Alerts Detection System

Release v1.5.0

Roberto Otero

Apr 24, 2023

CONTENTS:

CHAPTER

ONE

INITIAL SETUP

1.1 Configuring parameters

The `config` folder contains files where you can configure the necessary parameters to execute the System. They are duplicated in different environments: *DEV* and *PRO*. It is because, in order to improve the workflow, the changes in the config that you want to test before putting them into Production can be tested with *DEV* files. The following parameters have to be configured:

- *googleProject.project*: the name of the Google Cloud project. We talk more about this in *Google Configuration section*.
- *googleProject.tokenFile*: the name of your *.json* credential to connect with *BQ* and/or *GA*.
- *model.confidenceInterval*: the confidence of the confidence intervals in the model. Lower confidence means you are more stricted with the alerts (more alerts will raise).
- *model.nHistData*: number of historical data, in days. If you want to control a yearly seasonality you should introduce at least 365 days.
- *tableAlerts.styleTable*: the style of the table that will appear in the email. Different styles css have different coulours, format, etc.
- *tableAlerts.showUmbral*: not available for now.
- *tableAlerts.saveHistorical*: *True* if you want to save historical data in a *BQ* table; *False* if not. Please remember that you have to give access to your service account to Big Query.
- *tableAlerts.limSupAlert*: *True* if you also want to receive alerts when a metric overcomes the limsup confidence interval.
- *mail.typeSend*: select the type of the alert notification. For now, it is only available *email*.
- *mail.alertsTableTitle*: title of the alerts table in the email.

You also have to configure some email parameters in a Google Project Secret Manager. Specifically, it is neccesary to create the following secrets:

- *email_user*: the email sender account.
- *email_to*: the email recipients, separated by a comma.
- *email_password*: the password of the *email_user* account.
- *email_smtp_server*: the smtp server (for example, *smtp.gmail.com*).
- *email_smtp_port*: the smtp port (for example, 587).

You have to add the name of the "env" at the end of the name. For example: *email_user_dev*.

The secrets must be created in a Google project where the service account from which you obtained the *.json* credential has access. If you are executing the System inside a VM of the Google project, then you do not need the *.json* credential and these parameters will be read from your Secret Manager.

You have to change all of these parameters to make the Alert System works. We recommend to introduce this config files in *.gitignore* file so that your sensitive info is not shared.

You also can add a *logo.png* file in the folder `alerts_table_styles` so that the image appear as an icon in the alerts notification.

The styles of the alerts table, the predictions table and the historical alerts table can be changed with the *extra_styles_* files in `alerts_table_styles` folder. You can add any extra method you want in order to apply new styles to the tables. For example, if you want to add a percentage symbol in some metrics, then add a function that do that and it will return the table with the symbol.

1.2 Configuring metrics

The model needs two columns to build the model for each metric: the column with the dates (daily) and the column with the metric. You can find inside the folder `config` another file that is called `metrics_<env>.yml`. Here, the metrics have to be introduced. Metrics from both *BQ* and *GA* can be introduced. To do that, please follow these rules:

1. The *.yml* file is divided into tuples. Each tuple contains info about a metric and it is separated from the others by setting a new integer number: first tuple will begin with `"0"`, the second with `"1"`, and so on. You must specify the *source* of the metric in each tuple (*BQ* or *GA*).

2. To introduce metrics for **BQ**, each tuple of the *.yml* file contains the sql query field you can complete with the corresponding query to obtain the metrics. The field *date_name* is the name of your field *date* in your original table. You also have to specify the *dateFormat* of the date in you *BQ* table. You have to build the query so that you get a unique date by row (no dups).

There is a field called *kpiNamesMethod* where you have to introduce the name you want to give to your metrics so that they appear in the alert notification and the method and the model's parameters you are going to use to detect the alert. You can choose, for now, between *prophet*, *arima* and *constraint*. The first two use a ML model to predict the value of the metric and find alerts. The last one is to use a constraint and the alert will be send if the constraint is violated. The constraint must be added to the query as a zero-one column where 1 means that the constraint is violated. For example, you can create a column to check if a metric has overcome the value 25, or if the metric is greater than other metric. That column would have ones if the constraint is violated and zero if not. The Alert System would use that column to send you an alert if it exists.

You may find in some metrics that they have *True* in the field *isRelated*. It is because the variable is related to others and maybe it has to be treated in a different way. If this is *True*, new styles can be applied in the *extra_styles.ExtraStylesPreAlerts* method.

3. To introduce metrics for **GA**, each tuple of the *.yml* file must contain: the name of the view of GA (this can be the name that you want, it is used to send you the email with that name instead of the view number); the view ID; the KPI's name (the name that you want to give to the metric and it will appear in the email with that name) and the method and model parameters you want to use to build the model (see last paragraph); the name of the metric that has to be downloaded from *GA* (it must exist in *GA*); the segment and the filter clause.

1.3 Google configuration and credentials

In order to connect with *GA* and *BQ* to analyze the metrics, it is necessary to create a Google Project. Once you have created it, please create a service account. Then, the service account must have the following roles and privileges:

- BigQuery Data Visualizer role in the project you have your data.
- Owner role.
- Access to the specific GA accounts to download the metrics.

Then, please create a new *.json* key for this service account and save it in a folder called `credentials` in the root directory of the project. Finally, introduce the name of your credential in *googleProject.tokenFile* in the config file. If are executing the Alert Detection System inside Google Cloud with a service account that has already access to these scopes, then you do not have to specify any name in *tokenFile*.

CHAPTER

TWO

MODULES

2.1 alerts_detection_system (main module)

This module is the main script that contains the Alert Detection System's skeleton. It calls every single module from the folder `modules`. Therefore, this is the script that has to be executed by a *shell* script in order to analyze data and send the alerts.

These are the steps this script does:

1. Load all configuration you set up in the `config` folder (metrics and general config).
2. Download data from *GA* and/or *BQ* and save the metrics as a *DataFrame*.
3. Create models for every metric with Machine Learning algorithms in order to understand the stationarity and seasonality.
4. Predict what should be your yesterday's data value and compare it with the actual one in order to detect the alerts.
5. Gather the alerts information in a table and send them by email.

2.2 modules.download_data module

This module has the necessary functions to download data from *GA* and *BQ*. You can configure metrics for both sources.

class modules.download_data.**DownloadData**(*token_path*)

Bases: `object`

This class has the necessary methods to download the data to make the analysis.

get_data_BQ(*sql_query_input*)

Function to download data from BQ.

Parameters
sql_query_input (*string*) – The sql query to download the specific metric.

get_data_GA(*view_id*, *start_date*, *end_date*, *metrics_input*, *dimensions_input*, *segments_input=[]*, *filters_input=''*)

Function to get the final data from GA.

Parameters

- **view_id** (*string*) – The ID of the GA view.
- **start_date** (*string*) – The start date from which the data has to be downloaded, in the format YYYY-MM-DD.

- **end_date** (*string*) – Same but for the end date.
- **metrics_input** – The metrics to be downloaded.
- **dimensions_input** (*list*) – The dimensions that have to appear in the table (date, city...).
- **segments_input** (*list*) – The segments to be applied to the metrics.
- **filters_input** (*string*) – The filter to be applied to the metrics.

get_service_GA()

Function to get the service connection to GA.

logging_bq()

Function to log into BQ.

res_to_df(*res*)

Function to convert the results from GA into a pandas DataFrame.

Parameters
res (*dict*) – A dictionary with the results from GA.

upload_future_pred(*project_id*, *future_table*, *env*, *date*)

Function to upload historical alerts to BQ.

Parameters

- **project_id** (*string*) – Project Id where the data will be saved.
- **new_hist** (*pandas DataFrame*) – The table with the new alerts.
- **env** (*string*) – The environment you are executing the System.
- **date** (*timestamp*) – The start date of the predictions.

upload_historical_alerts(*project_id*, *new_hist*, *env*)

Function to upload historical alerts to BQ.

Parameters

- **project_id** (*string*) – Project Id where the data will be saved.
- **new_hist** (*pandas DataFrame*) – The table with the new alerts.
- **env** (*string*) – The environment you are executing the System.

2.3 modules.preprocess

Many times data is downloaded from *BQ* or *GA* with missing values. Then, it is important to apply the proper techniques in order to fill the gaps. This module does that. The module will be updated with more techniques as the project progresses.

class modules.preprocess.**Preprocess**

Bases: object

This class is to impute missing data.

fill_moving_avg(*data*)

Function to apply moving average to fill missing data.

Parameters
data (*Pandas DataFrame*) – Data with the metrics.

remove_outliers(*data*)

Function to remove outliers from the time series.

Parameters
data (*Pandas DataFrame*) – Data with the metrics.

2.4 modules.alert_detector module

This library uses a time series model in order to predict the alerts. Specifically, it uses a Prophet model and it is prepared to model daily data. Principally, we can summarize the process as follows:

- First, each metric that is introduced in the config is modeled (the stationarity and the seasonality) until the day before yesterday.
- After that, yesterday's data is predicted along with its confidence interval.
- Yesterday's actual data is compared with the confidence interval. If it is below or above any limit, then an alert is detected.

class modules.alert_detector.**AlertDetector**(*data*, *plot*, *date*)

Bases: object

This class has the corresponding methods to get the final table with the alerts summary for the metrics downloaded from *GA*/*BQ*.

calculate_days_remaining_month()

This function calculates the remaining days until the end of the month that correspond to the date.

get_alerts_table(*real_value_table*, *model_params*, *future_pred=False*, *limsup_alert=False*)

Function to extract the info from the predictions table and create a new table with the summary of the alerts for every single metric. When the actual yesterday's data is below or above the limits of the confidence interval, an alert will exist.

Parameters

- **real_value_table** (*Pandas DataFrame*) – The yesterday's real value table.
- **model_params** (*Pandas DataFrame*) – Table with all model params for every metric.
- **future_pred** (*boolean*) – True if future predictions must be calculated.
- **limsup_alert** (*boolean*) – True if alerts above limsup must be detected.

get_prediction_ARIMA(*metric*)

Function to get the predictions table given a single metric. It uses a time series model based on SARIMAX model.

Parameters
metric (*string*) – The metric to analyze.

get_prediction_PROPHET(*metric*, *confidence_interval=95*, *seasonality_mode='multiplicative'*, *change_prior=0.5*)

Function to get the predictions table given a single metric. It uses a time series model based on Prophet library.

Parameters

- **metric** (*string*) – The metric to analyze.
- **confidence_interval** (*integer*) – The confidence level to build the confidence interval. An integer between 1-99.
- **seasonality_model** (*string*) – The type of seasonality. It can be 'additive' or 'multiplicative'.
- **change_prior** (*float*) – The changepoint_prior_scale parameter. Higher values return more sensibility in the changes of the time series.

remove_decimals(*number*)

Function to remove decimals from metrics that should not have them. They must have been converted to string before.

Parameters

number (*string*) – The number to remove the decimals.

2.5 modules.email_generator module

This module uses the final alerts table summary to generate an email with the alerts information.

class modules.email_generator.**EmailGenerator**(*email_from*, *emails_to*, *password*, *smtp_server*, *port*)

Bases: object

This class creates the email corpus, sets the smtp service with email and sends the email with the alerts.

create_corpus(*only_alerts_table*, *date*, *alerts_table_title*, *style_table*, *future_table=None*)

Function to create the corpus of the email. If any alert is detected, the corpus is created with a message and a table with the metrics that have alerts.

Parameters

- **alerts_table** (*Pandas DataFrame*) – The table that has the summary of every analyzed metric and if they have an alert or not.
- **future_table** (*Pandas DataFrame*) – Table with future predictions
- **alerts_table_title** (*string*) – The title of the alerts table in the email.
- **style_table** (*string*) – You can choose between different table styles. They must appear with the same name as a *css* files in alerts_table_styles folder.
- **date** (*string*) – The date when the alerts System is evaluating the alert.

send_email(*msg*)

Function to send the email to the recipients.

Parameters

msg (*string*) – The message to send.

set_email_service()

Function to set the connection with email service.

2.6 modules.secret_manager module

modules.secret_manager.**get_secret**(*project_id*, *secret*, *token_path*)

Function to read secrets from the Google Cloud Secret Manager.

Parameters

- **project_id** (*string*) – The project id.
- **secret** (*string*) – The name of the secret to read.
- **token_path** (*string*) – The path of the token that will connect to Google Cloud project.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

S

U