R-Tree, then put spatially similar blocks into partitions. An individual model is then trained for each of the partition. Finally, the prediction result is calculated from a weighted average of all the individual models that are close to the prediction target.

Comparing with the other algorithms, this method has its own advantage. It is especially suitable for Human Geography datasets with some non-stationarity, but are not non-stationary everywhere. Blocks that are far away but still spatially similar to each other will be grouped into the same partition and explained by the same model. By doing so, the accuracy of the individual models is greatly improved because there are enough amount of observations to improve the models to a certain degree.

Another advantage of the Geographic R-Partition Tree is the customizability. The block creation process in the modified R-tree can be customized. This ability allows users of the algorithm to program prior knowledge into the algorithm so that the best result is achieved. If no prior knowledge is given, the default block creation function also works very well and can produce models that are accurate enough for the majority of scenarios.

4.1 Statement of Problem

When talking about geographic datasets, most of them can be categorized as either Physical Geography dataset, or Human Geography dataset. The categorization is based on the source of the data. Physical Geography datasets are generated from Earth's natural activities, whereas Human Geography datasets are generated from people's activities. Some examples are [Wil20]:

- Physical Geography Datasets
  - Land and Ocean Boundaries
  - Elevation
  - Weather and Climate
  - Mineral Resources
  - Natural Disasters
- Human Geography Datasets
  - Population

  - Transport and Communications
  - Land Use
  - Buildings, Roads and Points of Interest
  - Administrative Boundaries
  - Wars, Conflict and Crime

Traditionally, scientific researches were more focused on Physical Geography datasets for many important motivations such as learn how to survive and mitigate the damage of natural disasters [SA07], better discover and utilize mineral resources [PM12], understand how to take care of Earth [FFWF10] (and how much damage we've done to her), and so on.

While these tasks remain important and crucial to our daily lives, recent years saw an explosive growth in the number of Human Geography datasets [BBFRS12]. A majority part of the growth comes from the fact that we are now living in a digital world of information explosion. A good example is the widespread use of GPS enabled personal digital devices like cellphones, tablets, smart wearable and home devices. As people began to realize the tremendous underlying values in these datasets, it is now increasingly important to study how to analyze, understand and model these datasets.

As discussed in the previous chapter, a major difference in Physical Geography and Human Geography datasets, from machine learning's perspective, is the amount of stationarity in the data. Physical Geography data tends to be stationary whereas Human Geography data are usually non-stationary. For example, crime activities are decided by different factors depending on the location, whereas the distribution of a certain mineral resources tend to follow the same rule across the Earth.

Due to the history of extended researches on Physical Geography datasets, there was never a lack of methods in studying and modeling stationary datasets. For example, the Kriging method family (such as ordinary kriging, universal kriging, co-kriging and regression kriging) is the most popular one among them. But for datasets with non-stationarity, these methods don't do very well. In fact, according our study performed in Chapter 3, sometimes they perform even worse than the simplest algorithms built to handle non-stationarity such as decision tree.

Thus, algorithms that are specifically designed to handle non-stationarity are needed to build better models for geographic datasets with a lot of non-stationarity in them.

4.2 Literature Review

GWR (Geographically Weighted Regression)

The first renowned method for exploring spatial non-stationarity, known as Geographically Weighted Regression (GWR), was proposed by Brunsdon, Fotheringham, and Charlton in 1996 ([BFC96]). The "main characteristic of GWR is that it allows regression coefficients to vary across space, and so the values of the parameters can vary between locations" ([Mat10]).

The motivation for inventing GWR was that "a single global model cannot explain the relationship between some sets of variables" ([BFC96]). Thus, in order to solve this problem, GWR made it possible for relationships between features and labels to differ across spaces, rather than generating an average global model.

The basic idea of how GWR works is to learn a regression equation for every feature in the dataset, during which dependent and explanatory components are accounted for by examine neighboring data points. And the neighbors contribute differently to this process according to how far away it is, which is why it is called a "weighted" regression. The closer a data point is, the more weight it is assigned. According to Tobler's first law of geography, "everything is related to everything else, but near things are more related than distant things" ([Tob70]).

Extensions of GWR

Soon after GWR was invented, Semiparametric GWR (SGWR) was proposed in 2002 [FBC02]. In this method, GWR is improved to allow some features to have fixed regression equations across the space, whereas others can still be variable.

Another extension is called Multiscale GWR (MGWR), which was introduced in 2017 by Fotheringham, Yang and Kang. This method "is similar in intent to Bayesian nonseparable spatially varying coefficients (SVC) models, although potentially providing a more flexible and scalable framework in which to examine multiscale processes" ([FYK17]). And it improves GWR in a way that it not only adapts to datasets on different levels of non-stationarity, but also provides necessary information to evaluate the scales of different processes.

GRF (Geographical Random Forest)

The latest well-known research on this topic was performed by Stefanos, Tais, et al. in 2019. The method is called Geographical Random Forest (GRF) and was developed basing on the Random Forests algorithm ([Bre01]). The principle idea of this method is to "disaggregate of RF into geographical space in the form of local sub-models" ([GGG$^+$19]). As a result, it produces many local RFs, instead of a global one, for each of the locations including data points nearby.

4.3 Spatial Similarity

Although the methods mentioned above tried to tackle non-stationarity from different angles, they work under a similar way that non-stationarity is learned by creating local models for locations basing on information obtained from nearby observations. And there is a critical disadvantage of this type of methods. When one want to learn an accurate local model for a certain location, he want to include as many nearby observations as possible. But the number of nearby observations are sometimes very limited, as datasets are usually not evenly sampled over the area of interest due to many reasons. To include more observations in the calculation, the distance has to be increased. And as a result, local models will be not so "local". But if we reduce the number of observations, local models tend to be highly sensible to the randomness in the missing of observations.

Essentially, when observations are evenly distributed over the space and non-stationarity is relatively stable at small regions (meaning local models can include more nearby observations safely without losing locality), these methods will perform well. But when those assumption doesn't hold true, a decreased performance is expected.

4.3.1 Dataset

To better understand how algorithms' performances are affected by the size of the train data, we performed the test on a real estate transaction dataset. This dataset will be reused throughout the rest of the chapter so it's important to get acquainted with it.

The dataset was made available by a company named Zillow in the year 2017 during a Home Value Prediction Competition [Zil17]. It contains more than $160,000$

sale records of properties, with 56 features including information about the sale price (sale prices are not included directly, will explain later) and information about the property itself like latitude, longitude, square feet, year built, number of stories, number of bedrooms and bathrooms, tax records, zip code, garage info and so on.

The data itself comes from properties sold in the year 2016 and 2017 in California within counties of Los Angeles, Orange and Ventura. Zillow reserved a small portion of the transaction data in both years for the purpose of testing and calculating the accuracy of competitors' models. But the majority of the transactions are available to everyone hence making it a pretty good dataset with reasonable amount of data that makes it theoretically possible to create a very accurate model.

The goal of the competition, however, is not to predict the sale price. Zillow hid the sale price from the data but made available the log-error between their Zestimate and the actual sale price. According to Zillow, "Zestimates are estimated home values based on 7.5 million statistical and machine learning models that analyze hundreds of data points on each property" [Zil17]. And the logerror is defined as:

$$logerror = log(Zestimate) - log(SalePrice)$$

Here, whether the prediction target is the sale price or the logerror makes no big difference. It only makes feature engineering by human more difficult because the underlying rules of which feature will affect the prediction target in what way is totally unpredictable, as the exact mechanism of how Zestimate works is a secret. But from a modeling algorithm's perspective, it is the same thing as one only need to model how the prediction target relates to a series of observations spanned across the space.

A visualization of the dataset is shown in figure 4.1. The latitude and longitude of the dataset have been transformed but their relative relationship is kept. From the figure we can the data coverage is so good that it pretty much outlines the coastal line of California and some of the most populated places. The large blank areas in the figure is either the sea where nobody lives there, or mountainous and rural areas that are sparsely populated. This also reveals a common rule of Human Geography datasets that observations are usually unevenly distributed across the site of interest, because it's natural for people or activities of people to cluster at a few places but leave large areas sparsely populated.

4.3.2 Data Size and MAE

In order to study how an algorithm's performance would be impacted by the size of training dataset, we performed a test using the Zillow real estate dataset mentioned above.

Each time, observations within a randomly selected square area S (with a side length of L) is used as the training data and fed into the Random Forests (RF) algorithm. The fitted model is then used to predict the same observations from S. The MAE is then calculated between the prediction results and the ground truth. To make the result more reliable, for each side length L, 100 random locations are tested and the mean of all test results is used.

As shown by figure 4.2, the MAE is highly negatively correlated with side length L, meaning the more data is used, the more accurate the trained model would be. In this test, the best MAE is 0.05 whereas the worst one is 0.15. The difference here is definitely unignorable.

4.3.3 Defining Spatial Similarity

So, to learn accurate local models, we need to include as many nearby observations as possible. But the amount of nearby observations are always limited and if we go too far, the learned local models will lose locality. One way to solve the problem is to collect more data. But this is often very expensive and sometimes even impossible, hence a better solution is yet to be found.

Here, we introduce the concept of "Spatial Similarity". An area of a geographic dataset is said to be spatially similar with another area if both areas can be explained by the same or similar local models.

Let:

- A_i denotes an area in a geographic dataset.
- $X(A_i)$ denotes observations within A_i.
- $M(A_i)$ denotes the model trained from $X(A_i)$ using any underlying algorithm.
- $E(M(A_i), A_j)$ denotes the error of using $M(A_i)$ to predict area A_j

Then, A_i and A_j are said to be spatially similar if:

$$E(M(A_i), A_j) \approx E(M(A_j), A_j)$$
$$\text{or, } E(M(A_j), A_i) \approx E(M(A_i), A_i)$$

4.3.4 Utilizing Spatial Similarity

With spatial similarity defined, our insight becomes that many Human Geography datasets could have a lot of spatial similarity in them, which can be utilized to create local models. For example, for criminal activity data, it is totally possible that model learned from one region is also accurate in another remote, disconnected region. This will cause the dataset to be possibly described by very few models with each model fits multiple regions.

Therefore, if we know which regions are spatially similar to each other, we can use data from these regions aggregated to train a better model to describe them, in stead of training a local model for each of the region with very limited amount of data.

Spatial similarity test

To check if this idea works, a test was designed to see if there is any said spatial similarity in a given geographical dataset. Here the Zillow dataset is used as the test subject. From the dataset, 4000 random regions of side length 50 are selected to train models, which are then tested on randomly different regions.

The result is shown in figure 4.3. The average MAE for different distances between A_{train} and A_{test} are calculated and shown as a line plot. As can be seen from the diagram, the MAE is only low when the training area A_{train} and test area A_{test} are very close to each other.

When the distance is smaller than 50, the MAE is also very small. This is due to both A_{train} and A_{test} have a side length of 50. A distance of smaller than 50 means A_{train} and A_{test} are actually overlapping each other, thus giving the most accurate results. After a certain value, the distance doesn't have an obvious correlation with MAE any more. In fact, as distance between A_{train} and A_{test} increases, the MAE almost stays unchanged.

This rule is even more obvious when the test result is rendered as a scatter plot, as shown in figure 4.4. A lot of fluctuation in the MAE actually comes from very few outliers. The formation of outliers may come from multiple factors. One of the reason is that the observations are not evenly distributed in the dataset space. Thus some of the A_{train} or A_{test} may happen to have very few observations in them, thus creating those bad results. If those anomalies are removed from the dataset, the correlation between distance and MAE stays even flatter.

As can be observed from figure 4.4, no matter if the distance is 100, 200, or 400, there are always A_{train} and A_{test} pairs that produces lowest MAE, which implies the two areas are spatially similar to each other.

Also, when distance is in the range of $[400, 500]$, the MAE fluctuates a lot due to lack of sample size thus doesn't affect our conclusion here.

As a conclusion, a model learned from A_{train} won't decrease performance on A_{test} simply because the two areas are far away from each other. This also implies that for a certain A_{train}, all the other areas, no matter how far away from A_{train}, have an equal chance of be spatially similar to A_{train}.

Evaluating spatial similarity

The test above shows it possible that two areas that are far away from each other could still be spatially similar to each other. But it remains uncertain whether this spatial similarity is worth exploiting.

In the extreme case, a dataset whose all areas are spatially similar to each other won't need a specialized algorithm at all. Such dataset is actually a stationary dataset and would be better off if any traditional machine learning algorithm is applied.

Conversely, a dataset whose all areas are spatially dissimilar to each other won't fit our application scenario either. As our insight is that spatially similar areas can be grouped and modeled together to increase the accuracy of the model. If none of them are similar to each other, the insight would not be working any more. But generally speaking, a real world Human Geography dataset is almost impossible to appear such level of dissimilarity. If there exist a dataset like this, it would be inherently difficult to learn an effective model from it any way.

Consequently, to verify if the insight works, another test is performed on the Zillow dataset. This time, a model is trained using all the available data, and then applied to all grids of the dataset to see how many areas are actually spatially similar (has a low MAE with the trained model).

The result, as shown in figure 4.5 and figure 4.6, proves this idea. Only a relatively small portion of all areas (see the dark red blocks on the heat map) have the best MAE with the fitted model. And these spatially similar areas scatter all over the space, instead of being clustered at a few locations.

Also, the distribution of the MAE, as shown in the histogram, shows that MAE values for most areas are between 0.01 and 0.05. There is indeed a good chance that a better model (with non-stationarity taken into consideration) can be built to minimize the MAE for the poorly fitted areas.

4.4 Geographic R-Partition Tree

4.4.1 Algorithm Outline

With the studies from the previous section, the ideas become much more clearer. For geographic datasets (especially human geography datasets) which can be described by limited number of models, we can group spatially similar regions to train models which will then be used for prediction.

Such an algorithm will include these steps:

1. **Spatial Division**: divide the dataset space into small blocks.
2. **Partition**: divide blocks into spatially similar partitions.

3. **Training**: learn models for each of the partition, using any underlying machine learning algorithm.

4. **Prediction**: unknown observations will be predicted by calculating a weighted average of predictions made by all the nearby models.

4.4.2 Terms and Definitions

Before we start, there are some terms that need to be defined:

- Observation
  - An observation is a data record in a geographic dataset.
  - It is essentially a spatial object (which could be either a point, or a rectangle, or a cube, or any shape of any dimension) with non-spatial attributes associated with it. Here we're only dealing with two-dimensional spatial objects but the method can be easily extended to process higher-dimension objects.
  - As an example, for a real estate transaction dataset, an observation is a single transaction record. It has two spatial attributes that are latitude and longitude of the house, and many non-spatial attributes like number of bedrooms, number of stories, year built, tax information and sale price (which is also the target variable)
- Target Variable
  - A target variable is a specified attribute of the observations, like the sale price in the real estate example. It is the attribute that we are especially interested in, so that we want to also know the value for any unknown observations.

  - The task of any machine learning algorithm is to learn a model from known observations, which can be then used to predict the target variable (sale price) of an unknown observation (which we know all of its other attributes but don't know the sale price)

- Block
  - Blocks are generated by a modified R-tree which will be explained later.
  - Every block has a rectangle associated with it, and a list of observations who are bounded within the rectangle. For higher dimension applications, the rectangle can be replaced with a multi-dimensional rectangle and the rest of the method still applies.
  - At the very first step of the algorithm, a modified R-tree will be constructed with all the observations. Then, the lowest level nodes of the R-tree, which are the direct parents of the observations, will be saved as blocks.

- Partition
  - A Partition is a set of blocks which are spatially similar. All of these blocks can be explained by this partition's corresponding P-Model very well (meaning the validation error is minimum).
  - The key to a successful algorithm is therefore being able to divide blocks into the right partitions. The theoretically possible number of ways to partition is B_n, which is a Bell Number with exponential grow rate. Thus a heuristic search in the state space is needed in order to find a good enough partition schema within reasonable amount of time.

- P-Model
  - A P-Model is a individual model that corresponds with a Partition. All blocks in the Partition can be explained by the P-Model relatively well.
  - During the prediction stage, a group of P-Models will be used to predict multiple target values. Then a final prediction is made by calculating a weighted average of these individual predictions, basing on how far away the P-Models are from the prediction target.

4.4.3 Spatial Division

This is the first step of the algorithm where observations in datasets are divided into blocks. The ideal way of dividing the space is, obviously, putting observations who are both close and similar to each other into the same block. Here, the concept of "similar" is the same as in Spatial Similarity defined in the previous section: observations that can be explained or predicted accurately by the same model are said to be similar to each other.

However, despite of this attractive idea, the fact is that this is the very first stage of the algorithm. At this point of time, there is no information of which observation is similar to which at all. So the best we can get is to put spacially close observations into the same region.

Here, we want to quote the famous first law of geography one more time, "everything is related to everything else, but near things are more related than distant things" [Tob70]. Now that the information of observation similarity is missing, the closest one we would get is to assume nearby observations are also similar to each other. By doing this, spatial auto-correlation is also utilized in the best way. In order to do spatial division, there are already numerous candidates out there so it is unnecessary to re-invent the wheel. The most promising ones are the Grid method, Quadtree, and R-tree.

Grid Method

This is the most basic method that simply divides space by fixed number of grids. It is most often used when a fast and simple method is needed to divide the space. However, the major drawback of this algorithm is that, in spatial datasets that the density varies by location, it has a horrible performance as it will create lots of blank grids in sparse areas and create grids that are crowed with observations in dense areas.

Here in our scenario, this drawback is even more exaggerated, because Human Geography datasets usually tend to be highly unevenly distributed. A good example is figure 4.1 in which urban areas are extremely populated, whereas rural areas are either very sparse or totally blank. So, as a conclusion, the grid method is obviously not what we want.

Quadtree

Quadtree was invented by Finkel and Bentley in the year 1974 [FB74]. It is a hierarchical data structure based on the principle of recursive decomposition [Sam84].

The tree is constructed by dividing a 2-D space into four sub-regions, and these sub-regions can be further divided recursively. The sub-regions may be either square or rectangular. See figure 4.7 for an example. The Quadtree in the figure was used by Microsoft to create a Map Tile System and served as the basic structure of the Bing map. It is actually a very popular structure commonly used by most map systems due to its simplicity and recursive structure.

There are a few advantages of the Quadtree. One is it can self-balance on unevenly distributed datasets. This is because each of the Quadtree's subregion can be assigned a maximum capacity. When number of observations fall in a subregion making it reach the maximum capacity, it can further split into four subregions. In case of all the observations are still tightly packed in one of the subregion, the splitting will continue until the number of observations in a subregion do not exceed its capacity anymore.

However, Quadtree is still not what we want as the subregions generated by the algorithm is still divided by fixed grid lines. It is very likely a group of very close to each other observations (thus are highly possible that they're similar and can be explained by one model) happen to be divided into two totally different regions.

R-tree

R-tree was proposed by Antonin Guttman in the year 1984 [Gut84]. It is also a hierarchical tree structure which resembles other hierarchical trees like B-tree and Quadtree (as mentioned above). Although it can be applied to data of any dimensions, the most commonly application scenario is for indexing two-dimensional data.

The main idea of R-tree is to use a minimum bounding rectangle (MBR) to group objects within an area, and organize them into a hierarchical structure. An example of generated MBRs is shown in figure 4.8. Unlike Quadtree which can be built from top down, R-tree is constructed from bottom up. Each of the node also has a pre-defined maximum capacity, inserting new observations will cause it split and increase height when the maximum capacity is reached. For our user scenarios, the advantages of R-tree are obvious for several reasons:

- A minimum and maximum number of children can be specified for each of the node.
  - This flexibility makes it possible to create smaller MBRs in areas where observations are sparse, whereas also allowing reasonably amount of MBRs created in dense areas.
- MBRs are allowed to overlap each other.
  - For the typical scenarios in which R-tree is used as an index, this is an disadvantage. Because the more overlapped MBRs are, the worse overall performance of R-tree is due to more nodes need to be scanned in order to find the desired result.
  - However, in our situation, it doesn't matter or not whether MBRs overlap each other, as overlapped blocks will be likely grouped into the same partition, thus won't impact the performance of the final learned models. Allowing overlap can actually improve the performance as spatially similar blocks are more likely to be overlap each other in a region. In fact, there is a way to turn this overlapping issue into an advantage, which will be explained later in this chapter.
- The choose leaf procedure in the R-tree algorithm is highly customizable.
  - The classic R-tree can be easily customized by changing the choose leaf method, which decides newly inserted observation will be inserted into which leaf node.
  - Over the years, many R-tree variants were developed, like R*-tree [BKSS90], R+ tree, Hilbert R-tree [KF99] and so on.

– Later, we will show a modified version of R-tree which make it generates MBRs more suitable for our algorithm.

Modified R-tree

Despite of these advantages, there are two problem with the classic R-tree that makes it not desired as our choice of spatial division algorithm.

The first problem is that the shape of generated MBRs can be too narrow. This will cause problems sometimes because two groups of points far away from each other can be connected by one MBR in the classic R-tree. See figure 4.8 for an example. The MBR in the center of the figure connects two groups of points that are so far away, that the likelihood of these points are spatially dissimilar to each other has greatly increased.

This is definitely not a desired result that works in our favor. But the good news is that this behavior is fixable by changing R-tree's $ChooseLeaf$ function. To how $ChooseLeaf$ function works, we need to start with the $InsertEntry$ function, which is shown below:

```
// Insert a new Entry into the R-tree
Function InsertEntry(e)
    node = ChooseLeaf(e) // find a leaf node for placing E
    Add e into node, split if necessary
    Propagate change to ancestor nodes
    Increase height of tree if necessary
end
```

The function is called every time a new Entry need to be inserted into the R-tree. The Entry e can be either a point data (with location x and y), or an object with shape (then both location and shape must be given). Then, inside of the $InsertEntry$ function, $ChooseLeaf$ must be called first, in order to find out where is the best fit for this new Entry. After the location is found, the new Entry can then be inserted into the best node. Other necessary changes triggered by the insertion will also be performed like splitting and propagating changes to ancestor nodes, so that the R-tree can maintain its valid structure.

```
// Select the best leaf node for placing a new Entry
Function ChooseLeaf(e)
    $n = root\_node$
    while $n$ is not a leaf node do
        Initialize $best\_placement$
        foreach $c$ in $n.children$ do
            if place $e$ in $c$ cause less enlargement than $best\_placement$ then
                $best\_placement = c$
            end
        end
        $n = best\_placement$
    end
end
```

The $ChooseLeaf$ procedure will traverse the tree to find the best placement strategy for the newly inserted Entry. If the Entry to be inserted can fit any existing node, it will be inserted into that node, obviously. But when inserting the Entry must cause enlargement of an existing node, multiple candidates will be compared to decide where is the best fit. Here, for the classic R-tree, the simplest and good enough strategy is to place the new Entry where would cause the lease enlargement, which makes total sense as with smaller nodes, the R-tree will query faster. And the original purpose of inventing the classic R-tree is to use it as a spatial index to find spatial objects quickly. But in our situation, we don't care about R-tree's query performance, but only want to avoid nodes have their MBR too narrow. Thus, the $ChooseLeaf$ function can be modified as:

```
// A selection process that favors squareness
Function ChooseLeafNew(e)
    n = root_node
    while n is not a leaf node do
        Initialize best_placement
        foreach c in n.children do
            if n is the direct parent of a leaf node then
                if placing e in c has less impact score than best_placement
                 then best_placement = c ;
            else
                if place e in c cause less enlargement than best_placement
                 then best_placement = c ;
            end
        end
        n = best_placement
    end
end
```

Essentially, this $ChooseLeafNew$ function replaces the $ChooseLeaf$ function in the classic R-tree algorithm. For nodes that are not the direct parent of a leaf node, the original algorithm is used because we don't care about the upper level MBRs, but only care about the squareness of the lowest level MBRs.

For these lowest level nodes, the $ChooseLeafNew$ function will find out the *best_placement* that has the least *impact_score*, whereas *impact_score* is defined as:

$$impact\_score = enlargement * (\frac{long\ length\ of\ \text{MBR}}{short\ length\ of\ \text{MBR}})^2$$

Thus, the *impact_score* will penalize MBRs basing on how narrow they are, and these penalties grow exponentially as they become narrower. And the effect of this modification is obvious, as shown in figure 4.9. There are more overlaps in this figure but as explained before this has no impact on the algorithm we're about to implement. The narrowness of MBRs has been greatly reduced as expected.

Customizing the Impact Score Formula

An additional feature brought by the *impact_score* is that it makes the modified R-tree highly customizable. Being able to customize the R-tree may not be so useful for a general dataset without any prior knowledge, but would be really handy if there are some assumptions about the dataset.

For example, on a real estate dataset collected from a certain location, if the area of the largest community is known as A_{max}, the *impact_score* formula can be customized as:

$$impact\_score = enlargement * (\frac{long\ length\ of\ \text{MBR}}{short\ length\ of\ \text{MBR}})^2 * penalty$$

where

$$penalty = \begin{cases} 1, & \text{if } A <= A_{max} \\ \frac{A}{A_{max}}, & \text{if } A > A_{max} \end{cases}$$

This formula is just an example and not necessarily the best way to handle the situation. But it illustrates how to customize *impact_score* by adding a *penalty* to it and make generating blocks larger than A_{max} less desirable.

Thus, this customizability makes the algorithm more flexible. Users can program prior knowledge of the dataset into the formula case by case.

Conclusion

After reviewing different Spatial Division algorithms, we choose the Modified R-tree to perform the task. Because with no pre-knowledge of the dataset, this spatial dividing method has the best chance of putting observations close to each other into the same MBR, whereas keeping the size of all the MBRs balanced.

4.4.4 Partition

After blocks are generated, it is now time to find spatially similar blocks and group them into partitions.

The definition of partition here is the same as what the term "partition" means in mathematics, "a partition of a set is a grouping of its elements into non-empty subsets, in such a way that every element is included in exactly one subset" [Wik20].

Brute Force Search

The theoretical number of possible ways to partition a set of size n is called a Bell Number, which is named after mathematician Eric Temple Bell who studied this number in the 1930s. A Bell Number satisfies a recurrence relation [AKK01] :

$$B_{n+1} = \sum_{k=0}^{n} \begin{pmatrix} n \\ k \end{pmatrix} B_k$$

Here, we first need to know whether it is possible to use a brute force way to find the best partitioning schema by examining each of the partitioning schema one by one. But in reality, the Bell Number grows exponentially [Lov93]:

$$B_n \sim \frac{1}{\sqrt{n}} \left(\frac{n}{W(n)} \right)^{n+\frac{1}{2}} \exp\left(\frac{n}{W(n)} - n - 1 \right)$$

In the formula above, $W(n)$ is a Lambert W function which has a growth rate of logarithm. With this growth rate of B_n, it is impractical to iterate all the possible combinations of partitioning. Thus a heuristic search (or somethings work similar) is needed.

Heuristic Search

If we train a global model M_0 on the entire dataset, it is likely to fit some of the blocks, but doesn't work so well for other blocks. Then, a possible way to improve M_0, is to take those blocks that fits well with M_0 (let's call the set of blocks P_0), and train another model M_1.

Since M_1 is only trained with data from blocks in P_0, it has a much higher chance to fit P_0 than with the rest of the blocks. Thus we say that M_1 is biased towards

P_0. But, there are some blocks in P_0' (the complement of P_0, formal definition: $P_0' = \{block \notin P_0\}$) that may be left out, which could also fit M_1.

Next, M_1 should be tested again with B (all the blocks) to see which blocks fit M_1 best (likewise, let's call them P_1). This is essentially a recursion in which both M_i and P_i can keep improving to a certain degree.

But when does the recursion stop? Ideally, the best scenario would be P_{i+1} is the same with P_i and thus M_{i+1} is also the same as M_i (because if you train on the same data with the same algorithm, you get the same model). In practice, this is nearly impossible to happen. Thus, the stop condition can be set as when P_{i+1} is only slightly different with P_i. Alternatively, this can also be called P_{i+1} starts to converge with P_i. And the convergence threshold can be a constant or predetermined number.

Also, to avoid the rare cases in which P_i fails to converge, another threshold of maximum number of loops can be defined to prevent wasting CPU resources on such cases. This process is similar to a Heuristic Search process because every time it goes from M_i and P_i to M_{i+1} and P_{i+1}, it is in fact a "search" for a better partition than the current one by using the current model to predict the next model.

Implementation

The heuristic search process described above can be generalized into a function $SearchBestPartition$.

Here, $ImprovePartition$ corresponds with the train-and-refine process as described above. It takes a *partition* as the parameter, train a *model* with it, and create *partition_new* which is better than *partition* from the perspective of spatial

```
// search for the best partition
Function SearchBestPartition()
    $partition$ = All blocks
    while True do
        $partition,\ model = ImprovePartition(partition)$
        if partition has converged, or reached max loop count then
            return $partition, model$
        end
    end
end

// given an existing partition, improve it and return the new
   partition along with the new model
Function ImprovePartition(partition)
    $model = Train(partition)$
    $mae\_array = Test(model, all\_blocks)$
    $partition\_new = CreateNewPartition\_draft(mae\_array)$
    $return\ partition\_new, model$
end
```

similarity. Then, $SearchBestPartition$ will keep invoking $ImprovePartition$ until convergence has reached, or the preset maximum number of loops has been reached.

During the described process, a $CreateNewPartition\_draft$ function is needed. Its purpose is to find a better partition basing on the mae result of the current $model$ tested on all_blocks. According to the algorithm above, it should be as simple as returning the blocks with the lowest test error (for which we use mae as the performance indicator). Although there is a problem here (more discussion on this later, and this is also why it's called a draft), for now, it can be temporarily defined as below.

```
    // return the best partition according to given mae_array
1 Function CreateNewPartition_draft(mae_array)
2     sort blocks by mae from smallest to greatest
3     return the first a few blocks in the array
4 end
```

4.4.5 Training

With the ability to generate one partition and its corresponding model, it is now time to extend the algorithm to generate all the partitions. This function will be called $TrainAllModels\_draft$ and looks like below.

```
    // train all the individual models which collectively form the final
       model of this algorithm
1 Function TrainAllModels_draft()
2     foreach i in range(0, K) do
3         $partition, model = SearchBestPartition()$
4         $all\_models.append(partition, model)$
5     end
6 end
```

$TrainAllModels\_draft$ works as simple as invoking $SearchBestPartition$ for multiple times, and store these individually generated models and partitions in an array, which will be used as the final model. However, there are two problems with this drafted version of function:

Problem 1. How to make these partitions cover all the blocks (or, at least cover the majority of them)

- *SearchBestPartition* will return the same partition no matter how many times it is called.
- To cover all the partitions, some measures must be taken to prevent the same blocks from being included by a partition too many times.

Problem 2. How to determine parameter K

- K is the number of models (or partitions) that will be generated.
- The value of K will directly decide how many spatially dissimilar models are needed to describe a dataset.
- A larger K value would potentially be better than a smaller K, because overestimating the number of spatially dissimilar models will only cause duplications in the final model. But underestimating would cause some spatially dissimilar models to be missing, thus causing a potentially worse result when used to predict observations that match these models.

Similarity Score

To address the first problem, there must be a way to exclude blocks that have already been included in a partition from the next invoke of *SearchBestPartition*. But, as discussed above, the *SearchBestPartition* actually does not guarantee to find the best partition. It is only a heuristic search which may lead to a good result, but not guarantee it's the one and only best solution.

Thus, a block should not be excluded from the next calculation simply because it has been included in a previous generated partition, as there is a good chance this block shouldn't have been included in that partition in the first place. What's more, the $CreateNewPartition\_draft$ function only sort blocks by mae and return the ones with least mae values. It is in fact not an all-or-nothing situation here. Blocks with slightly greater mae values than the cutoff line may still influence the model to a certain degree.

So, $CreateNewPartition\_draft$ need to be modified first. The solution to the problem above is, instead of a clear line of whether a block should be included in a partition or not, a weighted result shall be calculated to represent how much relation does each of the block have with the corresponding model. Hence, a $similarity\_score$ function is created to facilitate with this process:

$$similarity\_score(block, model) = \sqrt{\frac{1}{mae(block, model)}}$$

The $similarity\_score$ measures how "similar" a block is to a certain model. It should have a negative correlation with mae because a lower mae value means this block better fits this model. At the same time, the highest mae can easily be ten if not hundred times larger than the lowest mae so a square root calculation is applied to make the final result less sensible to the mae values. In real application, this doesn't have to be a square root relationship but can be replaced by an β value that scales depending on the dataset.

With the definition of $similarity\_score$, the $CreateNewPartition\_draft$ function can be formalized into $CreateNewPartition$, which looks like below.

```
// return the best partition according to the given mae_array
Function CreateNewPartition(mae_array)
    initialize score_array
    foreach mae in mae_array do
        $score = similarity\_score(mae)$
        $score\_array.append(score)$
    end
    sort score_array from greatest to smallest
    return $score\_array.top(\alpha * score\_array.length)$
end
```

In the new version of $CreateNewPartition$, an α value within the range of $(0, 1)$ must be specified by the user of the algorithm, to determine how much percentage of the blocks will be used to train the next model. There is no absolute best value of α because it totally depends on the dataset. A default value of 0.1 can be used but that only provide a general idea. In practice, one can use grid search to find out the best α value.

Coverage Array

With the introduction of similarity score, it is now possible to proceed and solve the issue that $SearchBestPartition$ will always yield the same results. When $SearchBestPartition$ ends and about to return a result, it should not return *partition* and *model*, but *similarity_score_array* and *model*.

Here, *similarity_score_array* is an array of similarity scores of all the blocks to this model. This array can later be used to add to a global *coverage_array* which indicates which block has already been represented by how many models. The *coverage_array* can then be used by future invokes of the *SearchBestPartition* function to avoid covering a certain block too many times.

The formal version of *TrainAllModels* shall look like below.

```
// train all the individual models which collectively form the final
   model of this algorithm
Function TrainAllModels()
    foreach i in range(0, K) do
        $similarity\_score\_array, model = SearchBestPartition()$
        Add $similarity\_score\_array$ to $coverage\_array$
        $all\_models.append(similarity\_score\_array, model)$
    end
end
```

Coverage Score

By far, the algorithm is almost complete but lacking one last thing. Because obviously a *coverage_score* needs to be defined in order to determine how much a block have already been covered, and also for future *SearchBestPartition* calls to assign less weight to highly covered blocks.

The *coverage_score* should be associated with the *similarity_score*. It only needs to be calculated once after *SearchBestPartition* returns, because during the calculation process of *SearchBestPartition*, the algorithm should be consistently skipping the same high coverage blocks. The calculations inside of *SearchBestPartition* is for determining which blocks should be included in the current partition or not. Thus, *coverage_score* is calculated after *SearchBestPartition* returns. Then the result is added to the global *coverage_array*.

There are multiple expected characteristics of the *coverage_score*. First, a block with high *similarity_score* should have a much higher *coverage_score* than any block with lower *similarity_score*. The relationship should be more than linear because low *similarity_score* is really not helpful in building accurate models. One highly accurate model should be much more desirable than many inaccurate models.

But on the other hand, if the formula gives to much weight to a high *similarity_score*, those blocks with high *similarity_score* will then very likely to be only covered a few times by models thus resulting a highly biased model.

Thus, the final formula is a balance and it is defined as the following with γ has a default value of 2, but also adjustable if the user want to.

$$coverage\_score = (similarity\_score)^{\gamma}$$

Initially, all blocks has a *coverage_score* of zero. Every time $SearchBestPartition$ finishes running, all blocks gain some *coverage_score* depending on how similar they are to the current model returned by the $SearchBestPartition$ function.

Conversely, the change of *coverage_score* will affect the $CreateNewPartition$ function, thus also affect how $SearchBestPartition$ runs and priorities those with less coverage. Finally, the generated model will include all blocks but with different coverage on each of them.

Value of K

As mentioned before, K is the number of models that will be generated. Each of the model represents some blocks in the modified R-tree. Model and Block are in a many to many relationship, meaning one model covers many blocks and one block many be covered by multiple models.

In extreme scenarios, a few blocks may not have any models covering them. But that is by design and can be avoided by adjusting the γ value. However, it is not necessarily a good thing to have all blocks covered, because some outliers or erroneous observations may actually decrease the overall performance of the algorithm, and excluding them will in fact be beneficial.

Here, a larger K value could potentially be better than a smaller K but not necessarily always the case. This is due to the same reason that over-fitting is generally considered harmful. But under-fitting is not a good thing either. Thus, it should be left for the user of the algorithm to decide value of K and adjust it according to the dataset. Generally speaking, grid search and several other commonly seen techniques can be used to handle this situation and find the best K value for an actual dataset.

Complete Training Algorithm

Putting all the pieces together from the previous discussions, the completed training algorithm looks like below. Note that the modified R-tree must be created before running these functions to group observations into blocks.

```
Function SearchBestPartition()
    partition = All blocks
    while True do
        partition, model = ImprovePartition(partition)
        if partition has converged, or reached max loop count then
            return partition, model

Function ImprovePartition(partition)
    model = Train(partition)
    mae_array = Test(model, all_blocks)
    partition_new = CreateNewPartition(mae_array)
    return partition_new, model

Function CreateNewPartition(mae_array)
    initialize score_array
    foreach mae in mae_array do
        score = similarity_score(mae)
        score_array.append(score)
    sort score_array from greatest to smallest
    return score_array.top($\alpha * $score_array.length)

Function TrainAllModels()
    foreach i in range(0, K) do
        similarity_score_array, model = SearchBestPartition()
        Add similarity_score_array to coverage_array
        all_models.append(similarity_score_array, model)
```

4.4.6 Prediction

After train and learn the model, it is now time to use the model to predict unknown observations.

The final model of the Geographic R-Partition Tree is a series of models, with each of the model representing a Partition (some spatially similar blocks). Every model also comes with a *similarity_score* array with a *similarity_score* for each of the blocks in the dataset (not just blocks in its own partition).

For prediction, one should first use the location of the unknown observation to determine which block(s) it belongs to. Here, one can do a brute force scan, or use the traditional R-tree search algorithm on the modified R-tree. If it doesn't fall in any block, then the nearest block(s) will be used.

After the blocks are determined, for each of the block, there are K models with different *similarity_score* to this block. Here, a weighted result should be calculated. The calculation process will have the same idea with the IDW algorithm, which takes the following form [She68]:

$$u(\mathbf{x}) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(\mathbf{x}) u_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}, & \text{if } d(\mathbf{x}, \mathbf{x}_i) \neq 0 \text{ for all } i \\ u_i, & \text{if } d(\mathbf{x}, \mathbf{x}_i) = 0 \text{ for some } i \end{cases}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

Inspired by the IDW algorithm, the final prediction result can be defined as a weighted average of the prediction from all the individual models:

$$p(\mathbf{x}) = \frac{\sum_{i=1}^{K} w_i(\mathbf{x}) p_i}{\sum_{i=1}^{K} w_i(\mathbf{x})}$$

where

$$w_i(\mathbf{x}) = similarity\_score(model_i, block(\mathbf{x}))$$

In the formula, p_i is the prediction result given by the $model_i$, and $w_i(\mathbf{x})$ is defined as the *similarity_score* of $model_i$ to the block that $\mathbf{x}$ belongs to.

4.5 Conclusion

In this chapter, a novel Geographic R-Partition Tree is proposed to solve two main challenges in Human Geography datasets: non-stationarity and spatial autocorrelation.

4.5.1 Previous Studies

Although many researches had been done to tackle these challenges, none of them had utilized locality to the extreme. They all work under a similar way that only deals with non-stationarity locally by examine nearby observations.

The main disadvantage for these type of methods is that there are usually not enough nearby observations to train a local model that is accurate enough. So some of the methods simply behaves poorly when data is not dense enough to create accurate local models. For the rest of the methods, they generally use one of two ways to solve this problem.

The first one tries to model the relationship between distance and observation's target value as a function. Although this might work for some of the datasets, it would definitely fail for datasets that this function is non-stationary. For example, in a real estate dataset, the sale price of a property may be affected by the sale price of nearby properties. This distance is highly likely to be smaller in urban areas than in rural areas. And different locations may have their own special situations that make it even more complicated. Thus, this method might work for some datasets with weak non-stationarity, but won't work when non-stationarity is inherently strong. It will especially fail the cases when the relationship between target values and distance cannot be described by a single function.

The other type of methods will use a scalable structure (usually a hierarchical tree or something similar) to describe the dataset. These methods, however, will also fail when their finest level fails. Because no matter how many layers or scales have been created, the base layer usually decide the performance of the entire model. The upper layers are merely created for a purpose of preventing over-fitting, but cannot solve the problem if underlying is inaccurate.

4.5.2 Our Solution

In this chapter, we proposed a totally different to solve this problem. The challenge of not being able to create accurate local models due to limited number of nearby observations is overcame by creating partitions that put spatially similar observations together.

Although this method may not be suitable for stationary datasets, or datasets that are non-stationary everywhere (usually such kind of dataset is hard to exist and even if it exists, it's inherently difficulty to predict), it provides a novel way to deal with non-stationarity.

In the Geographic R-Partition Tree, observations that are far away but still spatially similar are put into the same partition. Individual models are then trained from these partitions. Comparing with the other methods that create a local model for every location, this method will greatly improve the individual models' accuracy because much more observations are used to build the models.

This characteristic is especially important. Because when the number of observations increase but the data density doesn't change, the other methods will not benefit from this growth, as again, the number of nearby observations are still the same if the density doesn't change. But our method can benefit from the growth as long as the non-stationarity doesn't grow (less likely), or it grows but at a slower rate than the number of observations grows (much more likely).

The Geographic R-Partition Tree is also high customizable. The creation of blocks are based on the R-tree process. And R-tree is especially famous for its customizability. During the years, many variants of the R-tree were developed like R*-tree, R+ tree, Hilbert R-tree and so on. Our algorithm keeps this customizability and allows users of the GRP-tree to add prior knowledge into the algorithm to produce even more accurate models.

4.5.3 Possible Extensions

Here, GPR-tree is only designed to deal with two-dimensional observations of any shape. But in fact it is capable of handling higher dimension objects as R-tree can theoretically deal with objects of any dimension.

However, higher than three dimensional objects are almost impossible to be handled effectively by the GPR-tree. As curse of dimensionality will soon render any distance-based algorithms useless. Although the training algorithm do not rely directly on calculation of space, the same assumptions that hold true in a 2D or 3D space doesn't necessarily still hold true for higher dimensions. Also, the prediction process needs to calculate the distance of nearby blocks, whereas the concept of "nearby" may change drastically in dimensions that are higher than 3.

Despite of this, GPR-tree is promising for dealing with 3D datasets, especially for datasets with time as the third dimension, although certain alterations are needed in order to make it work.

For example, in the first step of creating R-tree, bounding rectangles need to be changed from 2D to 3D. After that, all calculations that involves calculation of distance shall also be changed. The assumptions of non-stationarity that is the key to the success of the algorithm, need to be revisited and verified. Besides, whether partitioning still converges under a 3D dataset would also need to be carefully reviewed. During these changes, there might be new challenges that need new solutions. But no matter what, the GPR-tree still provides a totally new option for tackling the non-stationarity.

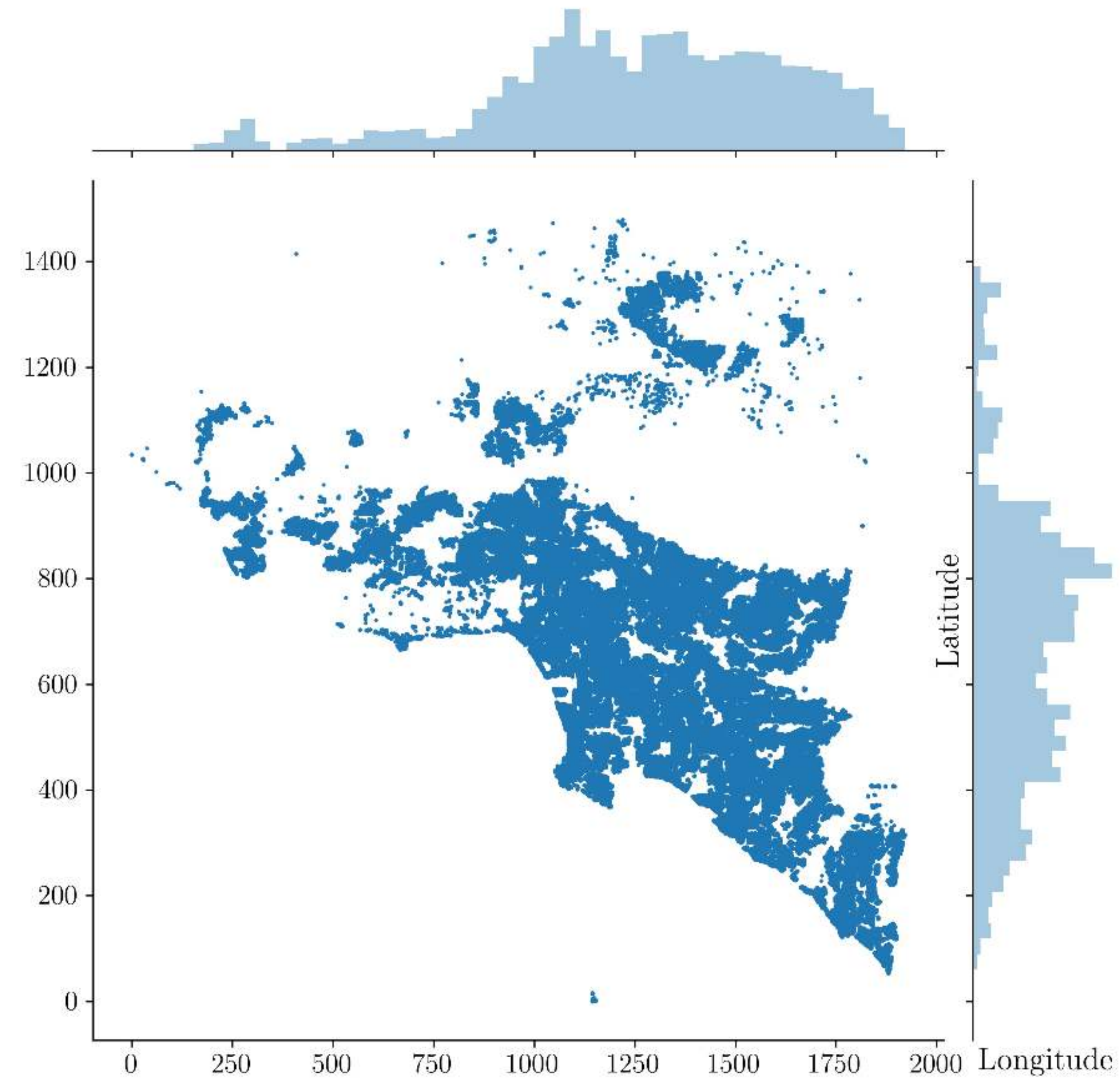


Figure 4.1: Visualization of the Zillow Home Value Prediction DataSet

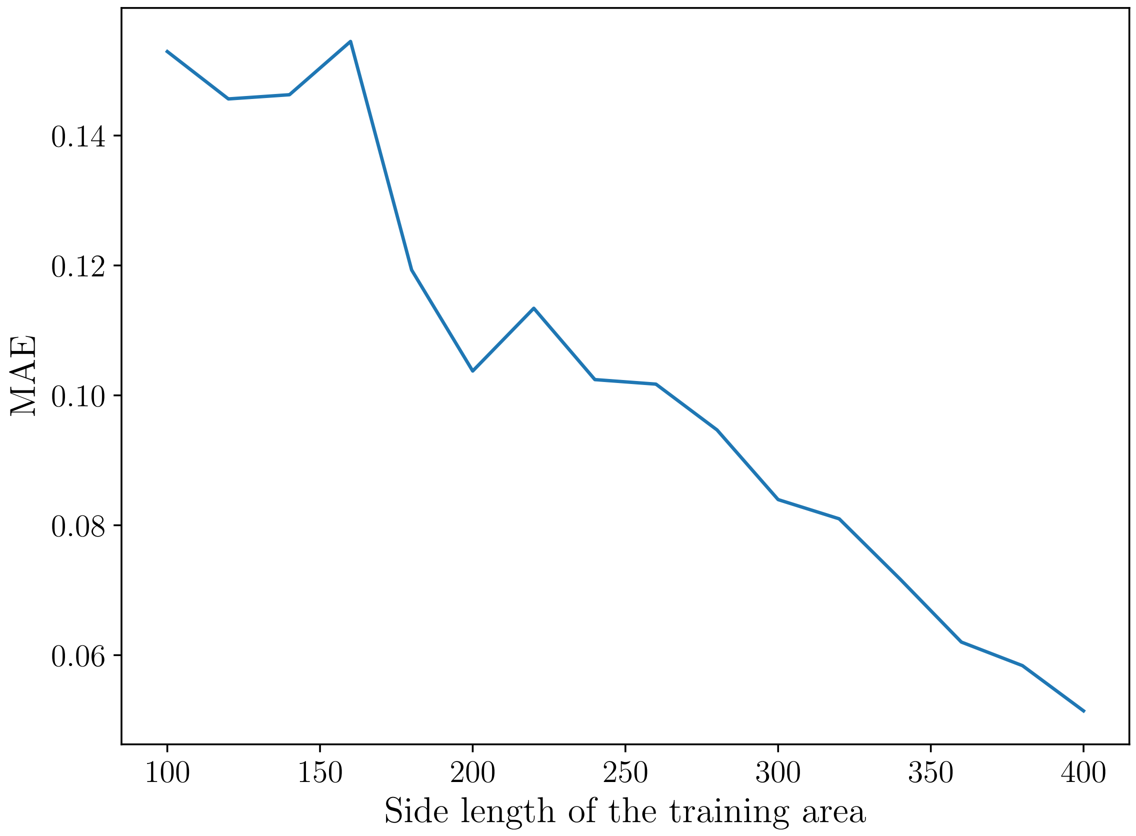


Figure 4.2: Relationship between training area and MAE

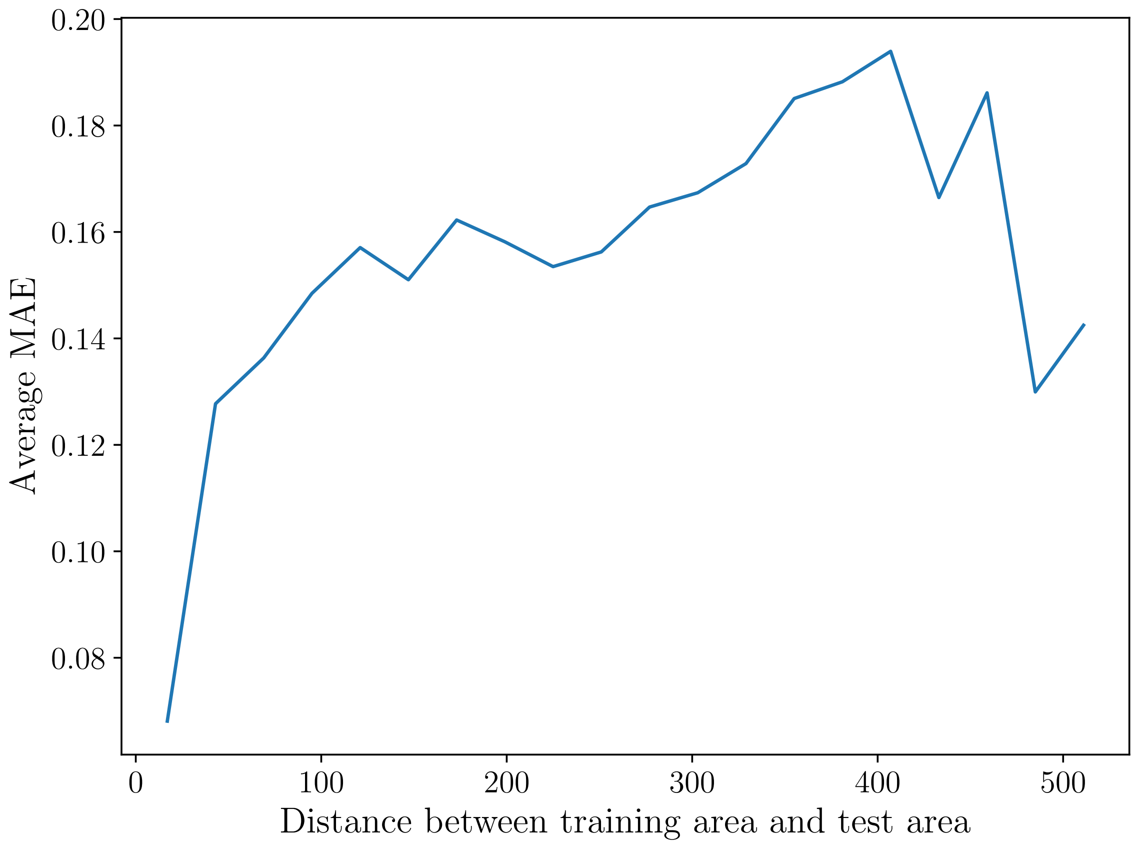


Figure 4.3: Spatial similarity test

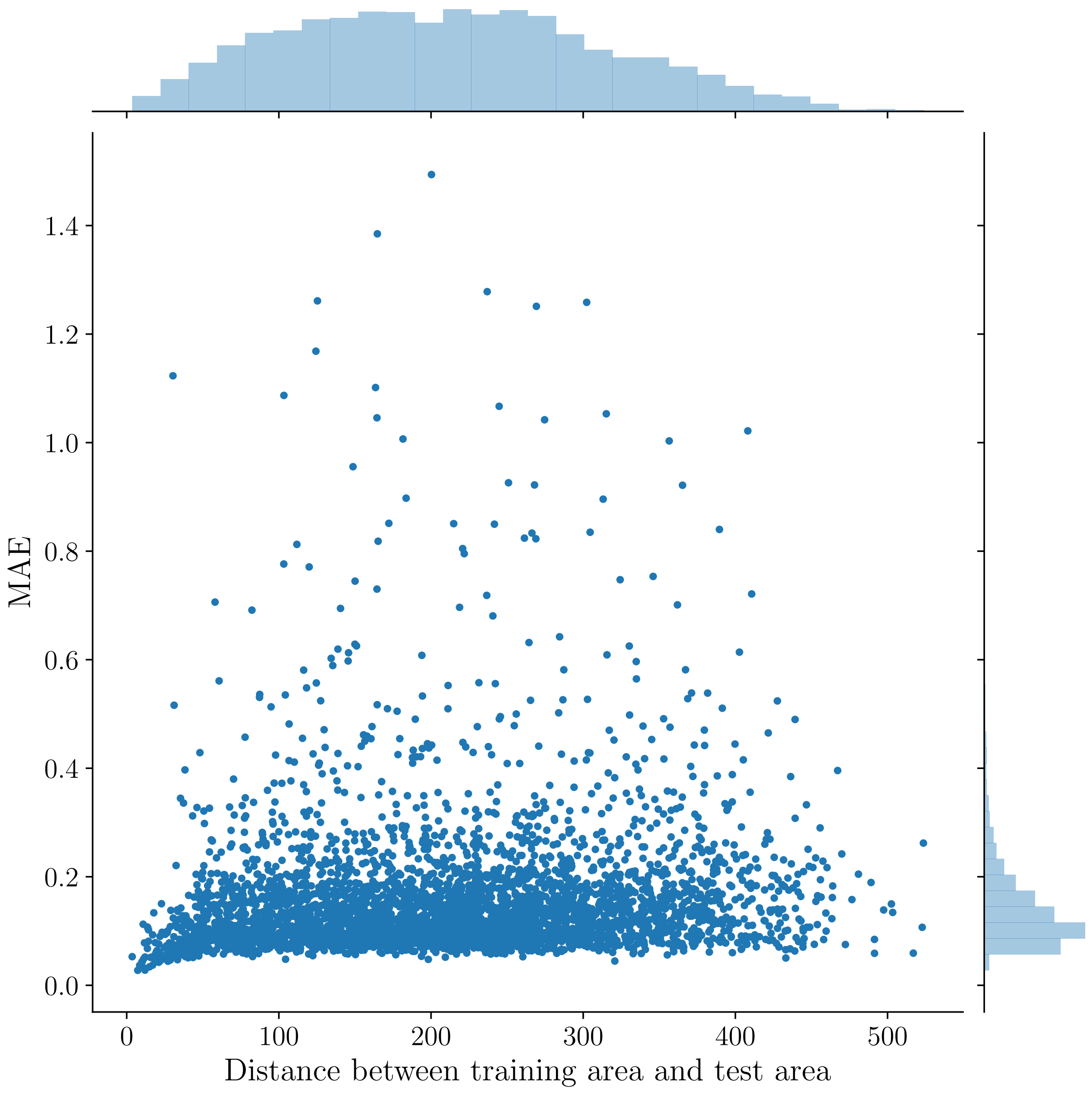


Figure 4.4: Spatial similarity test (scatter plot)

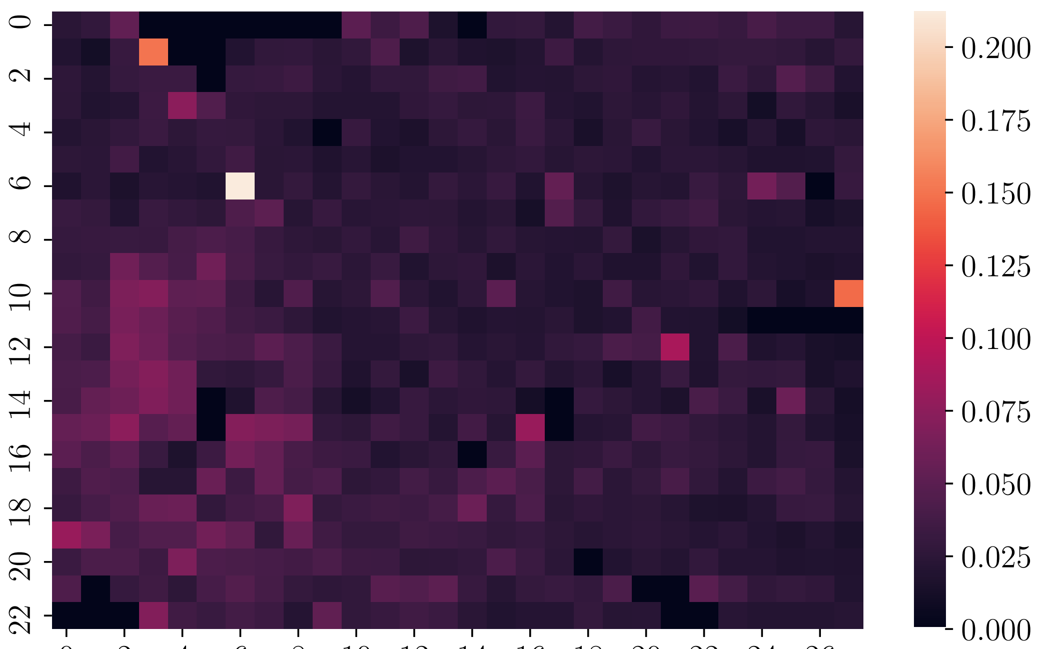


Figure 4.5: MAE of different areas (heat map)

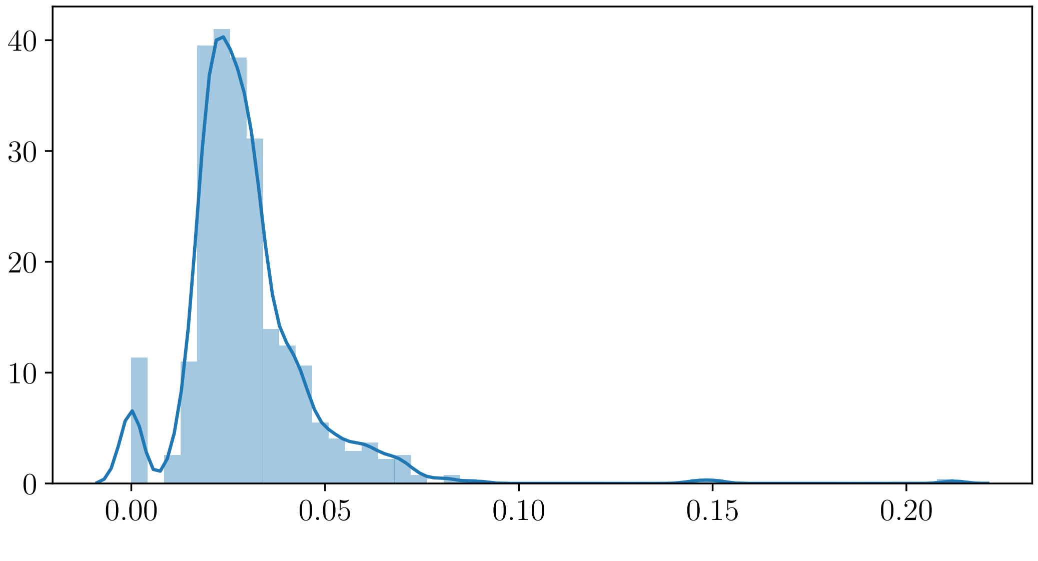


Figure 4.6: MAE of different areas (histogram)

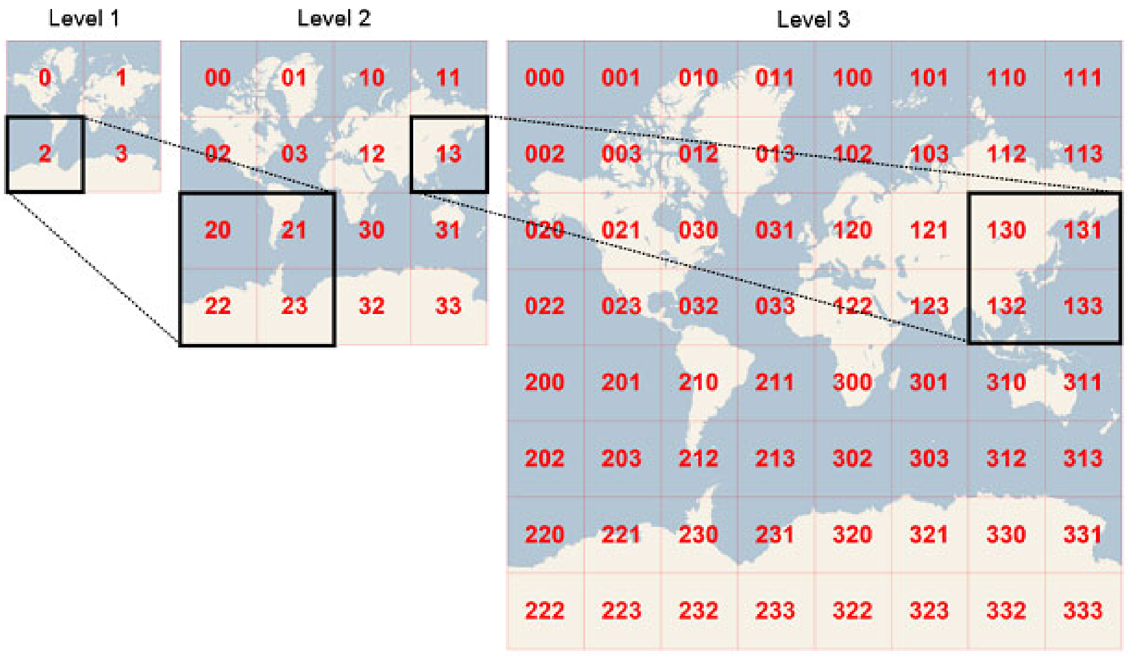


Figure 4.7: Basic structure of the Quadtree [Mic18]

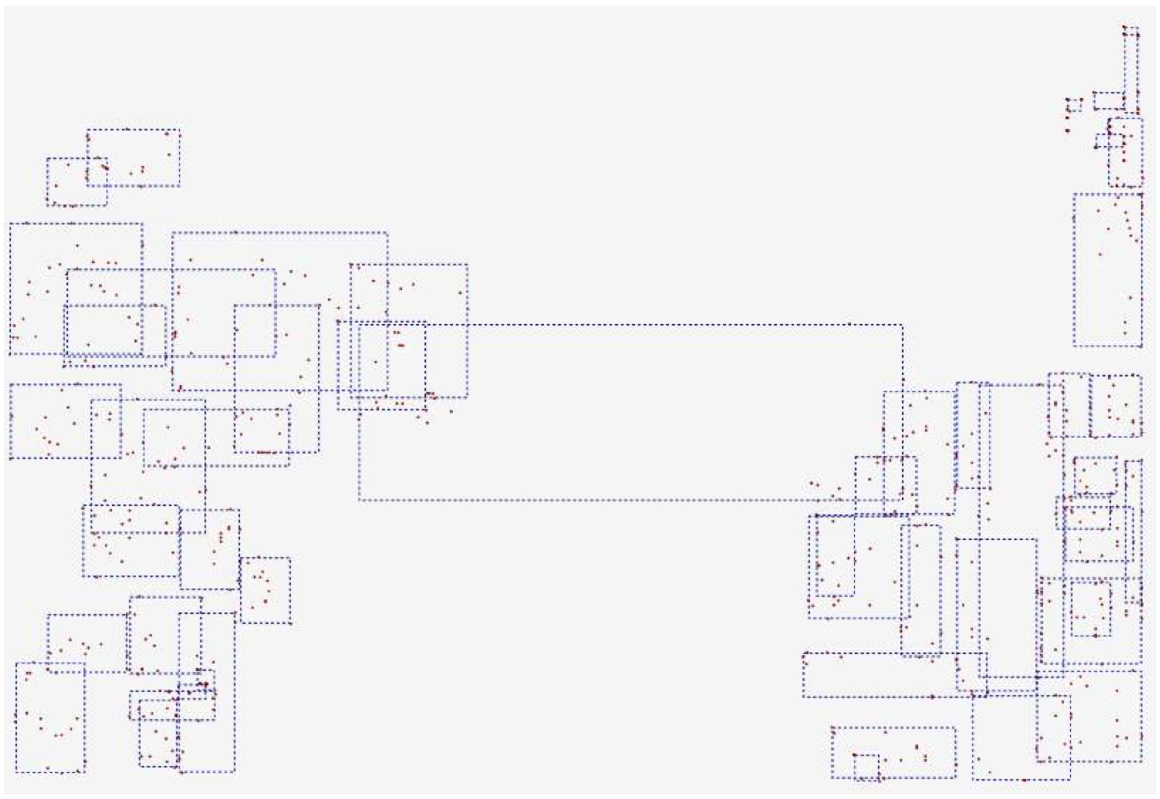


Figure 4.8: MBRs generated by classic R-tree

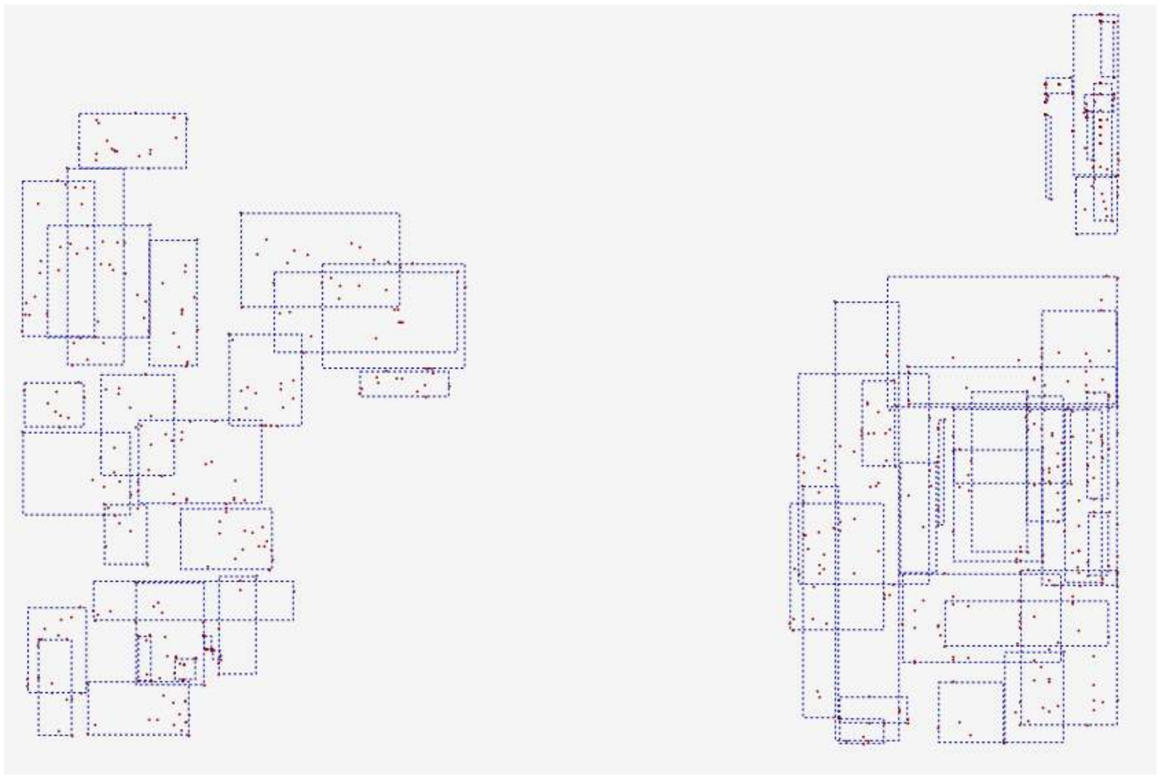


Figure 4.9: Modified R-tree

CHAPTER 5

INVERSE DISTANCE WEIGHTED RANDOM FORESTS: MODELING UNEVENLY DISTRIBUTED NON-STATIONARY GEOGRAPHIC DATA

* The content of this chapter is an extended version of an in-press paper ([DARssb]) which is accepted by the 2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS).

In the previous chapter, a Geographic R-Partition Tree was proposed to model non-stationarity geographic datasets. It proves the idea that for datasets with special characteristics, we need special designed algorithms to handle them.

For this chapter, another algorithm for dealing with non-stationary datasets is proposed. It takes a different approach than the previous R-tree based strategy.

5.1 Introduction

Geographic datasets are usually categorized by the phenomena they describe. Data collected about the natural processes of the Earth are categorized as Physical Geography datasets, such as mineral resources, hydrology, weather, and climate [Mas99]. In contrast, data generated about activities of people are called Human Geography datasets: housing, culture, traffic, disease, war, crime, etc. Historically, researchers were more interested in Physical Geography datasets in order to learn how to survive and mitigate the damage of natural disasters [SA07], discover and utilize mineral resources, understand environmental damage [FFWF10], and so on. But recent years also witnessed an explosive growth of Human Geography data, as GPS-enabled devices are omnipresent in everyday life[BBFRS12].

With new data come new challenges. Many of the tools and theories that worked well with Physical Geography data are incompatible with the new datasets, which have significantly different characteristics. Spatial non-stationarity is one of them. For Physical Geography data, researchers usually assume stationarity, meaning the relationships among features remain unchanged across space. This makes perfect sense because Earth's natural processes, such as the distribution of mineral deposits, will only be relevant to various environmental factors. Location doesn't matter as long as all the environmental factors have been sampled. However, such an assumption is not necessarily valid for Human Geography data due to the complicated nature of human activities. For instance, a house's sale price is affected by numerous factors like the number of bedrooms, garage space, square footage, and so on. When stationarity is assumed, the increase of garage space will cause the same amount of sale price increase everywhere, which is not the case, as garage space can be much more valuable in cities than in rural areas. Even for urban areas, it's unlikely that Paris shares the same pricing model with New York. Thus, it makes more sense to think that the sale price model is affected by "local knowledge" [BFC96], which shifts over space. This local knowledge is not directly included in the dataset because it's difficult to collect or measure, but its influence on the data is real and observable.

A viable solution to the non-stationarity problem is to build multiple local models instead of a single average global model. Each local model represents a subregion of the space within which the data is relatively stationarity. Many studies took this route (e.g., [BFC96], [GGG$^+$19] and [FYK17]) and obtained significant improvements. However, the fact that local models are only trained from nearby observations within a certain area (called the kernel) can be a double-edged sword, and the size of kernels (called the bandwidth) must be chosen carefully [Kal16].

According to [MDHC03], modeling accuracy will be severely impacted if the sample size drops below 1000. Thus, a smaller bandwidth will generate fine-grained models, but each of them is less accurate. In contrast, a larger bandwidth will be less sensitive to non-stationarity and tends to generate local models similar to each other. When data is extremely unevenly distributed, the majority of kernels will contain so few data that it's pointless to adopt the multiple-local-model approach. Figure 5.1 shows the Melbourne housing data (ranging from 2016 to 2018), which illustrates how unevenly distributed data could be for Human Geography datasets.

To solve this problem, we propose the IDW-RF (Inverse Distance Weighted Random Forests) algorithm, which adopts the multiple-local-model approach but allows kernels to overlap. Experiments show that IDW-RF performs as well as other state-of-the-art methods when data is evenly distributed and outperforms in uneven distribution.

5.2 Background

Researchers had been studying how to model non-stationary spatial data for a long time. Brunsdon, Fotheringham, and Charlton performed the first well-known research on this topic in 1996 [BFC96]. In the paper, an algorithm called Geographically Weighted Regression (GWR) was proposed, whose "main characteristic is that it allows regression coefficients to vary across space, so the values of the parameters can vary between locations" [Mat10]. GWR solves non-stationarity by making it possible for relationships between features and labels to differ across spaces, rather than generating an average global model. Many later studies are based on this multiple-local-model idea. For example, [GGG^{+}19] improves GWR by replacing Ordinary Least Squares (OLS) – which is used in GWR to generate local models

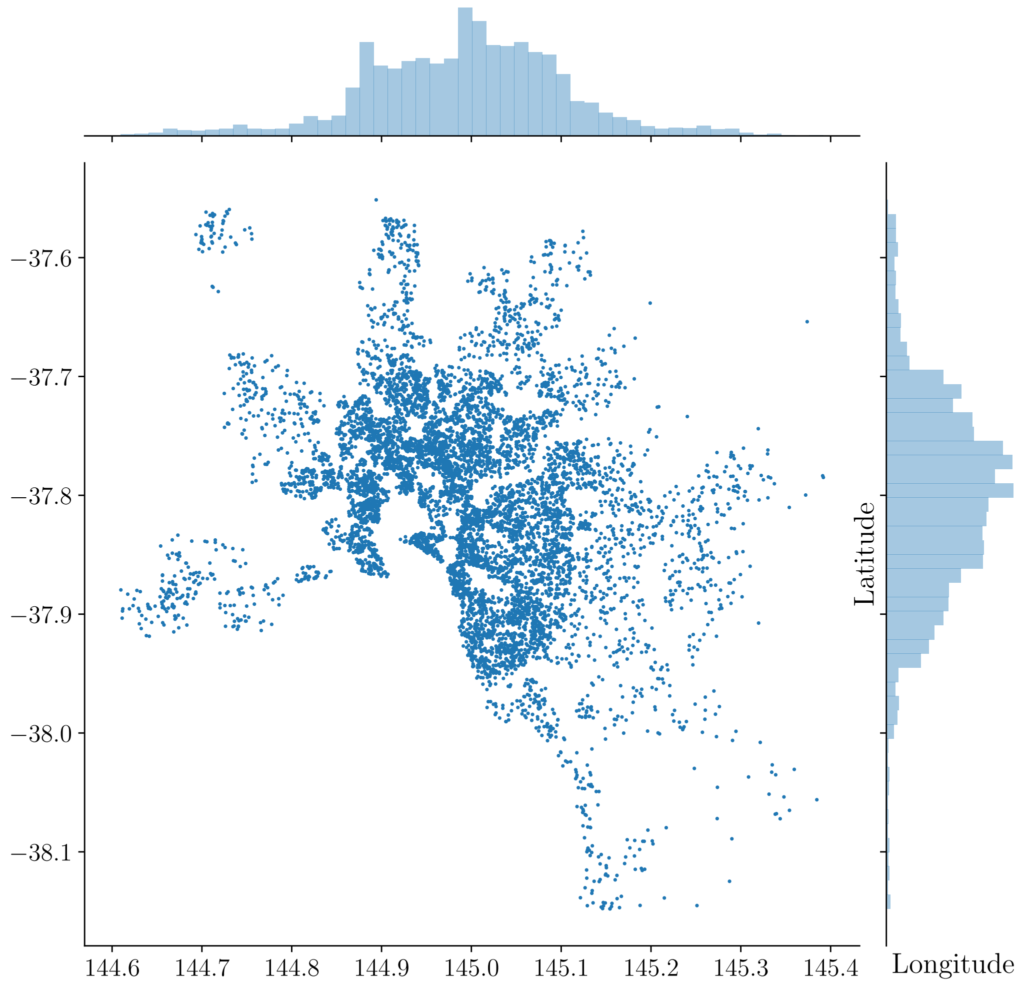


Figure 5.1: Spatial distribution of Melbourne housing data.

– with Random Forests and see substantial improvements, as the Random Forests algorithm is naturally superior to OLS at modeling spatial data. And [FYK17] allows different local models to operate at different spatial scales, thus building more flexible and scalable regression models.

These studies also discussed the problem mentioned above that the bandwidth of kernels must be carefully chosen to get any useful models. [BFC96] uses cross-validation to find the optimal bandwidth. [FYK17] has a complicated weighing method and invented a Back-Fitting algorithm to solve the bandwidth selection problem. [GGG+19] fuses prediction results from global and local models, to improve overall accuracy when the local models are not good enough. However, these methods fail to discuss the case in which data are so unevenly distributed that it's impossible to find a bandwidth that achieves both high accuracy (favors larger bandwidth) and non-stationarity (prefers smaller bandwidth). Here, we present a new approach that allows local models to be trained from very large kernels (typically greater than 10% of the entire dataset) so that bandwidth no longer plays a critical role in the success of the algorithm. The impact brought by letting kernels overlap each other is then minimized by fusing all prediction results with the IDW method.

5.3 Inverse Distance Weighted Random Forests

IDW-RF's design includes five key steps in training and predicting process, which we detail below.

5.3.1 Training: Select Kernel Centroids

As previously mentioned, a kernel is defined as an area in which a local model operates. The number, location, and radius of kernels will have a direct impact on the model performance and, thus, must be carefully chosen. A common way to do this – as adopted by many other researchers – is to use the locations of all the data points as kernel centroids, which also implies that the number of local models will be the same as the number of data points. The main disadvantage of this method is that it's computationally expensive and doesn't scale well when the data size increases. In our case, another drawback of this method is most of the calculations would be unnecessary and even harmful because the adoption of large kernels means most kernels will significantly overlap with each other if there are too many of them.

Here, we use a simple grid-based method to generate kernel centroids. Within the dataset's boundaries, space is evenly divided to grid cells, whose geometric centers are then used as kernel centroids. The value of G – the size of grid cells – can be determined by either prior knowledge of the data, an exhaustive grid search process, or a combination of both. As a general rule of thumb, if prior knowledge is to be used, G should be the best guess on the average range within which data points remain relatively stationary. For example, when predicting house sale prices, homes within the same community are generally believed to follow the same pricing model. In other words, data is stationary on the scale of communities. So for this case, G can be set as the average size of communities. However, in most scenarios, when prior knowledge doesn't exist or cannot be precisely determined, an exhaustive grid search will be used to find the optimal value of G, which will be explained in detail later.

5.3.2 Training: Determine Kernel Sizes

There are two types of kernels. Adaptive kernels are defined by n nearest neighbors, whereas fixed kernels have a predetermined radius r [Kal16]. In the present study, adaptive kernels are used, since they perform much better when data density varies across space, which is the main challenge this paper tries to solve. Instead of n, we use $\alpha = \frac{n}{N}$, where N is the total number of data points, to get a better idea of what percentage of data are used to train each of the local models. Similarly to G, this parameter is also tuned by the grid search process.

5.3.3 Training: Create Local Models

After kernels are chosen, local models can then be trained. Here we use the Random Forests [Bre01] algorithm to create local models. As the name suggests, Random Forests use multiple randomly generated decision trees to predict unknown observations. Each of the trees is trained by only part (about two-thirds) of the data points available. Moreover, during the feature selection process, each decision tree node only chooses from a random subset of features. The final prediction result is either a majority vote (for classification) or an average (for regression) of results from all the trees. The theory behind RF is that the bagging process will decrease the variance of the model without increasing the bias, leading to better overall model performance.

We choose RF to create local models for several reasons. First, RF will not over-fit no matter how the number of trees is increased. According to a study of the Random Forests [Lou14], expected generalization error of ensembling decision trees has a variance of:

$$var(x) = \rho(x) \cdot \sigma^2_{\mathcal{L},\theta}(x) + \frac{1-\rho(x)}{M} \cdot \sigma^2_{\mathcal{L},\theta}(x)$$

in which M is the size of the ensemble and $\rho(x)$ is Pearson's correlation coefficient between two randomized models trained from the same data. Thus if $\rho(x)$ is smaller than 1 (which is always the case for RF), increasing M (number of decision trees) will always cause $var(x)$ to decrease. This characteristic of RF is critical to the success of our algorithm, which is de facto an ensemble of decision trees that trained on the same or partially different data.

Second, RF is based on decision trees, which is naturally good at handling coordinates in geographic datasets. Many other models have trouble with them because latitude and longitude will be treated as independent variables if fed directly into the model. In such a case, loss of spatial information would be unavoidable, and

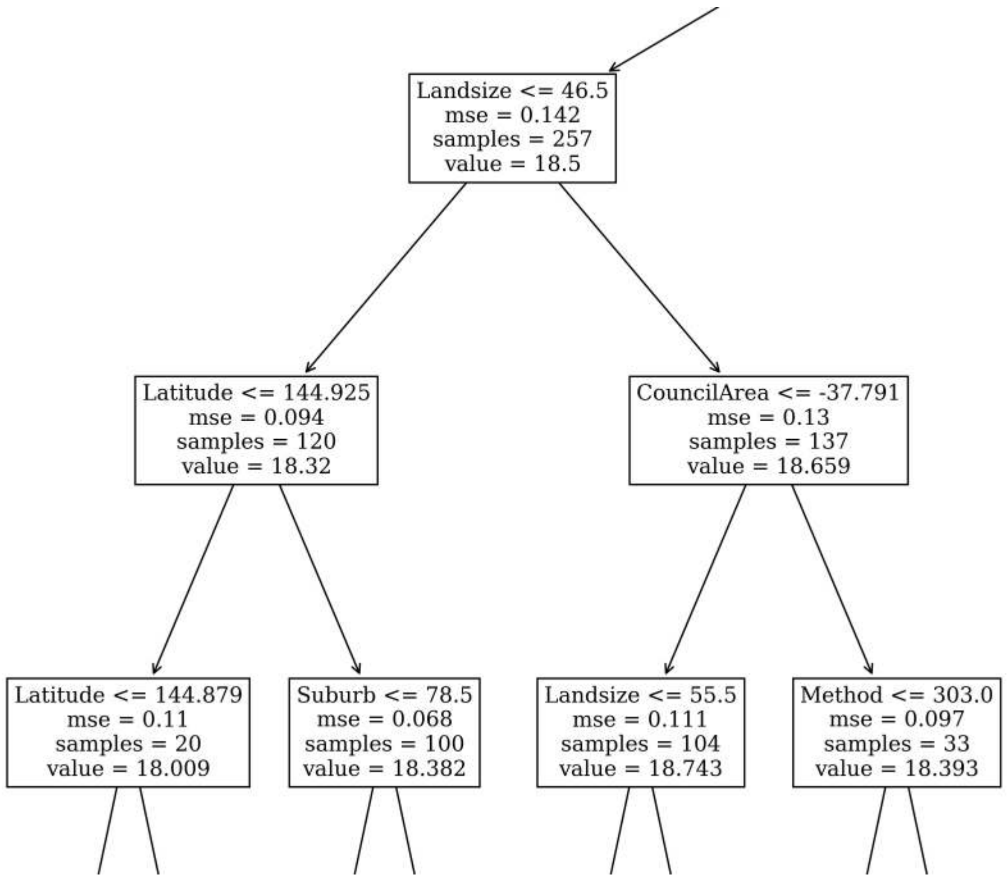


Figure 5.2: A branch of decision tree trained from Melbourne housing data.

the model's effectiveness would be undermined. As a workaround, researchers often run a feature engineering process on geographic datasets before training, to capture the information embedded within data locations and convert them into additional features. But this method is often unreliable, and the results largely depend on the skill of the person who performs the feature engineering process. But decision trees do not suffer from such complications. If a leaf node of the decision tree is examined, its criteria are determined only by the set of ancestor nodes all the way up to the root, for which the order doesn't matter. If latitude or longitude appears in any of its ancestor nodes, like in the example shown in Figure 5.2, the leaf node can be considered as operating within the area defined collectively by all its ancestor latitude/longitude nodes. This is precisely how we expect location information to be accounted for.

Additionally, Random Forests is one of the best machine learning algorithms available and often shows excellent potential when dealing with spatial data, as observed by many studies, including [BJKK12] and [NSB+18]. Using RF as the underlying local model will enable us to inherit all of these advantages.

5.3.4 Predicting: An Inverse Distance Weighted Approach

To predict an unknown observation, results from all local models are combined with the following formula:

$$f(x) = \frac{\sum_{i=1}^{N} w_i(x) f_i(x)}{\sum_{i=1}^{N} w_i(x)},$$

where

$$w_i(x) = \begin{cases} d(x, x_i)^{-p}, & \text{if } d(x, x_i) \geq L \\ L^{-p}, & \text{if } d(x, x_i) < L \end{cases}$$

Here, $f_i(x)$ is the prediction result for unknown observation x, given by the ith local model. $w_i(x)$ is the weight, which decreases as distance $d(x, x_i)$ increases between x and the local model's kernel centroid. p is a positive real number, called the power parameter. L applies a lower bound to the distance function, to avoid the situation in which x is so close to a local model that renders all others useless. This situation is generally not a problem when IDW is used for interpolation purposes but is harmful in our case. Being close to the centroid of a kernel doesn't make the data point a better fit for the local model than others. As a general rule of thumb, L should be smaller than G (grid cell size), and here we use $L = G/2$ in our model. Its value can be fine-tuned. But experiments show that as long as L stays close to $G/2$, its value doesn't have an observable impact on overall accuracy.

There are multiple reasons why this inverse distance weighted approach works. For one, the idea is in accordance with the first law of geography "everything is related to everything else, but near things are more related than distant things", which was proposed by Tobler in 1970 [Tob70]. Spatial heterogeneity is accounted for in this method by assigning larger weights to closer local models. Another reason is that the adoption of large kernels significantly improves the local model's accuracy, which then benefits the entire model. Although ensembling local models trained from the same data could increase the variance of the generalization error, this possibility is eliminated by adopting Random Forests as the underlying local model, which doesn't over-fit no matter how many decision trees are trained, as discussed in the previous section.

5.3.5 Exhaustive Grid Search with Cross-Validation

So far, the algorithm is almost complete except several parameters are not yet determined: grid cell size G, kernel size α, and the power parameter p in the IDW formula. Best values for these parameters depend on the dataset and have to be fine-tuned on a case-by-case basis. As a general idea, datasets with larger scales of non-stationarity tend to favor a large G value. The smaller the dataset is, the larger α should be to offset the impact on accuracy brought by small training data size. And large p values should be used when the data is highly non-stationary.

To get the best performance, our research utilizes the Grid Search method to determine the best G, α, and p. The grid search method runs an exhaustive search on a predetermined hyper-parameter space. Each parameter has a lower bound, an upper bound, and the number of steps. The method will attempt all parameter combinations to find the best one. This process is considered to be computationally expensive. However, since we only run this process during the training stage, it is generally not a problem for most applications that are not sensitive to long training time.

Still, Grid Search alone is not effective enough as it is prone to variance problems. Model performance obtained from one test may differ from the others due to randomness in the tests performed. If not dealt with, this variance may propagate further down the line and cause the parameters learned from the Grid Search process to be biased. For this reason, we add Cross-Validation [All74] to the evaluation process of the Grid Search. A straight-forward K-Fold Cross-Validation would be sufficient for a general problem, but the story is very different for a geographic dataset. A randomly generated training set from the regular K-Fold algorithm is not necessarily equally random at all locations. Many researches have noticed that this could lead to potential issues and proposed different cross-validation strategies [Lie11]. In our study, we adopt the Block Cross-Validation method, which splits data into blocks from which samples are equally withdrawn. For this method, there is no set rule on how large the blocks should be and how many folds (i.e., the value of K) work the best. Generally speaking, the blocks should be of the same scale on which the data remains stationary. And although a higher K gives better results, it would significantly extend the Grid Search process. Thus, K should be set as high as the computing time limitations allow.

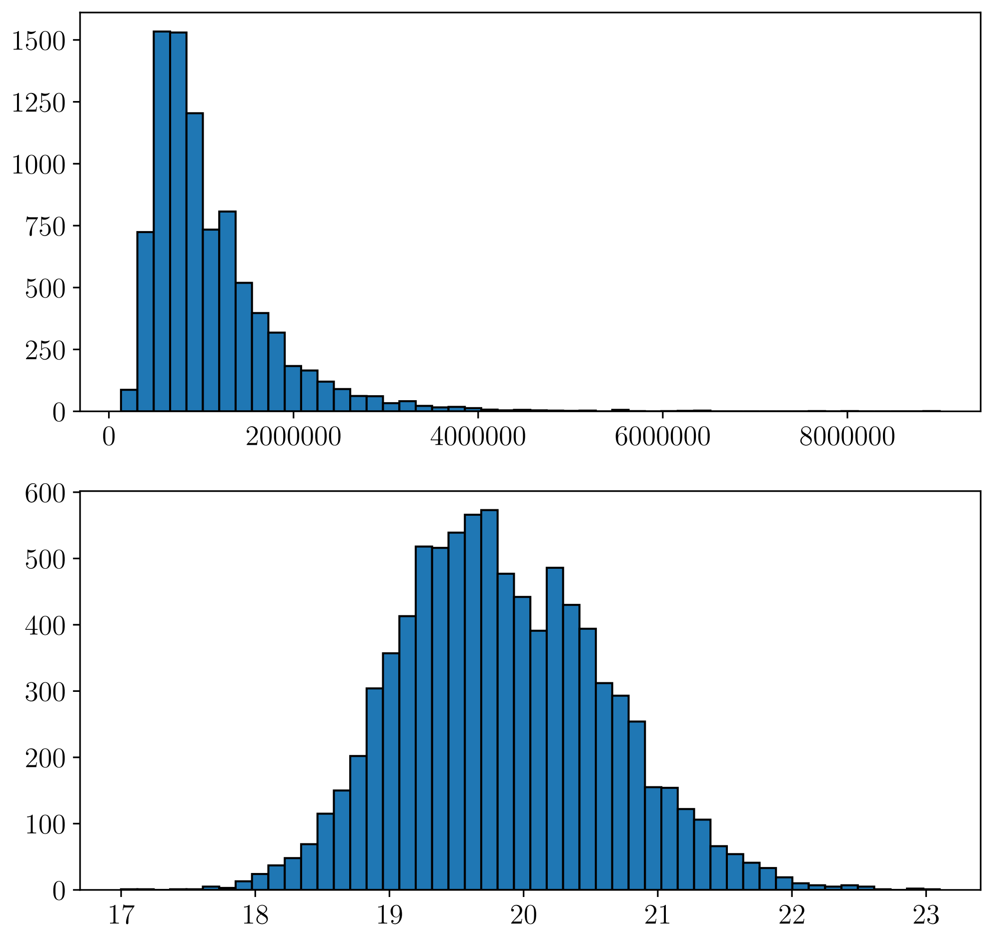


Figure 5.3: Histogram of Price vs. log(Price).

5.4 A Case Study: Melbourne Housing Market

In this section, the IDW-RF algorithm will be applied to the Melbourne Housing Market data (downloaded from the Kaggle website [Pin18]). The dataset contains 8,841 (records with missing fields are stripped) real estate transaction records in the city of Melbourne in Australia from 2016 to 2018, during which the area was experiencing a housing bubble. As shown in Figure 5.1, this is a very typical non-stationary human geography dataset in which data is extremely unevenly distributed. In densely populated areas, there are more than enough data points to outline the Port Phillip Bay's coast. But rural areas only see sparse data points scattered all over the space. Thus, this is a perfect dataset to test the capability of the IDW-RF algorithm.

5.4.1 Data Cleaning and Exploratory Analysis

The original dataset has 21 features which can be categorized into these groups:

- Location related features: latitude/longitude, zip code, suburb, region, etc.
- Transaction related features: sale price, date, seller and sale method, etc.
- House related features: the number of bedrooms and bathrooms, garage space, land size, etc.

Of all these features, the sale price is the target variable we would like to model and predict. After plotting the house price value as a histogram, we immediately realize that it spans over a broad range with a long tail, as illustrated in Figure 5.3. Thus we adopt log(Price) – which has a normal distribution – instead of using Price directly as the target variable. Among the rest of the features, Latitude and Longitude are the most important ones giving us the precise location of the house. The address field is unnecessary as it's inferior to the coordinates and cannot provide any other useful information. However, other location-related features, like zip code and suburb, are kept even though they are derivable form the coordinates, as they have sharp boundaries affecting the tax and school district of the house. And all transaction/house related features are also useful.

Figure 5.4 shows the Pearson correlation heatmap among the most important features. From the figure, the sale price has an obvious positive correlation with the number of rooms/bathrooms and the building's area (square meters of the living area). It also has a solid negative correlation with year built and house type. The

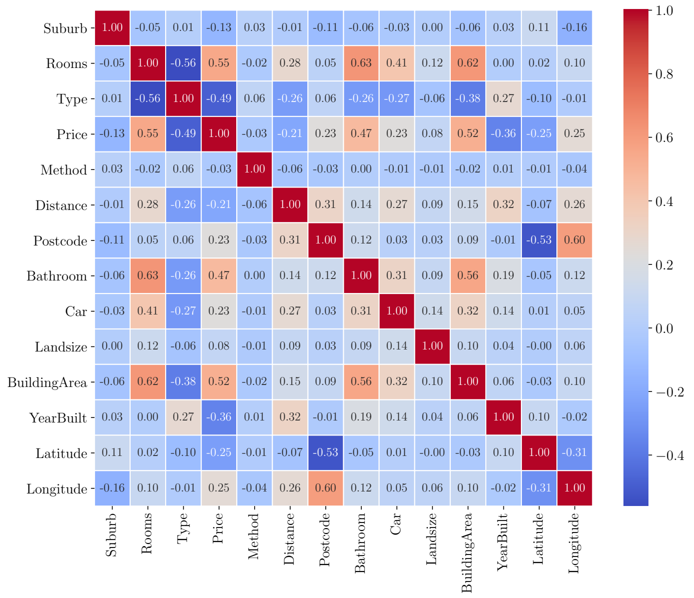


Figure 5.4: Pearson correlation of all important features.

sale method (how the house was sold) is not correlated with anything; thus, it can be removed from the modeling process. None of the rest of the feature pairs show a strong positive or negative correlation. Therefore, it's safe to retain all of them.

However, Fig. 5.4 only gives us the global average correlation among features, which doesn't tell anything about how it could vary across space. Therefore, we split the entire coordinate space into 80 grids ($10 * 8$). To reduce randomness introduced by the small sample, grids with too few data points are removed from the rest of the calculation. In each of the grids left, the Pearson correlation coefficient is calculated between all regular features and the target variable log(Price). Then, all the results are summarized to form Table 5.1. Here, we can see that Landsize has the largest standard deviation among all features, which means it's probably "more non-stationary" than the others. Nevertheless, almost all features show a great difference between the minimum and maximum correlation, which is an indication that the level of non-stationarity cannot be overlooked in this dataset.

Table 5.1: Statistics of Correlations across space

	mean	std	min	max
Suburb	-0.01	0.18	-0.28	0.43
Rooms	0.68	0.11	0.47	0.87
Type	-0.67	0.12	-0.84	-0.38
Distance	-0.09	0.19	-0.39	0.22
Postcode	0.03	0.15	-0.27	0.27
Bathroom	0.48	0.12	0.24	0.74
Car	0.35	0.11	0.14	0.55
Landsize	0.31	0.28	-0.16	0.80
BuildingArea	0.61	0.13	0.28	0.83
YearBuilt	-0.28	0.17	-0.57	-0.02

5.4.2 Assessment Measurements

Before proceeding, we still need to decide how to measure the accuracy of our models. The choice of measuring method would affect the creation of Random Forests and the Grid Search process during which parameters are optimized. The most commonly used error measurements are: mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). Here, we prefer MAE (see the equation below) as the latter two methods tend to penalize large errors, which makes them unfavorable in our situation.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

5.4.3 Results

Now that everything is ready, we apply the IDW-RF algorithm to the data. Results returned by the Grid Search are plotted as an Average-Max-Min chart, as shown in Figure 5.5. We can see that a valley is present in all the three graphs, where MAE is the lowest. According to the results, a kernel size of 0.22 (meaning 22% of all data points are used to form the kernel) is optimal for this particular dataset. This observation matches our theory that when data points are sparse in most areas, a large kernel will produce better overall results. But if the kernel becomes too big, the model will fail to capture non-stationarity, as all local models tend to behave the same.

Additionally, the optimal grid cell size is 0.039 (the difference in the longitude), which translates to about 3.4 kilometers in the city of Melbourne. And our assumption was that grid cell size should be roughly on the same scale with which the data remain stationary. Although for this dataset there is no way to know on what distance would the house price model remain stationary, it is reasonable to believe it's of the same scale of 3.4 kilometers. As for the power factor, its range of MAE is way smaller than the others. The implication of this is that the optimization of power factor p takes priority vs. the others. This makes sense as the local models' training areas overlap each other heavily, and the power factor must be carefully tuned to fuse them correctly.

Table 5.2: Results from different algorithms.

	MAE
Linear regression	0.308
Neural Network	0.317
Random Forests	0.187
MGWR	0.186
RFsp	0.191
IDW-RF	0.174

As a comparison, we run several other algorithms on this dataset and list the results in Table 5.2. Unsurprisingly, the non-geographic algorithms (Linear Regression and Neural Network) perform poorly, as they are not capable of handling non-stationary geographic data. We also tested two state-of-the-art geographic machine learning algorithms RFsp [HNW$^+$18] and MGWR [FYK17], using their R and Python implementations. Both of them adopt the multiple-local-model method and use RF as the underlying local model. Results show that their performance will degrade to that similar to the original RF for such an unevenly distributed dataset.

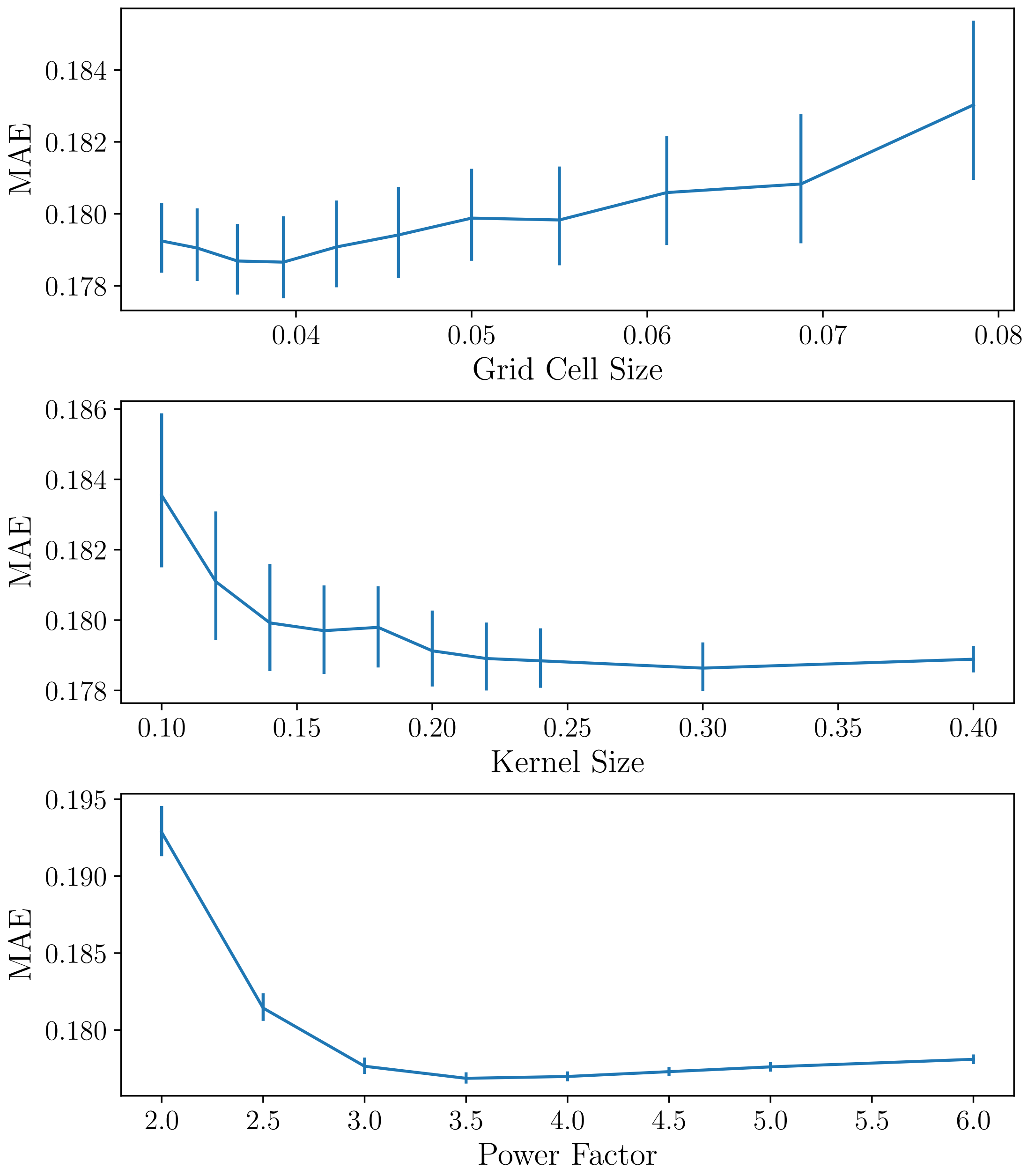


Figure 5.5: Calibrate Parameters with Grid Search.

CHAPTER 6

LVRF: A LATENT VARIABLE BASED APPROACH FOR EXPLORING GEOGRAPHIC DATASETS

Geographic datasets are usually accompanied with spatial non-stationarity – a phenomenon that the relationship between features vary across space. Naturally, non-stationarity can be interpreted as the underlying rule that decides how data are generated alters over space. Therefore, traditional machine learning algorithms are not suitable for handling non-stationary geographic datasets as they only render a single global model. To solve this problem, researchers usually adopt the multiple-local-model approach, which uses different models to account for different sub-regions of space. This approach is proven to be efficient, but not optimal because it's inherently difficult to decide the size of sub-regions. Also, the fact that local models are only trained from a subset of data will also limit its potential. This paper proposes a entirely different strategy that interpret non-stationarity as lack of data, and address it by introducing latent variables to the original dataset. Back-propagation is then used to find the best values of these latent variables. Experiments show that this method is at least as efficient as multiple-local-model based approaches and has a greater potential.

6.1 Introduction

Geographic data is defined as information that implicitly or explicitly associated with a location on the surface of earth [ISO11]. With the advancement in remote sensing technologies and wide-spread use of GPS-enabled devices, the number of available Physical and Human geography datasets has greatly increased during the last few years [BBFRS12]. These data are studied and used for social good such as mitigating the damage brought by natural disasters [SA07], discovering mineral resource [PM12], preventing crimes [WS20], improving traffics [MBS$^+$17], etc.

However, when dealing with geographic datasets, researchers find that many traditional machine learning algorithms do not work very well due to the existence of non-stationarity. For such data, the relationship between features does not necessarily remain the same everywhere, which means the underlying model that decides the data would alter over space. To solve this problem, a natural solution is to replace the global model with many local models. Each of the local model is only responsible for describing a much smaller region within which the data is supposed to be relatively stationary. Most of studies took this approach (like [BFC96], [GGG$^+$19] and [FYK17]) and saw much better results, comparing with the traditional algorithms which are not specifically designed to handle non-stationarity.

These multiple-local-model based approaches all suffer to similar challenges. For one, the dataset used to train local models is only a subset of all available data. And previous researches had shown that the accuracy of a model is strongly correlated with the amount of data used to train this model. There would be a significant performance decrease of the model if training data size drop below a certain degree [MDHC03]. For the other, the size of sub-regions within which local models corresponds to are hard to decide. A large size means more data could be used to train local models, but the region is more likely to be non-stationary. And a small size mean the opposite. Thus a compromise is always needed.

Our insight is that the source of non-stationarity can be explained as lack of data, i.e., some dimensions of the data not being collected. For example, a crime dataset could present a strong non-stationarity in it, as crime patterns in New York could be very different from Washington DC. Even within New York, it's hard to imagine Brooklyn shares the same crime pattern with Manhattan. But eventually, this difference is caused by various factors like household income, population composition, culture, number of police officers per capita, etc. If one is able to collect data on every single aspect of the area, the dataset would ultimately be stationary. This theory is also in accordance with the fact that non-stationarity is quite often observed in human geography datasets, but rarely found in physical geography data. Since physical geography data – which is generated by Earth's natural processes – have less determine factors and are usually simpler to collect. Whereas human geography data is all about activities of human thus much more complicated. Even some of the seemingly simplest datasets would have countless deciding factors that are impossible to collect. For example, house sale price data generally includes all features of the house itself, and nearby areas. But other factors – such as school, traffic, population, crime – are usually not included, although they're also important and certainly would affect the pricing model. The lack of these data would then be observed as non-stationarity in the data and affect the final model in a certain way.

Based on this insight, we propose a entirely different strategy that address non-stationarity by introducing latent variables to the original dataset. This latent variable would account for all the missing factors that not collected by the original dataset, but observable as non-stationarity. Theoretically, assuming we have unlimited calculating power, the optimal value of the latent variables can be easily found by a brute-force search of the entire vector space. But this solution is obviously impossible due to the tremendous size of the vector space. Thus, inspired by the neural network, we use a back-propagation algorithm to find the optimal values of the latent variable. Experiments show that this new approach can build models as accurate as the state-of-the-art algorithm, whereas having the potential of being further improved.

6.2 Background

The first renowned method for exploring spatial non-stationarity, known as Geographically Weighted Regression (GWR), was proposed by Brunsdon, Fotheringham, and Charlton in 1996 [BFC96]. The "main characteristic of GWR is that it allows regression coefficients to vary across space, and so the values of the parameters can vary between locations" [Mat10]. The motivation for inventing GWR was that "a single global model cannot explain the relationship between some sets of variables" [BFC96]. To address non-stationarity, GWR allows relationships between features and labels to differ across spaces. The basic idea of how GWR works is to learn a regression equation for every feature in the dataset, during which dependent and explanatory components are accounted for by examine neighboring data points. And the neighbors contribute differently to this process according to how far away it is, which is why it is called a "weighted" regression. The closer a data point is, the more weight it is assigned. This design complies with Tobler's first law of geography, "everything is related to everything else, but near things are more related than distant things" [Tob70]. Later in 2002, Brunsdon further improved this algorithm to Semiparametric GWR (SGWR) [FBC02] and allow some features to have fixed regression equations across the space, whereas others can still be variable.

Due to the success of GWR, many later studies followed this multiple-local-model design. One example is Multiscale GWR (MGWR), which was introduced in 2017 by Fotheringham, Yang and Kang. This method "is similar in intent to Bayesian non-separable spatially varying coefficients (SVC) models, although potentially providing a more flexible and scalable framework in which to examine multiscale processes" [FYK17]. It improves GWR in a way that it not only adapts to datasets on different levels of non-stationarity, but also provides necessary information to evaluate the scales of different processes. The latest research using this approach is Geographical Random Forest (GRF), which was proposed by Stefanos, Tais, et al. in 2019. It adopts Random Forests [Bre01] as the base algorithm to create local models. The principle idea of this method is to “disaggregate of RF into geographical space in the form of local sub-models” [GGG$^+$19], which is basically another version of the multiple-local-model approach.

As a conclusion, all these methods are directly or indirectly based on the multiple-local-model approach thus suffer from the same problems mentioned in the previous section. Here we propose a completely different approach with the goal that the intrinsic nature of non-stationarity can be better understood and accounted for.

6.3 Study Area

We choose the housing sales data from King County, US as the target study area (obtained from [Kag16]). There are 21,613 records in the dataset, with each record being a real estate transaction that happened between May 2014 and May 2015, during which the housing market stayed relatively stable in the King County.

In this dataset, there are 20 features regarding the house’s location (latitude, longitude, zip code), its basic information (size, number of stories and rooms, garage, air conditioning), and its transaction related information (sale date and price). Some of the features have missing values. This is not a problem for our algorithm, which is based on the Random Forests algorithm thus be able to handle missing values. However some other algorithms we use to compare the performance with are not capable of doing this. Thus during the data preparation stage, we fill the missing values with average value of that column.

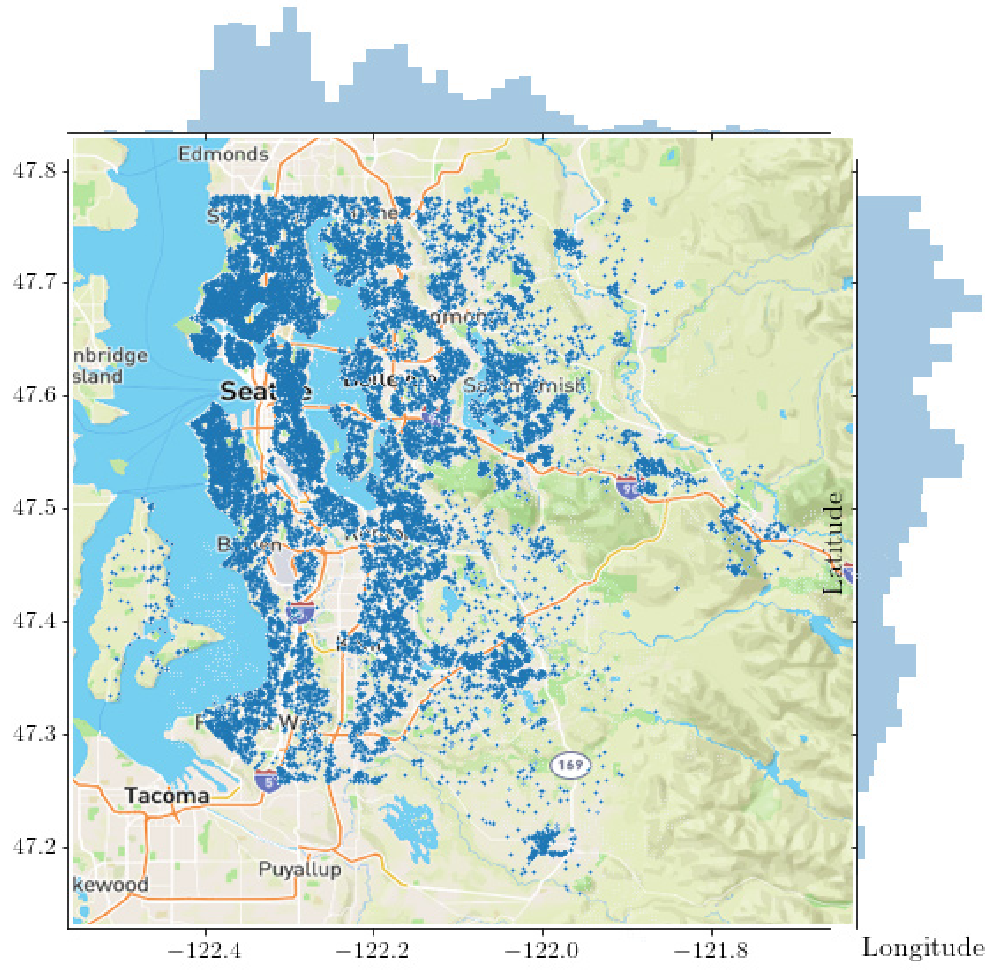


Figure 6.1: Distribution of the King County housing data.

The goal of this dataset is to build a predictive model that is able to estimate house sale price, given the house's location and some of its basic information. This is actually a well researched topic which had been studied for a long time. But even the state-of-the-art algorithm in this area still has a lot of room to improve due to the complicated nature of this task. It is also a very typical human geography dataset within which data availability varies depending on the amount of human activity. Figure 6.1 shows the distribution of the dataset over the map. As can be see from the figure, the downtown area in Seattle is populated with data, with some areas left blank which are mostly parks or commercial zones. The rural regions has much less data scattered all over the place. The fact that this dataset is extremely unevenly distributed over space brings additional challenges if the previously mentioned multiple-local-model was to be used, as local models which correspond with rural areas will see much fewer training samples, thus generating inaccurate results. Whereas in urban areas, over crowded data points will only bring marginal improvement to models built for that area.

One more problem for this dataset is that the house sale price spans over a fairly large range with a long tail, as shown in figure 6.2 , which is not desirable. To eliminate the tail, we convert Price to log(Price), which follows the normal distribution thus a much better target variable to deal with.

6.4 Latent Variable Random Forests

In this section, we detail the key designs of the Latent Variable Random Forests as below.

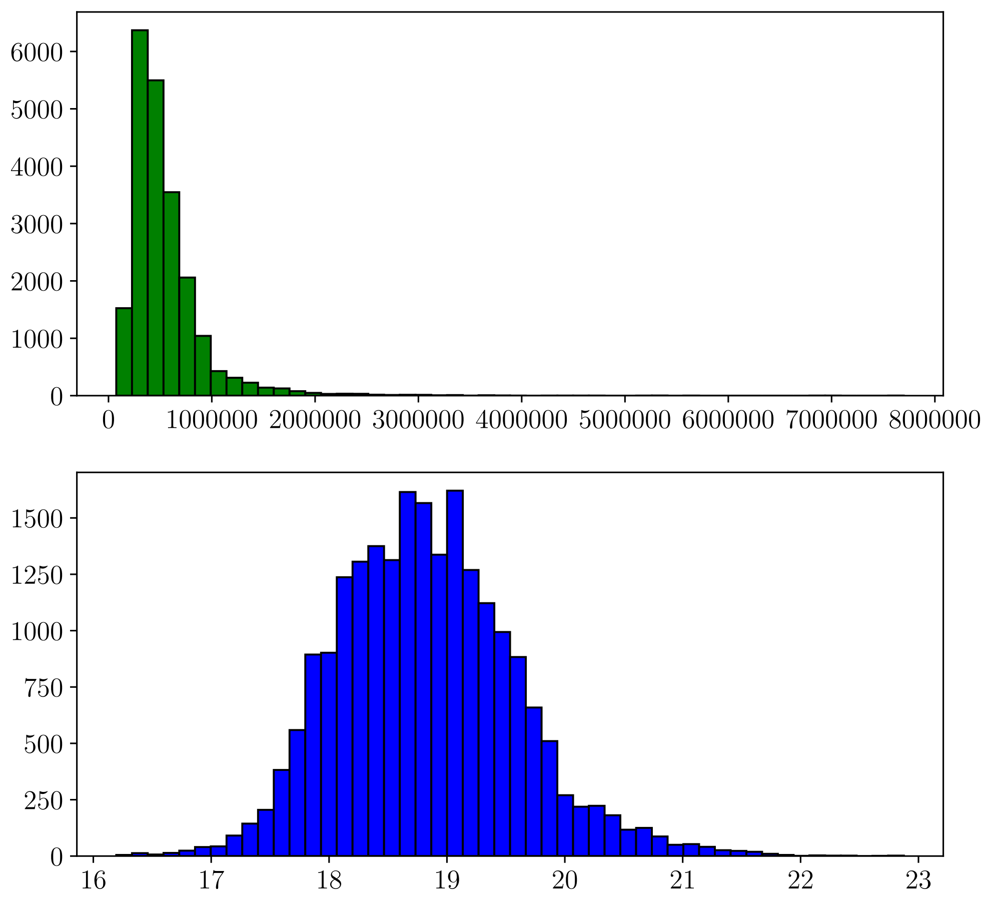


Figure 6.2: Distribution of Price vs. log(Price)

6.4.1 Key Design of the Latent Variable

By introducing a new latent variable, we hope to use it to represent the hidden factor that cause non-stationarity. In our housing price model example, it would be the various unknown factors combined that could affect how house price should be modeled. For instance, how secure a community is will obviously have an impact on the house value. Although we don't have any information on which area is secure and which is not, its influence on the sale price will be observable via non-stationarity. Note that the target variable might be affected by multiple hidden factors like security, traffic, nearby schools and so on. But no matter how many hidden factors there are, they will influence the target variable together. It is impossible to know which factor has a larger impact. Fortunately we don't need to care about that. Our only concern is how these hidden factors as a whole would affect the target variable we want to predict.

To better describe the problem, let $(f_1, f_2, ..., f_n)$ denote the features in the dataset and t denote the target variable to be modeled and predicted. After adding a latent variable lv, the feature vector becomes $F(lv) = (f_1, f_2, ..., f_n, lv)$. So, the task is converted to finding the best $\vec{lv}$ that makes the model trained from $F(lv)$ (with a predetermined regular machine learning algorithm) has the best accuracy.

The vector space $\vec{lv}$ is obviously unlimited. Thus we introduce a value range of $[0, 1]$ to lv and define a minimum step interval of 0.01. The reason why we limit the value range to $[0, 1]$ is that the value range of lv actually doesn't play any important role in the final model. If lv is multiplied by 2, the resulting model will still be the same. So, only the relative value matters and is what we should care about. Also, during the machine learning stage, all the features of original dataset need to be standardized and normalized anyway, thus a standardized lv will in fact benefit the entire procedure. For the minimum step, the smaller it is the more fine-grained the final model would be. But setting it too small will also considerably increase the calculation time and may not worth the marginal return. So we recommend setting it to 0.01 as a balance between speed and accuracy.

Theoretically, the value array of latent variable $\vec{lv}$ can be inferred by an exhaustive brute-force search of the entire vector space. The time complexity of doing so is as follows:

$$O(n) = (\frac{R}{S})^n * (T_{train} + T_{test}) \tag{6.1}$$

where n is the number of data points in the dataset, R is the value range, S is the step size, T_{train} and T_{test} are time needed for training and testing the model respectively. Note that the value of n is usually very large. Even for a very small dataset n will probably be greater than 1000. Thus, this brute-force method is completely impractical considering the amount of calculation is needed.

6.4.2 Grid Based Latent Variable System

To solve the time complexity problem, we clearly need a smarter algorithm, for example, a heuristic search which could greatly reduce the search space. But before that, let's examine the possibility of reducing the size of potential vector space, which would greatly benefit the entire procedure even if a heuristic search is to be adopted.

Here we introduce a grid based latent variable system. Let $(x_{min}, x_{max}, y_{min}, y_{max})$ denote the minimum bounding box that contains the entire dataset. A step size of s will evenly divide the space to this many grids:

$$G(s) = \lceil \frac{x_{max} - x_{min}}{s} \rceil * \lceil \frac{y_{max} - y_{min}}{s} \rceil \tag{6.2}$$

For each intersection of the grid system, we assign an *Influence Center* (abbreviated as IC) to it. For a data point with a coordinate of (x, y), we first determine which *grid* it is located in. Then calculate its latent variable value from all the nearby ICs located at the four corners of *grid*. Here we use an inverse distance weighted method to combine the values from nearby ICs, in accordance with the idea that nearby ICs should have stronger influence on the latent variable than remote ones. The detailed formula is as follows:

$$v(x, y) = \frac{\sum_{i=1}^{N} W(IC_i)V(IC_i)}{\sum_{i=1}^{N} W(IC_i)} \tag{6.3}$$

where $W(IC_i)$ is the weight for the ith influence center which equals to the inverse of the euclidean distance between the data point and the IC.

This design simulates how the hidden factors create non-stationarity in the dataset. No matter what hidden factors there are, as a general rule, it would affect nearby data points more than remote ones. Thus we simulate this procedure by introducing the concept of Influence Centers and make them impact nearby records in a similar way. Another benefit brought by this design is that now the search space is greatly reduced down to the number of ICs. Instead of finding the best values for all the records, we only need to optimize the values for ICs now, which is way less then the total number of records.

6.4.3 Random Forests as the Base Algorithm

Before proceeding, we still need to decide which base machine learning algorithm is to be used to train models. Here, our choice is the Random Forests [Bre01] algorithm. As suggested in the name, Random Forests will create many randomly generated decision trees to perform the prediction task together. For classification tasks, The final result would be a majority vote of results from all the decision trees. For regression, this would be an average of all results. The core idea of RF is to create a bagging procedure where the variance of the model is decreased but the bias remains unchanged, thus will generate a better result from sub-optimal models.

There are multiple reasons why we choose RF as our base algorithm. First, RF is based on decision tree which is naturally good at handling coordinates in geographic datasets. Then, Random Forests is among the top machine learning algorithms available and often shows exceedingly good results when handling spatial data, as proven by [BJKK12] and [NSB+18]. We will be able to inherit all of these advantages by using RF as the base algorithm.

6.4.4 Back Propagation

With a reduced search space, the time complexity is till massive as we are only replacing $(\frac{R}{S})^n$ in Formula 6.1 with X^n (X is the total number of influence centers) if a brute-force search is to be used. Thus we must find a way to further reduce the search space, i.e., a heuristic-search like method.

Here, inspired by the back propagation in the Neural Network algorithm [PW17], we design a back propagation process to search for the best values for influence centers, as detailed in algorithm $BackPropagation()$. In this function, a learning rate α is introduced. It decides how fast the back propagation converges. A large value will cause $BackPropagation()$ to converge faster, but the generated result will be more likely to be coarse-grained thus less than optimal. Conversely, a smaller value will converge slower but produce better results. Generally speaking, the best α value is recommended to set to the smallest value within acceptable training time.

The *converge* condition in the *BackPropagation()* algorithm is a little tricky. Ideally, if *IC_Array* remains the same after an iteration, one can definitely say the algorithm is converged as future iterations will keep generating the same results. But this could not necessarily happen as *IC_Array* could always change slightly with pretty much the same results. So, we insert a process at the end of each iteration, which will evaluate the test accuracy under the current *IC_Array.* If the test accuracy remains not improved for more than 5 iterations, we consider

```
Function BackPropagation()
    Initialize IC_Array
    while IC_Array has not converged do
        foreach IC in IC_Array do
            foreach learn_rate in [$\alpha$, -$\alpha$] do
                IC_new = IC + learn_rate
                if Trained model sees improvement in accuracy then
                    IC = IC_new
                else
                    continue
                end
            end
        end
    end
    return IC_Array
end
```

the algorithm as converged and stop the back propagation iteration. This extra calculation will slow down the entire algorithm but it's worth the cost.

6.4.5 Prediction

The prediction process is relatively simple. After *IC_Array* is returned by *Back-Propagation()*, the final *Model* will be trained from the orignal dataset plus the latent vector generated from *IC_Array*. When predicting an unknown observation, first calculate its latent variable by using inverse distance weighted method from Formula 6.3. Then, apply *Mode* to get the final prediction result.

6.4.6 Assessment Measurements and Results

One thing that wasn't mentioned in the previous sections is that a proper assessment measurement must be chosen. This actually plays an important role in the algorithm, as the evaluation result generated by the measurement will be used to determine how the back propagation process runs and guide it to generate a better result for each iteration.Some of the most commonly used measurements are [HOFL+15]: mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). In our case, MAE is preferred as the other ones will penalize large errors and cause bias in our algorithm.

Now that the algorithm complete, we run LVRF on the King County housing dataset and get an MAE of 0.263. As a comparison, we run unmodified Random Forests on the same dataset and get a result of 0.289. This means that the learned latent variables offset some of the non-stationarity and make it easier for the standard RF to generate a more accurate model. To compare with the others, we also run the same dataset on two state-of-the-art algorithms RFsp [HNW+18] and MGWR [FYK17], who are specifically designed to handle geographic datasets and non-stationarity. The results are 0.261 and 0.272 respectively. This means the idea of using latent variables to capture hidden factors that cause non-stationarity works at least as well as the best results one could get using the multiple-local-model approach.

6.5 Conclusion

This paper presents LVRF, a machine learning algorithm that is capable of creating predictive models for non-stationary geographic datasets. Unlike other algorithms, LVRF adopts a latent variable based approach, instead of the widely used multiple-local-model strategy. Experiments show that LVRF can build models as accurate as the state-of-the-art algorithms, whereas avoiding the common disadvantages of the multiple-local-model approach. LVRF first establish a grid-based influence centers. The latent variable value of any data point is decided by the nearby influence centers using an inverse distance weighted method. It then use a back propagation algorithm to train the values of the influence centers. The training process will finish after the value of influence centers converge. When used to predict unknown observations, the data point's latent variable will be calculated from the converged influence centers, and fed into the model with its other features.

The insight of LVRF is that the design of influence center can mimic the hidden factors which affect nearby data points in different ways depending the location. By learning these hidden factors with a back propagation algorithm and then include them in the model creation stage, the impact brought by non-stationarity will be offset and a single global model can be used to describe the features plus the hidden factors.

It is also worth mentioning that, although Random Forests is selected as the base algorithm, LVRF is capable of using any other regular machine learning algorithm as the base algorithm. Doing so may bring advantages in certain scenarios when there is preknowledge regarding the dataset.

BIBLIOGRAPHY

[AKK01] Nobuhiro Asai, Izumi Kubo, and Hui-Hsiung Kuo. Bell numbers, log-concavity, and log-convexity. *Acta Applicandae Mathematica*, 63, 05 2001.

[All74] David M. Allen. The relationship between variable selection and data agumentation and a method for prediction. *Technometrics*, 16(1):125–127, 1974.

[AMlBI17] Khawaja Asim, Francisco Martínez-Álvarez, Abdul Basit, and Talat Iqbal. Earthquake magnitude prediction in hindukush region using machine learning techniques. *Natural Hazards*, 85:471–486, 01 2017.

[Ans95] Luc Anselin. Local indicators of spatial association—lisa. *Geographical Analysis*, 27(2):93–115, 1995.

[BBFRS12] C. Beath, I. Becerra-Fernandez, J. Ross, and J. Short. Finding value in the information explosion. *MIT Sloan Management Review*, 53:18–20, 06 2012.

[BFC96] Chris Brunsdon, A. Stewart Fotheringham, and Martin E. Charlton. Geographically weighted regression: A method for exploring spatial nonstationarity. *Geographical Analysis*, 28(4):281–298, 1996.

[BJKK12] Anne-Laure Boulesteix, Silke Janitza, Jochen Kruppa, and Inke R. König. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *WIREs Data Mining and Knowledge Discovery*, 2(6):493–507, 2012.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, SIGMOD '90, page 322–331, New York, NY, USA, 1990. Association for Computing Machinery.

[Bre01] Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[Cre93] Noel Cressie. *Statistics For Spatial Data*, volume 35. 06 1993.

[DARssa] Liangdong Deng, Malek Adjouadi, and Naphtali Rishe. Geographic boosting tree: Modeling non-stationary spatial data. (in press).

[DARssb] Liangdong Deng, Malek Adjouadi, and Naphtali Rishe. Inverse distance weighted random forests: Modeling unevenly distributed nonstationary geographic data. (in press).

[FB74] Raphael Finkel and Jon Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 03 1974.

[FBC02] Alexander Fotheringham, Chris Brunsdon, and Martin Charlton. Geographically weighted regression: The analysis of spatially varying relationships. *John Wiley and Sons*, 13, 01 2002.

[FFWF10] Eloi Figueiredo, Charles Farrar, Keith Worden, and Joaquim Figueiras. Machine learning algorithms for damage detection under operational and environmental variability. *Structural Health Monitoring*, 10, 03 2010.

[FYK17] A. Stewart Fotheringham, Wenbai Yang, and Wei Kang. Multiscale geographically weighted regression (mgwr). *Annals of the American Association of Geographers*, 107(6):1247–1265, 2017.

[Gü17] Ralf Güting. *Moving Objects Databases and Tracking*, pages 1–8. 01 2017.

[GA89] Daniel Griffith and L. Anselin. Spatial econometrics: Methods and models. *Economic Geography*, 65:160, 04 1989.

[GGG+19] Stefanos Georganos, Tais Grippa, Assane Niang Gadiaga, Catherine Linard, Moritz Lennert, Sabine Vanhuysse, Nicholus Mboga, Eléonore Wolff, and Stamatis Kalogirou. Geographical random forests: a spatial extension of the random forest algorithm to address spatial heterogeneity in remote sensing and population modelling. *Geocarto International*, 0(0):1–16, 2019.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, page 47–57, New York, NY, USA, 1984. Association for Computing Machinery.

[HNW+18] Tomislav Hengl, Madlene Nussbaum, Marvin N. Wright, Gerard B.M. Heuvelink, and Benedikt Gräler. Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables. *PeerJ*, 6:e5518, August 2018.

[HOFL$^+$15] Jose Hernandez-Orallo, Cèsar Ferri, Nicolas Lachiche, Adolfo Martínez-Usó, and M. Ramírez-Quintana. Binarised regression tasks: methods and evaluation metrics. *Data Mining and Knowledge Discovery*, 30, 11 2015.

[Hol04] H. Holgersson. Testing for multivariate autocorrelation. *Journal of Applied Statistics*, 31:379–395, 05 2004.

[ISO11] ISO/TC 211 committee. ISO/TC 211. `https://www.iso.org/committee/54904.html`, 2011.

[JMF99] A.K. Jain, M. Murty, and P.J. Flynn. Data clustering: A review. *ACM Comput Surv*, 31:264–323, 01 1999.

[JSZ$^+$14] Zhe Jiang, Shashi Shekhar, Xun Zhou, Joseph Knight, and Jennifer Corcoran. Focal-test-based spatial decision tree learning. *IEEE Transactions on Knowledge and Data Engineering*, 27:1–1, 01 2014.

[Kag16] Kaggle. King County Housing Market data. `https://www.kaggle.com/harlfoxem/housesalesprediction`, 2016.

[Kal16] Stamatis Kalogirou. Destination choice of athenians: An application of geographically weighted versions of standard and zero inflated poisson spatial interaction models. *Geographical Analysis*, 48(2):191–230, 2016.

[KBL95] Karen Kafadar, Vic Barnett, and Toby Lewis. Outliers in statistical data. *Journal of the American Statistical Association*, 90:395, 03 1995.

[KF99] I. Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. *Proc. Twentieth Int. Conf. Very Large Databases*, 10 1999.

[KHS99] Krzysztof Koperski, Jiawei Han, and Nebojsa Stefanovic. An efficient two-step method for classification of spatial data. 02 1999.

[Kri52] D.G. Krige. A statistical analysis of some of the borehole values in the orange free state goldfield. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 53:47–64, 01 1952.

[LH11] Jin Li and Andrew D. Heap. A review of comparative studies of spatial interpolation methods in environmental sciences: Performance and impact factors. *Ecological Informatics*, 6(3):228 – 241, 2011.

[Lie11] David Lieske. A robust test of spatial predictive models: Geographic cross-validation. *Journal of Environmental Informatics*, 17:91–101, 06 2011.

[Lou14] Gilles Louppe. Understanding random forests: From theory to practice, 2014.

[Lov93] László Lovász. *Combinatorial problems and exercises.* Amsterdam: North-Holland, 2. ed. edition, 1993.

[LW01] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *Forest*, 23, 11 2001.

[Mas99] Doreen Massey. Space-time, 'science' and the relationship between physical geography and human geography. *Transactions of the Institute of British Geographers*, 24(3):261–276, 1999.

[Mat10] Jorge Mateu. Comments on: A general science-based framework for dynamical spatio-temporal models. *Test*, 19:452–455, 11 2010.

[MBS$^+$17] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. Profiliot: A machine learning approach for iot device identification based on network traffic analysis. In *Proceedings of the Symposium on Applied Computing*, SAC '17, page 506–509, New York, NY, USA, 2017. Association for Computing Machinery.

[MDHC03] James Morgan, Robert Dougherty, Allan Hilchie, and Bern Carey. Sample size and modeling accuracy with decision tree based data mining tools. *Acad Inf Manag Sci J*, 6, 01 2003.

[Mic18] Microsoft. Bing maps tile system. `https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system`, 2018.

[Mor50] P. A. P. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[NSB$^+$18] M. Nussbaum, K. Spiess, A. Baltensweiler, U. Grob, A. Keller, L. Greiner, M. E. Schaepman, and A. Papritz. Evaluation of digital soil mapping approaches with large sets of environmental covariates. *SOIL*, 4(1):1–22, 2018.

[OBK$^+$10] D. Oliver, A. Bannur, J. M. Kang, S. Shekhar, and R. Bousselaire. A k-main routes approach to spatial network activity summarization: A summary of results. In *2010 IEEE International Conference on Data Mining Workshops*, pages 265–272, 2010.

[Pin18] Tony Pino. Melbourne housing market data. `https://www.kaggle.com/anthonypino/melbourne-housing-market`, 2018.

[PM12] Panagiotis Partsinevelos and Zinovia Mitraka. Change detection of surface mining activity and reclamation based on a machine learning approach of multi-temporal landsat tm imagery. *Geocarto International*, 28:1–20, 01 2012.

[PW16] Hong-Gang Peng and Jianqiang Wang. A linguistic intuitionistic multicriteria decision-making method based on the frank heronian mean operator and its application in evaluating coal mine safety. *International Journal of Machine Learning and Cybernetics*, 12 2016.

[PW17] Sandy Putra and Anjar Wanto. Analysis of artificial neural network accuracy using backpropagation algorithm in predicting process (forecasting). *International Journal Of Information System & Technology (IJISTECH)*, 1:34–42, 11 2017.

[SA07] Evaggelos Spyrou and Yannis Avrithis. A region thesaurus approach for high-level concept detection in the natural disaster domain. volume 4816, pages 74–77, 12 2007.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, June 1984.

[SG16] Kristin Stock and Hans Guesgen. Chapter 10 - geospatial reasoning with open data. In Robert Layton and Paul A. Watters, editors, *Automating Open Source Intelligence*, pages 171 – 204. Syngress, Boston, 2016.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM Na-*

tional Conference, ACM '68, page 517–524, New York, NY, USA, 1968. Association for Computing Machinery.

[SSV$^+$02] S. Shekhar, P. R. Schrater, R. R. Vatsavai, Weili Wu, and S. Chawla. Spatial contextual classification and prediction models for mining geospatial data. *IEEE Transactions on Multimedia*, 4(2):174–188, 2002.

[SY18] Xingjian Shi and Dit-Yan Yeung. Machine learning for spatiotemporal sequence forecasting: A survey. *ArXiv*, abs/1808.06865, 2018.

[SYA11] Imas Sitanggang, Razali Yaakob, and Nuruddin A. An extended id3 decision tree algorithm for spatial data. pages 48–53, 06 2011.

[SZHV03] Shashi Shekhar, Pusheng Zhang, Yan Huang, and Ranga Vatsavai. Trends in spatial data mining. 10 2003.

[SZK14] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Local outlier detection reconsidered: A generalized view on locality with applications to spatial, video, and network outlier detection. *Data Mining and Knowledge Discovery*, 28, 01 2014.

[Tob70] W. R. Tobler. A computer movie simulating urban growth in the detroit region. *Economic Geography*, 46:234–240, 1970.

[Wik20] Wikipedia. Partition of a set. `https://en.wikipedia.org/wiki/Partition_of_a_set`, 2020.

[Wil20] Robin Wilson. Individual country/area datasets. `https://freegisdata.rtwilson.com/`, 2020.

[WS20] Andrew Wheeler and Wouter Steenbeek. Mapping the risk terrain for crime using machine learning, 01 2020.

[ZGHW07] Feng Zhou, Huai-Cheng Guo, Yuh-Shan Ho, and Chao-Zhong Wu. Scientometric analysis of geostatistics using multivariate methods. *Scientometrics*, 73:265–279, 12 2007.

[Zil17] Zillow. Zillow's home value prediction competition. `https://www.kaggle.com/c/zillow-prize-1/overview`, 2017.

VITA

LIANGDONG DENG

2014-Present	Ph.D., Computer Science Florida International Univerisity Miami, Florida
2013	M.S., Software Engineering Beihang University Beijing, China
2010	B.S., Software Engineering Beihang University Beijing, China

PUBLICATIONS AND PRESENTATIONS

Liangdong Deng, Yuzhou Feng, Dong Chen, and Naphtali Rishe. *Iotspot: Identifying the iot devices using their anonymous network traffic data.* In MILCOM 2019 – 2019 IEEE Military Communications Conference (MILCOM), pages 1-6, 2019.

Liangdong Deng, Malek Adjouadi, and Naphtali Rishe. *Inverse Distance Weighted Random Forests: Modeling Unevenly Distributed Non-Stationary Geographic Data.* ICACSIS 2020 (in press).

Liangdong Deng, Malek Adjouadi, and Naphtali Rishe. *Geographic Boosting Tree: Modeling Non-Stationary Spatial Data.* ICMLA 2020 (in press).

Liangdong Deng, Malek Adjouadi, and Naphtali Rishe. *LVRF: A Latent Variable Based Approach for Exploring Geographic Datasets.* (submitted to SDM21).

Yuzhou Feng, Liangdong Deng, and Dong Chen. 2019. *IoT devices discovery and identification using network traffic data: poster.* In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19), pages 338–339, 2019.