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 behaviors, 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 knows what happened in missing value. Meanwhile, imputation may affect our prediction and it takes too much 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:

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

It searches the largest and best slope from the end of data spending as little time as possible. The kernel part of our algorithm is how to find the best FinalSlope and FinalIntercept by cutting appropriate sections in 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, traditional linear regression will be implemented. Otherwise, we 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 slopes 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 slopes exist, return the maximum of them; if not, this function will calculate the slopes within [0,1/4] and [1/4,1/2] part and return the maximum. This design aims at nonlinear 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 replaced by any regression methods and fraction number can also be changed.

DetectJumpDown(i): This is another kernel function in our algorithm. As we mentioned before, there are many server clean and maintenance time where y_t decreases sharply, which are called great jump. With DetectJumpDown, any sharp decrease 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 resulting from outliers. Our rule contains coarse and fine filter which can save a lot of time. You can refer to our code for details. So what about sharp increase? For most situations, sharp increase caused by server maintenance usually follows sharp decrease, which occur at the start of data. Therefore our PartLinearRegression can usually avoid the impact of sharp increase, so we will not find the sharp increase.

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 returns, move to the next section. If there is no positive slope we just return 0. In most situations our function

will return output in the first section near to the end. By this design, not only our function can handle server maintenance and clean period, but can also perform well when there is no linear trend in the data.

Algorithm Performance

Additional to the examples given by professor, our group also designed some other data sets to test our algorithm performance. From the first seven figures, blue line represents maxcap, red line is the fitted line and green line is doing linear regression on the whole data set given by professor. The final figure shows the comparison of speed and scalability test between the two methods. Red dots shows the performance of our algorithm and green ones are the counterpart from professor. Note that our prediction will start from end of data with final slope instead of intercept directly from regression.

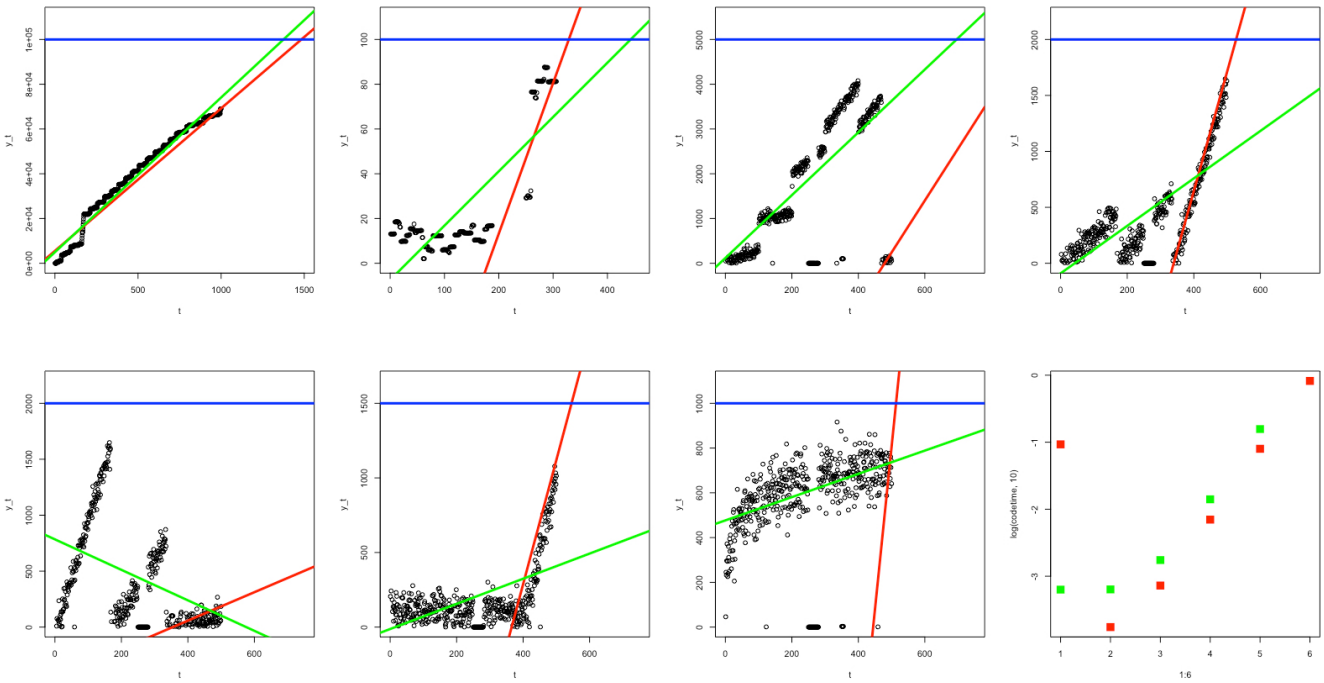

Figure 1: Algorithm performance between our function (Red) with lm (Green)

Our algorithm has better performance in piece-wise linear and exponential data, while works a little worse in logarithmic data (Pic 7). In Youtube and box data, estimated time reaching maxcap are 1481 and 327, which meet the example assumption. In speed test, our algorithm finishes in 0.076s, compared with 0.177s from professor's example.

Summary

For the problem in module 1, our group 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, performing well in linear, piece-wise linear and exponential data. however, there's still some space for further development and update. Parameters in our algorithm can also be adjusted under different data from various storage servers. Also, you can change both the rules of data partition and the regression methods.

Weakness: Our algorithm may not perform well for high frequency data. Our algorithm has too many parameters, which is another weakness. To meet various patterns, we need to adjust the parameters for specific situations.