Online Outlier Detection

Ratnik Gandhi, Ativ Joshi, Pratik Padalia

Contents

1 Introduction

Considering the scenario for real time data we propose an online algorithm to compute the outlier in constant time upon its arrival for a 2-D data series; where x is the periodic series of time stamp for each of the data point and y is the value of the corresponding observation value.

The general pattern of a dataset is maintained in the form of a line which fits the data best. Considering the line $y = a_0 + a_1x + a_2x^2 + \dots$ we need to find the best fit equation for all the data points, using the matrix form $Y = X\beta$.

We consider a matrix $Y_{n\times 1}$ where n is the number of observed data points received till that time instant. y_i represents ith observed data value.

$$Y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}^T$$

The matrix $X_{n\times p}$ represents the variable term of the equation of line, where x_i represents the timestamp of ith observation.

$$X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{bmatrix}^T$$

Then we have matrix $\beta_{p\times 1}$ where p is the number of coefficients of the line of best fit taken into consideration. β_i represents the coefficient of the ith term of the equation of line.

$$Y_{n\times 1} = X_{n\times p}\beta_{p\times 1} + \epsilon_{n\times 1}$$

where ϵ is the residue or error.

$$\therefore Y - \epsilon = X\beta$$

$$\therefore \hat{Y} = X\beta \tag{1.1}$$

Now, consider the equation $Y = X\beta$. Here, this equation has no solution as the datapoints do not lie on a line. Treat the equation as a least square problem.[1]

$$\begin{aligned} Y &= X\beta \\ \therefore\ X^TY &= X^TX\beta \\ \therefore\qquad \beta &= (X^TX)^{-1}X^TY \end{aligned} \tag{1.2}$$

Substituting 1.2 in 1.1,

$$\hat{Y} = X(X^TX)^{-1}X^TY$$
$$\hat{Y} = HY$$

where $H = X(X^TX)^{-1}X^T$ is called projection matrix.

The data point is to be classified as an outlier or not based on the distance from the general trend of the data set. We may consider different distances for various scenarios. Cook's Distance and Mahalanobis Distance suits the best here.

Cook's Distance gives the best results for general purpose data sets. Mahalanobis Distance however gives us a better accuracy for data set having a low number of outliers.

2 Cook's Distance

Cook's Distance of element i is given by

$$D_i = \frac{\Sigma_{j=1}^{n}(y_j - y_{j(i)})^2}{p.MSE} = \frac{(\beta - \beta^{(-i)})^T(X^TX)(\beta - \beta^{(-i)})}{p.MSE} \tag{2.1}$$

where,
y_j is the jth fitted response.
$y_{j(i)}$ is the jth fitted response without considering i.
MSE is mean square error.
p is number of coefficients in regression model.

Error is given by,

$$e = Y - \hat{Y} = (I - H)Y$$
$$So,\ MSE = n^{-1}e^Te$$

Making MSE online,

$$\begin{aligned} MSE &= \frac{1}{n} \times [(Y - X\beta)^T(Y - X\beta)] = \frac{1}{n} \times [(Y - \hat{Y})^T(Y - \hat{Y})] = \frac{1}{n} \times e^Te \\ &= \frac{1}{n} \times [(Y^T - \beta^TX^T)(Y - X\beta)] \\ &= \frac{1}{n} \times [(Y^TY)_{1\times1} - (Y^TX\beta)_{1\times1} - (\beta^TX^TY)_{1\times1} + \beta^TX^TX\beta_{\times1}] \end{aligned}$$

Makign β online,

$$\beta = \frac{1}{n\Sigma x_i^2 - (\Sigma x_i)^2}\begin{bmatrix} \Sigma y_i\Sigma x_i^2 - \Sigma x_i\Sigma y_ix_i \\ n\Sigma y_ix_i - \Sigma y_i\Sigma x_i \end{bmatrix} = \begin{bmatrix} \beta_1 & \beta_2 \end{bmatrix}^T$$

3 Mahalanobis Distance

Let $\vec{Z} = \begin{bmatrix} \vec{X} & \vec{Y} \end{bmatrix} = \begin{bmatrix} \vec{z_0} & \vec{z_1} & \dots & \vec{z_{n-1}} \end{bmatrix}^T$ where $\vec{X} = \begin{bmatrix} x_0 & x_1 & \dots & x_{n-1} \end{bmatrix}^T$ is the vector containing timestamps, $\vec{Y} = \begin{bmatrix} y_0 & y_1 & \dots & y_{n-1} \end{bmatrix}^T$ is the vector containing corresponding observed value and $\vec{z_i} = (x_i, y_i)$. (Each number in X can be treated as index and each corresponding number in Y can be treated as value).

Then, Mahalanobis Distance of $\vec{z_i} = (x_i, y_i)$ is given by

$$D_M(\vec{z_i}) = \sqrt{(\vec{z_i} - \vec{\mu})S^{-1}(\vec{z_i} - \vec{\mu})}$$

where,

$\vec{\mu} = mean(\vec{Z}) = [\ mean(\vec{X})\ mean(\vec{Y})\ ]$ is the mean of all elements of $\vec{X}$ and $\vec{Y}$.

S is the covariance matrix $S = \begin{bmatrix} cov(X,X) & cov(X,Y) \\ cov(Y,X) & cov(Y,Y) \end{bmatrix}$.

3.1 Calculate mean in online manner

Given the mean $\vec{\mu}$ of the points $\vec{Z_0}$ to $\vec{Z_{n-1}}$, we can find the mean $\vec{\mu'}$ of points $\vec{Z_0}$ to $\vec{Z_n}$ by

$$\vec{\mu'} = \frac{n\vec{\mu} + \vec{Z_n}}{n+1}$$

3.2 Calculating Online covariance

$$cov(\vec{X}, \vec{Y}) = \frac{1}{n}\Sigma_{i=1}^{n}(x_i - E[\vec{X}])(y_i - E[\vec{Y}])$$

Let, $E[\vec{X}] = e_X$ and $E[\vec{Y}] = e_Y$.

Also let $\vec{E}_X = [\ e_X\ e_X\ \dots\ e_X\ ]^T$ and $\vec{E}_Y = [\ e_Y\ e_Y\ \dots\ e_Y\ ]^T$

So,

$$\begin{aligned} cov(X,Y) &= (\vec{Y}^T - \vec{E}_Y^T)(\vec{X} - \vec{E}_X)\frac{1}{n} \\ &= \vec{Y}^T\vec{X} - \vec{Y}^T\vec{E}_X - \vec{E}_Y^T\vec{X} + \vec{E}_Y^T\vec{E}_X \\ &= \Sigma_{i=1}^{n}x_iy_i - e_X\Sigma_{i=1}^{n}y_i - e_Y\Sigma_{i=1}^{n}x_i + e_Xe_Yn \end{aligned}$$

4 Feeding the code on hardware

4.1 Advantages and Challanges

Advantages :

• Creating an application specific hardware help the hardware to be efficient in terms of memory as well as computationally efficient.

•Reduced cost for hardware.

•The application and its hardware is scalable.

Challenges :

•The memory constraints are currently set lower but are scalable.

•The precision is also compromised a bit however it also can be scaled to reduce the error.

•Division is difficult on hardware.

4.2 Verilog Code Blocks Description

4.2.1 top_module

The top most input output module which acts as the main block calling the other sub blocks.

INPUTS :

clk[1 bit] - representing the clock

new_number[1 bit] - posedge of which represents a new number has been feeded

reset[1 bit] - to reset all the previous computations

x[16 bits] - represents the time instance of the current point

y[16 bits] - represents the value of the current point

OUTPUT:

flag[1 bit] :- the final flag representing if the point is an outlier or not.

4.2.2 beta

This block is used to calculate Beta and calls the block to calculate MSE returning both the outputs to the top module. It needs division operation to be carried out twice and so the div module is also called by this block making two instances of the div block.

INPUTS:

n[16 bits] - representing the number of inputs

sig_x[16 bits] - sum of all time instance

sig_y[16 bits] - sum of all value of points

sig_xy[32 bits] - sum of product of time instances and value of points

sig_x2[32 bits] - sum of square of all time instances

sig_y2[32 bits] - sum of square of all value of data points

en[1 bit] - enable bit which initiates calculations

clk[1 bit]- representing the clock

start[1 bit] - Bit representing new number and initiating division calculation

OUTPUT :

beta0[32 bits] - returns value of beta0 in 16,16 fixed point representation

beta1[32 bits] - returns value of beta1 in 16,16 fixed point representation

n_times_mse_wire[32 bits] - returns value of n*MSE in 16,16 fixed point representation

4.2.3 nMSE

This block calculates MSE taking the Beta values as input parameters from it calling module. This is the block where the major calculations are done to make this method work online.

INPUTS:

n[16 bits] - representing the number of inputs (bypass from beta module)

sig_x[16 bits] - sum of all time instance (bypass from beta module)

sig_y[16 bits] - sum of all value of points (bypass from beta module)

sig_xy[32 bits] - sum of product of time instances and value of points (bypass from beta module)

sig_x2[32 bits] - sum of square of all time instances (bypass from beta module)

sig_y2[32 bits] - sum of square of all value of data points (bypass from beta module)

beta0[16,16 bits fxp] - value of beta 0 in 16,16 fixed point representation (output of beta module)

beta1[16,16 bits fxp] - value of beta 1 in 16,16 fixed point representation (output of beta module)

OUTPUT :

n_times_mse[16,16 bits fxp] - product of n and MSE in 16,16 fixed point representation

4.2.4 div

This division block acts as a black box for division which simply divides two fixed point numbers and outputs a fixed point number of same configuration in constant number of operations.

PARAMETERS :

Q - number of bits in the fractional part

N - total number of bits the solution

INPUTS :

n[16 bits] - representing the number of inputs

i_dividend [N bits] - dividend in the division operation

i_divisor[N bits]- divisor in the division operation

i_start[1 bit] - set at the start of new division operation

i_clk[1 bit] - clock for the module

OUTPUT :

o_quotient_out[N bits] - output of the division operation in fixed point

o_complete[1 bit] - flag to denote the division operation is completed

o_overflow[1 bit] - flag to denote there is an overflow condition

4.3 Comparison of Cook's Distance and Mahalanobis Distance

Computationally both are almost similar, both can be performed in online fashion but since they have different approach to find outliers, they turn out to have different performance statistics for different kind of data.

Mahalanobis Distance which classifies a point to be an outlier based on the covariance, turns out to be more efficient for the data sets where we have a prior information about the percentage of occurrence of outliers. However it fails to give a good accuracy for general data sets where the rate of outliers is not known. This approach leads to a large number of true negatives if the rate is estimated lower and large number of false positives if the rate of outliers is predicted higher than the actual rate.

Cook's DIstance on the other hand gives a better performance for general data sets, it analyses the data by itself as it considers mean square error of all the data points and is self capable to predict the number of outliers where Mahalanobis Distance fails to a large extent as the rate of error is not known.

Thus we can conclude that Cook's Distance would prove to be efficient in general cases where we do not really know about how clean is our data set. While we may consider to go with Mahalanobis Distance when the rate of occurrence of outliers is known.

4.3.1 Threshold

Both the distance based classification of outliers methods give us a distance value and it becomes important to have a good threshold value set to correctly analyse the outliers, one may change the threshold value as per the application specification and the requirements of the system.

To generalize the threshold for widely accepted applications we have maintained the standard threshold limit for both the approaches.

For Cook's DIstance the threshold varies with the number of points while in the case of Mahalanobis Distance it considers the for a particular dimension and changes with the number of dimensions considered.

Threshold for Cook's Distance = 4/n (where n is the total number of points)
Threshold for Mahalanobis Distance = $P(X \leq x)$; chi square distribution for a particular r (rate of error) and df (degree of freedom). Refer Chi-square table.

4.3.2 Benchmarks: Yahoo Data

This data has very small fraction of outliers.

Cook's distance average accuracy over 67 data sets is 83.8403%.

Mahalanobis distance average accuracy over 67 data sets is 98.58%.(considering p=0.001)

Index	Mahalanobis Dist	Cook's Distance
1	99.1549	80.4225
2	99.5136	65.9486
3	99.9316	84.668
4	99.7892	98.5242
5	99.9305	95.4135
6	99.9305	74.7741
7	95.9241	92.1996
8	99.7887	94.9296
9	99.6578	91.718
10	100	84.3641

results of 10 datasets