Case-Study 10 Implementation of an Open Source Event Detection System

CH

SPOTSeven Lab - Cologne University of Applied Sciences

April 28, 2018

Abstract. This document describes the implementation of parts of the offline mode of the CANARY software in R. The implementation is released under GPLv3 and the source code is available at `https://github.com/choeffer/canary`. The implementation was done during a student project at the Cologne University of Applied Sciences. The aim of the student project was to implement different features (e.g. GUI) of the CANARY software in R. Two parts of the project are the developement of a GUI and plots with Shiny. In addition, another part is the implementation of set-point proximity algorithms (SPPB and SPPE) in R. All parts are then combined to one functional programm. This document is focused on the part which is the implementation of the offline mode and the explanation of the developed code.

1 Introduction

The student project started with trying to port the existing code of the CANARY software, which is written in JAVA/Matlab, to Octave to be able to execute it with open-source software. After unsuccessful trials to get the code running in Octave it was decided to code everything from scratch in R. To achieve this, it was decided to focus first on four features and implement them. The four implemented parts were later combined to one functional programm. For the GUI the Shiny R-package was chosen because it is open-source and is easy to learn, even for beginners. The plots are also done in Shiny with some developed plotting functions. The third part is the implementation of set-point proximity algorithms (SPPB and SPPE, see [3]). The last part, which is also the focus of this document, is the implementation of the offline mode in R. For more information about the GUI, the set-point proximity algorithms and plotting the results see the other documentations of the project.

This document is structured in the following way. Section 2 gives a short overview of CANARY and its offline mode. Section 3 provides an overview of the implementation of the offline mode in R and some parts of the code, like exception handling. Section 4 describes the details of the implementation of the offline mode, which means the core routine which can be seen in Figure 2. Section 5 summarizes the results and Section 6 provides ideas for further implementations and other improvements.

2 CANARY

CANARY is a water quality event detection tool developed at Sandia National Laboratories. It is designed for online usage (online mode) to give alarms to e.g. a SCADA system and also for offline usage (offline mode) using historical data e.g. to test/improve the parameters of the algorithms to get e.g. less false positive alarms. For further informations about CANARY the manual is a good starting point [7] [3].

Figure 1 shows a flowchart of the programm structure of the original offline mode in CANARY. A data set (historical data) is passed to the offline mode. Then an observation (row), starting with the oldest data entry in the historical data set, is taken and added to the empty window. The size of the window (amount of rows) is a parameter of the offline mode. Then it is checked if the window is already filled with enough rows or not. If not, a new row is added to the window. This loop continuous until the window is full. The window will be moved over the data set. In the normalization step, the input variables are standardized to a mean of zero and a standard deviation of one. After that, the last added row of the window is taken as a reference where a prediction (estimation) is made for with the help of the rest of the data in the window by e.g. a linear model. This prediciton (estimation) for each column is then compared with the real value of each column of the last row. The difference between these two values is called residual and is stored. The largest residual is then compared with a threshold, which is another parameter of the offline mode [6][5]. If the largest residual is larger than the threshold, it is classified as an outlier (event=TRUE). If it is smaller, then it is classified as background (event=FALSE). This classification is called residual classification and the result is stored. Equation 1 describes the BED (Binomial Event Discriminator)[4].

$$b(r,n,p) = \frac{n!}{r! \cdot (n-r)!} \cdot p^r \cdot (1-p)^{(n-r)} \tag{1}$$

The BED takes as input parameters the number of TRUE events (r) in the last (n) residual classifications (last n analyzed rows of the data set) and an expected probability of any one event TRUE (p). The inputs r,n,p of the BED are also parameters of the offline mode. The result is the probability (P(random)) that r TRUE events randomly occur in the last n rows. Therefore, the probability P(event) of a real TRUE event is 1-(P(random)). As the given description is rather brief, please refer to [4] for more information. The result P(event) is then stored and compared with another threshold. This threshold is also a parameter of the offline mode. This comparison is the final decision (larger threshold, event = TRUE; smaller threshold, event = FALSE) if in this observation (row) an event occured or not. The result is stored.

This loop is repeated (starting with getting the next observation (row)) until the window reaches the end of the data set. For further information about the offline mode, see manual [7] [3].

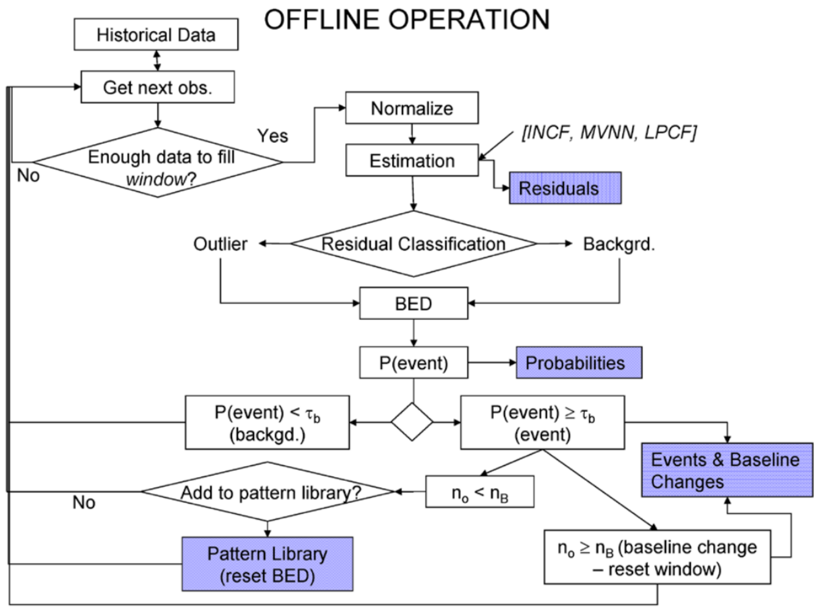

Fig. 1. Flowchart of the programm structure of the original offline mode in CANARY [7]

3 Overview Structure and Code

Figure 2 shows a flowchart of the programm structure which is implemented in R. As can be seen, Figure 2 is a shortened version of Figure 1. The detection of baseline changes and the pattern library are not implemented in this version. Because the structure of the offline mode is implemented in R from scratch without analyzing the original JAVA/Matlab code, the implementation in R is not guarenteed to behave exactly the same as CANARY.

Now an overview of the implemented offline mode is given. In the GUI (not part of this documentation) the parameters, which are passed to the offline mode function, are selected/set. The different parameters are the data set, a selection of the columns of the data set, and additional parameters which are later described in more detail. Afterwards the function is invoked. Inside the function, first some error and warning statements are checked and some necesarry data frames are created. Then the core routine (see Figure 2) is invoked. There, first the window is filled. Then the data inside the window is normalized. For each column (defined in **vals**) a model is trained and based on that model a prediction (estimation) is made. The difference between the real value and the prediction is called the residual. This residual is then stored and the residual classification for each column (defined in **vals**) is made. This means if the residual is larger than the defined threshold, it is classified as an outlier (event=TRUE). If it is smaller, it is classified as background (event=FALSE). Afterwards the P(event) of each column (defined in **vals**) is calculated and stored. After that, another residual classification, based on the largest residual of the columns, is made and stored. Then the P(event) of this row is calculated, stored and compared with another defined threshold. Based on this comparison, the final decision (larger threshold, event = TRUE; smaller threshold, event = FALSE) if in this observation (row) an event occured or not is made and stored.

The core routine is repeated (starting with getting the next observation (row) of the data set) until the window reaches the end of the given data set. Finally, the results are combined to one data frame and returned.

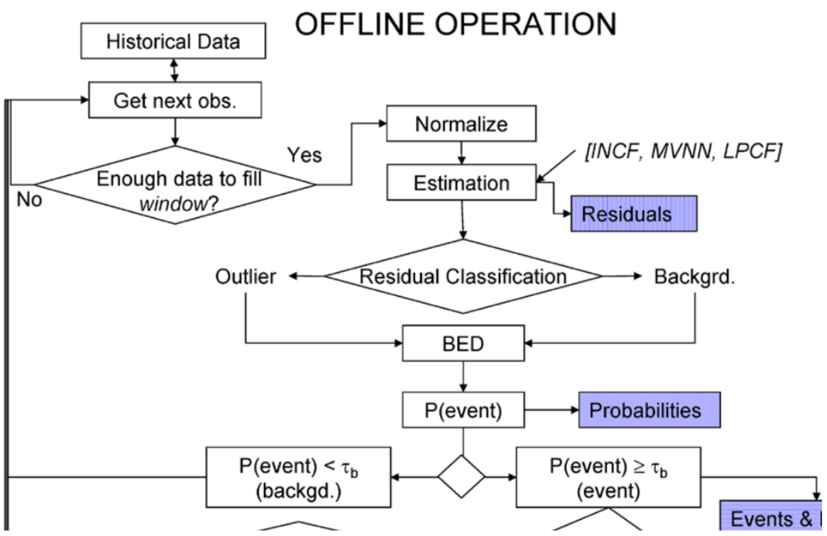

Fig. 2. Flowchart of the programm structure of the implemented offline mode in R [7]

The implemented offline mode can also be used standalone without the GUI. The following command invokes the function with the minimal set of needed parameters. In this case, the rest of the parameters are set to their default values.

```
returned_object <- offline_mode(loaded_data,vals,...)
```

Also a list containing the parameters can be used to invoke the implemented offline mode.

```
final_results <- do.call(offline_mode,param)
```

The following paragraph gives an overview of the parameters which can be passed to the function. All parameters have default values and the ellipsis (...) is used to pass additional arguments to the function which is used for prediction (estimation) [1] [2].

loaded_data data object (historical data) as data frame; default = NULL
vals vector of column names of **loaded_data** which are used for estimation, residual calculation etc.; default = NULL
window_size define the size (number of rows) of the window; default= 2000
threshold threshold used for residual classification; default = 1.5
used_algo defines which algorithm is used for prediction (estimation); default = lm
bed_window n parameter of the BED; default = 5
prob_bed p parameter of the BED; default = 0.01
bed_thresh threshold for final decision after BED; default = 0.975
debug_flag debug/test mode, if FALSE, all rows of **loaded_data** are analyzed; default = FALSE
debug_number_iteration if **debug_flag** = TRUE, just first n rows of **loaded_data** are analyzed; default = 2040
... additional parameters which are passed to the **used_algo** function

There is also a *test_canary_core.R* file available at `https://github.com/choeffer/canary` to test and run the *offline_mode* function.

3.1 Warnings and Errors, Debug Mode and Definition BED

After the function call, first some error and warning statements are checked inside the *offline_mode* function. The printouts in the *stop()* and *warning()* functions explain why each statement is checked.

It is checked if **loaded_data** is empty or not passed to the function.

```
if(is.null(loaded_data)){
    stop("Function needs data object loaded_data to run.")
}
```

It is checked if **loaded_data** is a data frame.

```
if(!is.data.frame(loaded_data)){
    stop("Function needs data object loaded_data in the form
    of a data.frame to run properly.")
}
```

It is checked if **loaded_data** contains one or more NA values.

```
if(anyNA(loaded_data)){
    warning("Given loaded_data data.frame contains one ore more
    NA values! This influences the behaviour of the used_algo and
    other parts of the code.")
}
```

It is checked if the first column of **loaded_data** is of data type factor which indicates a timestamp column. This timestamp column is then removed and a subset of **loaded_data** is created.

```
if(is.factor(loaded_data[0,1])){
    subset_loaded_data <- loaded_data[,-1]
}
else{
    stop("Unexpected structure in the given loaded_data data.frame.
    Function offline_mode of canary_core.R expects data type
    factor in the first column of loaded_data indicating that the
    first column contains the timestamps. Otherwise it will not
    run properly.")
}
```

It is checked if **vals** is empty or not passed to the function.

```
if(is.null(vals)){
    stop("Function needs data object vals to run.")
}
```

It is checked if **vals** is a vector.

```
if(!is.vector(vals)){
    stop("Function needs data object vals in the form of a vector
    to run properly")
}
```

It is checked if one ore more elements of the vector **vals** are NOT column names of **loaded_data**.

```
if(any(!(is.element(vals,colnames(loaded_data))))){
    stop("One or more elements in the list vals are NOT column
    names of the loaded_data data.frame. In this case function
    offline_mode will not run properly.")
}
```

It is checked if one ore more elements of the vector **vals** are duplicated.

```
if(any(duplicated(vals))){
    stop("One or more elements of the vector vals are duplicated.
    In this case function offline_mode will not run properly.")
}
```

It is checked if **debug_flag** = TRUE and if yes **debug_number_iteration** defines how many rows of **loaded_data** are analyzed. If **debug_flag** = FALSE all rows of **loaded_data** are analyzed.

```
if(debug_flag==TRUE){
    number_iteration<-debug_number_iteration
}
if(debug_flag==FALSE){
    number_iteration<-nrow(loaded_data)
}
```

It is checked if **window_size** + **bed_window** is smaller **number_iteration**.

```
if((window_size+bed_window)>number_iteration){
    stop("window size + bed_window=",window_size+bed_window,
    " is larger than number_iteration=",number_iteration,". In
    this case function offline_mode will not run properly.")
}
```

The BED function is defined which is later used in the code.

```
binevdis <- function(r,n,p){
    part1 <- factorial(n)/(factorial(r)*factorial(n-r))
    part2 <- (p^r)*(1-p)^(n-r)
    return(part1*part2)
}
```

3.2 Necessary Data Frames

The function requires some data frames which can be filled during the core routine, which will be described later in Section 4.

Two empty data frames for adding rows from *subset_loaded_data* and **loaded_data**[, **1**] are created.

```
subset_used <- subset_loaded_data[0,]
subset_used_timestamps <- loaded_data[0,1, drop = FALSE]
```

An empty data frame *temp_res_sd* for storing the residuals and the standard deviations of all columns (defined in **vals**) in the window is created. This data frame is later used for the residual classification.

```
temp_res_sd<-setNames(as.data.frame(matrix(ncol =
    length(vals), nrow = 2)),vals)
row.names(temp_res_sd)<-c("res","sd")
```

A data frame for storing the residuals of all columns (defined in **vals**) is created.

```
used_names_res <- NULL
used_names_res <- paste0(vals, "_residual")
calc_residuals <- setNames(as.data.frame(matrix(ncol =
    length(used_names_res), nrow = 0)),used_names_res)
calc_residuals[1:window_size,] <- as.numeric(NA)
```

In contrast to the CANARY offline mode, in this implementation the residual of each column (defined in **vals**) is compared with the **threshold**, not just the largest residual. Therefore, each column is separately classified as outlier (event =TRUE) or background (event=FALSE). So a data frame is created for storing the results of each column.

```
used_names_events <- NULL
used_names_events <- paste0(vals, "_event")
events_col <- setNames(as.data.frame(matrix(ncol =
    length(used_names_events), nrow = 0)),used_names_events)
events_col[1:window_size,] <- NA
```

A data frame for storing the residual classification result of comparing the largest residual with the **threshold** is created.

```
events <- read.csv(text="Event")
events[1:window_size,] <- NA
```

Because each column (defined in **vals**) is separately classified as outlier (event =TRUE) or background (event=FALSE), a data frame for storing the probability P(event) of each column is created.

```
used_names_bed <- NULL
used_names_bed <- paste0(vals, "_prob_event")
events_col_bed <- setNames(as.data.frame(matrix(ncol =
    length(used_names_bed), nrow = 0)),used_names_bed)
colnames(events_col_bed) <- used_names_bed
events_col_bed[1:(window_size+bed_window),] <- as.numeric(NA)
```

A data frame for storing probability P(event) of an observation (row) is created.

```
events_bed <- read.csv(text="Prob_event")
events_bed[1:(window_size+bed_window),] <- as.numeric(NA)
```

A data frame for storing the final decision, if in this observation (row) an event occured or not, is created.

```
events_final <- read.csv(text="Final_event")
events_final[1:(window_size+bed_window),] <- NA
```

4 Implementation Offline Mode

Now it is described in detail, how the outer for loop (core routine), which is repeated (starting with getting the next observation (row)) until the window reaches the end of the data set, is implemented. Figure 2 shows the implemented parts of the offline mode of CANARY.

First the data frames *subset_used_return* and *subset_used_timestamps* are filled. This is done for speed optimization of the code so that no for loop is needed to fill them. Also the data frame *subset_used* is filled until **window_size** $-$ 1 so that no for loop is needed to fill the window until enough observations are added to start the core routine.

```
subset_used <- subset_loaded_data[1:(window_size-1),]
subset_used_return <- subset_loaded_data[1:number_iteration,]
subset_used_timestamps <- loaded_data[1:number_iteration,1,
    drop = FALSE]
```

The outer for loop (core rountine) is started. Here, at each iteration, a new row is added to the data frame *subset_used* before performing the new calculations.

```
for(row in window_size:number_iteration){
    subset_used <- rbind(subset_used,subset_loaded_data[row,])
```

The data frame *temp_res_sd* is resetted back to initial state for each new iteration of the loop to avoid side effects from previous iterations.

```
temp_res_sd[]<-NA
```

A data frame for storing the data of the window is created.

```
defined_window <- tail(subset_used,window_size)
```

A data frame for storing the normalized data of the window is created.

```
scaled_defined_window <- as.data.frame(scale(defined_window))
```

It is checked if the normalized data of the window contains NaN values.

```
if(any(is.nan(as.matrix(scaled_defined_window)))){
    stop("Normalized data in the window contains NaN values.
    This might crash the used_algo, therefore function
    offline_mode stops")
}
```

Two data frames, one for storing the last row of the window, one for storing the rest of the rows inside the window, are created.

```
scaled_defined_window_last_row_rm <- head(scaled_defined_window,
    -1)
scaled_defined_window_last_row <- tail(scaled_defined_window, 1)
```

A inner for loop in the outer for loop is started. In each iteration of the loop one column (defined in **vals**) of the data set is analyzed.

```
for(i in 1:length(vals)){
    used_name <- vals[i]
```

A model for each column is trained.

```
    used_model <- used_algo(formula =
        as.formula(paste0(used_name, " ~ . ")),
        data = scaled_defined_window_last_row_rm,...)
```

A prediction (estimation) is made for each column and the residual is calculated and stored as an absolut value.

```
    pred_value <- predict(used_model,
        scaled_defined_window_last_row)
    temp_res_sd["res",used_name] <- abs(unname(pred_value-
        scaled_defined_window_last_row[,used_name]))
    calc_residuals[row,
        paste0(used_name, "_residual")] <- temp_res_sd["res",
        used_name]
```

The standard deviation for each column in the window is calculated.

```
    temp_res_sd["sd",used_name]<-sd(defined_window[,used_name],
        na.rm = TRUE)
```

Depending on the threshold, multiplied with the standard deviation, being larger or smaller than the residual of each column, the residual classification for each column is made and stored.

```
    if(!is.na(temp_res_sd["res",used_name])){
        if(temp_res_sd["res",used_name]>
                threshold*temp_res_sd["sd",used_name]){
            events_col[row,paste0(used_name, "_event")]=TRUE
        }
        if(temp_res_sd["res",used_name]<=
                threshold*temp_res_sd["sd",used_name]){
            events_col[row,paste0(used_name,"_event")]=FALSE
        }
    }
    else{
        events_col[row,paste0(used_name,"_event")]=NA
    }
```

At the end, P(event) for each column is calculated and stored.

```
    if(!any(is.na(tail(events_col[,paste0(used_name,"_event")],
            bed_window)))){
        occur_col_true<-sum(tail(events_col[,
            paste0(used_name,"_event")],bed_window),na.rm=TRUE)
        events_col_bed[row,paste0(used_name, "_prob_event")]
            <-1-binevdis(occur_col_true,bed_window,prob_bed)
    }
}
```

Back in the outer for loop, the residual classification, based on the largest residual of all columns (defined in **vals**), is made and stored. Also NA is written to the column of *subset_ used*, which is in this row the cause of the outlier classification, to exclude it from the values used to predict water quality at next time step.

```
index_biggest_res<-which.max(temp_res_sd["res",])
max_rs<-temp_res_sd["res",index_biggest_res]
```

```
if(length(max_rs) > 0){
    if(max_rs>threshold*temp_res_sd["sd",index_biggest_res]){
        events[row,"Event"] <- TRUE
        subset_used[row,colnames(
            temp_res_sd[index_biggest_res])]<-NA
    }
    if(max_rs<=threshold*temp_res_sd["sd",index_biggest_res]){
        events[row,"Event"] <- FALSE
    }
}
else{
      events[row,"Event"] <- NA
}
```

Then the P(event) is calculated and stored. Based on this results, the final decision, if in this observation (row) an event occured or not, is made and stored. This is the last part in the outer for loop.

```
if(!any(is.na(tail(events[,"Event"],bed_window)))){
    occur_true<-sum(tail(events[,"Event"],bed_window),na.rm=TRUE)
    temp_prob_event <- 1-binevdis(occur_true,bed_window,prob_bed)
    events_bed[row,"Prob_event"] <- temp_prob_event
    if(temp_prob_event>=bed_thresh){
        events_final[row,"Final_event"]<-TRUE
    }
    if(temp_prob_event<=bed_thresh){
        events_final[row,"Final_event"]<-FALSE
    }
}
```

Finally, after the window reaches the end of the *loaded_ data*, the outer for loop ends, the results are combined to one data frame and returned.

```
results <- cbind(subset_used_timestamps,subset_used_return,
    calc_residuals, events_col,events,events_col_bed,events_bed,
    events_final)
return(results)
```

5 Conclusion

So far, the implementation of the offline mode in R is a success. Most of the parts of the original offline mode could be implemented. The implementation is just using base R functions, so no additional packages are required. Also optimization of the code is done regarding the speed/runtime of the code. For loops are removed as much as possible and also special functions (e.g. anyNA instead of any(is.na)) are used. Also other small improvements where done to get a clean, fast and comprehensible code. The function is also designed to be flexible regarding which columns to include in the analyzis. The debug mode allows flexible testing of different *used_algo* with different passed through parameters to it with a user defined number of iterations. The warning and error messages are also giving a good feedback and catch a lot of possible loop holes and side effects. Further and more detailed comments to each line of the code can be found in the source code which is available at `https://github.com/choeffer/canary`. The code is released under GPLv3 so that it can be used and modified by everyone who is interested to.

6 Outlook

Additional research is required if writing NA to the column of *subset_used* (which is in this row the cause of the outlier classification) is the correct way to exclude it from the values used to predict water quality at next time step or if this should be solved/implemented in another way. Also the issue regarding the NaN check where scaling leads to NaN values (if columns have zero variance) needs to be investigated, because at the moment the function just stops if the return value of the scale() function includes NaN values.
To add more flexibility, there could be another parameter which gives the user access to the formula interface of the *used_algo* so that the prediction is not just done with "$name\_used \sim .$". Also the implementation of the pattern library and baseline changes should be considered to be implemented to get the full functionality.
In general further testing with different data and users will help to find bugs which need to be fixed or improvements which can be implemented. Also this will help to find further warning and error messages which need to be implemented and are not considered yet.

References

1. An introduction to r. Website, `https://cran.r-project.org/doc/manuals/r-devel/R-intro.html#Named-arguments-and-defaults`, visited on 2018-03-09
2. An introduction to r. Website, `https://cran.r-project.org/doc/manuals/r-devel/R-intro.html#The-three-dots-argument`, visited on 2018-03-09
3. Agency, U.E.P.: Canary user's manual version 4.3.2. Website, `https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P100HKY5.TXT`, visited on 2018-03-08
4. Hart, D., Klise, K.A., Cruz, V., McKenna, S.A., Wilson, M.P.: Event detection from water quality time series. (1 2007)
5. McKenna, SA, H.D.K.K.K.M.M.S.W.M.C.V.C.L.M.R.H.T.: Water quality event detection systems for drinking water contamination warning systems - development, testing and application of canary. Website, `https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P100780H.txt`, visited on 2018-03-11
6. McKenna, SA, H.D.K.K.K.M.M.S.W.M.C.V.C.L.M.R.H.T.: Water quality event detection systems for drinking water contamination warning systems - development, testing and application of canary. Tech. rep. (2010)
7. U.S. Environmental Protection Agency, Washington, DC: CANARY User's Manual version, 4.3.2 edn. (2012)