Ashvini Fulpagar, Enze Wang
Mengkun Chen, Qingchuan Ji

Module 1

Introduction

In this project, our group want to predict when hard drives storage will reach its maximum capacity after given a time period. We want to use t (Time), y_t (drives storage), and maxcap (maximum capacity) to predict the t when y_t will reach maxcap. All code and examples can be found in our group GitHub: https://github.com/moslandwez/Module1_Group

Background

Our data is time series data. It contains stochastic behavior, server down, maintenance period and clean time. Therefore our data contains many invalid value. In our algorithm, our prediction will be conservative and strict because it will be dangerous for storage reach maximum capacity without cleaning. For invalid data such as NA and NULL, we will not try to impute for nobody know what happen in missing value, meanwhile, imputation may affect our prediction and it takes too much of time. To save more time, our algorithm will pay more attention to data from the end because servers history behaviour near the end of time are the most important for our prediction. Between two alternatives: linear regression and ARIMA, our groups choose the former, it is not only computation consuming to decide the parameters in ARIMA but also time consuming to predict the following y_t according to previous ones.

Algorithm

The algorithm description is in text, if you want to have a better understanding, please refer to our code. Our basic model is linear model:

$$y(t) = FinalSlope * t + FinalIntercept$$

by searching the largest and best slope from the end of data with as less time as possible. The kernel part of our algorithm is how to find the best FinalSlope and FinalIntercept by cutting appropriate section of the whole data.

LR(X,Y): The kernel function our algorithm based is linear regression function LR(X,Y) whose input is X and Y, and output is slope and intercept. This function is designed by our group instead of lm() from R.

PartLinearRegression(X,Y): Based on our linear regression function, we define a function called PartLinearRegression(X,Y), For a piece of X, Y. If its length is smaller than 30, it will just use the linear regression. Else, it will cut the data into 4 equal parts. [0,1/4], [1/4,1/2], [1/2,3/4] and [3/4,1]. We will calculate the slope from [1/2,3/4] and [3/4,1] of X and Y first. As our assumption mentioned before, the default slope must be positive, if positive slope exists, return the maximum of them; if not, this function will calculate the slope from [0,1/4] and [1/4,1/2] part and return the maximum. This design aims at nonlinearity data such as exponential shape, and it can avoid large calculation of linear regression on the whole X and Y. The kernel of PartLinearRegression can be changed to any regression methods and fraction number can also be changed.

DetectJumpDown(i): This is another kernel function in our algorithm. As we mention before, there are many server clean and maintenance time where y_t decreasing sharply, which called great jump. With DetectJumpDown, any sharp decreasing can be found and saved in a list called GreatJump. After that function will also clean too short section cut by GreatJump. Our rule for data partition is very complex and strict to avoid accidental cut from outliers. Our rule contains coarse and fine filter which can save a lot of time. You can refer to our code for detail. So what about sharp increasing? For most of situations, sharp increasing caused by server maintenance usually follows sharp decreasing, which occur in the start of data. Therefore our PartLinearRegression can usually avoid the impact of sharp increasing, so we will not find the sharp increasing.

With the function before, the main part of function will run a loop from the end to start of data. For any section cut by great jump, we calculate the slope with PartLinearRegression, if no positive slope return, move to the next section until no positive slope we just return 0. In the majority situation our function will return output in the first section near the end. By this design, not only our function can handle server maintenance and clean period, but can also has good performance in nonlinearity data.

Algorithm Performance

Besides examples given by professor, our groups also design other data sets to test our algorithm performance. In 1-7 pictures, blue line is maxcap, red line is the our fitting line and orange line is that from linear regression given by professor. The final picture is speed and scalability test between them, red dot is from our algorithm and orange one is from professor's example.

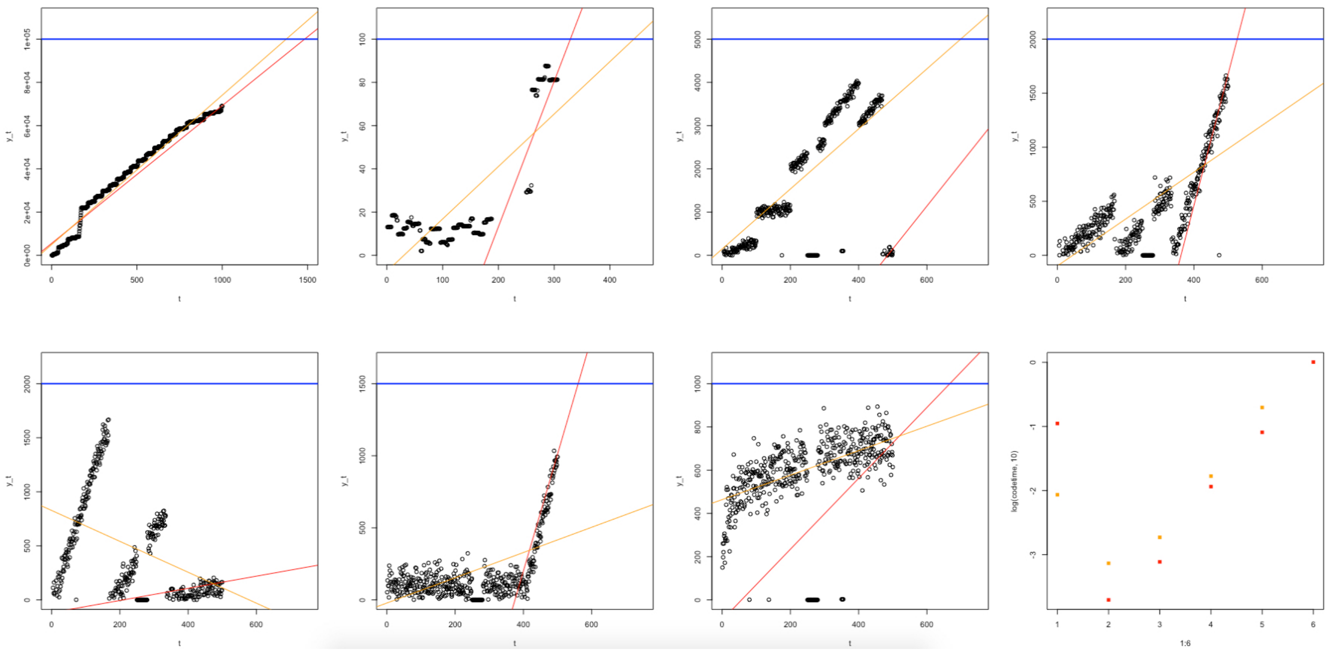

Figure 1: Algorithm performance on Data Sets

Our algorithm has good performance in piece-wise linear, exponential style data, but work a little bad in logarithmic data (Pic 7). In Youtube and box data, estimated time reaching maxcap are 1480 and 327, meet the example assumption. In speed test given by professor's code, our algorithm will finish in 0.117s, compared with 0.177s of professor's example.

Summary

For the problem in module 1, our groups built an algorithm based on simple linear regression: $y(t) = FinalSlope * t + FinalIntercept$ with cutting data by DetectJumpDown and fitting data by PartLinearRegression.

Strength: Our algorithm is a simple one, which works well at linear, piece-wise linear and exponential style data, it also has space for further development and update. The parameter in our algorithm can also be fixed for different kinds of data from different kinds of user habit of storage servers. And you can change both the rule of data partition and the regression method.

Weakness: Our algorithm may work not well for high frequency data. Our algorithm has many parameters, which is also a kind of weakness, to meet a type of user habit our algorithm may need to change the parameters.