12-2016

Divide and recombined for large complex data: Nonparametric-regression modelling of spatial and seasonal-temporal time series

Xiaosu Tong
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Xiaosu Tong

Entitled

DIVIDE AND RECOMBINED FOR LARGE COMPLEX DATA: NONPARAMETRIC-REGRESSION MODELLING OF SPATIAL AND SEASONAL-TEMPORAL TIME SERIES

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

William S. Cleveland Chair	Boyu Zhang
Ryan Hafen	
Hao Zhang	
Anindya Bhadra	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): William S. Cleveland

Approved by: Jun Xie 8/10/2016

Head of the Departmental Graduate Program Date

DIVIDE AND RECOMBINED FOR LARGE COMPLEX DATA: NONPARAMETRIC-REGRESSION MODELLING OF SPATIAL AND SEASONAL-TEMPORAL TIME SERIES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Xiaosu Tong

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

To my wife and my parents. I couldn't have done this without you.

ACKNOWLEDGMENTS

No one denies that finishing a PhD degree is a long and difficult journey. And I could not have done it without the help and support of a lot of people.

First of all I would like to express my deeply appreciation and thank to my advisor Dr. William S. Cleveland. I really treasured this rare opportunity to work with such an outstanding professor. Dr. Cleveland guided me with his unique thinking, needs for perfect in details of the visualization, and his enthusiasm to research, which indeed profoundly influenced me and will keep influencing and benefiting me in my future career.

At the same time, I really like to thank the other committee members, Dr.Ryan Hafen, Dr. Hao Zhang, Dr. Anindya Bhadra, and Dr. Boyu Zhang. Thank you all for your valuable suggestion and unselfish support and help to me during this journey.

Also I would like to give a special thank you to Dr.Mark Daniel Ward, Dr.Sergey Kirshner, Douglas G. Crabill, and Dr.Rebecca Doerge. Dr.Ward I really do not how to express my thank to you for helping me preparing the Qualify Exams and my wedding. That morning in the kitchen of your house discussing probability questions with you was a very special and unforgettable moment in my life. You are an wonderful professor. Dr.Sergey Kirshner thank you for leading me to the world of Computational Statistics in the class of STAT598G. Those sleepless night doing project really profoundly stimulated my interests to be a PhD student in the area of computational statistics. Doug, I can barely count how many times I have been sitting in your office. Thank you for tutoring me with all basic knowledge about Hadoop and cluster. And thank you for your time and patience to help me with my projects. Sincere thanks to Dr.Rebecca Doerge for all the support and encouragement during this six years. You taught me a lot of thing which made me to be a better man.

I want to say thank you to all my former and present colleagues, Jianfu, Xiang, Jiasen, Ashrith, Philip, Aritra, Yuying, Qi, Barret, Jeremy. Jianfu, thank you very much for your patience when you unreservedly passed on to me your understanding about Hadoop and RHIPE, I cannot be expert on this without your teaching. Barret thank you for all your technical support and unselfish help to me whenever I said "May I ask you a question" to you. And I miss the time playing pingpong with Philip at Purdue and at Amazon as well, I lost a lot of games though.

I also want to thank all my dear friends at Purdue. Because of you, this journey becomes shorter according to the theory of relativity. Cheng Liu, thank you for being my mentor since the first day I came to Purdue; Cheng and Ximing, thank you for all the fun and hangover we have had as roommates in the Lodge; Qiming, thank you for your patience and time to help me preparing the talk I gave in the Amazon, and all the happy hours we shared during our internship over there in that summer, and I am looking forward to our new life in Seattle; Pan, thank you for sharing your knowledge and experience when we were doing project about Scala and Spark; Yang, Longjie, Xiaoguang, Hanli, Zhuo, Shuai, Faye, April, Kelly-Ann, thank you for being such great classmates during all the classes we had before; this list keeps going on, and I thank you all for the happiness you bring to me.

I am very pleased to have an opportunity to spend the last six years in the Department of Statistics at Purdue. The department provides an excellent environment and various opportunities. I have learned so much from many faculty and stuff members, as well as our excellent graduate students.

Finally, I want to thank my parents, and my beloved wife, Fang, for all their patience, encouragement, and unconditional love, thank you so much.

TABLE OF CONTENTS

Page

LIST OF FIGURES

SYMBOLS

$\|\cdot\|$	L^2 norm
d	smoothing degree
α	smoothing parameter (span)
$B(\cdot)$	bisquare weight function
y_i	response variable
x_i	design points
β	coefficient vector of linear model
$\hat{\beta}$	estimate coefficient vector
$f(x)$	mean function at point x
$\omega_i(x)$	weight of ith observation when fitting at x
$K(\cdot)$	Kernel function
h	bandwidth of local smoothing
x_0	location at which local fitting are calculated
n_α	number of observations in a loess smoothing neighborhood
$T(\cdot)$	tricube weight function
$n_{(p)}$	STL parameter – number of observations per period
$d_\alpha(x_0)$	distance between x_0 and its n_αth nearest neighbor
X	design matrix
Y	the vector of response variable
W	weight matrix
ϵ_i	random error term
δ_i	robustness weights
W_{update}	updated weight matrix with robustness weights
t_i	trend component of STL model at time i
s_i	seasonal component of STL model at time i

r_i	remainder component of STL model at time i
t_i^k	estimated trend component of STL model at kth iteration at time i
s_i^k	estimated seasonal component of STL model at kth iteration at time i
C_i^k	estimated pre-seasonal component of STL model at kth iteration at time i
L_i^k	estimated low-frequency filtered series at kth iteration at time i
$\hat{r}_i$	final estimate of remainder component of STL model at time i
$\hat{s}_i$	final estimate of seasonal component of STL model at time i
$\hat{t}_i$	final estimate of trend component of STL model at time i
w_t	smoothing window for the trend component
w_s	smoothing window for the seasonal component
w_l	smoothing window for the low-pass smoothing
d_s	smoothing degree for the seasonal component
d_t	smoothing degree for the trend component
$n_{(I)}$	total iteration time for inner loop of STL
$n_{(O)}$	total iteration time for outer loop of STL
ϕ_a	latitude degree for location a
λ_a	longitude degree for location a
$\Delta\sigma$	central angle between to location
N_{map}	the total number of mappers in a given MapReduce job
N_{cont}	the maximum number of containers can be running simultaneously on a cluster

ABBREVIATIONS

COOP	Cooperative Observer Program
D&R	Divide-and-Recombine
GB	Gigabyte
GWR	Geographically Weighted Regression
HDFS	Hadoop Distributed File System
IBP	mapreduce.reduce.input.buffer.percent
IMAGe	Institute of Mathematics Applied to Geosciences
ISF	mapreduce.task.io.sort.factor
ISM	mapreduce.task.io.sort.mb
ITaP	Information Technology at Purdue
JVM	Java Virtual Machine
KB	Kilobyte
MAPE	Mean of Absolute value of Prediction Error
MB	Megabyte
MRCC	Midwestern Regional Climate Center
MSDPE	Mean of Standard Deviation of Prediction Error
NCDC	National Climatic Data Center
NCEI	National Centers for Environmental Information
NW	Nadaraya-Watson
NRCS	Natural Resources Conservation Service
NWS	National Weather Service
RHIPE	R and Hadoop Integrated Programming Environment
RTSK	mapreduce.job.reduces
RSC	mapreduce.job.reduce.slowstart.completedmaps
SDPE	Standard Deviation of Prediction Error

SSP	mapreduce.map.sort.spill.percent
SPC	mapreduce.reduce.shuffle.parallelcopies
SIBP	mapreduce.reduce.shuffle.input.buffer.percent
SMP	mapreduce.reduce.shuffle.merge.percent
SNOTEL	Snowpack telemetry
STL	Seasonal Trend Loess
USDA	United States Department of Agriculture
USHCN	United States Historical Climatology Network

ABSTRACT

Tong, Xiaosu PhD, Purdue University, December 2016. Divide and Recombined for Large Complex Data: Nonparametric-Regression Modelling of Spatial and Seasonal-Temporal Time Series. Major Professor: William S. Cleveland.

In the first chapter of this dissertation, I briefly introduce one type of nonparametric regression method, namely local polynomial regression, followed by emphasis on one specific application of loess on time series decomposition, called Seasonal Trend Loess (STL). The chapter is closed by the introduction of D&R (Divide and Recombined) statistical framework. Data can be divided into subsets, each of which is applied with a statistical analysis method. This is an embarrassing parallel procedure since there is no communication between each subset. Then the analysis result for each subset are combined together to be the final analysis outcome for the whole dataset. The main purpose of this chapter is to cover the foundation of the methodology and principle for the data analysis in later chapters, which crucially depends on these topics.

In the second chapter, a new statistical method for the analysis of spatial seasonal-temporal dataset is proposed, named as Spatial Seasonal Trend Loess (SSTL). The chapter starts with the illustration of main steps of analysis routine of SSTL. Next, a spatial-temporal dataset from National Climatic Data Center (NCDC) [1] is introduced and used as an example to explore the routine step by step. The modeling results are evaluated by diagnostic visual display among the third chapter.

In the third chapter, I illustrate procedure to choose the best smoothing parameters for spatial and temporal smoothing respectively. For the spatial smoothing, cross validation method is used to choose the best smoothing span and degree in all 576 months collectively. The training dataset and testing dataset are decided by using near exactly replicates framework which split the original dataset into subsets.

For the temporal STL+ fitting, an experiment of tunning smoothing parameters is explored based on data after year 1950 to choose the best smoothing parameters in terms of prediction ability.

In the fourth chapter, I discuss the generalization of SSTL routine under the divide and recombined framework for big and complex spatial-temporal dataset. Because of the flexibility of the SSTL for large and computationally heavy dataset, it comes free to embed the SSTL method into the divide and recombined framework, called drSSTL. Either by-month division or by-station division of the dataset can be generated, then a corresponding analysis method is applied on them. The drSSTL routine now consists of a series of MapReduce jobs, and all fitting results are saved on HDFS. Meanwhile some of the diagnostics procedures discussed in the third chapter are illustrated here but with more details about their parallel implementation. In the first section, I illustrate the MapReduce job to download the target dataset in parallel. Next I explore the details about the each steps of drSSTL as a MapReduce job to proceed the smoothing procedure on different divisions.

In the last chapter, I conduct the analysis of the performance of the drSSTL routine for large spatial-temporal dataset. There are two groups of tunning parameters have the potential influence on the performance of the routine. One is the tunning parameters of statistical model, which controls the complexity of the smoothing procedure of spatial loess and STL+ procedure. Another group of parameters are user-tunable Hadoop parameters. Within this chapter, I illustrate several pilot experiments and full factorial experiments to study the affect of these tunning parameters to the elapsed time of the drSSTL routine.

1. BACKGROUND

In this chapter, I briefly introduce one type of nonparametric regression method, namely local polynomial regression, followed by emphasis on one specific application of loess on time series decomposition, called Seasonal Trend Loess (STL). The chapter is closed by the introduction of D&R (Divide and Recombined) statistical framework. Data can be divided into subsets, each of which is applied with a statistical analysis method. This is an embarrassing parallel procedure since there is no communication between each subset. Then the analysis result for each subset are combined together to be the final analysis outcome for the whole dataset. The main purpose of this chapter is to cover the foundation of the methodology and principle for the data analysis in later chapters, which crucially depends on these topics.

1.1 Nonparametric Regression

In the family of regression analysis, the conditional expectation of a response variable (y) is assumed to be a function of one or several predictors (x's). Suppose the data contains n pairs of observations: $(x_1, y_1), \cdots, (x_n, y_n)$, the object of regression analysis is to model the relationship between y and x in the form of

$$y_i = f(x_i) + \epsilon_i \tag{1.1}$$

where f is an unknown function, ϵ_i is the random error term left in the observation which cannot be explained by the model. As it is usually proposed in parametric approach, regression analysis assumes a known form of function for $f(x_i)$, which is fully described by a finite set of parameters. For example, a linear function of predictor variables is commonly chosen:

$$f(x_i; \beta) = \beta_0 + \beta_1 x_i \tag{1.2}$$

where β_0 and β_1 are the parameters in the model to be estimated. The advantage of parametric regression method is that the relationship between response variable and predictor variable is described and summarized easily by the estimated parameters.

But in real data analysis, $f(x)$ is flexible and unknown, so we may have to relate y to x without assuming any pre-defined functional form. And this is when nonparametric model is preferred. Concretely, within nonparametric regression, the relationship between response variable and predictor does not have an explicit form but is constructed based on the information derived from the data.

1.1.1 Local Regression models

One approach in the nonparametric regression family to modeling an unknown and complex f is called local regression. Suppose we would like calculate the estimate at x_0. Instead of assuming $f(x)$ to have a parametric form overall, we assume that the data can be locally, within some neighborhood of x_0, well-approximated by a parametric form which may be polynomials or even a constant. Nadaraya [2] and Watson [3] proposed a local constant estimator, or kernel regression method, with a pre-defined kernel function as weighting function, called Nadaraya-Watson estimator:

$$\hat{f}(x_0) = \sum_{i=1}^{n} \omega_i(x_0) y_i \tag{1.3}$$

where

$$\omega_i(x_0) = \frac{K(\frac{x_i - x_0}{h})}{\sum_{i=1}^{n} K(\frac{x_i - x_0}{h})} \tag{1.4}$$

and $K(\cdot)$ is a non-negative real-valued integrable function, which satisfying the following two requirements

$$\int_{-\infty}^{\infty} K(\mu) d\mu = 1 \tag{1.5}$$

$$K(-\mu) = K(\mu) \tag{1.6}$$

The same idea can be illustrated when replacing a quantitative variable by indicator variables in linear regression. It uses a constant kernel function $K(\mu) = 1$ to assign equal weights to all neighbors within $x_0 \pm h$.

However, no matter which kernel function is chosen, or what the bandwidth h is, NW estimator has its limitation. It actually fits a weighted constant regression line in the neighborhood of x_0. The prediction variable only effects the weights of each neighbors of target point x_0. In [4], Cleveland has well indicated that kernel regression always introduces bias in the estimation of a linear surface, especially on the boundaries of the domain. A more sophisticate model than local constant model is to fit a low order polynomial in the neighborhood of x_0. So the predictor variable not only effects the weights of neighbors, it also provides more information to the local model. More theoretical results can be found in [5], [6], and [7].

1.1.2 Fitting a Local Polynomial Model

loess, short for locally weighted scatter-plot smoothing, also known as locally weighted polynomial regression, is a more general and flexible method than the kernel regression we discussed above. It was originally introduced by Cleveland in [8], and then well expanded in the references of [9], [10], [4], and [11] in term of its methodology, theory, and computation

Suppose we are trying to fit at target point x_0 in the predictor space with one dimension. The loess fit is obtained by locally weighted least square with a polynomial of degree λ using n_α nearest neighbors of x_0. Concretely, a smoothing parameter α is defined, which is the ratio of number of observations included in the model as neighbors. Then the number of neighbors are calculated as following since α could be larger than 1,

$$n_\alpha = \min(n, \lfloor \alpha n \rfloor) \tag{1.7}$$

A weight is assigned to each neighbors based on their distance to the target x_0. The closer neighbors is to the x_0, the higher weight it will get. The weight is defined as a function of the distance. Specifically, a tri-cube weight function is used.

$$T(\mu) = \begin{cases} (1-\mu^3)^3 & \text{for } 0 \le \mu < 1 \\ 0 & \text{for } \mu \ge 1 \end{cases} \tag{1.8}$$

It is flatter on the top when μ is around 0, and is differentiable at the boundary of its support. Meanwhile, the distance from each neighbor to the target point is calculated based on the choice of distance calculation, such as Euclidean distance, Great Circle distance, or Mahalanobis distance [12]. Suppose we use Euclidean distance L^2 norm $\|\cdot\|$, and $d_\alpha(x_0)$ is the distance between x_0 and its n_αth nearest neighbor. Then the weights for each x_i when fitting at x_0 are generated as

$$\omega_i(x_0) = T\left(\frac{\|x_i - x_0\|}{d(x_0)}\right) \tag{1.9}$$

which assigns 0 weight to x_i outside the circle region with radius of $d(x_0)$.

Once the weights of each x_i are calculated for a given x_0, the locally fitting is trivial. Let $Y = (y_1, y_2, \cdots, y_n)^T$ and $\beta = (\beta_0, \beta_1, \beta_2)^T$, the local quadratic regression fit at target point x_0 can be illustrated in terms of matrix notation:

$$Y = X\beta + \epsilon \tag{1.10}$$

and the minimization of objective function for a given target x_0 is the estimate of parameters.

$$\hat{\beta} = \arg\min \ (Y - X\beta)^T W (Y - X\beta) \tag{1.11}$$

where

$$X = \begin{pmatrix} 1 & (x_1 - x_0) & (x_1 - x_0)^2 \\ 1 & (x_2 - x_0) & (x_2 - x_0)^2 \\ \vdots & \vdots & \vdots \\ 1 & (x_n - x_0) & (x_n - x_0)^2 \end{pmatrix} \tag{1.12}$$

is the design matrix of each local quadratic regression model, and weight diagonal matrix is

$$W = \begin{pmatrix} \omega_1(x_0) & 0 & \cdots & 0 \\ 0 & \omega_2(x_0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega_n(x_0) \end{pmatrix} \tag{1.13}$$

For each target point x_0, we need solve the optimization problem in equation 1.11. It is easy to get the estimate based on weighted least square:

$$\hat{\beta} = (X^T W X)^{-1} X W Y \tag{1.14}$$

and the estimate of response at x_0 is just the intercept of the local quadratic curve:

$$\hat{y}_0 = \hat{\beta}_0 \tag{1.15}$$

1.1.3 Robust Locally Weighted Regression

One of shortcoming of local regression methods is its weakness to the effect of outliers. In [8], a robust local regression procedure was proposed. Specifically, after the original local weighted regression fitting at each x_i, residuals defined as following

$$r_i = y_i - \hat{y}_i \tag{1.16}$$

Meanwhile, let B be the bi-square function defined as:

$$B(\mu) = \begin{cases} (1-\mu^2)^2 & \text{for } \|\mu\| < 1 \\ 0 & \text{for } \|\mu\| \geq 1 \end{cases} \tag{1.17}$$

Let m be the median of the $|r_i|$, and robustness weights is defined as

$$\delta_i = B\left(\frac{r_i}{6m}\right) \tag{1.18}$$

Robustness weights is used to measure the outliers. If there is an outlier in the data, the corresponding r_i should be relatively larger than the others. And we definitely would like to shrink the weight of this outlier in all local regression fitting which includes it as neighbor. So δ_i leans to 1 if the residuals normalized by median is small, and is closed to 0 if normalized residual is large. Then we compute new $\hat{y}_i$ for each i using the same local quadratic model but with updated weights

$$\hat{\beta} = (X^T W_{update} X)^{-1} X W_{update} Y \tag{1.19}$$

where

$$W_{update} = \begin{pmatrix} \delta_1\omega_1(x_0) & 0 & \cdots & 0 \\ 0 & \delta_2\omega_2(x_0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_n\omega_n(x_0) \end{pmatrix} \tag{1.20}$$

Finally we repeatedly utilize this procedure with a number of times to get the robust locally weighted regression fitted value $\hat{y}_i$. Usually, four iteration time is enough to get reliable and converged result.

Because of its flexibility and simplicity, there are various application of locally weighted regression. In the next two sections, we illustrate two of main application of loess in different area of data analysis, time series and spatial analysis respectively.

1.1.4 Computation

The last piece of the loess method is the computation procedure. It will become more computationally intense to fit at every single observation in the data if the size of it is keeping growing. So modification has to be applied to the original loess to make the computation more feasible.

First of all, a kd-tree [13] is built to partition the predictor space into cells. The number of cells is decided by specifying the number of observation in each cell is no more than

$$\lfloor \frac{n\alpha}{5} \rfloor \tag{1.21}$$

Instead of getting local weighted regression estimate at every point, the directly estimation is only calculated at the vertices of the kd-tree. Derivatives of $\hat{y}$ are also estimated at the vertices by taking the slopes of the local models. And then cubic Hermite spline is used to interpolate any point within the corresponding cell. In [14], Hafen demonstrated with more details that the interpolated fit is very close to the exact fit, but make the computation to be linear in n.

1.1.5 Conditionally Parametric Model

The conditionally parametric fit, introduced in [15], provides a generalization of the semi-parametric model, without the complexity of a full local fit. Under the context of the dataset, we can easily divide the three predictor variables into two disjoint groups, one includes longitude and latitude which is the horizontal direction, and another is elevation in the vertical direction. It is a well known fact that the pressure is decreasing when altitude is going up. Therefore at a given location, the temperature is decreasing as well when altitude is high. It is plausible to specify the maximum temperature to be conditionally parametric in longitude and latitude. Making such a specification when it is valid can result in a more parsimonious fit. An exceedingly simple modification of loess fitting yields a conditionally parametric surface. We simply ignore the elevation factor in computing the distances that are used in the definition of the neighborhood weights, $W_i(\mathrm{x})$.

Let us demonstrate the concept and implementation with more details. The method for making a loess fit conditionally parametric in a proper subset of predictors is simple. The subset is ignored in computing the distances that are used in the definition of the neighborhood weights, $\omega_i(x)$. Suppose that there are two predictors in order to make the point straight forward, μ and ν. Suppose we specify μ to be the global variable of the conditionally parametric model. Since the weight function ignores the variable μ the ith weight, $\omega_i(\mu, \nu)$ for the fit at (μ, ν), is the same as the ith weight, $\omega_i(\mu + \delta, \nu)$, for the fit at $(\mu + \delta, \nu)$. Thus the quadratic polynomial that is fitted locally for the fit at (μ, ν) is the same as the quadratic polynomial that is fitted locally for the fit at $(\mu + \delta, \nu)$, whatever the value of δ. So for the fixed value of ν, the surface is exactly this quadratic as a function of the μ.

A growing list of practical applications has shown that conditionally parametric fits add structure in a way that is quite useful in practice [10] and [16]. In [17], Cleveland illustrate the conditional parametric model using Taylor series approximations as a handrail, which is very delightful. Specifically, consider the maximum temperature

surface as a function over all factor longitude, latitude, and elevation. And suppose the surface is very complex and has a lot of peaks and valleys. By Taylor series, if the elevation is varied by a sufficiently small amount, the temperature surface can be well approximated by a very simple function, linear or quadratic given the value of longitude and latitude. In [15], The conditionally parametric model is compared to the partially linear model, which is similar with it. However, the conditionally parametric model allows all coefficients of the parametric variables to depend on the smoothing variables, which is not valid in partially linear model.

1.2 Seasonal Trend Decomposition using Loess (STL)

Seasonal Trend Loess is first introduced by Cleveland in [18]. It is a simple but powerful design for seasonal time series, which is built on a series of applications of the locally weighted regression. It can also be understood as a form of general additive model utilizing back-fitting algorithm to iteratively update several components of the time series at each time point.

Compared with other seasonal adjustment methods of time series such as X-11, STL method does not just remove the seasonal component from the original time series. It actually decomposes time series into seasonal, trend, and remainder components, so the property and variability of each component including seasonal is able to be analyzed.

1.2.1 Basic Procedure of STL

As a filtering procedure, STL is trying to decompose a time series y_i, $i = 1, \cdots, n$, to be

$$y_i = t_i + s_i + r_i \tag{1.22}$$

where t_i, s_i, and r_i are the trend, seasonal, and remainder components for ith time point respectively. With an pre-defined initial value of s_i and t_i (usually set to be 0), STL starts to smooth the seasonal and trend components iteratively, and the

remainder is whatever left in y_i besides the other two. The seasonal component has to be determined based on a prior knowledge of the time series itself, which is the seasonal periodicity. For example, the dataset we will demonstrate in the second chapter is the about monthly maximum temperature in the United State. There are 12 observations in each annual period, so the $n_{(p)}$ is equal to 12. If the time series is hourly temperature observation, then the $n_{(p)}$ is 24, and so forth.

STL is comprised of two nested iterative procedures, an outer loop with an inter loop inside. Collectively, all smoothing operation in STL are based on loess method. Smoothing parameters are required to be specified for each smoothing operation, and the smoothing parameter α is replaced by window size $w = \lfloor \alpha n \rfloor$ in those operation, which is just specifying the number of observation used in local model instead of the ratio.

First in the smoothing procedure of seasonal component, the original time series is split into $n_{(p)}$ *cycle-subseries*, each of which is defined to be the subseries at each time point of the seasonal cycle [14], [18]. For instance, for monthly maximum temperature observation, all the observations of January will be the first subseries, and all values of February is the second subseries and so forth, so totally there are 12 subseries for monthly data. Each subseries is smoothed separately and independently, and then they are reassembled back together. Next, smoothing procedure of trend component is applied to the *deseasonalized* series, which exclusive the seasonal component fit from the y_i. The outer loop starts with the inner loop in which two components are estimated back and forth using back-fitting algorithm [19]. After the inner loop, a robustness weight is calculated before the end of current iteration of outer loop, and it will be passed into the next iteration in order to eliminate the effect of outliers.

Inner loop

The inner loop is nothing but the estimation procedure of seasonal and trend components iteratively. Concretely, the trend component is initialized as 0, then

the seasonal component is obtained by applying smoothing separately to each *cycle-subseries* of the *detrending* series using local weighted regression with window size w_s and degree d_s; trend component is calculated by another local weighted smoothing to the *deseasonalized* series with window size w_t and degree d_t. Moreover, seasonal and trend smoothing procedure actually are two filtering procedure. Unfortunately two filtering procedure compete with each other the variation in the original time series. So another high-pass filtering procedure is applied between the seasonal and trend smoothing.

There are mainly 5 steps in the inner loop, and these steps will be iterated for $n_{(I)}$ times. In each iteration time k, we utilize the same smoothing parameter for all *subseries*, which are w_s and d_s, to get the current estimate of seasonal component $s_i^{(k)}$, and use smoothing parameter w_t and d_t to get the current estimate of trend component $t_i^{(k)}$. Details are illustrated as following:

1. *Detrending*: We start with subtracting the previous estimation of trend component from the y_i: $y_i - t_i^{(k-1)}$. If $k = 1$, then we initialize $t_i^{(0)} = 0$. Notice, if y_i is missing, the detrended series is also missed at the time point i.

2. *Subseries Smoothing*: A loess smoothing procedure is applied separately to each *subseries* of the detrended series $y_i - t_i^{(k)}$ with smoothing window size w_s and smoothing degree d_s. For each subseries, smoothed value is not only calculated at every time point in the original series, but also one time point before and one time point after the original series. Missing values in the original series are also fitted at this step. Consequently, all $n_{(p)}$ smoothed subseries are pulled back together to be the temporal seasonal series $C_i^{(k)}$ with length of $n + 2n_{(p)}$, because we extrapolated each of $n_{(p)}$ subseries with one extra fitting value on each side. i is from $-n_{(p)} + 1$ to $n + n_{(p)}$.

3. *Low-pass Filtering*: As we mentioned previously, the smoothing procedure which generates $C_i^{(k)}$, it also captured low-frequency variation which truly should belong into the trend component, we call this the competition between seasonal

smoothing and trend smoothing. In order to solve this issue, a low-pass filter is applied to $C_i^{(k)}$ to capture any low-frequency variation, or any long-term trend in $C_i^{(k)}$, which we denote as $L_i^{(k)}$. Specifically, this low-pass filtering is done by three moving average procedure with length of $n_{(p)}$, another $n_{(p)}$, and 3 respectively. Since moving average drop the ending points, the length of the series is back to n after them. Then a loess smoothing procedure with parameter w_l and d_l is applied to generate the $L_i^{(k)}$ series. Finally the seasonal component is obtained as $s_i^{(k)} = C_i^{(k)} - L_i^{(k)}$.

4. *Deseasonalizing*: Once the seasonal component is estimated, it is subtracted out from y_i. If y_i is missing, then $y_i - s_i^{(k)}$ is also kept as missing at the i time point.

5. *Trend Smoothing*: The last step for inner loop is applying a loess smoothing with parameter w_t and d_t to the deseasonalized series $y_i - s_i^{(k)}$. Smoothed value is calculated at all time points including those with missing values

The inner loop is run $n_{(I)}$ times, and remainder is calculated based on the estimates of seasonal component $\hat{s}_i$ and trend component $\hat{t}_i$ once the inner loop is finished:

$$\hat{r}_i = y_i - \hat{t}_i - \hat{s}_i \tag{1.23}$$

Outer loop

It is very likely to have an outlier in the given time series. For example, like the monthly maximum temperature data in chapter two, extremely hot or cold weather always exist. So similar with robust locally weighted regression mentioned in previous subsection, robust weights are calculated to lessen the effect of outliers.

Let m be the median of the $|r_i|$, and robustness weights is defined as

$$\delta_i = B\left(\frac{|r_i|}{6m}\right) \tag{1.24}$$

B is still the bi-square function defined in 1.17. So the weights for each observation used in the inner loop is now multiplied by the robustness factor δ_i. The inner loop and robustness calculation together form the outer loop, and it iterates for $n_{(O)}$ times.

1.2.2 Choosing Smoothing Parameters

Collectively, there are 8 smoothing parameters needed to be specified in STL procedure besides $n_{(p)}$, which are:w_s, d_s for seasonal smoothing, w_l, d_l for the low-pass filtering procedure, w_t, d_t for trend smoothing, and iteration time $n_{(I)}$ and $n_{(O)}$. Fortunately, most of smoothing parameters have very nice default value except w_s, d_s, and w_t. In the current implantation of STL in base R called *stl* function, d_s and d_t are restricted to be either local constant or local linear fit, which is not quite efficient for complex seasonal or trend component. It is reasonable to restrict d_l for low-pass filter to be local linear since we absolutely do not want any variation with high frequency is filtered out from seasonal component. As illustrated in [18], by setting w_l to be the least odd integer greater than or equal to $n_{(p)}$ can help to reduce the competing for same variation between trend and seasonal component. With respect to the iteration time, robust fit is needed if extreme value behavior exist based on the domain knowledge of the data. With $n_{(I)} = 1$, 5 will be a fair value for $n_{(O)}$, otherwise, $n_{(I)} = 2$ is sufficient for non-robust fit in general.

Visualization diagnostic plots are very helpful for us to decide the value of w_s, d_s, and w_t. For estimate of seasonal component, $s_i + r_i$ and centralized s_i are plotted against to time i conditional on each subseries which is called as *Seasonal-Diagnostic plot*. It helps us to balance the bias-variance tradoff within seasonal smoothing procedure. Examples can be found in [18]. For estimate of trend component $\hat{t}_i$, $t_i + r_i$ and t_i itself are plotted against to time i. By assessing the visual graph, we can tell if too much variation (high variance) or very few variation (high bias) is included in the trend component.

1.2.3 STL with Added Feature in `stlplus`

In [14], Hafen demonstrated a new implementation of STL procedure in R called *stlplus* package. Compared with original *stl* function in base R, it has following benefits:

1. Enable local quadratic fit for seasonal and trend component. Based on the behavior of the time series, it is very likely we need local quadratic fitting to the smoothing procedure.

2. Be able to handle missing value. Even though the STL procedure illustrated in [18] is able to handle missing values, the old implementation *stl* function in base R cannot. *stlplus* function now is capable to fit at any missing value time point.

3. Give a theoretical lower bound for smoothing window for local quadratic fit. In the original STL article [18], a lower bound for w_t, and w_l is provided only for local linear fit. In the *stlplus* implementation, however, [14] provides theoretical lower bound for w_t, w_l when utilizing local quadratic fit for seasonal and trend smoothing.

4. Blending endpoints to lower degree polynomial. One of the most attractive problem for time series analysis is forecasting the future observations. However, the estimate of endpoints always suffers the asymmetric weights. Blending the fitting at endpoints can help to eliminate the variance of estimate [14]. According the empirical and theoretical analysis in [14], if the degree of smoothing procedure is 1, we blend the loess at endpoints to be local constant fit with window size same as other time points. If the degree is 2, we blend the loess to be local constant fit with window size as the next odd integer of $(w_s - 1)/2$ or $(w_t - 1)/2$. Moreover, a gradually blending mechanism, which controlling the degree of blending by δ, is also introduced in [14] to deal with all of the endpoints.

It has been shown in [14] that blending endpoints can be viewed as a shrinkage method, which reduces the mean squared error of endpoint estimates by sacrificing some of bias. Specifically, let X_d be the design matrix for the local regression with degree d at the right endpoint, here we omit rows that are outside of w window of neighborhood. Then

$$X_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & w-2 \\ 1 & w-1 \end{pmatrix}_{w \times 2} \tag{1.25}$$

where w is the window size (or span). And the weight matrix W for the corresponding X_1 is

$$W = \begin{pmatrix} T(0/(w-1)) & 0 & \cdots & 0 \\ 0 & T(1/(w-1)) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T((w-1)/(w-1)) \end{pmatrix}_{w \times w} \tag{1.26}$$

where T is the tricube function defined in 1.8. So if we blend degree from 1 to 0 with blending proportion δ, and let $e_2^T = (0, 1)$, estimate of coefficient is

$$\hat{\beta} = \arg\min (y - X_1\beta)^T W (y - X_1\beta) + c\beta_1^2 \tag{1.27}$$

where

$$c = \left(\frac{e_2^T (X_1^T W X_1)^{-1} e_2}{\delta - e_2^T (X_1^T W X_1)^{-1} e_2} \right)^{-1} \tag{1.28}$$

More mathematical details can be found in [14].

1.3 Divide and Recombine (D&R) for Large Complex Data

1.3.1 D&R Statistical Framework

D&R [20] is a powerful and practical statistical framework for the analysis of large and complex data. It consists of three main steps which are divide, apply,

and recombine. The analyst first divide the whole original dataset into subsets or divisions. The division methods can be either defined by the analyst or based on a conditional variable in the dataset itself. For instance, if the dataset is a spatial-temporal data, then it is reasonable to divide the data either by the time unit or by the location unit.

There are two different categories with respect to the analytic method applied to each subset or division. One is the numerical analysis method whose results are numerical values such as fitting results of a statistical procedure applied to each subset; another type of method is visualization method whose results are visual displays created by plotting routine such as `lattice` in R. However sometimes due to the enormous number of the subsets, only a sample of visual displays of subset can be evaluated carefully [21].

Next a statistical analysis method is applied to each subsets of the division in parallel. It is an embarrassingly parallel computation which means there is no communication between each subset. Finally, the analysis results are recombined together with a selected recombination method. It can be a computational method which applied to the outputs across all subsets to generate the final result for the whole dataset, or it can just simply combine the results of each subsets.

1.3.2 Computational Environment

The front end of the computational environment of D&R is R [22], a widely used software environment for statistical computing and graphics. On the other side, the back end is the Hadoop which consists of Hadoop Distributed File System (HDFS) [23] for storage and processing engine (MapReduce) [24]. And RHIPE [25], the R and Hadoop Integrated Programming Environment, bridges the gap between these two ends.

As analyst, we only need to specify R commands to carry out a D&R job which consists of following three steps:

- Divide the whole dataset into subsets. It can be randomly dividing or can be done conditional on a given categorical variable.
- Apply the analytic method to each subset in parallel.
- recombine the outputs of the A computations and write results to the HDFS.

The first step can be achieved by one MapReduce job using RHIPE R commands which creates the subsets from the original raw data set sat on HDFS and distributes the subsets across the servers of the cluster onto the Hadoop Distributed File System (HDFS) as key-value pairs. Most of situation, the original raw data can be a raw text file saved on HDFS.

Thereafter, the second and third steps can be implemented with another MapReduce job also specified by analyst in RHIPE R commands. In this second MapReduce job, a group of Map computation procedures are running embarrassingly parallel with a free core assigned to each, which means independently with no communication among those Map procedures. We call those Map procedures the Mapper. Then, a data block or a collection of subsets will be passed into those Mappers, and each subset will be applied with the analytic method independently.

2. SPATIAL SEASONAL TREND LOESS (SSTL)

In this chapter, a new statistical method for the analysis of spatial seasonal-temporal dataset is proposed, named as Spatial Seasonal Trend Loess (SSTL). The chapter starts with the illustration of main steps of analysis routine of SSTL. Next, a spatial-temporal dataset from National Climatic Data Center (NCDC) [1] is introduced and used as an example to explore the routine step by step.

2.1 Road-map of the Routine

Spatial Seasonal Trend Loess (SSTL) is a new procedure for fitting spatial seasonal-temporal dataset. It consists of four main steps of modeling, which are spatial and temporal fitting procedures respectively. Suppose the spatial-temporal dataset is about monthly observations at different weather stations. Each observation is measured at a given station in a given month. Then procedure is shown as follows:

Step I The raw data in text format is first read into R session, and a data.frame is generated from it. Each row represents an observation at a given station in a given month. Location information of each station and the month index are saved as other columns besides the response variable in the data.frame.

Step II For each month, spatial loess smoothing is applied to the observations at all stations in the given month. Concretely, a kd-tree is built based on the location (longitude and latitude) of all stations including those with missing values. Then a local polynomial regression is fitted at each vertex of the kd-tree. Distance of nearest neighbors and corresponding weights are calculated based on the Great-circle distance. Cubic interpolation is carried out for any locations fall into the area covered by the kd-tree. The smoothing neighborhoods are set to be small,

since the purpose of this fitting is to eliminate spatial noise and infill missing values. Over smoothing is much worse than under smoothing in this case.

Step III Next, the spatial smoothed value and corresponding month index of each station are passed into `stlplus` function in R to proceed the STL+ fitting. Smoothing parameters such as s_w and s_d, smoothing window and degree for seasonal component, t_w and t_d, smoothing window and degree for trend component, are specified for the temporal fitting of each station. The fitted values are seasonal, trend, and temporal remainder components.

Step IV The final step is spatial smoothing, which is applied on the temporal remainder component for each month. After STL+ fitting in temporal dimension at each station, remainder component is the variation that cannot be explained or captured by the STL+ procedure. However, it does have correlation with spatial predictor variables in each month. The spatial surface of remainder and the corresponding spatial predictor variables are fed into `spaloess` function with predefined smoothing parameters `span` and `degree`, which controls the span of neighborhood and polynomial degree of local polynomial regression in the spatial loess.

More details with examples for each step of the procedure are explored in the next several sections after a brief introduction of the example dataset we use.

2.2 The Source of Dataset

The dataset we are going to analyze is a compendium of different levels of weather data ranging from stations taking regular hourly measurements, such as those at airports, to cooperative observer where the records may only include daily values and have gaps in time. Some of original data sources hosted by NCDC [1] are shown as follows:

- COOP: Through the National Weather Service (NWS) Cooperative Observer Program (COOP), more than 10,000 volunteers take daily weather observations at National Parks, seashores, mountaintops, and farms as well as in urban and suburban areas. COOP data usually consist of daily maximum and minimum temperatures [26].

- SNOTEL: The Natural Resources Conservation Service (NRCS) operates and maintains an extensive and automated system (SNOwpack TELemetry or SNOTEL) designed to collect snowpack and related climatic data like air temperature in the Western United States begins in 1978 [27].

- AG: Agricultural climate data for southeastern Washington from United States Department of Agriculture Natural Resources Conservation Service (USDA-NRCS) [28].

- MRCC: Midwest Climate Data Center data, mainly for period between 1895 and 1948. The MRCC [29] serves the nine-state Midwest region (Illinois, Indiana, Iowa, Kentucky, Michigan, Minnesota, Missouri, Ohio, and Wisconsin).

- USHCN: Historical Climate Network data provides summary of the month temperature and precipitation observations for 1,218 stations across the contiguous United States. Temperature observations have been homogeneity corrected to remove biases associated with non-climatic influences, such as changes in instrumentation and observing practices, and changes to the environment including station relocations [30].

In 2002, based on the data set we listed above, the NCDC processed them to a consolidated and uniform 103-year spatial temporal data set by combining stations at similar locations, eliminating stations with short records, and aggregating some of daily measurement to be monthly. In summary, the data we are going to analyze is about observed monthly average maximum daily temperatures for the conterminous

US from 1895 to 1997. There are 8,125 stations reporting monthly average maximum daily temperatures at some time in this period which includes 1,236 months.

2.3 Step I: Read in

The raw dataset is a collection of 103 files in fixed-width text format. Each of file names is in the form of `tmax.complete.Ynnn` with `nnn = 001, 002, ··· , 103` such as `001 = 1895` and `103 = 1997`. Each separate data file consists of maximum temperature for one of the year. Each line of the files contains 12 monthly maximum temperature observations and 12 observed/missing value codes (`1=missing`, `0=observed`). We only keep the observed monthly maximum temperature in our analysis, and use `NA` to represent those missing values. Totally there are 8,125 stations, and the temperature appears as a integer in tenths of Celsius degree. For instance, 73 in the raw text file should be interpreted as 7.3 degrees Celsius.

Additionally, besides the 103 raw data files, there is metadata about stations' information which includes:

1. The station id, which uniquely identifies each station.
2. The degree of longitude of station.
3. The degree of latitude of station.
4. The meter of elevation of station.

The metadata set is in text format of 244 KB.

We consider to transform the raw text files to be a more analysis friendly structure, data.frame object in R, which only has one monthly observation per row. Concretely, the data.frame should include following fields (columns):

1. `station.id`, character variable, the station ID, which uniquely identifies each station.
2. `lon`, numeric variable, the degree of longitude of the station.

3. `lat`, numeric variable, the degree of latitude of the station.
4. `elev`, numeric variable, the meter of elevation of the station.
5. `year`, numeric variable, the year of the observation.
6. `month`, character variable, the month of the observation.
7. `tmax`, numeric variable, the observation of the maximum temperature for the given month.

2.3.1 Summary of the Data

In summary, the dataset covers monthly maximum temperature over the period of 103 years, from Jan 1895 to Dec 1997. 8,125 stations are scattered across the conterminous US excluding Hawaii and Alaska. Concretely, the elevation of stations ranges from -59 to 3810 meters, the longitude ranges from -124.73 degree to -67 degree, and the latitude ranges from 24.55 degree to 49 degree. The whole dataset is saved as data.frame object in R with 10,042,500 rows and 7 columns.

Location of Stations

Figure 2.1 illustrates the location of all 8,125 stations on a map of United States.

Link to figure

Figure 2.1.: The location of all stations

Meanwhile, we equally split the stations into 8 groups based on their elevation. Cutting points are at 58m, 162m, 244m, 351m, 581m, 1098m, and 1647m. And then location of stations are demonstrated on a map of United States conditional on the elevation.

Link to figure

Figure 2.2.: The location of stations conditional on elevation

Besides the demonstration of stations' location based on longitude and latitude, the distribution of elevation of stations is also shown in Figure 2.3.

Link to figure

Figure 2.3.: Quantiles of elevation of stations

In Figure 2.3, horizontal dash lines are the cutting points of elevation when we split stations into groups. It shows that more than 60% of stations are below 581 meters. Figure 2.2 tells us that those stations are mainly outside of the Pacific Coast Ranges, which includes the Rocky Mountains, Columbia Mountains, Interior Mountains, the Interior Plateau, Sierra Nevada Mountains, the Great Basin mountain ranges. Also, there are 724 stations are on the west coast of United States, which is between the Pacific Ocean and the Pacific Coast Ranges.

The Number of Observation

As we mentioned before, missing values have been saved as `NA` in the dataset. We would like to reveal where and when are those missing values. In Figure 2.4, the log base 2 number of observation in each month is drawn against to month index which is from 1 to 1,236. The red solid reference line in the plot is the $\log_2(8125)$, which is the number of observations without any missing value in one month. The number of observations increased gradually from Jan 1895 to Jan 1915. But then there was a dramatically jump in 1931. The valid observation number arose from 1,619 on Dec

1930 to 2,953 on Jan 1931. After Jan 1950, the count of observation stayed around $5,000$, which is 65% of the number of all stations.

Link to figure

Figure 2.4.: The number of observations over time

A deeper look into the dataset shows that January has minimum number of observation in 70 out of 103 years. The observation count is plotted against month index superposed on month of year, only for January, February, and December in Figure 2.5

Link to figure

Figure 2.5.: The number of observations around Jan 1982

There is not any enormous difference between the twelve month-of-year with respect to the observation count before 1981. However, Figure 2.4 and Figure 2.5 both indicate there is a significant drop of observation count on Jan 1982. And among the 16 years after that, January always has the lowest observation count.

Moreover, we found that this drop on Jan 1982 is not caused by adjustment of location of the stations, such as one station shifts its location to its neighborhood. Actually 688 stations were missing observation on Jan 1982, but then they were active again on Feb 1982. As shown in Figure 2.6, the blue circles represent stations whose status were changed to be missing from December 1981 to January 1982. Meanwhile the magenta solid points represent stations whose observation status switched back to be valid from January 1982 to February 1982. If a location has both blue circle

and magenta solid point, it means that weather station suddenly failed on January 1982, but then went back to normal on February 1982.

Link to figure

Figure 2.6.: The location of active stations around Jan 1982

We believe this astonishing number of failure of stations on January 1982 is caused by severe weather, which may have caused the battery in the weather station to die. From January 11 to January 17, A brutal cold snap sends temperatures to record lows in dozens of cities throughout the Midwestern United States. Especially on January 17, 1982, so called "Cold Sunday" [31], in numerous cities temperatures fall to their lowest levels in over 100 years.

2.3.2 Subset after Year 1950

As shown in the Figure 2.4, we notice the number of active stations is gradually stabilized after Jan 1950. Since long period of missing observation is not the main concern of this thesis, we decide only consider stations with observations after Jan 1950. So the subset includes 7,738 stations with 576 months observation in each, which is saved as a new data.frame in R with 4,457,088 rows and 7 columns.

Location of Stations

Figure 2.7 illustrates the location of 7,738 stations on the map of United States. These stations are still densely scatter over the entire US map even though it is 387 stations less than the original dataset.

Link to figure

Figure 2.7.: The location of stations after 1950

Link to figure

Figure 2.8.: The observation status of stations in each month

For each month, it is a spatial regularization of the maximum temperature over 7,738 locations, but with around 2,000 missing values. For instance, in Figure 2.8, we illustrate the observation status of 7,738 stations for all 576 months.

As we will see in later sections, diagnostic plots play a critical rule in our analysis. However it is not feasible to visualize the analytic results of all 7,738 stations for example. A natural approach to compromise this issue is to visualize a reasonable part of all stations. Randomly sampling stations may be plausible, but it is likely that we sampled stations densely or sparsely in a specific region on the map. In order to get well presentation of all stations, a sampling procedure is proposed based on near exactly replicate.

As shown in Figure 2.9 A kd-tree with 512 cells is built based on 7,738 stations. Then one station is sampled from each cell randomly. The blue dots on the graph show the locations of these 512 sampled stations. An index number from 1 to 512 is assigned to each station to illustrate which cell the station comes from.

Link to figure

Figure 2.9.: The location of 512 stations sampled from each cell of a kd-tree

We also graph the quantile plot of the log base 2 of elevation of stations in each cell in Figure 2.10. The elevation in majority of the 512 cells varies in a very limited range. However, some of the cells, such as cells 3, 4, 6, 9, 17, 18, in which stations are concentrated in the area of west coast of the United State. The distribution of elevation has a much more expanded range than that in the area of the central United States.

Link to figure

Figure 2.10.: The quantiles of elevation of stations conditional on cell

The Number of Observation

As shown in Figure 2.8, the location of stations with missing values in each month are relatively sparse and random. There is not any region covered by nothing but missing values. However the missing value is not quite random in the temporal dimension. Here we provide visualization evidence about the high rate of missing value in the time series at each station.

Link to figure

Figure 2.11.: The quantiles of number of observation in one station

Link to figure

Figure 2.12.: The quantiles of rate of valid observation number in one station

In Figure 2.11, the number of observation for each station, in log base 2, is plotted against to their corresponding f-value. The black dash line with "Full Obs" label represents the $\log_2(576)$, which is the number of observations in a station without any missing value. Only less than 20 percent of stations do not have missing value during that 576 months. Meanwhile, about 30 percent of stations have only 256 or less observations in total. It is about 40 percent of total observations for a station based on the Figure 2.12, in which the rate of valid observation of each station is drawn against to the corresponding f-value.

If the missing values of each station are sparse only, we still can handle them well in the temporal dimension using fitting procedure like `stlplus`. However, as shown in Figure 2.13 in which the log based 2 of the maximum length of consecutive missing values in a station is plotted against to its f-value, more than 20 percent of stations have maximum length of consecutive missing values longer than 16, which is more than a year.

Link to figure

Figure 2.13.: The quantiles of maximum length of consecutive missing value

Because of the consecutive missing values in time series, we start the SSTL procedure with spatial loess fitting instead of temporal fitting. Another critical reason for starting with spatial loess fitting is to eliminate spatial noise in the spatial dimension.

Distribution of Maximum Temperature

Another main question about the dataset is how the response variable maximum temperature distributes. Figure 2.14 demonstrates the distribution of maximum temperature. 10,000 quantiles of maximum temperature from 0.0001 quantile to 0.9999 quantile are plotted against to their corresponding f-value, as well as the minimum

and maximum value of the response variable. The maximum temperature varies from -23 degree to 49 degree. The median is about 19.6 degree, and the mean is 18.3 degree overall.

Link to figure

Figure 2.14.: The distribution of maximum temperature

After the overall distribution of the maximum temperature, we are also interested in how it varies across the US map for each month. Next, the distribution of response variable is plotted in Figure 2.15. There are 576 pages in total, one for each month. Collectively, the maximum temperature in the winter has larger variation than in the summer. And in July and August, the distribution is more symmetric than other months.

Link to figure

Figure 2.15.: The distribution of maximum temperature in Year 1950

Besides checking by each month, we also plot the maximum temperature against to the month index for given number of stations. In Figure 2.16, it shows that a great amount of stations have missing value issue with varying degrees of severity. For example, station `046483` from cell 15, it did not have any observation after October 1955. Or station `421418` from cell 48, it only had 3 observations in a year from 1975 to 1984.

Link to figure

Figure 2.16.: The maximum temperature vs. month

Meanwhile, we notice that several stations such as `047905` from cell 1, `047916` from cell 3, `047807` from cell 5 and so on, have very vague seasonality compared with other stations. Mainly this is because those stations are all located on the west coast in California State where is commonly recognized as Mediterranean climate zones in the US.

2.4 Step II: Spatial Smoothing of Raw Observation

As we briefly introduced in section 2.1, the second step of SSTL routine is the spatial smoothing procedure for each month independently. Specifically, for each month, spatial fitted value at a target station is calculated based on a local weighted regression model only using n_α closest neighbors. Location information of stations, longitude, latitude, and elevation are included in the local regression model as predictor variables. Spatial loess smoothing is a modification of the original loess method. The distance calculation of neighbors and therefore the weights of them in the model are defined based on Great-Circle distance instead of Euclidean distance, since the predictor space is on the surface of a spheres.

Moreover, in order to improve the computational efficiency without losing too much of fitting accuracy, the local weighted regression model is not applied to all 7,738 stations in a given month. Instead, it is only carried out at several knots on the surface. Fitted values at any other locations are smoothed by cubic interpolation. The more dense the knots are, more heavy the computation is. One extreme case is that the knots are exactly all 7,738 fitting locations. The knots are chosen based on a two-dimension kd-tree, which are exactly the vertexes of the kd-tree.

2.4.1 Distance Calculation of Neighbors

loess method calculates the weights $\omega_i(x)$ for weighted least square based on the distance of neighbors x_i to the target point x_0. The distance calculation of $d(x_i, x_0)$ is based on Euclidean distance (in two dimension for example):

$$d(x_i, x_0) = \|x_0 - x_i\|_2 \tag{2.1}$$

In the spatial loess fitting, however, the Euclidean distance is apparently not appropriate. Because one unit increment in longitude degree is different than that in latitude degree [32]. A more reasonable and realistic distance definition in spatial dimension is the Great-circle distance, which is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere (as opposed to a straight line through the sphere's interior) [33].

Explicitly, let ϕ_a, λ_a and ϕ_b, λ_b be the geographical latitude and longitude of two points a and b, and $\Delta\phi$, $\Delta\lambda$ their absolute differences; then $\Delta\sigma$ the central angle between them, is given by:

$$\Delta\sigma_{a,b} = 2\arcsin\sqrt{\sin^2(\frac{\Delta\phi}{2}) + \cos\phi_a \cdot \cos\phi_b \cdot \sin^2(\frac{\Delta\lambda}{2})} \tag{2.2}$$

Then the distance $d(a, b)$, i.e. the arc length, for a sphere of radius r is

$$d(a, b) = r\Delta\lambda_{a,b} \tag{2.3}$$

2.4.2 Spatial Smoothing with Elevation

Very few of spatial or spatial-temporal analysis in the literature includes elevation as one of the predictors in the model. Experientially for spatial local regression model, if we simply add elevation in the local regression model directly, the calculation of weights of neighbors needs modification based on the three-dimension predictor space. However, the definition of distance in this three-dimension space is not trivial, since the temperature is not isotropic in the vertical and horizontal direction. As a result, Euclidean distance in three-dimension space is not appropriate for this situation.

Meanwhile, it is more intuitively to assume that the neighborhood or distance to neighbors are defined only based on longitude and latitude, as the weights introduced in Geographically Weighted Regression [34].

Consequently, a conditional parametric local model is proposed with elevation as the global variable. For a given value of longitude and latitude, the elevation variable is fitted globally. This can be easily achieved by excluding the elevation from the distance and weight calculation of each local regression fitting. The distance is just simply the Great-Circle distance based on longitude and latitude as we proposed in the previous subsection.

We specify the span of spatial smoothing to be 0.015, which means about 75 stations in the neighborhood of the target location will be used for local weighted regression fit since there are roughly 5,000 valid observations in total. The degree for longitude and latitude variables are set as quadratic, and the degree for conditional parameter elevation is also specified as quadratic. We demonstrate and assess the spatial smoothing result for this fit in the following diagnostics plots .

The quantiles between 0.015 and 0.095 of spatial smoothing residual is plotted against to the corresponding quantiles from a unit Normal distribution in Figure 2.17. As we can notice, the distribution of the smoothing residual has a longer tail than unit Normal distribution, and the range of it for each month is from -2 to 2 collectively.

Link to figure

Figure 2.17.: The normal quantiles between 0.015 and 0.985 of residual of spatial spatial smoothing

Another crucial diagnostic plot for the residuals are drawn in the Figure 2.18. The spatial smoothing residuals are plotted against to the corresponding smoothed values. The residuals are randomly distributed around 0 horizontal reference line

mainly between ±2 across all months collectively, no matter what the smoothed value is.

Link to figure

Figure 2.18.: Smoothing residual vs. fitted value

Next, we generate the diagnostics plots about the residuals against to one of the predictor variables to detect any pattern left in the residual with respect to the predictors space. As shown in Figure 2.19, spatial smoothing residuals are drawn against to latitude conditional on longitude. Longitude is converted into a categorical variable with 20 levels. The cutting point of those levels are decided such that the number of stations in each level of longitude is even. A loess smoothing curve with span equal to 0.25 and degree 1 is superposed in each panel as well. The loess curve is almost a flat horizontal line around 0, which reveal that no more variation can be explained by the predictor space.

Link to figure

Figure 2.19.: The residual of spatial smoothing against to latitude conditional on longitude

Then, we switch longitude and latitude, plot the residuals against to longitude conditional on latitude, which is converted into a categorical variable with 20 levels. Each page is the smoothing residual against to longitude for a given month. A loess smoothing line with span equal to 0.25, degree equal to 1 is superposed in each panel to summarize the overall behavior of residuals. Similar conclusion can be made for latitude variable as well.

Link to figure

Figure 2.20.: The residual of spatial smoothing against to longitude conditional on latitude

In the Figure 2.20, the mean of spatial smoothing residuals over each month is plotted against the corresponding f-value. Clearly, all the monthly mean are very close to 0. However, if we gather the spatial residuals of the spatial smoothing by stations, we find that the mean of the spatial residuals shift from the 0 for some of the stations. As shown in Figure 2.21, the residuals are plotted against the the month index for 512 sample stations. The residuals are mostly distributed around 0 reference line without any pattern. But at several stations, the mean of residuals shifts from the 0 reference line without any temporal pattern.

Link to figure

Figure 2.21.: The residual of spatial smoothing against to month index

Link to figure

Figure 2.22.: The distribution of the mean of spatial smoothing residuals

In the Figure 2.22, we plot the mean of residuals over months for each station against to the corresponding f-value. The station mean only shift from 0 within ± 1, but some of them shift from 0 by ± 4. Possible reason for this shifting mean is that the temperature of several stations are always lower or higher than its neighbors in

each month. Consequently, the mean of residuals in each station is added back to the spatial smoothed values to be the final smoothed values which will be passed into the STL+ fitting in the next step of routine as response.

Of course, we are not trying to perfectly capture the spatial variation in the maximum temperature within this spatial smoothing. It is only to fill in the missing values and smooth out some spatial noise, which will be a rigorous problem in `stlplus` fit in the next section. Another crucial reason for smoothing spatially first is for the prediction at a new location. Suppose we would like to carry out prediction at a new location where there is no historical data at all. So all information we can borrow for predicting is based on spatial dimension instead of time dimension. We smooth by assuming there is only very limited relation between different locations in two different months. After the spatial smoothing at the new location for each month in parallel, we then will carry out the fitting in time dimension as demonstrated in the following section since we generated an estimated time series at the new location.

2.5 Step III: Temporal Fitting by STL+

After the first spatial smoothing, we continue the analysis routine in the temporal dimension. In order to extract the seasonal and trend components from the time series of each station, we apply the STL+ method to the spatial smoothed value of each station separately. Similar with spatial loess procedure, a group of predefined smoothing parameters is specified for the STL+ method.

2.5.1 Seasonal Trend Decomposition with `stlplus`

As we discussed in section 1.2, Seasonal Trend Loess decomposition is a very powerful decomposition procedure that can capture seasonality and long term trend in a given time series. The underlying computation engine is really a series of loess smoothing procedures, such as loess smoothing on each subseries. So for all those loess smoothing procedures in STL+, smoothing parameters controlling the span and

degree of those local regression models should be chosen wisely. Theoretically, we can pass into different smoothing parameters for different stations. But here, we consider the same group of smoothing parameters for all stations.

More details about the procedure of choosing smoothing parameters can be found in the next chapter of diagnostics. Here we demonstrate the fitting results based on the best prediction model we found.

s.window ="periodic", s.degree = 1, t.window = 241, t.degree = 1

The smoothing span, or smoothing window as mentioned frequently in STL+ literature, for the seasonal component is specified to be `periodic`. Instead of a particular odd integer, `periodic` smoothing window means the smoothing of each subseries is effectively replaced by taking the mean, which is global constant fitting for the seasonal component. The first group of smoothing parameter we chose is considering local linear fitting for both seasonal component and trend components. The smoothing window for trend component is 241, which is about 42% of total observations. Meanwhile we set the iteration time of inner loop to be 2 and outer loop to be 1. The fit is evaluated using diagnostic visualization methods as following.

The first visualization display demonstrates the fitted value against month. In Figure 2.23, each page is one of the 512 sampled stations. The whole time series with 576 time points for each station is chunked into 6 panels from top to bottom, each of which has 108 observations. The spatial smoothed values are drawn with blue points, and the seasonal component plus trend component, which is the fitted value, are drawn as magenta curve.

Link to figure

Figure 2.23.: The fitted value vs. month

Collectively, no matter where the station is, the temporally fitted value (seasonal + trend) can precisely capture the temporal pattern of each station, especially the seasonality. Next, we carry out the diagnostics for each components based on corresponding diagnostic plots.

seasonal fitted values plot [14], in which seasonal plus remainder component with mean of seasonal subtracted, is drawn against to year is not necessary here. Since the seasonal component is the mean of each subseries. *seasonal fitted values plot* is same as *seasonal residual plot* [14], which drawn in Figure 2.24. The remainder is plotted against to year with a superposed loess smoothing line. The loess smoothing line here can be helpful to judge the lack of fit for the seasonal component. Clearly, there is not any lack of fit problem left in the remainder over all stations since loess smoothing line are all around horizontal line at zero.

Link to figure

Figure 2.24.: The seasonal residual vs. year

Since the seasonal components are the same in each subseries of the time series, we can plot the seasonal component unique value against to the corresponding subseries index, from 1 to 12. It is shown in Figure 2.25. The 12 unique seasonal component values are plotted against to the their subseries index for each station. The characteristics of seasonality varies dramatically among different stations even with the same smoothing parameters. For instance, the amplitude of the seasonality at stations from cell 1, 3, 5 are much smaller than the others. These three stations are all located on the west coast of California State where is commonly recognized as Mediterranean climate zones. One of the most crucial characteristics of Mediterranean climate is that temperatures are generally moderate with a comparatively small range between the winter low and summer high. So the seasonality is quite small in those stations.

Link to figure

Figure 2.25.: The seasonal fitted values vs. month

Besides the seasonal component, in Figure 2.26, the trend component of one station is drawn against to month index in blue curve in each panel. In order to optimize the perception of the trend curve, we include 12 panels on each page, which follows the 45 degree banking rule [35]. However this sacrifices the room in each panel, which make the visual assessing of trend component much harder if all remainder of each station are included in corresponding panel. So instead of superposing the remainder on each panel, the moving average of yearly mean of monthly maximum temperature is superposed on each panel as magenta points, which can help to judge the bias-variance trade-off of trend fitting. Collectively, the `stlplus` procedure can capture the different trend behavior across all different stations with the same parameters setting. Stations closed with each other in location have very similar long term trend.

Link to figure

Figure 2.26.: The trend components vs. month

Finally, Remainder is plotted against to month index for each station with a loess smoothing line superposed on each page in Figure 2.27. This diagnostic plot can help us to assess lack of fit or if there is any pattern left in remainder. Notably loess smoothing line in magenta color of each station is flat and close to horizontal reference line at zero.

Link to figure

Figure 2.27.: The remainder vs. month

2.6 Step IV: Spatial Smoothing of Remainder

In the previous section, we carried out `stlplus` fit with same parameters over all 7,738 stations, and results are very promising. The STL+ remainders are the variation left in the time series which cannot be captured by the STL+ procedure. Based on the diagnostic plots in the previous section, it is reasonable to claim that there is no more pattern left in the remainder temporally. However, temporal fitting is not the end of the analysis. In the next subsection, we can see the remainder components are spatially correlated in each month.

2.6.1 Spatial Correlation among Remainders

In the Figure 2.28, the remainder is plotted against to latitude of stations conditional on the longitude. Longitude is equally cut into 20 levels so each level includes about 250 stations. Clearly there is quadratic relationship between the remainder and latitude in all different levels of longitude cross all months, while the pattern is quite different among different months. For instance, in September 1950, there exists a convex quadratic shape in the most of longitude levels. However in October 1950, the shape of the pattern is changed to be concave mostly.

Link to figure

Figure 2.28.: STL remainder vs. latitude conditional on longitude

In Figure 2.29, similar situation is found that part of the variation of remainder component can be explained by the spatial factors after the temporally fitting. Remainders are plotted against to the longitude conditional on the latitude in the given month. Latitude is converted into a categorical variable with 20 levels. Specifically, `cut` function is utilized to accomplish this purpose. Obviously, a quadratic relationship exists between the remainders and longitude in each month.

Link to figure

Figure 2.29.: STL remainder vs. longitude conditional on latitude

2.6.2 Spatial Smoothing with Elevation

Conditionally parametric spatial loess is considered to regress the remainder onto the three spatial factors by using functions from `Spaloess` package. Elevation in log based 2 is again excluded from the distance and weights calculation of neighbors, which guarantees it to be the global parameter in the model. The smoothing span is set to be 0.015 based on the cross validation results discussed in chapter three. Meanwhile the smoothing degree is chosen to be local quadratic. So the degree of freedom of each local model is roughly about $75 - 10 = 65$.

In Figure 2.30, the final fitted residual is plotted against to longitude conditional on equally cut interval of latitude for each month. There are roughly 250 points in each panel of latitude interval. Also a loess smoothing curve with 0.25 span and degree 1 is superposed on each panel. Collectively, the loess smoothing curve in each panel of a given month is surprisingly almost flat and closing to 0. Meantime, residuals are randomly scattered around 0 and the width is equal throughout all panels of each month. But we notice there are some outliers with extreme residuals (outside of ± 2). We will analyze their property in later subsection.

Link to figure

Figure 2.30.: Residual against longitude conditional on latitude

Similarly, the final residuals are also plotted against to latitude conditional on 20 equally cut interval of longitude for each month in Figure 2.31. Same phenomenon can be found according to the loess smoothing line added in each panel. Residuals are randomly scattered around 0 without any pattern with respect to latitude in all panels of longitude in all of 576 months. And majority of the residuals fall into the range of -2 to 2.

Link to figure

Figure 2.31.: Residual against latitude conditional on longitude

After the longitude and latitude, residuals should be also assessed with elevation. As shown in Figure 2.32 and Figure 2.33, the residuals are graphed against the log base 2 of elevation plus 128 conditional on intervals of latitude and longitude respectively. The reason of adding a base line of 128 is because there are some of stations with negative elevation. Collectively, there is not any pattern related to elevation that is remained in residual. The loess smoothing curves are all surprisingly around 0 within each panel of longitude or latitude in each month.

Link to figure

Figure 2.32.: Residual against elevation conditional on latitude

Link to figure

Figure 2.33.: Residual against elevation conditional on longitude

So far, all four main steps of the SSTL routine have been explored with examples. In the next subsection, the package we create in R for spatial smoothing is introduced briefly.

2.6.3 `Spaloess` Package

`Spaloess` is a R package for spatial loess smoothing. It is highly depends on the original `loess` function in the `stats` package in base R. There are two main functions in the `Spaloess` package, which are `spaloess` function for spatial loess smoothing, and `predloess` for the spatial prediction using loess smoothing. Most of implementation in these two functions are kept the same as the `loess` and `predict.loess` functions, which is R wrapper functions. All memory allocation are done in C, and real computation engine is implemented in FORTRAN. `Spaloess` does offer the following advantages compared with original `loess`:

- Two different distance calculation are available. Euclidean distance and Great-Circle distance are allowed. For Great-Circle distance, the input spatial attributes must be longitude and latitude degree. Great-Circle distance calculation is implemented in FORTRAN.
- Interpolation kd-tree is built based on all locations instead of only non-NA locations as in `loess` function.
- Missing values in the dataset can be handled directly within `spaloess`.

In the original implementation of `loess` function, the kd-tree [36] for interpolation is only built based on observations that are not missing values. Those missing

observations are directly excluded from the analysis. It makes extremely harder and computational expensive to predict at those missing value especially if those missing value are outside the boundary of the space spanned by all independent variables. Instead, in the `spaloess` function, kd-tree is built based on all observations including those with missing value. Then interpolation can be easily conducted at every location including missing value.

In the next subsection, we are going to decide the best smoothing parameter for both spatial smoothing and STL+ temporal smoothing procedures. This is achieved by utilizing cross validation procedure with near exact replication framework to get the training and testing dataset.

3. DIAGNOSTICS METHOD FOR TUNNING PARAMETERS SELECTION

In this chapter, I illustrate procedure to choose the best smoothing parameters for spatial and temporal smoothing respectively. For the spatial smoothing, cross validation method is used to choose the best smoothing span and degree in all 576 months collectively. The training dataset and testing dataset are decided by using near exactly replicates framework which split the original dataset into subsets. For the temporal STL+ fitting, an experiment of tunning smoothing parameters is explored based on data after year 1950 to choose the best smoothing parameters in terms of prediction ability.

3.1 Tuning Parameters of Seasonal Trend Loess

In order to decompose the trend and seasonal components from each observation, we fit `stlplus` on the time series of each station independently. However, according to the literature of STL+, there is not any rule of thumb about choosing of smoothing parameters w_s, w_t, d_s, and d_t. So a data-driven approach to tun the parameters is considered.

Concretely, we are trying to use consecutive 270 observations, which is about 50% of the total number of observation for each station, as training data to predict oncoming 36 observations which treated as testing data. Then the error between the predicted value and the true observation is calculated to measure the prediction ability. For instance, we use the first 270 observations to predict 36 oncoming observations (271st to 306th). Then the time window of training dataset is moved one observation ahead (2nd to 271st), and the corresponding testing data is the next 36 observations. We count one run of prediction of 36 observations in the future as one

replicate. Since there are 576 observations for each station, we conduct 271 replicates for prediction of 36 observations within each station.

Then, we want to find the best set of smoothing parameters based on the prediction ability, which is namely accuracy measurement. There are many measurements which all have their strength and weakness. Research indicates that the performances of different methods are related to the purpose of forecasting and the specific concerns of the situation using the forecasts. That is why from a theoretical point of view there is no single best method [37]. In this thesis, the error measurement is therefore defined as following:

$$Error = |y_i - \hat{y}_i| \tag{3.1}$$

where y_i is the observation at ith month, and $\hat{y}_i$ is the corresponding prediction value which is equal to the summation of seasonal and trend components at ith month.

The usual choice: Mean Squared Error (MSE) is not suitable in this case because it is influenced by the outliers significantly. MSE gives too much weights to the outliers and the entire loss function is dominated by the extreme outliers in the data. What we want is a robust solution. The absolute prediction error does not discriminate against the regular values and limits the influence of the extreme outliers, which provides robustness in the assessment of the performance. On the other hand, the absolute value of the error makes sure the influence from positive and negative errors are treated equally.

The smoothing parameters we are going to tun are w_t, w_s, d_t, and d_s. A series of full factorial experiments are explored in the following subsections. Two parameters are varied with several possible values in each experiment, while the other two are kept as fixed. The number of inner iterations and the number of outer iterations are kept constant. The number of trials in each experiment may vary. For example, in the experiment 1, the factors we choose are w_s and w_t, each of which has three levels. The total number of trails is 9.

In the next several section, we detail the experiment set up, visualization for the diagnostics, and the parameter selection based on prediction error.

3.1.1 Experiment I

In the first experiment, we vary the smoothing window of seasonal and trend components. There are three values for w_s, 21, 30, 39. And three values for w_t, 231, 313, 451. The smoothing degree is fixed as linear and quadratic for d_s and d_t respectively. We first illustrate the distribution of prediction error of each station for a given trial of parameter setting conditional on lag distance, which is from 1 to 36. The 9 PDF files are generated as shown in Figure 3.1. The parameter setting for each trial are

trial 1: $w_s = 21$, $w_t = 231$; trial 2: $w_s = 30$, $w_t = 231$; trial 3: $w_s = 39$, $w_t = 231$;
trial 4: $w_s = 21$, $w_t = 313$; trial 5: $w_s = 30$, $w_t = 313$; trial 6: $w_s = 39$, $w_t = 313$;
trial 7: $w_s = 21$, $w_t = 451$; trial 8: $w_s = 30$, $w_t = 451$; trial 9: $w_s = 39$, $w_t = 451$.

Link to trial 1	Link to trial 2	Link to trial 3
Link to trial 4	Link to trial 5	Link to trial 6
Link to trial 7	Link to trial 8	Link to trial 9

Figure 3.1.: The normal quantiles of prediction error conditional on lag for 512 stations with a given group of parameter setting in Experiment 1

Each station is plotted on one page, and the order of the stations for a given trial of parameter set up is decided as following: based on the normal quantile plot of prediction error for a given station across all 36 lag values, the total amount of deviation of the distribution of prediction error from normal distribution is calculated as

$$Dev = \sum_{i=1}^{36} \sum_{j=1}^{271} |r_{ij} - z_{(j-0.5)/271}|^2$$

where r_{ij} is the remainder at lag i replicate j, and $z_{(j-0.5)/271}$ is the $(j-0.5)/271$ quantiles of unit Normal distribution. Then the stations are ordered from the smallest

to the largest in term of deviation. Surprisingly, with a given parameter setting of STL+, the prediction error for all stations with different lag are well-approximated by unit Normal distribution, which guarantee us to use the summary statistics, mean and standard deviation of the prediction error for a given station and a given lag to compare the prediction ability of different groups of smoothing parameters. As we mentioned previously, we sampled 512 stations from the 7,738 stations for the visualization purpose. The location of the 512 stations are shown in Figure 2.9. The index 1 to 512 is helpful to easily identify the stations.

In Figure 3.2, the mean of absolute value of prediction error over 271 replicates for a given station given lag distance is plotted against lag distance conditional on w_s and superposed on w_t with different color. Each page is for one station. Even though 512 stations all come from very different locations, the prediction ability of different parameter setting are surprisingly consistent. Within different w_s values, trend window size with 451 always illustrates a lower prediction error over 36 lag distance.

Link to figure

Figure 3.2.: The mean of abs error vs. lag conditional on w_s

And in Figure 3.3, the standard deviation of prediction error multiplies 1.96 over 271 replicates for a given station given lag distance is plotted against lag distance conditional on w_s and superposed on w_t. Again, trend window size with 451 returns the lowest variance of error collectively. While we do notice that the advantage of window size 451 of trend smoothing, with respect to the variance and mean of the prediction error, is negligible.

Link to figure

Figure 3.3.: The standard deviation of error vs. lag conditional on w_s

Similarly, we plot the mean of absolute value of prediction error and standard deviation of prediction error, respectively, against to lag distance conditional on w_t and superposed on w_s as well.

Link to figure

Figure 3.4.: The mean of abs error vs.lag conditional on w_t

Link to figure

Figure 3.5.: The standard deviation of error vs. lag conditional on w_t

It is noticeable that within three different value of w_t, seasonal window size 39 provides the lowest mean and standard deviation of prediction error based on Figure 3.4 and Figure 3.5. Moreover, three curves with different seasonal windows under given trend window are approximately parallel, and the range of standard deviation and mean of absolute value of prediction error over lag distance is negligible comparing to the value itself, we summarize the graphs above to make the comparison over different group parameter setting more straightforward, which is the overall mean and standard deviation of the prediction error. We define the MAPE to be the mean of absolute value of prediction error, and MSDPE to be the mean of standard deviation of prediction error multiplied by 1.96.

Link to figure

Figure 3.6.: The dotplot of mean of abs error vs. station conditional on w_s

Link to figure

Figure 3.7.: The dotplot of mean of standard deviation of error vs.station conditional on w_s

In Figure 3.6 and Figure 3.7, the mean of absolute prediction error and the mean of standard deviation of prediction error of each station are plotted against the corresponding station respectively, conditional on w_s and superposed on w_t. Both standard deviation and mean of prediction error illustrates that prediction ability is maximized when trend window size is 451. But we also notice that the difference between different trend window size with respect to the prediction error is negligible cross all 512 stations.

Link to figure

Figure 3.8.: The dotplot of mean of abs error vs. station conditional on w_t

Link to figure

Figure 3.9.: The dotplot of mean of standard deviation of error vs.station conditional on w_t

Similarly, in Figure 3.8 and Figure 3.9, we plot the mean of absolute prediction error and the mean of standard deviation of prediction error of each station against the corresponding station conditional on w_s and superposed on w_t. We also notice that even though seasonal window size of 39 provides the best prediction ability with different trend window size, the advantage is very small. This implies we should also enlarge the seasonal window.

3.1.2 Experiment II

In the experiment 2, we still vary the smoothing window for seasonal and trend, but with a larger range for seasonal smoothing window. w_s is now varied among 11, 41, and "periodic". While w_t is valued from 123, 241, and 451. The parameter setting for each trial are

trial 1: $w_s = 11$, $w_t = 123$; trial 2: $w_s = 41$, $w_t = 123$;

trial 3: $w_s =$ "*periodic*", $w_t = 123$;

trial 4: $w_s = 11$, $w_t = 241$; trial 5: $w_s = 41$, $w_t = 241$;

trial 6: $w_s =$ "*periodic*", $w_t = 241$;

trial 7: $w_s = 11$, $w_t = 451$; trial 8: $w_s = 41$, $w_t = 451$;

trial 9: $w_s =$ "*periodic*", $w_t = 451$;

We include "periodic" as one of the value for seasonal smoothing window. "periodic" means for each subseries, local weighted regression is conducted with span equal to infinite and degree equal to 0. Infinite span window is actually same as global weighted regression with equal weight 1 for all points. And we make the rang of trend window to be more variable than experiment 1. Again, the normal quantiles plots of prediction error conditional on lag distance for 512 stations are demonstrated in the following 9 graphs. Each page is about the normal-quantile plot of a given station, which is conditional on the lag distance varied from 1 to 36. The black solid lines represents the theoretical normal distribution.

Link to trial 1	Link to trial 2	Link to trial 3
Link to trial 4	Link to trial 5	Link to trial 6
Link to trial 7	Link to trial 8	Link to trial 9

Figure 3.10.: The normal quantiles of prediction error conditional on lag for 512 stations with a given group of parameter setting in Experiment 2

The stations are ordered from the smallest to the largest in term of deviation from the unit normal distribution. Still, the prediction error for all 512 stations with different lag are well-approximated by unit Normal distribution. As a result, we evaluate the prediction ability of each group of parameter setting based on the mean of absolute value of prediction error and 1.96 times standard deviation of prediction error collectively. However, there is one thing should be pointed out which did not happen in the first experiment. With trial 1 to trial 3 parameter setting, even though the distribution of mean of absolute value of prediction error for each lag distance is approximated by a unit Normal distribution, the variance of the unit Normal is enlarging when the lag distance becomes further. This implies that the prediction error is more unstable when the w_t is relatively small.

Again, the MAPE and SDPE of a given lag distance given station are drawn against to lag distance conditional on the w_s in Figure 3.11 and Figure 3.12. Different values of w_t are superposed in each panel with different colors. No doubt, different than the first experiment, this time it is trivial to notice that the difference between trend window of 123 and the other two value of trend window is significant in all three different values of seasonal window for all stations. But the curves for 313 and 451 of trend window are almost overlapping. These plots imply that the prediction ability decrease when trend window is enlarged until a stable level after trend window is around 241.

Link to figure

Figure 3.11.: The mean of abs error vs. lag conditional on w_s

Link to figure

Figure 3.12.: The standard deviation of error vs. lag conditional on w_s

In Figure 3.13 and Figure 3.14, the prediction ability varies dramatically between different seasonal smoothing window cross all three trend windows for all stations. "periodic" gives the lowest prediction error collectively in all stations. Moreover, we also notice that when trend smoothing window is 123, the mean and standard deviation of prediction error increasing dramatically as lag distance increases. This also confirms our finding for the trend smoothing window, which 123 of w_t gives the worst prediction ability.

Link to figure

Figure 3.13.: The mean of abs error vs.lag conditional on w_t

Link to figure

Figure 3.14.: The standard deviation of error vs. lag conditional on w_t

Finally, we summarize the prediction ability of each trial of smoothing parameters by the overall MAPE and MSDPE, which are the overall mean of absolute prediction error over all prediction error and mean of standard deviation multiplied by 1.96 of a given station respectively. As shown in Figure 3.15 and Figure 3.16, the summary statistics MAPE and MSDPE are plotted against to the corresponding station conditional on different w_t. Different values of w_s are superposed in each panel with different colors. Stations are order based on the MAPE with $w_s = periodic$ and $w_t = 451$. "periodic" smoothing window for seasonal component always provides the lowest prediction error across all stations under all three different values of w_t.

Link to figure

Figure 3.15.: The dotplot of mean of abs error vs. station conditional on w_t

Link to figure

Figure 3.16.: The dotplot of mean of standard deviation of error vs.station conditional on w_t

In Figure 3.17 and Figure 3.18, trend span window equal to 123 collectively has undoubtedly higher mean of absolute value and standard deviation of the prediction error cross all stations. But there is not clearly difference after trend smoothing window larger than 241 with respect to prediction error.

Link to figure

Figure 3.17.: The dotplot of mean of abs error vs. station conditional on w_s

Link to figure

Figure 3.18.: The dotplot of mean of standard deviation of error vs.station conditional on w_s

3.1.3 Experiment III

In the third experiment, we vary the trend smoothing window w_t with 5 different values, 41, 83, 123, 241, 451, which is a much wider range than the first two experiments, and trend degree d_t is varied with 2 values, local linear or local quadratic. Seasonal smoothing window is fixed as "periodic". So totally there are 10 runs in the experiment.

trial 1: $w_t = 41$, $d_t = 1$; trial 2: $w_t = 83$, $d_t = 1$; trial 3: $w_t = 123$, $d_t = 1$;
trial 4: $w_t = 241$, $d_t = 1$; trial 5: $w_t = 451$, $d_t = 1$; trial 6: $w_t = 41$, $d_t = 2$;
trial 7: $w_t = 83$, $d_t = 2$; trial 8: $w_t = 123$, $d_t = 2$; trial 9: $w_t = 241$, $d_t = 2$;
trial 10: $w_t = 451$, $d_t = 2$;

Link to trial 1 Link to trial 2 Link to trial 3

Link to trial 4 Link to trial 5 Link to trial 6

Link to trial 7 Link to trial 8 Link to trial 9

Link to trial 10

Figure 3.19.: The normal quantiles of prediction error conditional on lag for 512 stations with a given group of parameter setting in Experiment 3

The first batch of graphs are still the normal quantile plots of prediction error for each station with different parameter settings. We notice that when d_t is quadratic and w_d is less than 123, the trial 6 to trial 8, the slop of normal quantile plot increases dramatically as lag distance becomes larger. This means when we try to apply quadratic polynomial regression in a relatively small local time period, the prediction for longer lag distance turns to be more unstable.

Next, mean of absolute prediction error (MAPE) and standard deviation error (SDPE) are plotted against lag distance conditional on d_t and superposed on w_t in the Figure 3.20 and Figure 3.21. It is shown that MAPE and SDPE are minimized and stabilized after w_t is larger than 123. Moreover, when d_t is equal to 2, small value of w_t makes the prediction error inflated as lag distance is larger.

Link to figure

Figure 3.20.: The mean of abs error vs. lag conditional on d_t

Link to figure

Figure 3.21.: The standard deviation of error vs. lag conditional on d_t

Figure 3.22 and Figure 3.23 illustrate the MAPE and SDPE against lag distance conditional on w_t and superposed on d_t for each of 512 stations. It is noticeable that prediction error inflates dramatically as lag distance goes up when the w_t is 41 and d_t is 1. As w_t is larger than 123, there is not significant difference between the local linear or quadratic fit for trend component.

Link to figure

Figure 3.22.: The mean of abs error vs. lag conditional on w_t

Link to figure

Figure 3.23.: The standard deviation of error vs. lag conditional on w_t

We summarize the results in the third experiment by the dotplot of overall mean of absolute prediction error (MAPE) and mean of standard deviation of prediction error (MSDPE) against station conditional on w_t (d_t) and superposed on d_t (w_t). As dedicated in Figure 3.24 and Figure 3.25, conditional on d_t value, the prediction error is minimized at w_t of value 241 collectively.

Link to figure

Figure 3.24.: The dotplot of mean of abs error vs. station conditional on d_t

Link to figure

Figure 3.25.: The dotplot of mean of standard deviation of error vs.station conditional on d_t

Figure 3.26 and Figure 3.27 demonstrate that local linear fit for trend component always provide the lowest prediction error, especially for small value of w_t. When

the smoothing window of trend component is greater or equal to 241, the difference between local linear and local quadratic in term of prediction error is negligible.

Link to figure

Figure 3.26.: The dotplot of mean of abs error vs. station conditional on w_t

Link to figure

Figure 3.27.: The dotplot of mean of standard deviation of error vs.station conditional on w_t

3.1.4 Experiment IV

In the fourth experiment, which is the last run, we vary the smoothing window and polynomial degree for seasonal component w_s and d_s only, and fix the parameter for the trend component based on the result of previous experiment, which are $w_t = 241$ and $d_t = 1$. The w_s is varied among 11, 21, 31, 47, and "periodic", and d_s is varied between 1 and 2. Totally there are 9 runs in the experiment, since there is only one run for w_s is "periodic".

trial 1: $w_s = 11$, $d_s = 1$; trial 2: $w_s = 21$, $d_s = 1$; trial 3: $w_s = 31$, $d_s = 1$;

trial 4: $w_s = 47$, $d_s = 1$; trial 5: $w_s =$ "*periodic*", $d_s = 1$;

trial 6: $w_s = 11$, $d_s = 2$; trial 7: $w_s = 21$, $d_s = 2$; trial 8: $w_s = 31$, $d_s = 2$;

trial 9: $w_s = 47$, $d_s = 2$;

Link to trial 1	Link to trial 2	Link to trial 3
Link to trial 4	Link to trial 5	Link to trial 6
Link to trial 7	Link to trial 8	Link to trial 9

Figure 3.28.: The normal quantiles of prediction error conditional on lag for 512 stations with a given group of parameter setting in Experiment 4

Collectively, the prediction error is well approximated by an unit Normal distribution for given lag distance of each station with a given group of parameter setting. However, in trial 4, which w_s is 11 and d_s is 2, the variance of the prediction error for lag distance farther than 12 becomes much larger than that for shorter lag distance. Prediction is more unstable for longer lag distance under this group of parameter setting. Similar situation can also be found in trial 7, which also has relatively small w_s but quadratic local fit for the seasonal component.

Then the mean of the absolute value of prediction error over all lag distance for each station is plotted against to the lag distance on each page as in Figure 3.29. Clearly, with smaller value of w_s the MAPE with shorter lag distance is consistently much smaller than it with further lag distance. Moreover, this difference is amplified when d_s is equal to 2. For example, in the first panel on the left hand side of each page, the MAPE increases drastically from the first year to the second year and then the third year when d_s is 2. And when the w_s is raised to a higher value, the MAPE becomes more stable over the value of lag distance until it is minimized as w_s becomes "periodic".

Link to figure

Figure 3.29.: The mean of abs error vs. lag conditional on w_s

Same phenomena can be found in the Figure 3.30, which the standard deviation of prediction error multiplied by 1.96 is plotted against to the value of lag distance. w_s of "periodic" gives the lowest and the most stable standard deviation over all different setting of w_s and d_s over all 512 stations collectively.

Link to figure

Figure 3.30.: The standard deviation of error vs. lag conditional on w_s

If we switch the conditional variable from w_s to d_s, it is undoubted the "periodic" seasonal component will be the best choice with respect to the prediction error. As shown in Figure 3.31 and Figure 3.32, there is a dramatically difference between the "periodic" setting and smaller value of w_s for both MAPE and SDPE. Meanwhile, MAPE and SDPE corresponding to small w_s does not increased smoothly as the lag distance increases. There surely is a jump every 12 months of lag distance. So the prediction is not stable over longer lag distance with smaller value of w_s compared with the "periodic" setting.

Link to figure

Figure 3.31.: The mean of abs error vs. lag conditional on d_s

Link to figure

Figure 3.32.: The standard deviation of error vs. lag conditional on d_s

Finally, we summarize the prediction error for each group of parameter setting by the overall mean of absolute value of prediction error and the overall mean of standard deviation of prediction error multiplied by 1.96. In the Figure 3.33 and Figure 3.34, The overall MAPE and MSDPE respectively are plotted against to the corresponding station in order conditional on w_s and superposed on d_s. Apparently, within a given value of w_s, local linear fit consistently provides a lower value of overall MAPE and MSDPE over all 512 stations. And the superiority of local linear fit is maximized when w_s is 11.

Link to figure

Figure 3.33.: The dotplot of mean of abs error vs. station conditional on w_s

Link to figure

Figure 3.34.: The dotplot of mean of standard deviation of error vs.station conditional on w_s

Similar conclusion can be also made if we switch the conditional variable from w_s to d_s, and overall MAPE and MSDPE is plotted against to the corresponding station superposed on w_s, as in figure 3.35 and figure 3.36. "periodic" seasonal component overcome other values of w_s in both local linear and quadratic fit.

Link to figure

Figure 3.35.: The dotplot of mean of abs error vs. station conditional on d_s

Link to figure

Figure 3.36.: The dotplot of mean of standard deviation of error vs.station conditional on d_s

In summary, we vary two of w_s, w_t, d_s, and d_t four factors each time to find the best prediction STL model with respect to the prediction error. The best smoothing parameters for the prediction is "periodic" seasonal smoothing and local linear fit with smoothing window of 241 for trend component.

3.2 Tunning Smoothing Parameters of Spatial Loess

In the spatial loess fitting, there are two smoothing parameters, span and degree, which are used to control the number of neighbors and the complexity of each local model. But just as most of nonparametric method, there is no rule of thumb to determinate the choice of these smoothing parameters. Consequently, we consider the leave-p-out cross validation method to find the best model based on the mean squared error. Specifically, 128 stations are randomly sampled as testing dataset from the 7,738 stations of each month. In order to guarantee these 128 stations are not clustering together, we sample them from a kd-tree built based on all stations with 128 cells, one station from each cell. Then the remainders at 128 stations of given month are predicted using the rest of training stations, and prediction error of each

128 stations are calculated. This procedure is repeated until the prediction error is calculated at all stations,

3.2.1 Experiment V

In the second step of SSTL routine, the spatial loess smoothing is applied to the original observation in each month separately. The two main purpose of this smoothing procedure are infilling missing values and smoothing out spatial noise. The best smoothing parameters are chosen based on the mean squared error (MSE). Concretely, suppose each month has N_m valid observations, m varies from 1 to 576. N_m should be some integers around 5,000 as we have seen in section 2.3.2. Leave-128-out cross validation is conducted until all prediction error are calculated at N_m stations in each month. The mean squared error as the criteria of prediction ability is calculated as

$$MSE_m = \sum_{l=1}^{N_m} (y_{ml} - \hat{y}_{ml})^2 \tag{3.2}$$

The distribution of the MSE over 576 months is evaluated for different smoothing parameter settings. A full factorial experiment is considered for the smoothing span and degree two factors. We set the smoothing span to be varied among 0.005, 0.015, 0.035, and 0.095. Smoothing degree for longitude and latitude are fixed as quadratic, but the degree for elevation is varied among 0, 1, and 2. So totally there are 12 trials in the experiment.

Link to figure

Figure 3.37.: The distribution of MSE conditional on degree

In Figure 3.37, monthly MSE are plotted against to their corresponding f-value conditional on degree of elevation. Different values of smoothing span are superposed

in four different colors in each panel. Clearly, for the spatial smoothing without elevation, span equal to 0.005 provides the lowest MSE. And MSE becomes larger as span increases. However, when elevation is included in the local conditional parametric models, span of 0.005 is not optimal anymore. Especially when the degree of elevation is quadratic, the span 0.005 is facing a severe high-variance problem. For both degree 1 and 2 of elevation, span 0.015 minimize the MSE collectively.

Link to figure

Figure 3.38.: The distribution of MSE conditional on span

Meanwhile, the monthly MSE are plotted against to their corresponding f-value conditional on the smoothing span factor in Figure 3.38. Quadratic elevation included in the smoothing model always minimizes the MSE for all span values except 0.005 in which degree 1 is the optimal setting. In 0.005, local regression model with elevation degree 2 is suffering serious high-variance issue, and elevation degree 1 provides the optimal prediction ability. However, over all the degree and span values, the spatial smoothing model with degree 2, span 0.015 is the best prediction parameters.

3.2.2 Experiment VI

and the mean squared error (MSE) of given month is calculated as following

$$MSE_m = \sum_{l=1}^{7738} \left(y_{ml} - \hat{s}_{ml} - \hat{t}_{ml} - \hat{g}_{ml}\right)^2 \tag{3.3}$$

where m is from 1 to 576, y_{ml} is the maximum temperature at month m station l. $\hat{s}_{ml}$, $\hat{t}_{ml}$ are the estimated seasonal and trend components respectively. g_{ml} is the spatially smoothed remainder value. The MSE for each month is calculated in parallel.

span = c(0.005, 0.015, 0.05, 0.15), degree = c(1, 2). We conduct a experiment with 8 different smoothing parameters of the spatial loess model which are the combination

of 4 different values of span and 2 values of degree. Then 8 MapReduce jobs are executed sequentially, one for each group pf smoothing parameter, and MSE of each month are calculated as we described above.

In Figure 3.39, the quantiles of MSE are plotted against to the corresponding f-value conditional on the span value. And within each panel, local linear and local quadratic fitting are differentiated by different color of points. In the panel of span is equal to 0.005, local linear fitting has lower value of MSE than local quadratic. This is a reasonable result because with span equal to 0.005, very limited number of neighbors were used in the local fitting. Quadratic local model will make the high variance issue even more serious. However, besides 0.005, local quadratic fitting collectively gives a better prediction results than local linear fitting under all other span values.

Link to figure

Figure 3.39.: The distribution of MSE conditional on span

On the other hand, the quantiles of MSE are plotted against to the corresponding f-value conditional on the degree superposed on span value in Figure 3.40. In both linear and quadratic local fitting, model with span equal to 0.015 provides lowest prediction error compared with other span values.

Link to figure

Figure 3.40.: The distribution of MSE conditional on degree

3.3 Diagnostics of Residuals

3.3.1 Final Residuals

In Figure 3.41, sequence of quantiles from 0.005 quantiles to 0.995 quantiles increment by 0.005 are plotted against to their corresponding f-value. It shows that, 99% of residuals are symmetrically distributed within ±2, and are centering at 0 as well. 90% of residuals, which are 1,887,140, are distributed within ±1.

Link to figure

Figure 3.41.: The distribution of residual

Moreover, a quantile-quantile plot of residual is shown in Figure 3.42. The 199 quantiles from 0.005 to 0.995 are drawn against to the corresponding quantiles from a t-distribution with degree of freedom 4. Clearly, The linearity of the points suggests that the data are linearly related to a t-distribution centered at 0 with degree of freedom 4.

Link to figure

Figure 3.42.: The quantile-quantile plot of residual

3.3.2 Outliers

Outliers or extreme weather temperature is a very interesting problem which is highly concerned by people. In our analysis, we identify the residuals outside the range of ±2 as outliers of our fitting, which is defined by the 0.005 and 0.995 quantiles.

First we calculate the number of outliers in each month and plot it against the date in Figure 3.43. Also a loess smoothing curve with degree equal to 1 and 0.2 span is superposed on the plot as a red curve.

Link to figure

Figure 3.43.: The number of outliers vs. date

As we can see in Figure 3.43, the number of outliers appreciably decreased between January 1950 to January 1958. Then it keeps as flat mostly less than 50 after January 1950 till January 1980. After that, there is an obvious hump between January 1982 to December 1997. The largest number of outliers happens on December 1985, which has 183 outliers. We also notice that there is yearly seasonality in the number of outliers as well. The next plot we plot the number of outliers in each of 12 month-in-year.

Link to figure

Figure 3.44.: The number of outliers in month of year

Figure 3.44 illustrates that months in winter, such as January, December, February have more outliers than other months. On the other hand, the number of outliers is much smaller in spring and summer than that in winter. However, this may be caused by several months since we are only consider the overall count of outliers in each month-in-year. In the next Figure 3.45, the distribution of the number of outliers in each of 576 months is visualized superposed on different groups of month, winter or non-winter month. Winter includes December, January, and February.

Link to figure

Figure 3.45.: The distribution of number of outliers in each month

The log base 2 number of outliers in each month is plotted against to their corresponding f-value in the Figure 3.45. The magenta points represent the distribution of the count of outliers in the winter months, and the blue points represent the distribution of all other months. Clearly, the number of outliers in all winter months is larger than that in other months in distribution. There is a shift between two distribution with no doubt.

Link to figure

Figure 3.46.: The distribution of number of outliers in each station

Figure 3.45 illustrates the distribution of the log base 2 number of outliers in each station. We can tell that around 50% of stations do not have any outliers, 95% of stations only have less than 8 outliers, which is only about 2% number of observations in the time series of each station.

4. SSTL UNDER DIVIDE AND RECOMBINED FRAMEWORK FOR BIG AND COMPLEX DATA

In this chapter, I discuss the generalization of SSTL routine under the divide and recombined framework for big and complex spatial-temporal dataset. Because of the flexibility of the SSTL for large and computationally heavy dataset, it comes free to embed the SSTL method into the divide and recombined framework, called drSSTL. Either by-month division or by-station division of the dataset can be generated, then a corresponding analysis method is applied on them. The drSSTL routine now consists of a series of MapReduce jobs, and all fitting results are saved on HDFS. Meanwhile some of the diagnostics procedures discussed in the third chapter are illustrated here but with more details about their parallel implementation. In the first section, I illustrate the MapReduce job to download the target dataset in parallel. Next I explore the details about the each steps of drSSTL as a MapReduce job to proceed the smoothing procedure on different divisions.

4.1 Data Download

Downloading process can be accelerated under the divide and recombined framework. When dataset is extremely large, it is usually saved as multiple files; one for each year or for each location. Files are downloaded in sequence in general. However, under the divide and recombined frame work, the process can be achieved in parallel, which is much faster and more efficient than in sequence. Meanwhile, the dataset is directly saved on HDFS and ready for analysis. By calling functions from the RHIPE [20] packages, a MapReduce job is implemented to download several files in each mapper simultaneously. For example, the following MapReduce job parallely downloads 103 data files from IMAGe database, and save them to the HDFS once

the job is finished. Within the entire thesis, map task and mapper can be treated as equivalent. Same for the reduce task and reducer.

Input Integers from 1 to 103. There is no real input files fed into this MapReduce job. Instead, we simulate the key of input key-value pairs to be sequence of integer from 1 to 103, which is exactly the number of data set files. Each of them is passed to one Mapper.

Output The 103 text files saved on HDFS.

Map Download one file in each mapper. For each Mapper, the download link is generated by concatenating the string `https://www.image.ucar.edu/pub/nychka/NCAR_tinfill/tmax.complete.Y` with the integer passed into this Mapper. The corresponding data file is downloaded into a temporal directory on the local file system of the server on which the Mapper is running, and then this local copy of the file is automatically copied onto HDFS after Map is finished.

Reduce Reduce is not needed in this MapReduce job.

The total raw text files saved on HDFS are around 90MB. There are 836,875 lines in text files collectively. Each line of the raw text file corresponded to twelve monthly maximum temperature observations for a given station in a given year, followed by their status code of observation (observed or missing) in terms of 0/1. These text files will be read in as inputs to the next MapReduce jobs to continue the SSTL routine. Since Hadoop is good at handling small number of large files compared with large number of small files [38], it will be more efficient to merge those files into one big text file as the inputs to the later jobs instead of 103 separate files.

4.2 Modeling Routine

4.2.1 Step I: Read in

By-month division of the original dataset is generated in the first step using raw text file as input. The observations are divided into one key-value pair per month. The key is the month index, and the value are observations at all stations in the corresponding month. In the following paragraph we describe the MapReduce job used to create such division.

Input The merged text data file on HDFS. Each row is read in as a input key-value pair. The key is a unique row index, and the value is the corresponding string in that row.

Output By-month division, in the form of 576 key-value pairs. The keys are the month index, and the corresponding value is a R data.frame containing observations from 7,738 stations in the corresponding month.

Map Every block of raw text dataset, in size of 128 MB, is sent to one mapper with each row as one key-value pair. R function `strsplit` is called to split each filed of the row string. An intermediate key-value pair is generated from each input key-value pair. The key is the month index , and the value is a one-row data.frame with `station.id`, `tmax` two columns.

Reduce 7,738 of one-row data.frame corresponding to one month are aggregated to be one data.frame. Specifically, all one-row data.frame who share the same month index are shuffled and transferred to one reducer, and then R function `rbind` is called in that reducer to combine those one-row data.frames.

Instead of including all spatial predictors, longitude, latitude, and elevation, for each station in the outputs of the Step I, a separate RData file is saved on HDFS, which contains a data.frame including longitude, latitude, and elevation of all 7,738 stations. The benefit of saving spatial information separately is to eliminate the

redundant copies of same information in every key-value pair. The RData file on the HDFS is copied into every mapper of the next MapReduce job as a shared object.

4.2.2 Step II: Spatial Smoothing of Original Observation

By taking the by-month division as input, the second MapReduce job applies spatial smoothing fitting to each month independently. Meanwhile, the RData file which contains meta-data of spatial information of all 7,738 stations are copied to the R session of each mapper. The information is combined with each of input value and passed into the `spaloess` function. After the spatial fitting, those spatial predictors are dropped from the outputs to eliminate the size of outputs.

Input 576 key-value pairs of by-month division. Key is the month index; value is the data.frame object with 7,738 number of rows, each of which is observation for one station.

Output 576 key-value pairs. Key is same as input, and value is a vector of spatial smoothed values of all stations for the corresponding month.

Map The shared RData file containing all location information is first copied and load into the global environment of the R session in each mapper. The input values are merged with the data.frame of station information separately. In each mapper, `spaloess` function from package `Spaloess` is called to calculate the spatial smoothed value for all stations. After the fitting, all spatial information of each location is dropped. The key of each intermediate key-value pair is kept as month index, but value is updated to be a vector of spatial smoothing value in the same order of stations as in the shared RData file.

Reduce Reduce is not needed in this MapReduce job. The intermediate key-value pairs are directly written to HDFS after the Map.

In order to limit the size of the outputs, the spatial fitting results are vectorized before saved on HDFS. Vector also benefits the computation in Map of the next

MapReduce job, since looping over each element of a vector is much faster than looping over each row of a data.frame in general.

4.2.3 Step III: Swapping to By-location Division

The third MapReduce job is served as a swapping process, from by-month division to by-station division. After the MapReduce job in Step II, the spatial smoothed values are still saved as by-month division on HDFS. In order to proceed the temporal smoothing in each station, we need to generate a by-station division. Fortunately, by using RHIPE, which integrates the R and Hadoop, the division by-station can be handily generated from the by-month division. The details about Step III are illustrated in the following MapReduce job.

Input 576 key-value pairs of by-month division. The key is the month index `date`, and corresponding value is a vector with length 7,738, which contains the spatial smoothed values.

Output Division by-station in form of 7,738 key-value pairs, with `station.id` as the key, and corresponding value is a R data.frame containing 576 rows and 2 columns, which are month index and spatial smoothed values.

Map An intermediate key-value pair is generated from each element of each input key-value pair. Totally, there are 576×7738 intermediate key-value pairs generated after the Map. The keys are changed to be the `station.id`, and the corresponding value is a one-row data.frame with spatial smoothed value and the corresponding month index `date`.

Reduce All intermediate key-value pairs that belong to the same station are sorted, shuffled, and sent to one reducer. In the Reduce, the 576 one-row data.frames are rbind accumulatively to be one data.frame with two columns, `spaofit` and `date`.

4.2.4 Step IV: Temporal Fitting in Parallel

The fourth MapReduce job in the routine is mainly focus on the temporal fitting on the time series at each location. This job is similar with the Step II, which proceeds the spatial loess smoothing on the original observations in each month. Only Map stage is necessary, shuffle and sort stage and even reduce stage should be avoid. The outputs of Map are directly wrote to HDFS.

Input 7,738 key-value pairs from the by-station division generated in section 4.2.3. 7,738 unique `station.id` are the keys, and a R data.frame with 576 rows and 2 columns as the corresponding value.

Output 7,738 key-value pairs as outputs. Keys are kept as same as inputs, but each value is a vectorized data.frame with 4 columns: `date`, `spaofit`, `seasonal`, and `trend`.

Map For each of input key-value pairs, we pass into the `stlplus` function the `spaofit` and `date` columns in the value, as well as a group of predefined smoothing parameter, which are `s.window`, `s.degree`, `t.window`, `t.degree`, `inner`, and `outer`. The result of the `stlplus` function is a new data.frame which includes `seasonal` and `trend` two columns. After being column-combined with the input value, the new data.frame is vectorized and saved with the corresponding key as the intermediate key-value pairs.

Reduce No Reduce step is needed. The intermediate key-value pairs are directly written to HDFS.

Similar with Step II, a vectorized data.frame is saved as the value of each key-value pair. The purpose is to accelerate the loop in each mapper of the next MapReduce job, since querying elements from vector is much faster than querying elements by row from data.frame in R.

4.2.5 Step V: Swapping to By-month Division

Once the temporal fitting is done, the by-station division with all temporal fitted values is read into a new MapReduce job in Step V to generate the second by-month division. The MapReduce job in Step V is detailed as following:

Input $7,738$ key-value pairs. Key is the station index, and the corresponding value is a vector with length 576×4, which includes temporal fitting results. The first 576 elements are 1 to 576 which represents the `date` variable; the second 576 elements are the `spaofit` variable; the third 576 elements are the `seasonal` variable ; and the last 576 elements are the `trend` variable.

Output 576 output key-value pairs are generated. The key is changed to month index, the corresponding value is updated to a data.frame of $7,738$ rows and 4 columns which are `trend`, `seasonal`, `station.id`, and spatial smoothed value `spaofit`.

Map For each input key-value pair, 576 intermediate key-value pairs are produced. As the looping index i varies from 1 to 576, the i'th, $(i+576)$'th, $(i+1152)$'th, and $(i+1728)$'th elements of the input value vector are pulled out to form a one-row data.frame with 4 columns. Totally $4,457,088$ intermediate key-value pairs are generated and copied to the local disks after the Map is finished.

Reduce Intermediate key-value pairs that share the same month index are shuffled, sorted, and sent to one reducer, then are merged together by calling `rbind` function. The final by-month division is saved on HDFS.

The MapReduce job in Step V is similar with the job in Step III, since both of them generates $4,457,088$ intermediate key-value pairs, which is the number of observations in the whole dataset. However, they differs in the way of generating them and the elapsed time of two jobs. In general, no matter which MapReduce job it is, each mapper is always fed by several input key-value pairs. Each mapper from

Step V is looping over 576 elements from each input key-value pairs. But each mapper from Step III is looping over 7,738 elements from each input key-value pairs. A large number of observations either in one month or at one station will dramatically explode the overhead caused by `rhcollect` function called in the mappers. Consequently, the elapsed time of these two jobs can be quite different depending on the number of observations.

4.2.6 Step VI: Spatial Fitting of Remainder

The sixth step of the routine is the spatial fitting of the remainder component from the STL+ fitting. The MapReduce job reads in the by-month division with all STL+ fitted values, and applies spatial smoothing procedure on the remainders in each month separately. Meanwhile, the RData file which contains the spatial predictors for all stations is copied to each mapper of the job as a shared R object as well.

Input 576 key-value pairs are read in; keys are the month index, and the corresponding value is a data.frame with dimension $7,738$ by 4, which includes temporal fitted values.

Output 576 output key-value pairs are generated; the keys are kept as month index, but the corresponding value is updated to a data.frame of $7,738$ rows and five columns, which includes a new column of spatial fitted value of remainder `Rspa`.

Map The shared RData file containing all location information is copied load into the global environment of the R session in each mapper. The input value is merged with spatial predictors by location index, then is passed into the `spaloess` function. After the spatial smoothing is finished, the location information with longitude, latitude and elevation are removed from the intermediate values in order to illuminate the size of the data written to local disk.

Reduce Reduce is not needed in this job. The outputs of each mapper are directly written to HDFS as by-month division.

The outputs of Step VI is one of the two final output files of the SSTL routine, containing all fitted values. Another output file includes the same fitted values but in the form of by-station division.

4.2.7 Step VII: By-location Division of Final Results

The final step of the entire SSTL routine is a MapReduce job to generate by-station division with all estimated components from the previous by-month division. It is a job with Map, shuffle and sorting, and Reduce stage.

Input 576 key-value pairs of by month-division; keys are the month index varying from 1 to 576, and corresponding value is a data.frame with dimension $7,738$ by 4, which includes two temporal fitting components, the spatial smoothed value of original observations, and the spatial smoothed value of the remainder.

Output $7,738$ output key-value pairs are generated. The keys are changed to be the location index, the corresponding value is restructured to be a data.frame of 576 rows and 5 columns which are trend, seasonal, location index, spatial smoothed value of original observation, and spatial smoothed value of remainder.

Map Each input value, a data.frame with dimension 7,738 by 4, is vectorized as a numeric vector with length 7738×4. A one-row data.frame with 4 columns is generated based on the i'th, $(i + 7738)$'th, $(i + 15476)$'th, and $(i + 23214)$'th elements of the numeric vector. As i varies from 1 to 7,738, 7,738 intermediate key-value pairs are produced from each input key-value pair.

Reduce Intermediate key-value pairs who share the same location index are shuffled and sent to one reducer, and then are merged together by `rbind` function.

In summary, there are 7 steps in the SSTL routine under the divide and recombined framework. Compared with the original SSTL routine, there are three extra steps. Even with the same raw dataset, the advantage of drSSTL is dramatical, which

shrink the elapsed time of finishing the whole routine from 35 minutes to about 10 minutes. These three steps are served for the purpose of generating different divisions of the whole dataset in the form of key-value pairs. They are quite different than the rest of MapReduce jobs which carries out the smoothing procedures in parallel. Shuffle and sorting stage heavily involve and play a critical role in these three MapReduce jobs, which swap one type of division to another. Several Hadoop parameters can be tuned wisely to improve the performance of these jobs in terms of elapsed time. Chapter five provide more details about the experiment of tuning Hadoop parameters.

4.3 Generating Visual Display in Parallel

As we discussed in section 2.4.2, the smoothing results are evaluated heavily based on those diagnostic plots, such as Figure 2.17 and Figure 2.18. When the dataset becomes larger and larger, the number of pages and the size of these plots can be enormous, therefore the elapsed time to generate them becomes unacceptable. Procedure of generating large plots in parallel is considered and implemented to reduce the elapsed time of producing large plots.

Suppose we want to visualize the spatial surface of the final fitted value for each month. Since the temperature surface is in three-dimension, a level plot for each month should be considered. But the restriction on the `levelplot` function requires that fitted values has to be evaluated on a rectangular grid on the US map. Consequently, the fitted values are predicted at 20,389 new locations whose longitude and latitude are varied by 0.2 degree respectively. The prediction procedure is straightforward. The new locations with their value spatial predictors are passed into the Step II of the SSTL routine to calculate the spatial smoothed values with original observations at 7,738 stations. Then we just follow the routine described in section 4.2 to get the final prediction at new locations in the form of by-location division and by-month division respectively. Taking the by-month division of the prediction re-

sults, the following MapReduce job generates the temperature surface in each month in parallel.

Input The by-month division with 576 key-value pairs. Keys are month index, and the corresponding value is a data.frame object with 20,389 rows and 4 columns, which includes the final fitted values.

Output Still 576 key-value pairs; Keys are same as input, but each value is updated to be a serialized `trellis` object.

Map For every input key-value pair, `levelplot` function from the `lattice` package is applied on the value, and then `serialize` function is applied to the `trellis` object return by `levelplot` function in order to be able to save it as the value of intermediate key-value pairs, since saving `trellis` object directly as the value is forbidden.

Reduce Reduce is not needed in this MapReduce job. Intermediate key-value pairs are directly save to HDFS.

The prediction results is plotted in Figure 4.1. The fitted temperature over the entire US is a smoothed surface. Red represents high temperature, and blue represents low temperature. Collectively, the Rockey mountains region has lower maximum temperature than the neighborhood regions. South California, South Florida, and the most area of Texas have the maximum temperature over all months.

Link to figure

Figure 4.1.: The level plot of fitted value in each month

Most of visual displays we shown in the chapter two can also be generated in the similar way, which creates each page of the display in one mapper. After the the job

is done, we only need to read back those serialized `trellis` objects and print them in a given order such as by month or by station.

4.4 Generating Database for Diagnostic Experiment

In the section 3.1 and 3.2, we demonstrate a procedure to choose the best smoothing parameters for temporal fitting and spatial smoothing respectively. They are both cross-validation style, which use partial data as training dataset to predict the testing dataset. Once the dataset gets larger, the computation for the cross-validation will become dramatically heavy. In this section, we introduce the implementation of those cross-validation procedures under the divide and recombined framework. Still, the subset of maximum temperature after year 1950 is used as example to illustrate the details.

4.4.1 Experiment of Temporal Prediction

If we conduct the experiment in 3.1 in series, we have to run $271 \times 7738 = 2,096,998$ number of STL fitting for all stations, which is extremely computational inefficient. Fortunately, because of RHIPE (R and Hadoop Integrated Programming Environment), we can easily handle this experiment in parallel. Based on the division by-station created in section 4.2.2, a map function which executes 271 STL fittings is applied to each of 7,738 stations. However, this is still computationally heavy for each mapper since all 271 STL fittings have to be sequentially executed in one mapper within either a `for` loop or `lapply` function. We have to consider other more efficient parallel procedure for the experiment.

There are two ways to proceed the parallel procedure, which are much litter for each mapper with respect to the computation. One is to create a share R object and save on HDFS beforehand. It is a data.frame containing all 576 observations of all 7,738 stations ordered by station ID and month index. Then in the MapReduce job, we simulate the key of input key-value pairs to be a sequence of integer from

1 to 2,096,998. Each of them represents the row index of the first observation of the training data from the original full dataset. The corresponding value is the same integer as the key. For instance, if input key-value pair is `(3001,3001)`, it means the corresponding mapper applied to this key-value pair will use observations from August 1951 to Jan 1974 (20th observation to 289th observation inclusively) as training dataset to predict the next 36 observations from Feb 1974 to Jan 1977 for the 12th station. But this method still requires each mapper to load the same shared R data.frame into the memory of the server on which it is running. Since each server of the cluster is assigned with multiple mappers, the shared R object is actually loaded into the memory with multiple copies.

A more computationally efficient way to conduct the tunning experiment is actually based on a new databased on HDFS. Each of key-value pairs from the by-station division is split into multiple key-value pairs in the map function, which called FlatMap function. Specifically, the map function reads in one key-value pair, and generates 271 key-value pairs. The keys are the vector of station ID and replicates index i varies from 1 to 271, and the corresponding value is the data.frame contains the training dataset. Totally, there are $2,096,998$ new key-value pairs saved on HDFS as the database for the experiments.

In the following, we detail the procedure of generating experiment database in a MapReduce job.

Input 7,738 key-value pairs from the by-station division, with `station.id` as the key, and a data.frame including 576 observations as the value.

Output experiment database in form of 2,096,998 key-value pairs, with a vector of `station.id` and month index `i` of starting date of training data as the key, and a data.frame containing 270 training data spatial smoothing value and 36 months of testing dataset as the value.

Map For each input key-value pairs, we create 271 intermediate key-value pairs. We loop over month index `i` from 1 to 271, which represents the starting month of

the training data of a given station. The key of intermediate key-value pairs is vector of `station.id` and `i`, the value contains 270 training data (starting from `i`th month) and 36 testing data, which is a R data.frame with 307 rows and 7 columns.

Reduce Identity reduce function do not do anything to the intermediate key-value pairs besides evenly distributes all intermediate key-value pairs to multiple files, and save them on HDFS.

Compared with other implementations of the experiment, using the above database as input for the experiment relieves the workload for each mapper to be only several runs of 271 replicates for each station, and it only load into memory one copy of necessary training dataset. Next, we demonstrate the MapReduce job that reads in above database from HDFS and process the STL+ fitting on each key-value pairs. Within each experiment, a given number of trials of parameters setting is compared. For instance, if 9 sets of smoothing parameters are compared, the following MapReduce job will be executed 9 times, one for each set of smoothing parameters.

Input 2,096,998 key-value pairs are read in from experiment database on HDFS. Each key is a vector of `station.id` and month index `i` of starting date of training data, and the value is a data.frame including training data of 270 spatial smoothing value and testing data of the spatial smoothing value of the oncoming 36 months.

Output Totally there are 7,738 key-value pairs. Each key is vector of `station.id` and `group` which is the index of smoothing parameters set; the corresponding value is a R data.frame with $36 \times 271 = 9,756$ rows and 7 columns, which are (1) `date`, the month index, (2) `seasonal`, (3)`trend`, (4) `remainder`, (5) `tmax`, (6) `lag`, prediction lag distance from 1 to 36, and (7) `rep`, the replicates index of STL+ fitting, values from 1 to 271.

Map For each input key-value pairs, 270 training observations are first extended 270 with 36 `NA`s. And then the time series of 306 observations is fed into `stlplus` function with pre-defined set of smoothing parameters. Lastly a data.frame with 36 rows and 7 columns is generated as the intermediate value, which includes the raw observations of the testing data, as well as the predicted `trend` and `seasonal` components of them.

Reduce No reduce step is needed, the intermediate key-value pairs are generated from map step and saved on HDFS directly.

The comparison of different sets of smoothing parameters based on prediction error can be achieved by the following MapReduce job, which reads in the results from all 9 of previous MapReduce jobs, and calculates the mean of absolute error over 271 replicates of a given lag and given station.

Input 9 sequence files generated by 9 of previous MapReduce job, which includes $7,738 \times 9 = 69,642$ key-value pairs in total with vector of `station.id` and `group` as the key. The corresponding value is a data.frame including 36×271 STL+ fitted value of seasonal and trend components for a given station given group of parameters setting.

Output $7,738$ output key-value pairs are generated, one for each station. Keys are the `station.id`, and the value is a data.frame including the mean of absolute error, the `lag`, and the `group`.

Map For each input key-value pair, the key is changed from the vector of `station.id` and `group` to be just `station.id`. The intermediate value is a data.frame with 36 rows and 3 columns, `m.abser` the mean of absolute error over 271 replicates for each lag and each station; `std.abser`, 1.96 times one standard deviation of absolute error; and the `lag`.

Reduce All intermediate key-value pairs who share the same `station.id` are shuffled and sent to one reducer, and then are combined by row accumulatively. Final key-value pairs are saved on HDFS.

The overall summary statistics for each station can be produced in the similar way. Only the keys of intermediate outputs needs to be modified to be a unique integer, all summary statistics of each station are grouped into one reducer, and then saved on HDFS. The final output of data.frame can be read back into the R session on the front end of the cluster, and the dot-plots we showed in 3.1 can be drawn based on it.

4.4.2 Cross Validation for Spatial Smoothing

The leave-128-out cross validation for each month in section 3.2 can also be carried out in parallel using a MapReduce job. Apparently, this requires each mapper to execute the leave-128-out cross validation in a for loop. Instead, we make the computation to be much litter for each mapper by generate a new experiment database.

Input 576 input key-value pairs, keys are the month index, and the corresponding value is the data.frame including 7,738 rows of `tmax`, `trend`, `seasonal`, `remainder`, and station information.

Output 35,136 key-value pairs are generated as output, whose keys are a combination of month index and index from 1 to 61 of the observation in each cell of the kd-tree. And the corresponding value is same as the input value but with a new column `flag` with value of 0 or 1, which identifies 128 stations for prediction.

Map A FlatMap function is defined in which 61 intermediate key-value pairs are generated from each input key-value pair. For each input key-value pair, a kd-tree with 128 cells is created based on the location of 7,738 stations. Stations in each cell are assigned randomly with a index of 1 to 61. By looping over the index i from 1 to 61, an intermediate key-value pair is produced. Each key is

vector of month index and i; and each value is a copy of input value but with an added column `flag`, which is all 0 except 128 stations with the same index i as in the corresponding key. Consequently, there are 35,136 intermediate key-value pairs are generated in total.

Reduce No Reduce step is needed. All intermediate key-value pairs are saved directly on HDFS right after Map step.

Next, the database above is read into another MapReduce job to proceed the cross validation for the spatial smoothing in parallel.

Input database of cross validation with 35,136 key-value pairs are read in.

Output 576 output key-value pairs are generated. The key is the month index; the value is the mean squared error (MSE) for the corresponding month.

Map For each input key-value pair, spatial smoothed values are calculated at the 128 locations. `span`, `degree` smoothing parameters are pre-defined and sent to each mapper. Next, the summation of squared error based on the testing dataset is calculated and saved as the value of intermediate key-value pairs. The corresponding key is jut the month index.

Reduce Summation of squared error over 128 stations, which belongs to the same month, are shuffled and sent to one reducer. The final MSE is calculated for each month. The final 576 key-value pairs, one for each month, are saved on HDFS.

4.5 Residual Diagnostic

Generating diagnostic plots for the final residuals after the SSTL routine can be difficult once the dataset becomes larger. Even with the maximum temperature dataset we discussed previously, the visualization of residuals is not trivial. Totally, there are 2,096,822 residuals. The first diagnostic plot for the residuals is to visualize

the distribution of residuals. However, not mention the size of the visual graph, generating the graph itself can be very time consuming. The easiest way to simplify the plot is to only includes specific number of quantiles instead of all of them in the plot. As shown in the Figure 3.41, the approximated 10,000 quantiles, from 0.00001 to 0.99999, is plotted against to the corresponding f-values. With respect to the implementation method, we borrow the idea from the `drQuantile` function of `datadr` package [39], which we calculate the quantiles in parallel. This process is accomplished in a sequence of MapReduce jobs described as below. The first information about the residuals is the range, which is collected by the following MapReduce job.

Input 576 input key-value pairs; keys are the month index, and the value is the data.frame including 7,738 rows of `tmax`, `trend`, `seasonal`, and `Rspa`. It is the output from Step VII of SSTL routine described in section 4.2.7.

Output one key-value pair whose value is a vector with the maximum and minimum of the residuals.

Map For each input key-value pair, the monthly minimum and maximum value of residual is calculated and saved as a numeric vector. Meanwhile, the input key is changed to be 1, which is meaningless. The only purpose is to make sure all monthly minimum and maximum value can be sent to one reducer. The intermediate key can be any integer as long as all 576 of them are the same.

Reduce The overall range of residuals is calculated based on the monthly minimum and maximum who are sent to the only one reducer.

The whole range of residual is read back from the HDFS, and equally cut into 10,000 small intervals or bins. The cutting points of the 10,000 intervals are saved as a RData file on the HDFS. The idea is trying to count the frequency of residuals falling into each bin and using the center of each bin to represent the value of all residuals in the bin. It can be implemented lightly in parallel. This procedure is demonstrated in the following MapReduce job.

Input 576 input key-value pairs, key is the `date`, and value is the data.frame including 7,738 rows of `tmax`, `trend`, `seasonal`, and `spafit`.

Output 10,000 key-value pairs whose key is the center point of each bin, and value is the number of residuals falling into the corresponding bin.

Map For each input key-value pair, the frequency of residuals in each of 10,000 bins are calculated by R function `cut`. Then we generate 10,000 intermediate key-value pairs, one for each bin. The key is the index of bin, and value is the count of residuals in the corresponding bin. The cutting points of 10,000 intervals are passed and load into each mapper as a shared RData file.

Reduce All counts who share the same bin index are shuffled, sorted, and sent to one reducer to be summed together. The final 10,000 frequencies of residuals in bins are saved as key-value pairs on HDFS.

Once we got the frequency table of residuals in each bin, it is trivial to get the accumulated frequency of the residuals, and calculate the quantiles at the pre-defined f values f_i by left-continuous constant interpolation.

4.6 `drsstl` Package

`drsstl` is a R package for SSTL analysis under divide and recombined framework. It is highly depends on three exist R packages: `Rhipe`, `stlplus`, and `Spaloess`, which are all open source and available on Github [40]. Detailed documentation and examples can be found in the appendix.

The main function in the package is `drsstl()`, which can take two types of input to conduct the SSTL routine. One type of input is a data.frame sitting in the memory of local machine. The real computation engine for this situation is the `sstl_local()` function. Another type of input is a HDFS path where the spatial-temporal data is saved. For this situation, there are seven steps included in the function call of `sstl_mr()`, each of which are implemented in a function from `drsstl` package. Such

as `readIn()` to read in the raw text files and generating by-month division on HDFS; `spaofit()` to produce spatial smoothing fit of original observation in each month; `swaptoLoc()` to generate by-location division from the by-month division including the spatial smoothed values; `stlfit()` to apply temporal fitting on the spatial smoothed values in each location by calling `stlplus` function; `swaptoTime()` to generate by-month division from the by-location division which includes all STL+ fitted components; `sparfit()` to carry out spatial smoothing fitting on the remainder component of STL+ fit.

Besides the wrapper function `drsstl()` and those functions which represents each step of the modeling routine, there are other two functions `spacetime.control()` and `mapreduce.control()`, which are both returning a R list object. Concretely, the `mapreduce.control()` returns a list including all user tunable Hadoop parameters used in a given MapReduce job. `spacetime.control()`, on the other hand, returns a list with all smoothing parameters needed either for spatial smoothing or temporal smoothing.

Finally, the last function in the package is `predNewLocs()`. It is a wrapper function for function `predNew_local()` and `predNew_mr()`. They are used to conduct prediction at new locations based on the fitting results of the original dataset. `predNew_local()` is used for the situation when the fitting results of original dataset is in local memory. And `predNew_mr()` is for the prediction under the divide and recombined framework.

5. MULTI-FACTOR DESIGNED EXPERIMENT FOR PERFORMANCE OF THE NONPARAMETRIC-REGRESSION MODELING

In this chapter, I conduct the analysis of the performance of the drSSTL routine for large spatial-temporal dataset. There are two groups of tunning parameters have the potential influence on the performance of the routine. One is the tunning parameters of statistical model, which controls the complexity of the smoothing procedure of spatial loess and STL+ procedure. Another group of parameters are user-tunable Hadoop parameters. Within this chapter, I illustrate several pilot experiments and full factorial experiments to study the affect of these tunning parameters to the elapsed time of the drSSTL routine.

5.1 Type of MapReudce Jobs

There are three main steps in a MapReduce job: Map, Shuffle and Sort, and Reduce. Every MapReduce job can be easily labeled as one of two different types based on these three steps. One type of MapReduce jobs only have Map. The intermediate outputs from mappers are directly wrote on HDFS. Reduce function is not defined, and the Shuffle and Sorting stage is avoid as well. Other MapReduce jobs, however, include all Map, Reduce, and Shuffle and Sorting stages.

Within our drSSTL routine under the divide and recombined framework, we categorize all necessary MapReduce jobs into two groups. MapReduce jobs of the first type are mainly focus on one of the smoothing procedures (spatial smoothing or STL+ smoothing). We name them as *model-fitting* jobs. Any MapReduce job of this type reads in a given type of division (either by-month or by-station) from HDFS, and then

carries out a smoothing procedure, depends on what the division is, in its mappers. There is no need for neither Reduce nor Shuffle and Sort stage in this type of jobs because the division is not changed after the Map. So the outputs of the Map are directly written onto HDFS.

MapReduce jobs from another type are focusing on generating different division of data, which we name as *swapping* jobs. For this type of jobs, they read in a given type of division or raw text file from the HDFS, and generates another type of division. Concretely, for each input key-value pair, the Map function generates multiple intermediate key-value pairs with different keys. For instance, if by-month division is read in, 7,738 intermediate output key-value pairs are created for each input key-value pair, with station ID as key for each of input key-value pair in each mapper. The outputs of mapper is first written to local disk of servers. Next the intermediate outputs are shuffled and sorted, then copied to corresponding reducer based on keys. Several user-tunable Hadoop parameters can be considered and varied during this stage to improve the performance of jobs. In each reducer, all key-value pairs shared the same key are grouped and combined together. Final outputs of Reduce are written to HDFS as multiple files, one for each reducer.

5.1.1 Map Only Jobs

Among all seven steps of the drSSTL routine, step II, IV, and VI belongs to the *model-fitting* jobs in which only Map is involved. Shuffle and Sort stage is taken out from the job with Reduce to eliminate unnecessary network traffic and multiple trips to the local disks. The outputs from mappers are directly written to the HDFS. Consequently, the Hadoop tuning parameters which affect the Shuffle and Sort stage do not have any effect on these jobs. But the performance of this job in term of elapsed time can be improved by the statistical parameters. In step II and VI, each mapper applies spatial smoothing procedure to each of input key-value pairs. A statistical parameter named `cell` can be tuned to speed up the computation process of the

spatial smoothing. Meanwhile, for step IV, another job with Map only, Shuffle and Sort and Reduce are all excluded from the job. There is also a statistical parameter called `jump` controlling the speed of seasonal smoothing and trend smoothing.

5.1.2 MapReduce Jobs

The rest of steps, I, III, V, and VII of the routine under divide and recombined framework are MapReduce jobs with all Map, Shuffle and Sorting, and Reduce stages. Notice that the computation complexity of each mapper is very light, mainly just `strsplit` and `rhcollect` functions in R. Meanwhile the size of intermediate outputs from each mapper is roughly close or even larger than the inputs. A lot of intermediate outputs, in term of size, are shuffled and sorted and then sent to each reducer. For this type of jobs, the best performance can be obtained by assigning more memory to be used for Shuffle and Sorting, which can avoid heavy I/O to disks. Reduce stage is also very light with respect to computation. Only `rbind` or `c` concatenating function are called accumulatively. So the performance of job can be boosted when the intermediate outputs can be resided entirely in memory of the reducer's JVM. This can be achieved by assigning more memory from the heap size of reducer's JVM to hold as many as possible intermediate outputs. There is another critical Hadoop parameter should be considered which controls the starting time of reducers. Since in MapReduce2, reducers and mappers all request computational resource (cores and memory) from the Resource Manager of the cluster [41]. And each reducer cannot start the real computation until the corresponding partition from all mapper are finished and copied to the it. If reducers start before most of mappers finished, they are just occupying those computation resource doing nothing but waiting. Consequently, the starting time of reducers should be specified wisely to avoid the waist of computational resource. More details about Hadoop parameters are explored in the next section.

5.2 Experiment Design

We are considering three full factorial experiments with replicates to help us having a better understanding of the effect of those parameters to the performance of the routine. Each of Hadoop parameters and statistical parameters are treated as a factor, and the elapsed time associated with those MapReduce jobs are measured as response. Since the Hadoop parameters only influence the MapReduce jobs, and statistical parameters (`cell` and `jump`) only exist in Map only jobs, they will be assessed separately without interaction. There are 8 Hadoop parameters and 2 statistical parameters. Each of Hadoop parameters has 2 levels, the other two statistical factors has 4. One full factorial experiment for the Hadoop parameters, one for the `cell` parameter, and one for the `jump` parameter since there is no interaction between `cell` and `jump` either. The whole routine and the Hadoop system we are studying is very complex. There have been several researches of Hadoop performance analysis such as [42], [43]; some of them have been trying to explore the best configuration settings of the Hadoop cluster such as [44], [45]. But none of them are specifically targeting on the Shuffle and Sorting stage as we do. Also none of those work is under the MR2 (YARN) framework.

A series of pilot experiments on each factors independently is performed to provide us a great amount of knowledge about choosing the levels of these factors. However, any insight about the interactions among them is not included. We have to rely on the empirical study and visual display to discover the interaction effect of those Hadoop factors. Therefore a full factorial experiment is chosen. There are 256 combinations of the Hadoop factors. Each of which has 3 replicates, and there are 4 MapReduce jobs (*swapping* jobs) needs to be measured. So in total there are 3072 runs in the experiment. And each run has full use of the cluster without any other Hadoop jobs or HDFS activity.

Meanwhile, for the `jump` factor, there are 4 levels, each of which has 3 replications, and there are two spatial smoothing Map only jobs. So in total there are 24 runs for

the second experiment. Finally for the `cell` factor, there are 4 levels as well, with 3 replicates. Only one Map only job is measured. So totally there are 12 runs for the third experiment. Similarly, no other Hadoop jobs or HDFS activity is submitted to the cluster by any other users to guarantee the accuracy of the measurement of timing.

In order to mimic the potential effect of all factors to a real large dataset, we decide to duplicate the original text data by $1,365$ times, so the input file for the experiments is about 46.2 GB. Basically we keep the number of stations the same, but extend the length of each time series.

5.2.1 Hadoop User-Tunable Factors

All eight Hadoop factors have very different but highly correlated role. Six of them controls the Shuffle and Sorting stage, either on the map side or the reduce side of the Shuffle and Sorting. One of them controls when reducers should start. The last one controls the number of reducers. In the next several subsection, we explore the meaning and function of each Hadoop factors.

`mapreduce.task.io.sort.mb` (ISM)

ISM parameter controls the size of memory buffer, in megabytes, which is used to hold the intermediate outputs in each mapper. The value for ISM can be any integer varies from 1 to 2047, but the default value is 100. That means only 100MB of memory from heap size of the mapper's JVM is allocated for saving the outputs. It is quite small in general. For jobs like the *swapping* jobs in the drSSTL routine, it is definitely worth to increase the value of ISM to give more memory for holding outputs in each mapper. In the full factorial experiment for Hadoop factors, we vary the ISM factor between 128 and 512 based on the pilot experiment of ISM.

`mapreduce.map.sort.spill.percent` (SSP)

SSP parameter works with ISM collectively to control the memory used for holding Map outputs. Concretely, a circular memory buffer with size ISM is allocated for each mapper to write intermediate outputs to. When the map outputs occupy SSP percents of this memory buffer, the outputs are spilled into the local disks of the server where mappers are running. Meanwhile the mapper is keeping writing outputs to the memory buffer when buffer is spilling to disks. However if the memory buffer is filled up totally during this time period, the mapper is paused until the spill is finished. Clearly, it is critical to set this parameter as well as the ISM to be high enough to avoid spilling process or the pause of mapper. However, ISM and SSP together should not set to be too large. Because the memory buffer is still belongs to the mapper's JVM heap size, which is decided by `mapreduce.map.java.opts`. In our experiments, the mapper heap size is fixed at 3.5 GB. Reserving too much memory from the total heap size only for holding outputs of mapper will leave limited memory usage for other processes spoused by the JVM, and indeed force JVM to involve more garbage clean, which in turn hurt the job performance. Preliminary study of the SSP factor suggests the two levels of it to be 0.2 and 0.8.

`mapreduce.job.reduce.slowstart.completedmaps` (RSC)

The factor RSC controls when reducers should be launched. Specifically, it specifies the fraction of completed mappers before any reducer starts. Under MapReduce2 (YARN) [41] framework, this parameter becomes critical if the Map of MapReduce job is time consuming and the number of reducers is more than 20% of the total number of tasks can be running simultaneously on the cluster. Setting RSC parameter to be a small value as default, which is 0.05, will start the reducers doing nothing but waiting for the outputs from mappers once 5% of mappers are finished. However those containers assigned for reducers are requested from the Resource Manager of the cluster. Instead of being assigned to reducers waiting for outputs from mappers,

they can be assigned to the mappers who are still under the pending statues. In the experiment, RSC is varied between 0.05, which is the default, and 0.5.

`mapreduce.reduce.shuffle.parallelcopies` (SPC)

After the Map step is finished, the intermediate outputs are partitioned into several partitions, one for each reducer. They are located on the local disks of the server where mappers ran. Then each reducer copies the corresponding outputs partition from all mappers through the network. And actually each reducer evokes a number of copy-threads, which controlled by SPC parameter, to fetch the intermediate outputs in parallel. SPC parameter should be set wisely. Too large value of copy threads will waste for the CPU, but too small number of copy threads will slow down the copy step of reducers. The default value is 5, and we vary the value of SPC in the experiment between 5 and 20.

`mapreduce.reduce.shuffle.input.buffer.percent` (SIBP)

During the copying stage, the outputs of Map are copied to a memory buffer on the reduce side. SIBP parameter controls how large the memory buffer is. The function of SIBP is very similar with ISM but on reducer side. And it specifies the memory buffer by proportion of JVM heap size instead of the exact amount as ISM does. The default value of this parameter is 0.7. Suppose the total heap size of the JVM of reducer is 4GB, then 2.8 GB of JVM's memory is used for holding the outputs copied from the local disk of servers ran mappers. We vary it as 0.3 and 0.9 in the experiment.

`mapreduce.reduce.shuffle.merge.percent` (SMP)

Similar with the Map, there is also a spilling mechanism in the Reduce stage, which attempts to avoid the memory overflow issue. Specifically, the Map outputs

are consistently copied to a memory buffer in the JVM of reducer whose size is controlled by the SIBP. It is the proportion of memory buffer to the heap space of the reducer's JVM. Once the content in the memory buffer reaches the threshold controlled by two parameters collectively, the intermediate outputs are spilled to the local disks of the server where reducer is running. These two parameters are the SMP and the `mapreduce.reduce.merge.inmem.threshold` (MIT). The SMP controls the threshold in term of percent of the memory buffer size. If the memory buffer is occupied over the percent of SMP, the content in the buffer will be spilled to the local disks. By default, the SMP is set to be 0.66. For instance, suppose the heap size of reducer's JVM is 4 GB as well, the size of memory buffer is 2.8 GB by setting the SIBP to be 0.7. Then the spilling threshold is 1.85 GB. Another parameter MIT controls the threshold of the memory buffer in term of the counts of key-value pairs. So the intermediate outputs held in the memory buffer cannot be too many or too large, otherwise they will be spilled to the local disks as well. The default value of MIT is 1,000. But setting this parameter to 0 can hand over the control of spilling process fully to the SMP. In order to simplify the experiment, we fix the value of MIT at 0, and controls the reducer memory buffer only by SMP.

`mapreduce.reduce.input.buffer.percent` (IBP)

Once the last piece of intermediate outputs are copied into the memory buffer, reducer can decide either also spill current contents in the memory buffer to the local disks on the server where reducer is running, or hold an amount of outputs in the memory buffer and directly feed them to the reduce function. IBP parameter is specifying the proportion of size of outputs can be reserved in the memory buffer to the total heap size of each reducer's JVM. By default, the IBP is set to be 0, which forces all intermediate outputs to be spilled to the local disk and leave all memory for the computation of reduce function. However, if the memory utilization of reduce computation is extremely light, then we can increase this parameter to be value close

to 1 to keep more intermediate outputs in the memory on the reduce side and save strips to the local disks. Less I/O will definitely improve the job performance. In the experiment, we vary the IBP as 0 or 1.

`mapreduce.job.reduces` (RTSK)

The RTSK factor controls the number of reducer, which plays a critical rule on the performance of the whole routine. One fact of our analysis routine is that it is a sequence of MapReduce jobs. The sequence file generated from one MapReduce job is the input to the next. So it is crucial to choose the number of reducer because it also controls the number of files in the output path of the job on HDFS, which in turn controls the size of intermediate outputs sent to each reducer under the assumption that each key-value pair has roughly the same size in term of memory usage. In other words, this parameter controls the amount of data that will be processed by each reducer. If too few of reducer are specified, then more intermediate outputs are sent to one reducer. Then it is more likely to invoke the multiple trips to local disk on the reduce side as we mentioned in the paragraph about the SIBP and the SMP. Once the MapReduce job finished, each reducer generated a file on HDFS hold the output key-value pairs. The next MapReduce job read in those files and initialize one or several mappers for each file, and the number of mappers for each file is determined by

$$\frac{\text{size of the file}}{\text{block size of HDFS}} \tag{5.1}$$

The remainder of the division will be also stored as a block on HDFS, and it occupies a mapper as well even if it were only 1 MB in size. If too much files are generated in the previous MapReduce job by a large number of reducer, each file is highly likely much smaller than an usual block. Those small blocks still occupies the same amount of resource as a full block does, which dramatically reduce the efficiency of the job.

5.2.2 Statistical Factors

Two separated experiments are performed to study the effect of the statistical factors, `cell` and `jump`, to the computational performance of *model-fitting* jobs in the drSSTL routine respectively. As we have already mentioned before, two statistical factors affect the performance of jobs in a very similar way, except one is for spatial smoothing and another one is for STL+ fitting. Since there is no interaction between the two factors or any other 8 Hadoop factors, we conduct the two experiment independently with all 8 Hadoop parameters fixed at the optimal values chosen from the first experiment.

cell

The spatial smoothing procedure can be accelerated by including kd-tree and blending mechanism. The real local weighted regression fitting is conducted only at vertexes of the kd-tree. By "vertex", we just refer to a corner of a cell. The two extreme cases are all points in the tree are vertexes or one cell is in the tree with four vertexes only. In the first situation, the local weighted regression is fitted at all locations, no interpolation is introduced. However for the second case, the real local regression only fitted at four vertexes. All other locations inside of this cell are fitted by interpolation. The computational speed of interpolation is much faster than weighted least square procedure.

So a parameter named `cell` in spatial smoothing procedure, which varies from 0 to 1, is specified to control the the maximum number of points in a cell of the kd-tree. Cells with more than $floor(\texttt{n} \times \texttt{span} \times \texttt{cell})$ points are subdivided, where n is the total number of points. Clearly, parameter `cell` controls the number of vertexes of the kd-tree which in turn controls the number of points where actual local regression fitting is calculated. Varying `cell` from small to large values between 0 and 1 can dramatically affect the computation time of each mapper.

jump

Meanwhile, for step IV, the STL+ fitting for each time series can be accelerated by interpolation between some knots. The fitted values of seasonal or trend component are calculated at those knots directly by the STL+ procedure. The fitted value at the rest of time points are interpolated based on the fitted value at the knots. Apparently, less knots we have, the faster the computational speed will be. So we specify that linear interpolation happens between every $ceiling(w_s/\texttt{jump})$th value for each subseries. Same for the trend smoothing as $ceiling(w_t/\texttt{jump})$. The larger `jump` is, less knots is in the time series.

5.2.3 Hardware

The experiments run on the WSC cluster [46] configured and maintained by the ITaP. It consists of 9 HP compute nodes with two 10-core Intel Xeon-E5 processors (20 cores per node) and 128 GB of physical memory. But we configure the Hadoop containers to use 120 GB and reserve 8GB for OS and other daemons. All nodes have 56 Gb FDR Infiniband interconnect. The cluster runs Red Hat Enterprise Linux 6 (RHEL6); Cloudera Hadoop 2.3.0-cdh5.1.3; protobuf-2.5.0 protocal buffer software; RHIPE 0.75.2_cdh5; and Java 1.7.0_75.

For the Hadoop setting, the number of core for each container is set to be 1. Totally, 180 containers can be running simultaneously on the cluster. However, for any MapReduce job (application) submitted to the cluster with Hadoop YARN, one of the container will be assigned to launch the MRAppMaster (MapReduce Application Master) process. Consequently, 179 containers can be assigned with either a map task or reduce task and be running simultaneously. We denote this number as N_{cont}. Also the memory limit for each container is configured to be 5 GB, and the heap size for the JVM of each mapper is configured to be 3.5 GB, and the heap size for the JVM of each reducer is 4 GB.

5.3 Results

In this section we present our findings of the impact of those Hadoop factors and two statistical factors on the performance of the MapReduce jobs in the drSSTL routine, including the main effects of these factors and the interactions among them. We relied very heavily on trellis displays to effectively study the effects and interactions of these factors. We start with eight pilot experiments to help us to clear our thoughts about the levels for each factor to be included in the final full factorial experiment.

5.3.1 Preliminary Results of Pilot Experiments

There are 8 pilot experiments before the full factorial experiment for the Hadoop parameters. One of eight Hadoop factors is varied with three or more levels while the rest of seven factors are fixed at a given level. The first pilot experiment is about the ISM factor. For each of four *swapping* jobs in the drSSTL routine, ISM is varied from 128 to 768 with 3 replicates. As shown in the Figure 5.1, the log base 2 of elapsed time of each job is plotted against to the ISM conditional on the job. The mean of elapsed time of three replicates at each level of ISM is superposed as a red line in each panel.

Link to figure

Figure 5.1.: Log base 2 elapsed time against to ISM

First of all, the graph shows that the elapsed time of Step V and Step VII are almost double the elapsed time of Step I and Step III. The reasons for Step VII is that three new columns, seasonal, trend, and spatial fitted value of remainder, are included in the outputs of Step VI. The amount of data that Step VII read in is more than four times the size of the inputs of Step III. While, the reason for Step V is

the overhead caused by calling 768,432 times of `rhcollect` function in each mapper. For Step I and III, the performance is increasing as ISM gets larger until 512 MB which provides the minimal elapsed time. While after the ISM gets larger than 512, the elapsed time is raised up again. On the other hand, for Step V and VII, the performance is also increasing while ISM increases until 512 MB. After 512 MB, the elapsed time roughly keeps at the same value. As a result, we decide to include 128 and 512 as two levels for the ISM in the full factorial experiment, since the difference of elapsed time between this two levels is maximal among the four values.

The second pilot experiment is about the SSP factor. In Figure 5.2, the elapsed time of each job is plotted against to the SSP conditional on the job. The SSP is varied among 0.2, 0.8, and 1. ISM is fixed at 512 MB. Still the elapsed time of Step V and VII are doubled. For Step I and III, the elapsed time keeps decreasing as the SSP increases. The optimal performance is obtained when SSP is 1. While, the situation in Step V an VII is a little bit different from Step I and III. Elapsed time is decreasing as the SSP varies from 0.2 to 0.8. When the SSP keeps increasing until 1, the elapsed time, however, increases with a small amount. Setting the SSP to be 1 is risky. It is actually more likely to pause the Map stage if the SSP is 1. Based on the finding in the Figure, 0.2 and 0.8 are included as two levels of the SSP in the later experiment.

Link to figure

Figure 5.2.: Log base 2 elapsed time against to SSP

The third pilot experiment is about the RSC factor. The RSC controls when the reducers should start. The number of reducers is fixed at 179, which is also the number of containers can be run simultaneously on the cluster. The number of mappers are varied among four jobs, which is decided by the number and size of input files. There are 345 mappers in Step I, 358 mappers in Step III, 338 mappers in Step V, and 1790

mappers in Step VII. The RSC factor is varied among 0.2, 0.5, 0.8, 0.9 and 1.0. Each value is assigned to each of four MapReduce jobs with 3 replicates. In Figure 5.3, the log based 2 elapsed time is plotted against to the RSC value conditional on the type of jobs. The mean of elapsed time over three replicates at each level of RSC is superposed in each panel as the red lines. Clearly, the performances of Step I, III, and V are optimized when the RSC is 0.5, and after 0.5 the elapsed time raises up again. For Step VII, the elapsed time keeps decreasing as the RSC increases to 0.8. In the final experiment of Hadoop factors, we consider 0.05 (default) and 0.5 as two levels of the RSC factor.

Link to figure

Figure 5.3.: Log base 2 elapsed time against to RSC

The results of experiment for the SPC is illustrated in Figure 5.4. The elapsed time is plotted against to three different values of SPC, 5, 20, and 50, conditional on type of jobs. The mean of elapsed time over three replicates at each level of the SPC is superposed in each panel as a red line. The number of reducers is also fixed at 179 here. Collectively, the performance of four jobs are optimized when the SPC is 20. Either smaller value, 5, or larger value 50 cause the increasing of elapsed time of jobs. But the difference among the elapsed time with three different levels of the SPC is negligible compared with the effect of the ISM or RSC factors.

Link to figure

Figure 5.4.: Log base 2 elapsed time against to SPC

The fifth pilot experiment is about the SIBP, which controls the size of the memory buffer on the reduce side. The SIBP factor is varied among 0.4, 0.7, 0.9, and 1, each has 3 replicates on all four jobs. Since the SMP factor is fixed at 0.6 in this experiment, and the JVM of reducer is assigned with 4 GB of memory limit, the spilling procedure is triggered when the size of content in the memory buffer of each reducer is over 0.96 GB, 1.68 GB, 2.16 GB, and 2.4 GB correspondingly. In Figure 5.5, the elapsed time is plotted against to the SIBP conditional on the type of job. The mean over 3 replicates of elapsed time is superposed in each panel as a red line. The effect of SIBP to the performance of each job is not as obvious as those previous factors. In Step I, elapsed time first decreases, then keeps as flat when SIBP is larger than 0.7. In Step V, elapsed time also decreases to the minimum when SIBP is 0.7, but raises up after SIBP is over 0.7. While in Step VII, elapsed time keeps increasing as SIBP increases. 0.4 of SIBP seems to provide the optimal performance only for Step V. We decide to include 0.3 and 0.9 as two levels of the SIBP in the later experiment.

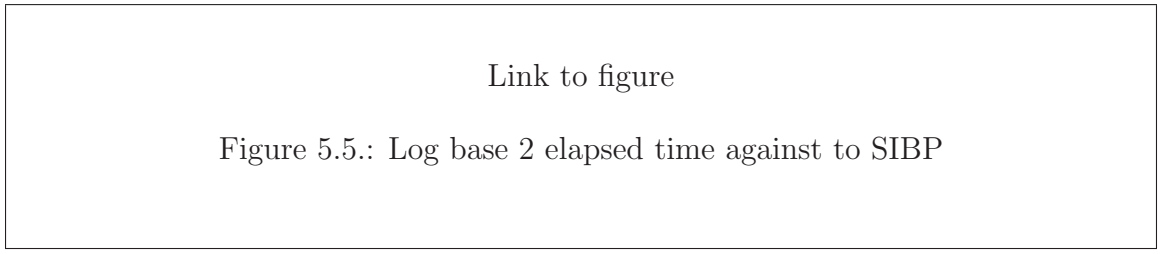

Figure 5.5.: Log base 2 elapsed time against to SIBP

The next experiment is about the SMP factor, which affects the performance of jobs interactively with the SIBP. The SIBP is fixed at 0.7 this time, which is the default value. And the SMP is varied among 0.3, 0.6, and 0.9. Since the heap size of JVM of each reducers is set as 4 GB, the spilling process is triggered when the size of content in the memory buffer is over 0.84 GB, 1.68 GB, and 2.52 GB correspondingly.

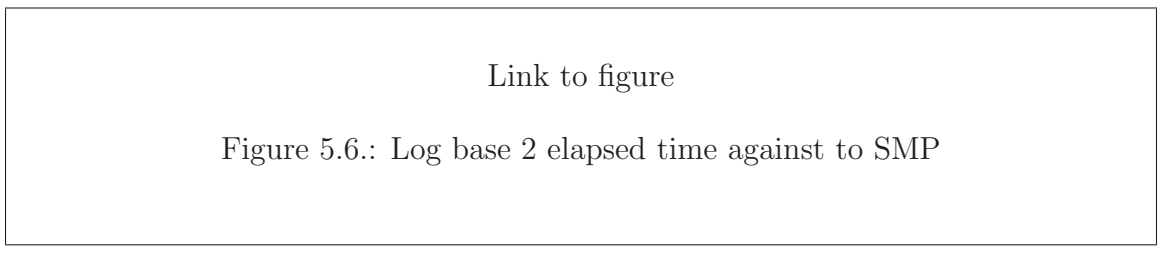

Figure 5.6.: Log base 2 elapsed time against to SMP

Link to figure

Figure 5.7.: Log base 2 elapsed time against to IBP

The last pilot experiment is about the RTSK factor which controls the number of reducers. Recall that there are 179 containers can run simultaneously on the cluster. We vary the RTSK factor among 90, 179, 269, and 358, which are 0.5, 1, 1.5 and 2 times the number of containers. Still 3 replicates are considered for each level of the RTSK. In Figure 5.8, the elapsed time is plotted against to the RTSK conditional on the type of job. The mean over 3 replicates at each level of RTSK is also superposed in each panel. Collectively, RTSK at 179 optimizes the performance of all four jobs. The elapsed time drops dramatically as the RTSK changed from 90 to 179, the it gradually raises up when RTSK keeps increasing. Consequently, 90 and 179 are included in the later full factorial experiment as two levels of the RTSK factor.

Link to figure

Figure 5.8.: Log base 2 elapsed time against to RTSK

5.3.2 Hadoop Factors

After the pilot experiments, we choose 2 levels for each of 8 Hadoop factors in the full factorial experiment. In this subsection, we illustrate the main and interaction effect of these factors to the performance of the routine in term of elapsed time of each job respectively.

Step I

There is clear interaction between the ISM and SSP factors. But they are quite independent with others. As shown in Figure 5.9, there is not significant difference in term of elapsed time between the two values of the SSP, 0.2 and 0.8, when the ISM is set to be 128 MB. But when the SSP is 0.8, by increasing the ISM to 512 MB, the performance of the job is boosted about 13% in term of elapsed time. Each mapper from Step I takes 128 MB data as input, and generates about 260 MB intermediate outputs. If the memory buffer specified by the ISM for each mapper is set to be 128 MB, increasing the SSP threshold from 0.2 to 0.8 cannot avoid the process of spilling to the local disks. Even we set the ISM to be 512 MB, keeping the SSP as 0.2 cannot avoid the spilling process either. However, it is perfectly avoided by increasing the SSP from 0.2 to 0.8 as the ISM is 512 MB, since only when the contents in the memory buffer exceeds 409 MB, which is much larger than the 260 MB, the spilling will be triggered.

Link to figure

Figure 5.9.: Log base 2 elapsed time against to ISM

Factor RTSK, factor IBP, factor SIBP and factor SMP are highly correlated with each other. The last three factors collectively controls the memory buffer on the reduce side; the RTSK, on the other hand, controls the amount of intermediate outputs copied to each reducer indirectly. For the job in Step I, setting RTSK to be 179 make each reducer getting about 506 MB intermediate outputs from mappers as inputs. The total heap size of the JVM is set to be 4 GB for each reducer. In the Figure 5.10, the log base 2 elapsed time is plotted against to the IBP condition on other 6 factors with the RTSK superposed in each panel. When the SIBP and the SMP are both 0.3, the threshold of spilling is about 360 MB; increasing the IBP from

0 to 1 does not improve the performance of the job. While if either or both of SIBP and SMP are 0.9, increasing the IBP can actually reduce the elapsed time by 6%. If the RTSK is changed to be 90, the amount of intermediate outputs sent to each reducer is increased to be 1012 MB. When the SIBP and SMP are both fixed at 0.3, increasing IBP from 0 to 1 still does not improve the performance of the job. But when the SIBP and SMP are both 0.9, increasing the IBP from 0 to 1 can reduce the elapsed time more than 16%. For the other two situations which the SIBP is 0.3 and SMP is 0.9 or the SIBP is 0.9 and SMP is 0.3, the threshold of spilling process in the memory buffer are both 1080 MB, which is slightly more than the 1012 MB inputs for each reducer. Increasing the IBP can still reduce the elapsed time but only less than 10%.

Link to figure

Figure 5.10.: Log base 2 elapsed time against to IBP

In the Figure 5.11, the log base 2 elapsed time is plotted against to the SIBP with the RTSK superposed in each panel. When the RTSK is 179, the IBP is 1, and the SMP is 0.3, increasing the SIBP from 0.3 to 0.9 can totally avoid the spilling to local disks, which reduce the elapsed time by 13%. But when the SMP is 0.9, the intermediate outputs to each reducer can be hold in the memory buffer no matter what the value of SIBP is. Then the spilling process is controlled by the IBP, which decides if the contents in the memory buffer should be spilled to disk right before the reduce function starts; and increasing the SIBP does not vary the elapsed time. On the other hand, when the RTSK is 90, the difference among the elapsed times with different values of SIBP is negligible, except when the RSC is 0.5 and IBP is 1, increasing the SIBP can reduce the elapsed time only about 6%.

Link to figure

Figure 5.11.: Log base 2 elapsed time against to SIBP

The interaction behavior between the factor RTSK and factor RSC depends on the combination of the SIBP and SMP. As shown in Figure 5.12, when the SIBP and SMP are both 0.9, increasing RTSK collectively reduces the elapsed time about 12%. No significant interaction exists between RTSK and RSC. However, when the SIBP and SMP are both 0.3 or either one is 0.9, and RSC is 0.5, the elapsed time of the job can be reduced around 19% by increasing the RTSK from 90 to 179; when the RSC is 0.05, on the other hand, the elapsed time only decreases about 8% or less.

Link to figure

Figure 5.12.: Log base 2 elapsed time against to RTSK

In the Figure 5.13, the log base 2 elapsed time is plotted against to the RSC with the RTSK superposed in each panel. Similarly, the effect of the RSC factor to the elapsed time is also depends on the RTSK factor and the combination of the SIBP and SMP. When the RTSK is 90 and both the SIBP and SMP are 0.3, increasing the RSC from 0.05 to 0.5 can reduce the elapsed time by 13%; but if either or both of SIBP and SMP are 0.9, in which case the spilling process is avoid, increasing the RSC can reduce the elapsed time 19%. When the RTSK is 179, amount of data copied to each reducer is shrunk, increasing the RSC can reduce the elapsed time by 24% collectively.

Link to figure

Figure 5.13.: Log base 2 elapsed time against to RSC

In summary, the RTSK and the RSC factors are the factors boosting the performance of the job the most. Factor SPC is independent with all other factors, and it does not seem to have significant effect on the performance of the MapReduce job. The best performance of the Step I can be obtained when set the ISM as 512 MB, SSP as 0.8, RTSK as 179, RSC as 0.5, SIBP as 0.3 or 0.9, SMP as 0.9, and IBP as 1.

Step III

The MapReduce job in Step III is similar with the Step I in terms of the amount of inputs and outputs for each mappers, which generates 260 MB outputs from 128 MB inputs.

Link to figure

Figure 5.14.: Log base 2 elapsed time against to ISM

In the Figure 5.14, when the SSP is 0.8, increasing the ISM from 128 MB to 512 MB can reduce the elapsed time by more than 13%, since 512 MB with 0.8 threshold can totally avoid the process of spilling to local disks. But 128 MB of memory buffer with 0.8 of the SSP cannot avoid that. When the SSP is set as 0.2, the elapsed time of the job is almost stable as the ISM increases, since the process of spilling to local disks cannot be avoid with either value of the ISM. In some of cases, when the ISM is 128 MB, increasing the SSP from 0.2 to 0.8 can even hurt the performance. This is because a large value of SSP can pause the mapper to clean up the memory buffer

when the ISM is relatively small compared with the size of map outputs. In this case, 128 MB is only half of the outputs size 260 MB.

Still the factor IBP is highly correlated with the SIBP and SMP factors. In the Figure 5.15, the log base 2 elapsed time is plotted against to the IBP with RTSK superposed in each panel. When the RTSK is 179, the size of intermediate outputs copied to each reducer is about 506 MB. When the SIBP and SMP are both 0.3, the threshold of spilling is about 360 MB, the contents in the memory buffer on the reduce side are spilled to local disks. Increasing the IBP from 0 to 1 cannot improve the performance of the job since very limited amount of contents are left in the memory buffer after the spilling process. Even increasing the IBP to 1 does not hold very much of contents in the memory compared with 0 IBP. But if the SIBP and SMP are both 0.9, increasing the IBP from 0 to 1 can reduce the elapsed time of the job by 6%. But when the SIBP and SMP have different values, one is 0.3 and another is 0.9, increasing the IBP can reduce the elapsed time by 6% only if the RSC is 0.5. On the other hand, when the RTSK is 90, the size of inputs to each reducer is about 1 GB. When the SIBP and SMP are both 0.9, increasing the IBP can reduce about 6% of the elapsed time collectively. When the SIBP and SMP are both 0.3, the elapsed time is not varied by increasing the IBP. When the SIBP and SMP have different values, increasing the IBP can only reduce the elapsed time by less than 6% when RSC is 0.5, but no difference when RSC is 0.05.

Link to figure

Figure 5.15.: Log base 2 elapsed time against to IBP

The RTSK is slightly interacted with the RSC factor. In the Figure 5.16, the log base 2 elapsed time is drawn against to the RSC with RTSK superposed in each panel. Collectively, the elapsed time can be reduce more than 24% by increasing the RSC from 0.05 to 0.5. And RSC does not show any strong interaction with other

factors. While increasing the RTSK from 90 to 179 can reduce the elapsed time about 6% only if the SIBP and SMP are both 0.9.

Link to figure

Figure 5.16.: Log base 2 elapsed time against to RSC

In summary, the ISM and the RSC factors are the factors boosting the performance of the Step III the most. Factor SPC is still quite independent with all other factors, and has no effect to the performance. The recommended values of those factors for Step III are: ISM as 512 MB, SSP as 0.8, RTSK as 179, RSC as 0.5, IBP as 1, and SIBP are both 0.9.

Step V

Each mapper from the MapReduce job in Step V takes 128 MB inputs but generates about 686 MB intermediate outputs. The spilling process cannot be perfectly avoided, even setting the ISM and the SSP to be 512 MB and 0.8 respectively. As shown in Figure 5.17, when the ISM is 512 MB, the elapsed time of the job is almost stable or even increased as the SSP increasing from 0.2 to 0.8. When the ISM is 128 MB, increasing the SSP slightly reduce the elapsed time by 7%. It is because a larger value of SSP reduces the number of spilling files on the local disks when the amount of intermediates outputs from each mapper is relatively larger than the inputs. But when the SSP is set as 0.8, increasing the ISM can reducer the elapsed time by 24%.

Link to figure

Figure 5.17.: Log base 2 elapsed time against to ISM

In the Figure 5.18, the log base 2 elapsed time is plotted against to the SSP with ISM superposed in each panel. Collectively, increasing the ISM from 128 MB to 512 MB can shrink the elapsed time of the job by 19%.

Link to figure

Figure 5.18.: Log base 2 elapsed time against to SSP

The interaction between the RTSK and the RSC factors is shown in Figure 5.19. When the RTSK is 179, increasing the RSC factor from 0.05 to 0.5 can dramatically reduce the elapsed time by 24%. But when the RTSK is 90, increasing the RSC can only reduce the elapsed time by 10%.

Link to figure

Figure 5.19.: Log base 2 elapsed time against to RSC

On the other hand, as in the Figure 5.20, if the RSC is fixed at 0.05, increasing the RTSK does not improve the performance of the job at all. While if the RSC is set at 0.5, increasing the RTSK can reduce the elapsed time about 13%.

Link to figure

Figure 5.20.: Log base 2 elapsed time against to RTSK

The factor RTSK, RSC, IBP, SIBP and SMP are highly correlated with each other. When the RTSK is fixed at 90, the amount of intermediate outputs copied to

each reducer is about 2.59 GB, only when the SIBP and the SMP factor are both 0.9, spilling to local disks on the reduce side can be fully avoid. Meanwhile, if the RSC is set as 0.5, increasing the IBP from 0 to 1 only reduces the elapsed time of the job about 6%, or even less if the RSC is 0.05. On the other hand, when the RTSK is set at 179, the amount of intermediate outputs copied to each reducer is about 1.36 GB, but the spilling process still cannot be avoid unless the SIBP and the SMP are both 0.9. Increasing the IBP from 0 to 1 can reduce the elapsed time about 13% or more. If either or both of the SMP and SIBP are set to be 0.3, spilling process is triggered on the reduce side, increasing the IBP does not improve the performance of the MapReduce job no matter what the value of RTSK is.

Link to figure

Figure 5.21.: Log base 2 elapsed time against to IBP

The interaction among the IBP, SIBP, and the SMP is even clearer in Figure 5.22. Only when the IBP is 1 and the SMP is 0.9, increasing the SIBP can reduce the elapsed time by 12%, while if the IBP is 1 but SMP is 0.3, increasing the SIBP does not improve the performance since spilling process on the reduce side is triggered. When the IBP is 0, increasing the SIBP cannot reduce the elapsed time even the SMP is fixed at 0.9, because all contents hold in the memory buffer have to be spilled to disk all at once right before the reduce stage starts.

Link to figure

Figure 5.22.: Log base 2 elapsed time against to SIBP

Similarly as before, the SPC is independent with all other factors, and does not seem to have significant effect on the performance of the MapReduce job. In summary, the best performance of the Step V can be obtained when set the ISM as 512 MB, SSP as 0.8, RTSK as 179, RSC as 0.5, SIBP and SMP both as 0.9, and IBP as 1.

Step VII

For the last step, the mappers from Step VII are the lightest one compared with previous three MapReduce jobs. Each mapper generates 126 MB intermediate outputs from 128 MB inputs data. When the ISM is set as 128 MB, increasing the SSP cannot avoid the spilling to local process. Consequently, the value of elapsed time is indistinguishable for the two situations. If the SSP is fixed at 0.2, increasing the ISM cannot improve the performance neither, since the threshold for spilling process is still much smaller than 126 MB. But if the SSP is fixed at 0.8, then increasing the ISM to 512 MB can perfectly avoid the spilling process on the map side, which reduce the elapsed time about 13%.

Link to figure

Figure 5.23.: Log base 2 elapsed time against to ISM

On the other hand, when the ISM is fixed at 128 MB, increasing the SSP does not help to improve the performance. But if the ISM is set at 512 MB, the elapsed time decreases about 13% by increasing the SSP.

Link to figure

Figure 5.24.: Log base 2 elapsed time against to SSP

The factor RTSK, IBP, SIBP and SMP influence the performance collectively. When the RTSK is set as 90, the amount of intermediate outputs copied to each reducer is about 2.52 GB. The spilling process on the reduce side can be avoid only if the SIBP and SMP are both 0.9. Increasing the IBP from 0 to 1 can only reduce the elapsed time of the job by 6%. If either or both of SIBP and SMP are 0.3, the effect of IBP to the elapsed time is negligible. On the other hand, if the RTSK is set as 179, the amount of intermediate outputs copied to each reducer is shrunk to be 1.26 GB. The elapsed time can be reduced by 13% only if the SIBP and SMP are both 0.9. The difference among the elapsed times are eligible with any other combination of SIBP and SMP.

Link to figure

Figure 5.25.: Log base 2 elapsed time against to IBP

Meanwhile, in Figure 5.26, the log base 2 elapsed time is drawn against to the SIBP with SMP superposed in each panel. When the RTSK is 90 and IBP is 0, increasing the SIBP from 0.3 to 0.9 can reduce the elapsed time about 5% for both SMP values. When the RTSK is 179, the situation is more complex. If the IBP is 0, increasing the SIBP actually increases the elapsed time by 6% as SMP is 0.9. But if the IBP is 1, increasing the SIBP does decrease the elapsed time by 6% as SMP is 0.9. While in both situations, the elapsed time does not change if SMP is 0.3.

Link to figure

Figure 5.26.: Log base 2 elapsed time against to SIBP

The interaction between the RTSK and the RSC factors becomes indistinguishable in the Step VII. There are 1790 mappers in the job of Step VII. Collectively, increasing the RSC from 0.05 to 0.5 can reduce the elapsed time by 13% or more. But the effect of RTSK to the elapsed time is complicated. When the SIBP and SMP are both 0.3, each reducer is extremely heavy in term of disk I/O. Increasing the RTSK from 90 to 179 can reduce the elapsed time by 17%. But if either the SIBP or the SMP is increased to 0.9, increasing the RTSK can only reduce the elapsed time by 7%. Even when the SIBP and SMP are both 0.9, and the IBP is 0, all intermediate outputs copied to each reducer still have to be spilled from memory to local disks. In this case, increasing RTSK can only reduce the elapsed time of the job about 6% or less. When the SIBP and SMP are both 0.9, but the IBP is 1, increasing the RTSK can reduce the elapsed time by 13%.

Link to figure

Figure 5.27.: Log base 2 elapsed time against to RSC

Link to figure

Figure 5.28.: Log base 2 elapsed time against to RTSK

In summary, for the MapReduce job in Step VII, the best performance can be achieved when set the ISM as 512 MB, SSP as 0.8, RTSK as 179, RSC as 0.5 or higher, SIBP and SMP both as 0.9, and IBP as 1. In the next subsection we will illustrate more details about the RSC factor and why it may be better to set it higher than 0.5 in Step VII.

5.3.3 More about RSC

Since the RSC factor is the most effective factor to the performance of all jobs, we consider an extra experiment for the RSC only. In this experiment, we fix all other factors except the RTSK at the level which benefits the performance the most. We then vary the RTSK and the N_{cont}, the maximum number of containers can be running simultaneously on the cluster. Both of two factors are varied between 89 and 179. There are 4 different combination, and each one with 3 replicates for each of 4 Steps in the routine. Under each combination, the RSC is varied with 5 values, 0.2, 0.5, 0.8, 0.9, and 1. Totally, there are 240 runs in the experiment.

Link to figure

Figure 5.29.: Log base 2 elapsed time against to RSC

In the Figure 5.29, the elapsed time of jobs are plotted against to the RSC conditional on the combination of the RTSK and the N_{cont} for each Step. For all the jobs, when the N_{cont} is 89, the optimal performance is obtained when the RSC is 0.8. But if the N_{cont} is increased to be 179, the optimal performance is obtained when the RSC is 0.5 for Step I, III, and V, but 0.8 for Step VII. Based on our finding about the RSC parameter, we propose a general formula to calculate the optimal value of RSC. Let N_{map} to be the total number of mappers of a given job, and N_{cont} to be the maximum number of containers can be running simultaneously on the cluster. Then we have

$$\text{RSC} = \min\left(0.8,\ \max\left(1 - \frac{N_{cont}}{N_{map}},\ 0\right)\right) \tag{5.2}$$

0.8 is a upper bound for RSC based on the empirical results in previous subsection. If the RSC is set to be larger than 0.8, then it force the coping process in reducers all start within a very short time period, which in turn hurts the performance of the

job because of the heavy load of network communication among servers. So for the Step VII, the optimal value of RSC in theory should be 0.8.

5.3.4 Statistical Factors

Since the cell factor only affect the MapReduce job in Step II and VI, there is not any interaction between it and either jump or any other Hadoop factors. A full factorial experiment is performed with one factor of three levels: 0.2, 0.5, and 0.7. Three replicates on each.

The `jump` factor only affect the MapReduce job in Step IV. A full factorial experiment is performed. Four levels of jump factor are 5, 10, 50, and 100. Each level of `jump` is applied to the Step IV with three replicates, and the results of elapsed time is shown in Figure 5.30. Elapsed time is plotted against to the reciprocal of `jump` value, and the mean over 3 replicates at each level are connected with a red line. The default value for jump in `stlplus` function is 10. By decreasing the jump value from 10 to 5, the elapsed time is only reduced by 24 seconds, which is about 3% of total elapsed time of Step IV. However, the elapsed time increases around 65 seconds when the jump is increased from 10 to 100. Considering the accuracy and the computation efficiency, we recommend to set the `jump` parameter in `stlplus` function to be 10.

Link to figure

Figure 5.30.: Log base 2 elapsed time against to Jump^{-1}

The `cell` factor while affect the MapReduce jobs in both Step II and Step VI. Four levels of cell factor are 0.2, 0.22. 0.26, and 0.28. Each level of cell is applied to both Step II and Step VI with three replicates, and the results of elapsed time is shown in Figure 5.31. In both Step II and Step VI, increasing the `cell` factor from 0.2 to 0.28 dramatically reduce the elapsed time about 34%. Even though keeping

increasing the `cell` to larger value can reduce the elapsed time even more than 45%, we do not recommend any value larger than 0.28 since that will explode the overall MSE of the final fitted values due to over smoothing caused by cubic interpolation.

Link to figure

Figure 5.31.: Log base 2 elapsed time against to Cell

5.4 Summary

Table 5.1.: Performance improvement

Step	I		III		V		VII	
RTSK	90	179	90	179	90	179	90	179
Default	612.3	532.0	467.0	478.4	1203.5	1186.8	1158.6	1087.9
Best	386.3	321.6	311.0	294.4	782.0	702.7	771.4	666.6
Diff	36.9%	39.6%	33.4%	38.5%	35%	40.8%	33.4%	38.7%

The results from the experiments are very promising. By tunning the Hadoop and statistical factors can dramatically boost the performance of the drSSTL routine. For the Hadoop factors, we find out the best setting and would like to compare with the default settings. The values of 8 Hadoop factors for both default and best setting are listed below:

Default ISM: 100 MB, SSP: 0.8, RSC: 0.05, SIBP: 0.7, SMP: 0.66, IBP: 0

Best ISM: 512 MB, SSP: 0.8, RSC: 0.5(0.8), SIBP: 0.9, SMP: 0.9, IBP: 1

The reason we list two values for the best setting of the RSC is that the optimal value of the RSC is different for the Step VII than others according to the formula

we described in previous subsection. For Step I, III, V, it is 0.5, but for Step VII it is 0.8. We calculate the mean of three replicates of each setting. And both 179 and 90 of the RTSK are considered since we always have to specify a RTSK for a MapReduce job. The default is 1, which is not reasonable for our case. The mean elapsed time for each job and each RTSK are listed in Table 5.1. The improvement of performance is huge! For RTSK is 179, which is the better choice, the elapsed time of all steps are reduced by 40% in total. Overall, the hole routine with 45 GB input dataset can be finished less than 2 hours.

REFERENCES

REFERENCES

[1] Ncdc: National climatic data center. http://www.ncdc.noaa.gov.

[2] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[3] Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.

[4] William S Cleveland, Susan J Devlin, and Eric Grosse. Regression by local fitting: methods, properties, and computational algorithms. *Journal of econometrics*, 37(1):87–114, 1988.

[5] Jianqing Fan. Design-adaptive nonparametric regression. *Journal of the American statistical Association*, 87(420):998–1004, 1992.

[6] Jianqing Fan. Local linear regression smoothers and their minimax efficiencies. *The Annals of Statistics*, pages 196–216, 1993.

[7] Trevor Hastie and Robert Tibshirani. Varying-coefficient models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 757–796, 1993.

[8] William S Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, 1979.

[9] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610, 1988.

[10] William S Cleveland and Eric Grosse. Computational methods for local regression. *Statistics and Computing*, 1(1):47–62, 1991.

[11] William S Cleveland and Clive Loader. Smoothing by local regression: Principles and methods. In *Statistical theory and computational aspects of smoothing*, pages 10–49. Springer, 1996.

[12] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[13] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[14] Ryan P Hafen. *Local regression models: Advancements, applications, and new methods*. PURDUE UNIVERSITY, 2010.

[15] William S Cleveland, Eric Grosse, and William M Shyu. Local regression models. *Statistical models in S*, pages 309–376, 1992.

[16] Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*, volume 43. CRC Press, 1990.

[17] William S Cleveland. Coplots, nonparametric regression, and conditionally parametric fits. *Lecture Notes-Monograph Series*, pages 21–36, 1994.

[18] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[19] Leo Breiman and Jerome H Friedman. Estimating optimal transformations for multiple regression and correlation. *Journal of the American statistical Association*, 80(391):580–598, 1985.

[20] Saptarshi Guha, Ryan Hafen, Jeremiah Rounds, Jin Xia, Jianfu Li, Bowei Xi, and William S Cleveland. Large complex data: divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012.

[21] Ryan Hafen, Luke Gosink, Jason McDermott, Karin Rodland, Kerstin Kleese-van Dam, and William S Cleveland. Trelliscope: a system for detailed visualization in the deep analysis of large complex data. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 105–112. IEEE, 2013.

[22] R core team et al. R: A Language and Environment for Statistical Computing, 2012.

[23] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[25] Saptarshi Guha. *Computing Environment for the Statistical Analysis of Large and Complex Data*. PhD thesis, Purdue University Department of Statistics, 2010.

[26] Noaa: National oceanic and atmospheric administration. `http://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/cooperative-observer-network-coop`.

[27] Nrsc: Natural resources conservation service. `http://www.wcc.nrcs.usda.gov/snow/`.

[28] Usda-nrsc: United states department of agriculture natural resources conservation service. `http://www.nrcs.usda.gov/wps/portal/nrcs/site/national/home/`.

[29] Mrcc: Midwestern regional climate center. `http://mrcc.isws.illinois.edu/data_serv/dataTypes.jsp`.

[30] Noaa: National oceanic and atmospheric administration. `http://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/us-historical-climatology-network-ushcn`.

[31] Cold sunday. `https://en.wikipedia.org/wiki/Cold_Sunday`.

[32] Sudipto Banerjee. On geodetic distance computations in spatial modeling. *Biometrics*, 61(2):617–625, 2005.

[33] Great-circle distance. `https://en.wikipedia.org/wiki/Great-circle_distance`.

[34] Stewart Fotheringham, Martin Charlton, and Chris Brunsdon. Geographically weighted regression: a natural evolution of the expansion method for spatial data analysis. *Environment and planning A*, 30(11):1905–1927, 1998.

[35] W. S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[36] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[37] Peter J Brockwell and Richard A Davis. Introduction to time series and forecasting, springer-verlag. *Inc., New York*, 2002.

[38] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[39] R package: datadr. `http://tessera.io/docs-datadr/rd.html`.

[40] Github. `https://github.com/XiaosuTong`.

[41] Apache hadoop yarn. `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`.

[42] Shrinivas B Joshi. Apache Hadoop Performance-tuning Methodologies and Best Practices. In *Proceedings of the third joint WOSP/SIPEW International Conference on Performance Engineering*, pages 241–242. ACM, 2012.

[43] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. Using Realistic Simulation for Performance Analysis of MapReduce Setups. In *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance*, pages 19–26. ACM, 2009.

[44] Guanying Wang, Ali Raza Butt, Prashant Pandey, and Karan Gupta. A Simulation Approach to Evaluating Design Decisions in MapReduce Setups. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pages 1–11. IEEE, 2009.

[45] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. The Performance of MapReduce: An in-depth Study. *Proceedings of the VLDB Endowment*, 3(1-2):472–483, 2010.

[46] Rice cluster. `https://www.rcac.purdue.edu/compute/rice/`.

APPENDICES

Package 'Spaloess'

June 4, 2016

Type Package

Title Spatially Local Polynomial Regression

Version 1.0

Date 2015-12-03

Description In the orginial loess, the weights for local fit is calculated based on distance between observations, which is defined as Euclid distance. However, in the Spatial Statistics, the distance between two locations is more reasonable to be defined in Great-circle distance. In this package, all functions of original loess are rewrote to include a distance feature to specify which distance calculation wants to be carried out.

License MIT

Suggests roxygen2 (>= 5.0.0)

Roxygen list(wrap = FALSE)

RoxygenNote 5.0.1

R topics documented:

`predloess` *Spatially Local Polynomial Regression Prediction*

Description

The first layer of the prediction of Spatial locally weighted regression. Mainly used as prediction function for the NAs in the original data set.

Usage

```
predloess(object, newdata = NULL, se = FALSE, na.action = na.pass, ...)
```

Arguments

object	an object fitted by 'spaloess'.
newdata	an optional data frame in which to look for variables with which to predict, or a matrix or vector containing exactly the variables needs for prediction. If missing, the original data points are used.
se	should standard errors be computed? Default is FALSE
na.action	function determining what should be done with missing values in data frame 'newdata'. The default is to predict 'NA'.
...	arguments passed to or from other methods.

Details

This is the first layer of prediction function of spatial locally weigted regression. In the spaloess function, NA will be removed from the fitting. By passing the spaloess object and NA observations to predloess, predction at the locations of NA is carried out.

When the fit was made using 'surface = "interpolate"' (the default), 'predloess' will not extrapolate - so points outside an axis-aligned hypercube enclosing the original data will have missing ('NA') predictions and standard errors.

Author(s)

Xiaosu Tong, based on 'loess' function of B. D. Ripley, and 'cloess' package of Cleveland, Grosse and Shyu.

Examples

```
set.seed(66)
x1 <- rnorm(100, mean=-100, sd=10)
x2 <- rnorm(100, mean=38, sd=4)
y <- 0.1*x1 + 1*x2 - 10 + rnorm(100, 0, 1.3); y[1:2] <- NA
testdata <- data.frame(LON = x1, LAT = x2, tmax = y)
cars.lo <- spaloess(tmax ~ LON + LAT, testdata, distance = "Latlong")
```

---

spaloess *Spatially Local Polynomial Regression Fitting*

Description

The first layer of the Spatial locally weighted regression, using local fitting with different type of distance calculation.

Usage

```
spaloess(formula, data, weights, subset, na.action, model = FALSE,
  napred = TRUE, span = 0.75, enp.target, degree = 2L,
  parametric = FALSE, distance = "Latlong", alltree = FALSE,
  drop.square = FALSE, normalize = FALSE, family = c("gaussian",
  "symmetric"), method = c("loess", "model.frame"),
  control = loess.control(...), ...)
```

Arguments

formula	a formula specifying the numeric response and one to four numeric predictors.
data	an optional data from, list or environment containing the variables in the model. If not found in 'data', the variables are taken from 'environment', typically the environment from which 'loess' is called.
weights	optional weights for each case
subset	an optional specification of a subset of the data to be used
na.action	the action to be taken with missing values in the response or predictors. The default is given by 'getOption("na.action")'.
model	Should the model frame be returned?
napred	Should missing observations in the dataset be predicted. Default is TRUE.
span	The parameter alpha which controls the portion of data points used in the local fit.
enp.target	An alternative way to specify 'span', as the approximate equivalent number of parameters to be used.
degree	The degree of the polynomials to be used, normally 1 or 2. (Degree 0 is also allowed, but see the 'Note'.)
parametric	should any terms be fitted globally rather than locally? Terms can be specified by name, number or as a logical vector of the same length as the number of predictors.
distance	Options: "Euclid", or "Latlong" which is for great circle distance
alltree	Should the kd-tree built based on all observations or only non-NA observations.
drop.square	For fits with more than one predictor and 'degree = 2', should the quadratic term be dropped for particular predictors? Terms are specified in the same way as for 'parametric'.
normalize	Should the predictors be normalized to a common scale if there is more than one? The normalization used is to set the 10 "Latlong" distance.
family	If 'gaussian' fitting is by least-squares, and if 'symmetric' a re-descending M estimator is used with Tukey's bi-weight function.
method	Fit the model or just extract the model frame.
control	control parameters: see 'loess.control'.
...	arguments passed to or from other methods.

Details

This spaloess function is the first wrapper of the spatial loess fitting procedure. It checks all the validity of all input arguments, and formats arguments like drop,square, parametric. Also generate other important arguments, like iteration, and pass all arguments into the second wrapper function: newsimpleLoess

Author(s)

Xiaosu Tong, based on 'loess' function of B. D. Ripley, and 'cloess' package of Cleveland, Grosse and Shyu.

Examples

```
set.seed(66)
x1 <- rnorm(100, mean=-100, sd=10)
x2 <- rnorm(100, mean=38, sd=4)
y <- 0.1*x1 + 1*x2 - 10 + rnorm(100, 0, 1.3)
testdata <- data.frame(LON = x1, LAT = x2, tmax = y)
cars.lo <- spaloess(tmax ~ LON + LAT, testdata, distance = "Latlong")
```

Index

Package 'drsstl'

August 8, 2016

Title Spatial Seasonal Trend Loess analysis routine using divide and recombined

Version 1.0

Description Apply loess smoothing to spatial-temporal data using divide and recombined concept. Divide the data either by time or by location, and then apply either spatial loess smoothing fit to the by time division or stlplus fit to the by location division.

Date 2016-06-06

Depends R (>= 3.2.1)

Imports datadr,
Spaloess,
roxygen2,
plyr,
Rcpp,
yaImpute,
stlplus,
testthat,
maps

License MIT + file LICENSE

LazyData true

RoxygenNote 5.0.1

Suggests knitr,
rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp

Remotes xiaosutong/Spaloess

R topics documented:

`drsstl` *Apply sstl routine to spatial-temporal dataset*

Description

sstl routine is applied to a spatial-temporal dataset either in memory or on HDFS.

Usage

```
drsstl(data, output = NULL, stat_info = NULL,
  model_control = spacetime.control(), cluster_control = NULL)
```

Arguments

`data`	The input path of raw text file on HDFS or a data.frame object in memory
`output`	The output path of fitting results on HDFS. If data is a data.frame object, the output should be set as default NULL. Since the drsstl function will return the fitting results in memory.
`stat_info`	The RData on HDFS which contains all station metadata. Make sure copy the RData of station_info to HDFS first using rhput.
`model_control`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`cluster_control`	Should be a list object generated from `mapreduce.control` function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters. It is only necessary for data on HDFS situation. If data is data.frame in memory, this parameter should be kept as default NULL.

Author(s)

Xiaosu Tong

Examples

```
## Not run:
    head(tmax_all)
    mcontrol <- spacetime.control(
      vari="tmax", n=576, n.p=12, stat_n=7738,
      s.window=13, t.window = 241, degree=2, span=0.015, Edeg=2
    )
    ccontrol <- mapreduce.control(
      libLoc= NULL, reduceTask=169, io_sort=128, slow_starts = 0.5,
      map_jvm = "-Xmx200m", reduce_jvm = "-Xmx200m",
      map_memory = 1024, reduce_memory = 1024,
      reduce_input_buffer_percent=0.4, reduce_parallelcopies=10,
      reduce_merge_inmem=0, task_io_sort_factor=100,
      spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.8,
      reduce_shuffle_merge_percent = 0.4
    )

    #If the data is on HDFS
    drsstl(
      data = "/tmp/tmax.txt", output = "/tmp/output",
      stat_info = "/tmp/station_info.RData", model_control = mcontrol,
      cluster_control=ccontrol
    )

    #If the data is tmax_all which is in memory
    drsstl(tmax_all, model_control=mcontrol, cluster_control=ccontrol)

## End(Not run)
```

`mapreduce.control` *Set mapreduce parameter for drsstl fitting*

Description

Set control parameters of mapreduce for drsstl fits

Usage

```
mapreduce.control(reduceTask = 0, libLoc = NULL, BLK = 128,
  map_jvm = "-Xmx200m", reduce_jvm = "-Xmx200m", map_memory = 1024,
  reduce_memory = 1024, slow_starts = 0.5, spill_percent = 0.8,
  io_sort = 128, task_io_sort_factor = 100, reduce_parallelcopies = 5,
  reduce_shuffle_input_buffer_percent = 0.7,
  reduce_shuffle_merge_percent = 0.66, reduce_merge_inmem = 0,
  reduce_input_buffer_percent = 0, reduce_buffer_read = 150,
  map_buffer_read = 150, reduce_buffer_size = 10000,
  map_buffer_size = 10000)
```

Arguments

`reduceTask`	The reduce task number, also the number of output files. If set to be 0, then there is no shuffle and sort stage after map.
`libLoc`	The library path where each worker should go to load R packages. If all R packages have been push to HDFS, leave this parameter as default NULL.
`BLK`	The block size of output file on HDFS.
`map_jvm, reduce_jvm`	For mapreduce.map.java.opts and mapreduce.reduce.java.opts
`map_memory, reduce_memory`	For mapreduce.map.memory.mb and mapreduce.reduce.memory.mb
`slow_starts`	For mapreduce.job.reduce.slowstart.completedmaps
`spill_percent`	For mapreduce.sort.spill.percent parameter
`io_sort`	For mapreduce.task.io.sort.mb, the size, in megabytes, of the memory buffer to use while sorting map output.
`task_io_sort_factor`	For mapreduce.task.io.sort.factor
`reduce_parallelcopies`	For mapreduce.reduce.shuffle.parallelcopies
`reduce_shuffle_input_buffer_percent`	For mapreduce.reduce.shuffle.input.buffer.percent
`reduce_shuffle_merge_percent`	For mapreduce.reduce.shuffle.merge.percent
`reduce_merge_inmem`	For mapreduce.reduce.merge.inmem.shreshold
`reduce_input_buffer_percent`	For mapreduce.reduce.input.buffer.percent
`reduce_buffer_read, map_buffer_read, reduce_buffer_size, map_buffer_size`	For all rhipe arguments.

Value

A list with mapreduce tuning parameters.

Author(s)

Xiaosu Tong

Examples

```
mapreduce.control()
```

predNewLocs *Prediction at new locations based on the fitting results of original dataset*

Description

Prediction at new locations based on the fitting results of original dataset

Usage

```
predNewLocs(fitted, newdata, output = NULL, stat_info = NULL,
  model_control = spacetime.control(), cluster_control = NULL)
```

Arguments

fitted	Can be either a data.frame in memory or HDFS path which contains all fitting results of original dataset.
newdata	A data.frame includes all locations' longitude, latitude, and elevation, where the prediction is to be calculated.
output	The output path of fitting results on HDFS. If data is a data.frame object, the output should be set as default NULL. Since the function will return the fitting results in memory.
stat_info	The RData on HDFS which contains all station metadata. Make sure copy the RData of station_info to HDFS first using rhput.
model_control	Should be a list object generated from spacetime.control function. The list including all necessary smoothing parameters of nonparametric fitting.
cluster_control	Should be a list object generated from mapreduce.control function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters. It is only necessary for data on HDFS situation. If data is data.frame in memory, this parameter should be kept as default NULL.

Author(s)

Xiaosu Tong

See Also

spacetime.control, mapreduce.control

Examples

```
## Not run:
    mcontrol <- spacetime.control(
      vari="resp", time="date", n=576, n.p=12, stat_n=7738, surf = "interpolate",
      s.window="periodic", t.window = 241, degree=2, span=0.015, Edeg=2
    )
```

```
ccontrol <- mapreduce.control(
  libLoc= NULL, reduceTask=169, io_sort=128, slow_starts = 0.5,
  map_jvm = "-Xmx200m", reduce_jvm = "-Xmx200m",
  map_memory = 1024, reduce_memory = 1024,
  reduce_input_buffer_percent=0.4, reduce_parallelcopies=10,
  reduce_merge_inmem=0, task_io_sort_factor=100,
  spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.8,
  reduce_shuffle_merge_percent = 0.4
)
new.grid <- expand.grid(
  lon = seq(-126, -67, by = 0.5),
  lat = seq(25, 49, by = 0.5)
)
instate <- !is.na(map.where("state", new.grid$lon, new.grid$lat))
new.grid <- new.grid[instate, ]

elev.fit <- spaloess( elev ~ lon + lat,
  data = station_info,
  degree = 2,
  span = 0.015,
  distance = "Latlong",
  normalize = FALSE,
  napred = FALSE,
  alltree = FALSE,
  family="symmetric",
  control=loess.control(surface = "direct")
)
grid.fit <- predloess(
  object = elev.fit,
  newdata = data.frame(
    lon = new.grid$lon,
    lat = new.grid$lat
  )
)
new.grid$elev2 <- log2(grid.fit + 128)

#if the original fitting results are in memory
fitted <- drsstl(
  data=tmax_all,
  output=NULL,
  stat_info="station_info",
  model_control=mcontrol
)
predNewLocs(
  original = fitted, newdata = new.grid, model_control = mcontrol
)

#if the fitting results are on HDFS
predNewLocs(
  fitted="/tmp/output/output_bymth", newdata=new.grid, output = "/tmp",
  station_info="/tmp/station_info.RData", model_control = mcontrol,
  cluster_control = ccontrol
)
```

```
## End(Not run)
```

---

predNew_local *Prediction at new locations based on the fitting results in memory.*

Description

The prediction at new locations are calculated based on the fitting results saved in memory based on the original dataset.

Usage

```
predNew_local(original, newdata, mlcontrol = spacetime.control())
```

Arguments

original	The data.frame which contains all fitting results of original dataset. The data.frame is saved in memory, not on HDFS.
newdata	A data.frame includes all locations' longitude, latitude, and elevation, where the prediction is to be calculated.
mlcontrol	Should be a list object generated from spacetime.control function. The list including all necessary smoothing parameters of nonparametric fitting.

Author(s)

Xiaosu Tong

See Also

spacetime.control, mapreduce.control

Examples

```
## Not run:
    library(maps)
    library(Spaloess)
    library(datadr)
    new.grid <- expand.grid(
      lon = seq(-126, -67, by = 1),
      lat = seq(25, 49, by = 1)
    )
    instate <- !is.na(map.where("state", new.grid$lon, new.grid$lat))
    new.grid <- new.grid[instate, ]

    elev.fit <- spaloess( elev ~ lon + lat,
      data = station_info,
      degree = 2,
```

```
  span = 0.05,
  distance = "Latlong",
  normalize = FALSE,
  napred = FALSE,
  alltree = FALSE,
  family="symmetric",
  control=loess.control(surface = "direct")
)
grid.fit <- predloess(
  object = elev.fit,
  newdata = data.frame(
    lon = new.grid$lon,
    lat = new.grid$lat
  )
)
new.grid$elev <- grid.fit

n <- 5000 # just use 5000 stations as example
set.seed(99)
first_stations <- sample(unique(tmax_all$station.id), n)
small_dt <- subset(tmax_all, station.id %in% first_stations)
small_dt$station.id <- as.character(small_dt$station.id)
small_dt$month <- as.character(small_dt$month)
mlcontrol <- spacetime.control(
  vari="tmax", time="date", n=576, n.p=12, stat_n=n, surf = "interpolate",
  s.window="periodic", t.window = 241, degree=2, span=0.75, Edeg=0
)

fitted <- drsstl(
  data=small_dt,
  output=NULL,
  model_control=mlcontrol
)
rst <- predNew_local(
  original = recombine(fitted, combRbind), newdata = new.grid, mlcontrol = mlcontrol
)

## End(Not run)
```

---

predNew_mr *Prediction at new locations based on the fitting results on HDFS.*

Description

The prediction at new locations are calculated based on the fitting results saved on HDFS based on the original dataset.

Usage

```
predNew_mr(newdata, input, output, info, mlcontrol = spacetime.control(),
  clcontrol = mapreduce.control())
```

Arguments

newdata	A data.frame includes all locations' longitude, latitude, and elevation, where the prediction is to be calculated.
input	The path of input file on HDFS. It should be by-month division with all fitting results of original dataset
output	The path of output on HDFS where all the intermediate outputs will be saved.
info	The RData path on HDFS which contains all station metadata of original dataset
mlcontrol	Should be a list object generated from spacetime.control function. The list including all necessary smoothing parameters of nonparametric fitting.
clcontrol	Should be a list object generated from mapreduce.control function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.

Author(s)

Xiaosu Tong

See Also

spacetime.control, mapreduce.control

Examples

```
## Not run:
   clcontrol <- mapreduce.control(libLoc=NULL, reduceTask=95, io_sort=100, slow_starts = 0.5)
    mlcontrol <- spacetime.control(
      vari="resp", time="date", n=576, n.p=12, stat_n=7738,
      s.window="periodic", t.window = 241, degree=2, span=0.015, Edeg=2
    )

    new.grid <- expand.grid(
      lon = seq(-126, -67, by = 0.1),
      lat = seq(25, 49, by = 0.1)
    )
    instate <- !is.na(map.where("state", new.grid$lon, new.grid$lat))
    new.grid <- new.grid[instate, ]

    elev.fit <- spaloess( elev ~ lon + lat,
      data = station_info,
      degree = 2,
      span = 0.015,
      distance = "Latlong",
      normalize = FALSE,
      napred = FALSE,
      alltree = FALSE,
      family="symmetric",
      control=loess.control(surface = "direct")
    )
    grid.fit <- predloess(
```

```
      object = elev.fit,
      newdata = data.frame(
        lon = new.grid$lon,
        lat = new.grid$lat
      )
    )
    new.grid$elev2 <- log2(grid.fit + 128)

    predNew_mr(
      newdata=new.grid, input="/tmp/output_bymth", output = "/tmp",
      info="/tmp/station_info.RData", mlcontrol=mlcontrol, clcontrol=clcontrol
    )

## End(Not run)
```

`readIn` *Readin Raw text data files and save it as by time division on HDFS.*

Description

Input raw text data file is download from NCDC, and is available in the drsstl package in ./inst/extdata. It is read in and divided into by-month division saved on HDFS

Usage

```
readIn(input, output, info, cluster_control = mapreduce.control(),
  model_control = spacetime.control(), cshift = 1)
```

Arguments

`input`	The path of input file on HDFS. It should be raw text file.
`output`	The path of output file on HDFS. It is by time division.
`info`	The RData on HDFS which contains all station metadata. Make sure copy the RData of station_info.RData, which is also available in the drsstl package, to HDFS first using rhput.
`cluster_control`	all parameters that are needed for mapreduce job
`model_control`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`cshift`	number of columns to be shifted when reading raw text file

Author(s)

Xiaosu Tong

Examples

```
## Not run:
    rhput("./station_info.RData", "/tmp/station_info.RData")
    FileInput <- "/tmp/tmax.txt"
    FileOutput <- "/tmp/bymth"
    ccontrol <- mapreduce.control(
      libLoc=NULL, reduceTask=5, io_sort=100, slow_starts = 0.5,
      reduce_input_buffer_percent=0.9, reduce_parallelcopies=5,
      spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.9,
      reduce_shuffle_merge_percent = 0.5
    )
    readIn(
      FileInput, FileOutput, info="/tmp/station_info.RData", cluster_control=ccontrol
    )

## End(Not run)
```

---

spacetime.control *Set smoothing parameters for the drsstl fitting*

Description

Set control parameters for the smoothing fit of stl and spatial smoothing

Usage

```
spacetime.control(vari = "resp", time = "date", n, stat_n, n.p = 12,
  s.window, s.degree = 1, t.window = NULL, t.degree = 1, inner = 2,
  outer = 1, mthbytime = 1, s.jump = 10, t.jump = 10, cell = 0.2,
  degree, span, Edeg, surf = c("direct", "interpolate"),
  family = c("symmetric", "gaussian"), siter = 2)
```

Arguments

vari	variable name in string of the response variable. The default is "resp"
time	variable name in string of time index of the whole time series. The default is "date". In the final results on HDFS, the index of time will be changed to this variable instead of year and month.
n	the number of total observations in the time series at each location.
stat_n	The number of stations.
n.p	the number of observations in each subseries. It should be 12 for monthly data for example.
s.window	either the character string "periodic" or the span (in lags) of the loess window for seasonal extraction, which should be odd. This has no default.
s.degree	degree of locally-fitted polynomial in seasonal extraction. Should be 0, 1, or 2.

t.window	the span (in lags) of the loess window for trend extraction, which should be odd. If NULL, the default, nextodd(ceiling((1.5*period) / (1-(1.5/s.window)))), is taken.
t.degree	degree of locally-fitted polynomial in trend extraction. Should be 0, 1, or 2.
inner	The iteration time for inner loop of stlplus for time dimension fitting
outer	The iteration time for outer loop of stlplus for time dimension fitting
mthbytime	The number of months will be grouped together in the by time division after swaptoTime. The parameter is only used for swaptoTime. Since there may be to many time point in each location, the swaptoTime looping all time point will add to much overhead caused by rhcollect. So every mthbytime time point are collect into one key-value pair. It is save to leave it as default 1. If the time series is extremely long, it can be set to be 2.
s.jump, t.jump	integers at least one to increase speed of the respective smoother. Linear interpolation happens between every '*.jump'th value.
cell	if interpolation is used this controls the accuracy of the approximation via the maximum number of points in a cell in the kd-tree. Cells with more than 'floor(n*span*cell)' points are subdivided.
degree	smoothing degree for the spatial loess smoothing. It can be 0, 1, or 2.
span	smoothing span for the spatial loess smoothing.
Edeg	the degree for the conditioanl parametric model including elevation.
surf	should the fitted surface be computed exactly or via interpolation from a kd tree.
family	if '"gaussian"' fitting is by least-squares, and if '"symmetric"' a re-descending M estimator is used with Tukey's biweight function.
siter	the number of iterations used for the spatial smoothing procedure if family is set to be '"symmetric"', which is for robust fitting.

Value

A list with space-time fitting parameters.

Author(s)

Xiaosu Tong

References

R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning (1990) STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, **6**, 3–73.

Examples

```
spacetime.control(
 n = 576, stat_n = 7738, n.p = 12, s.window = 21, s.degree = 1, t.window = 241,
  t.degree = 1, degree = 2, span = 0.015, Edeg = 2, surf = "interpolate"
)
```

`spaofit` *Apply the spatial loess fitting to the original observations at all locations in each month in parallel*

Description

Call `spaloess` function on the spatial domain in each month in parallel. Every spatial domain uses the same smoothing parameters. NA observations will be imputed.

Usage

```
spaofit(input, output, info, model_control = spacetime.control(),
  cluster_control = mapreduce.control())
```

Arguments

`input`	The path of input sequence file on HDFS. It should be by-month division.
`output`	The path of output sequence file on HDFS. It is also by-month division but with seasonal and trend components
`info`	The RData on HDFS which contains all station metadata. Make sure copy the RData of station_info to HDFS first using rhput.
`model_control`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`cluster_control`	Should be a list object generated from `mapreduce.control` function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.

Author(s)

Xiaosu Tong

Examples

```
## Not run:
    FileInput <- "/tmp/bymth"
    FileOutput <- "/tmp/bymthfit"
    ccontrol <- mapreduce.control(libLoc=NULL, reduceTask=0)
    mcontrol <- spacetime.control(
      vari="resp", time="date", n=576, n.p=12, stat_n=7738,
      s.window=13, t.window = 241, degree=2, span=0.015, Edeg=2
    )

    spaofit(
      FileInput, FileOutput,
      info="/tmp/station_info.RData",
      model_control=mcontrol, cluster_control=ccontrol
```

```
    )

## End(Not run)
```

`sparfit` *Apply the spatial loess fitting on the remainder at all location for each month in parallel*

Description

Call `spaloess` function on the spatial domain at each time point in parallel. Every spatial domain uses the same smoothing parameters

Usage

```
sparfit(input, output, info, model_control = spacetime.control(),
  cluster_control = mapreduce.control())
```

Arguments

`input`	The path of input file on HDFS. It should be by-month division with temporal fitting results.
`output`	The path of output file on HDFS. It is by-month division but with added spatial fitted value for remainder, named Rspa.
`info`	The RData path on HDFS which contains all station metadata
`model_control`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`cluster_control`	Should be a list object generated from `mapreduce.control` function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.

Author(s)

Xiaosu Tong

Examples

```
## Not run:
    FileInput <- "/tmp/bymthse"
    FileOutput <- "/tmp/bymthfitse"
    ccontrol <- mapreduce.control(libLoc=NULL, reduceTask=0, BLK=128)
    mcontrol <- spacetime.control(
      vari="remainder", time="date", n=786432, n.p=12,
      s.window=13, t.window = 241, degree=2, span=0.015, Edeg=2
    )
```

```
  sparfit(
    FileInput, FileOutput, info="/tmp/station_info.RData",
    model_control=mcontrol, cluster_control=ccontrol
  )

## End(Not run)
```

`sstl_local` *Apply sstl routine to a data.frame of spatial-temporal dataset in the memory.*

Description

Assuming data has been read into the memory as a data.frame. Each row of the data.frame contains the observation in a given month at given location. Month index and station location information are saved in 5 columns of the data.frame. The name of these columns should be "lon", "lat", "elev", "year", and "month".

Usage

```
sstl_local(data, mlcontrol = spacetime.control())
```

Arguments

`data`	The input data.frame which contains observation, lon, lat, elev, and year, month
`mlcontrol`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.

Author(s)

Xiaosu Tong

Examples

```
head(tmax_all)
n <- 1000 # just use 1000 stations as example
set.seed(99)
first_stations <- sample(unique(tmax_all$station.id), n)
small_dt <- subset(tmax_all, station.id %in% first_stations)
small_dt$station.id <- as.character(small_dt$station.id)
small_dt$month <- as.character(small_dt$month)

mcontrol <- spacetime.control(
  vari="tmax", n=576, n.p=12, stat_n=n, surf = "interpolate",
  s.window=13, t.window = 241, degree=2, span=0.15, Edeg=0
)

rst <- sstl_local(small_dt, mlcontrol=mcontrol)
```

`sstl_local_dr` *Apply sstl routine to a data.frame of spatial-temporal dataset in the memory using datadr.*

Description

Assuming data has been read into the memory as a data.frame. Each row of the data.frame contains the observation in a given month at given location. Month index and station location information are saved in 5 columns of the data.frame. The name of these columns should be "lon", "lat", "elev", "year", and "month".

Usage

```
sstl_local_dr(data, mlcontrol = spacetime.control(), outdiv)
```

Arguments

`data`	The input data.frame which contains observation, lon, lat, elev, and year, month
`mlcontrol`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`outdiv`	Specify the output division type.

Author(s)

Xiaosu Tong

Examples

```
head(tmax_all)
n <- 100 # just use 1000 stations as example
set.seed(99)
first_stations <- sample(unique(tmax_all$station.id), n)
small_dt <- subset(tmax_all, station.id %in% first_stations)
small_dt$station.id <- as.character(small_dt$station.id)
small_dt$month <- as.character(small_dt$month)

mcontrol <- spacetime.control(
  vari="tmax", n=576, n.p=12, stat_n=n, surf = "interpolate",
  s.window=13, t.window = 241, degree=2, span=0.35, Edeg=0
)

rst <- sstl_local_dr(small_dt, mlcontrol=mcontrol, outdiv="loc")
```

`sstl_mr` *Apply sstl routine to dataset saved on HDFS*

Description

Input raw text file on HDFS is the original input dataset. After a series of MapReudce jobs, the final fitting results are saved as output_bymth and output_bystat subdirectory inside of output path on HDFS.

Usage

```
sstl_mr(input, output, stat_info, mlcontrol = spacetime.control(),
  clcontrol = mapreduce.control())
```

Arguments

`input`	The input path of raw text file on HDFS
`output`	The output path of final fitting results on HDFS.
`stat_info`	The RData on HDFS which contains all station metadata. Make sure copy the RData of station_info to HDFS first using rhput.
`mlcontrol`	Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.
`clcontrol`	Should be a list object generated from `mapreduce.control` function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.

Author(s)

Xiaosu Tong

Examples

```
## Not run:
    mcontrol <- spacetime.control(
      vari="tmax", n=576, n.p=12, stat_n=7738,
      s.window=13, t.window = 241, degree=2, span=0.015, Edeg=2
    )
    ccontrol <- mapreduce.control(
      libLoc= NULL, reduceTask=169, io_sort=512, BLK=128, slow_starts = 0.5,
      map_jvm = "-Xmx200m", reduce_jvm = "-Xmx200m",
      map_memory = 1024, reduce_memory = 1024,
      reduce_input_buffer_percent=0.4, reduce_parallelcopies=10,
      reduce_merge_inmem=0, task_io_sort_factor=100,
      spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.8,
      reduce_shuffle_merge_percent = 0.4
    )
    sstl_mr(
```

```
    input = "/tmp/tmax.txt", output = "/tmp/output",
    stat_info = "/tmp/station_info.RData", mlcontrol = mcontrol,
    clcontrol = clcontrol
  )

## End(Not run)
```

`station_info` *Metadata of Stations*

Description

Metadata for all stations including longitude, latitude, elevation, station.id

Source

All station metadata: http://www.image.ucar.edu/Data/US.monthly.met/FullData.shtml#temp

---

`stlfit` *Apply the stlplus fitting at each location in parallel*

Description

Calling `stlplus` function from Ryan Hafen's `stlplus` package on time series at each location in parallel. Every station uses the same smoothing parameter

Usage

```
stlfit(input, output, model_control = spacetime.control(),
  cluster_control = mapreduce.control())
```

Arguments

`input`	The path of input sequence file on HDFS. It should be by location division.
`output`	The path of output sequence file on HDFS. It is by location division but with seasonal and trend components
`model_control`	The list contains all smoothing parameters
`cluster_control`	A list contains all mapreduce tuning parameters.

Author(s)

Xiaosu Tong

References

R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning (1990) STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, **6**, 3–73.

Ryan Hafen (2010): stlplus: Local regression models: Advancements, applications, and new methods. *Purdue University*

See Also

spacetime.control, mapreduce.control

Examples

```
## Not run:
    FileInput <- "/tmp/bystat"
    FileOutput <- "/tmp/bystatfit"
    ccontrol <- mapreduce.control(libLoc=NULL, reduceTask=0)
    mcontrol <- spacetime.control(
     vari = "resp", time = "date", n = 576, stat_n=7738, n.p = 12, s.window = "periodic",
      t.window = 241, degree = 2, span = 0.015, Edeg = 2
    )
    stlfit(FileInput, FileOutput, model_control=mcontrol, cluster_control=ccontrol)

## End(Not run)
```

---

swaptoLoc *Swap to division by-location*

Description

Switch input key-value pairs which is division by-month to the key-value pairs which is division by-location.

Usage

```
swaptoLoc(input, output, final = FALSE,
  cluster_control = mapreduce.control(),
  model_control = spacetime.control())
```

Arguments

input	The path of input file on HDFS. It should be by-month division.
output	The path of output file on HDFS. It is by-location division.
final	There two steps of switching to by-location division in the routine. In the first one, which final is set to be FALSE, the intermediate value is vectorized to minimize the size. In the second one, which final is set to be TRUE, the output value is saved as data.frame.

`cluster_control`
: Should be a list object generated from `mapreduce.control` function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.

`model_control`
: Should be a list object generated from `spacetime.control` function. The list including all necessary smoothing parameters of nonparametric fitting.

Author(s)

Xiaosu Tong

See Also

`spacetime.control`, `mapreduce.control`

Examples

```
## Not run:
    FileInput <- "/tmp/bymthfit"
    FileOutput <- "/tmp/bystat"
    ccontrol <- mapreduce.control(
      libLoc=NULL, reduceTask=5, io_sort=128, slow_starts = 0.5,
      reduce_input_buffer_percent=0.4, reduce_parallelcopies=10,
      reduce_merge_inmem=0, task_io_sort_factor=100,
      spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.8,
      reduce_shuffle_merge_percent = 0.4
    )
    mcontrol <- spacetime.control(
      vari="resp", time="date", n=576, n.p=12, stat_n=7738,
      s.window=13, t.window = 241, degree=2, span=0.015, Edeg=2
    )
    swaptoLoc(FileInput, FileOutput, cluster_control=ccontrol, model_control=mcontrol)

## End(Not run)
```

`swaptoTime` *Swap to division by-month from by-location division*

Description

Switch input key-value pairs which is division by-location to the key-value pairs which is division by-month.

Usage

```
swaptoTime(input, output, cluster_control = mapreduce.control(),
  model_control = spacetime.control())
```

Arguments

input	The path of input file on HDFS. It should be by-location division.
output	The path of output file on HDFS. It is by-month division.
cluster_control	Should be a list object generated from mapreduce.control function. The list including all necessary Rhipe parameters and also user tunable MapReduce parameters.
model_control	Should be a list object generated from spacetime.control function. The list including all necessary smoothing parameters of nonparametric fitting.

Details

swaptoTime is used for switching division by-location to division by-month. The input key is location index, and input value is a vectorized matrix with Mlcontrol$n rows and 3 columns in order of smoothed, seasonal, trend. For each row of matrix, a new key-value pair is generated. Since the matrix is vectorized by column, the trend in ith row is i+Mlcontrol$n. Index j controls the index of multiple location in one time point.

Author(s)

Xiaosu Tong

See Also

spacetime.control, mapreduce.control

Examples

```
## Not run:
    FileInput <- "/tmp/bystatfit"
    FileOutput <- "/tmp/bymthse"

    ccontrol <- mapreduce.control(
      libLoc=NULL, reduceTask=5, io_sort=128, slow_starts = 0.5,
      reduce_input_buffer_percent=0.2, reduce_parallelcopies=10,
      reduce_merge_inmem=0, task_io_sort_factor=100,
      spill_percent=0.9, reduce_shuffle_input_buffer_percent = 0.7,
      reduce_shuffle_merge_percent = 0.5
    )
    mcontrol <- spacetime.control(
      vari = "resp", time = "date", n = 576, stat_n=7738, n.p = 12,
      s.window = "periodic", t.window = 241,
      degree = 2, span = 0.015, Edeg = 2
    )

    swaptoTime(FileInput, FileOutput, cluster_control=ccontrol, model_control=mcontrol)

## End(Not run)
```

`tmax_all` *US Maximum Temperature*

Description

A data.frame contains maximum temperature data from NCAR weside

Source

Full data on NCAR: http://www.image.ucar.edu/Data/US.monthly.met/FullData.shtml#temp

VITA

VITA

Xiaosu Tong was born in Xichang, Sichuan, China in 1988. He earned his B.S. in Information and Calculation Science at College of Science, Shenyang Jianzhu University, China in 2010. Then he pursued his M.S. in Applied Statistics at Purdue University, Indiana in 2012. His academic interests include exploratory data analysis, data visualization, time series, and statistical computation with large and complex data.