RADSSo

RIASC Automated Decision Support Software

Manual for the

updated version to python 3.5

Index of Contents

1-Initial considerations for the RADSSo version adapated to python 3.5

RIASC Automated Decision Support Software (RADSSo) is a tool developed by the **Research Institute of Applied Sciences (RIASC) in Cybersecurity** at the **Universidad de León** that is able to solve the multi-CASH machine learning problem making use of the library scikit-learn that can be found at https://github.com/scikit-learn/scikit-learn. This new version of RADSSo can be **executed using python 3.5.**

This software does not include a preprocessing step, so the values of the features in the provided dataset must be numbers (**except the name of the problems to solve**). If input datasets contain **nan values, the software will replace those empty values by 0.**

There are available two feature selection algorithms. The first one, **Mutual Information Function**, was already included in the previous version; the new one, **Fisher Score,** can be selected in this updated version using the conf.ini; by this way the tool will use it instead of the mutual information function selection method. Those methods are run before training the models to select the relevant features for the specific event and target in an automated way.

The tool is able to solve problems in the same dataset with a variable number of values in the target. The dimensions of the confusion matrix will be automatically adapted when the corresponding parameter is set to **True** in the conf.ini file and the **False Positives and False Negatives** can be penalized when penalization is set to **True** in the conf.ini.

RHOASo algorithm, also developed by RIASC, is used to get the optimal parameters for each model, adjusting them for the current event and target. The project can be obtained at *https://github.com/amunc/RHOASo.* The module has been improved to obtain hyperparameters of higher dimensions**; to achieve this, it must be used the corresponding key in the conf.ini.**

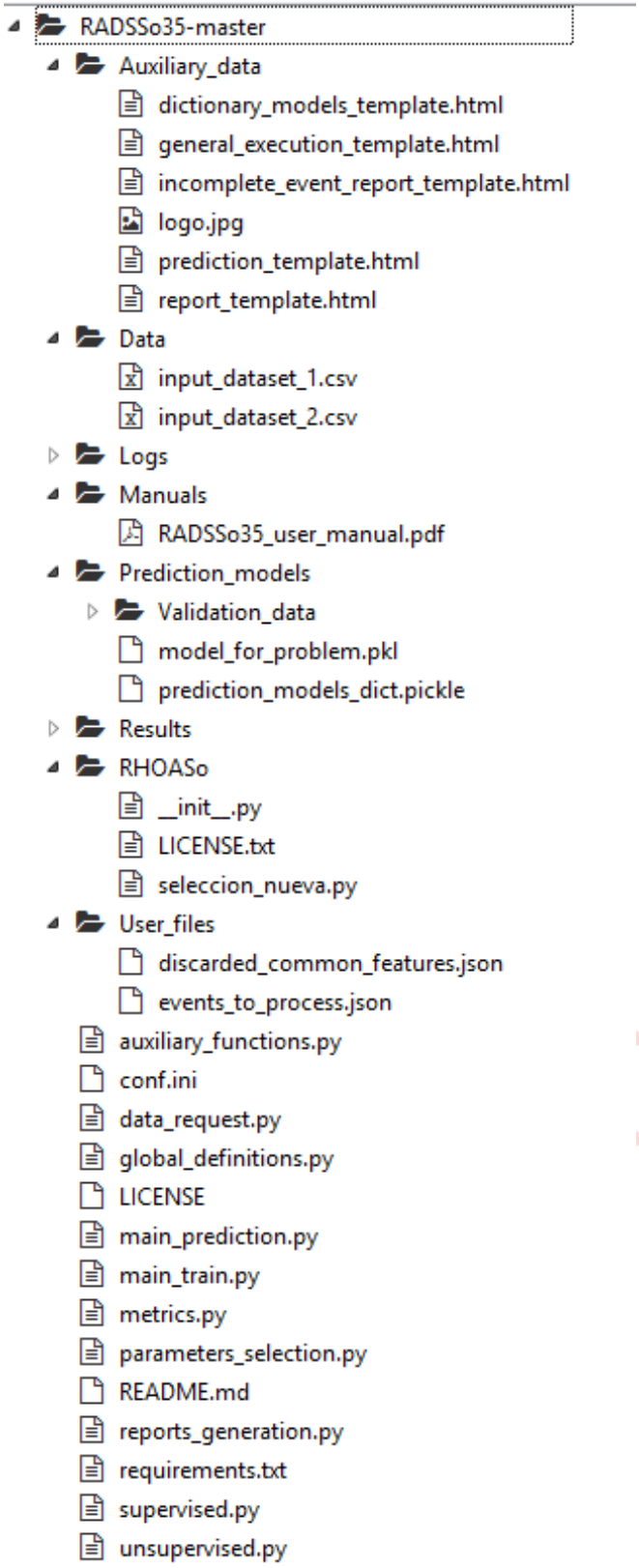
The available models are Decision Tree, Ada Boost, Gradient Boosting, Random Forest, Multi-Layer Perceptron and Kmeans.

The optimal model for each event and target is obtained according to a ranking based on the metrics computed during the training process. The string sorting is achieve using the functions from Natural string sorting.

After the training-test phase, where the optimal model is computed, is generated one pdf with the summary of the performance of each of the models trained and tested for the problems in the dataset. **In this new version, after the prediction phase, a pdf with the summary of the prediction is generated too.**

2-Directories structure

The next structure of directories must be maintained in order to use the tool. It can be modified changing the right parameters in the *conf.ini* file, but it is recommended to maintain it:



Auxiliary_Data: This directory contains the templates of the pdf files that will be generated during the processing of the input files and the creation of the models.

dictionary_models_template.html: It allows to generate a pdf with the current status of the dictionary that contains the relationships between the events and the best model obtained after the execution.

general_execution_template.html: It allows to create a pdf with an overview of the execution process (events to process, features, models to generate, etc.).

incomplete_report_template.html: It allows to create a pdf that indicates why an execution for a concrete event can not be achieved.

prediction_template.html: It allows to generate a pdf with the summary of the accuracy obtained (with the confusion matrix) by the best model, for each problem in the provided datasets, in the prediction phase where a dataset with labeled observations must be provided. If only unknown observations are provided the accuracy will not be computed.

report_template.html: It allows to create a pdf with the results of the performance of each model.

Data: This directory contains the input dataset/s with the features for each event in csv format. If **must not contain subdirectories and, if there are more than one input dataset.**

Logs:* The log files of the process, that can be checked to obtain additional information about the process or to verify that there were no errors, are generated under this directory. **This directory is created at the beginning of the execution.

Prediction_models:* This directory contains the best model for each event and the dictionary that allows to link events and prediction models in the **training step. The detected registers with an unknown target are placed in a subdirectory called **Validation_data, otherwise it will not contain unkown data.** In addition**,** if validation option is set to True in the *conf.ini*, the validation data composed by registers with known target will be placed there too; in other case **the user must provide the data to execute the prediction phase/ validation phase.**

**Results:* This directory will contain the pdf with the general overview of the current execution, a pdf with the current status of the dictionary that relates events and models and one directory for each processed event that contains the intermediate output.

RHOASo: This directory contains a python module obtained from another open source github project that selects optimal parameters for each model using an own algorithm.

User_files: This directory contains two JavaScript Object Notation files:

discarded_common_features.json: It is used to create a 'simple dictionary', easy to modify and to be adapted to the current problem (target), that allows to simplify the removal process of features from the start.

events_to_process.json: It **is used in semi-automated execution mode,** in order to process only the events with the relevant and discarded features specified by the user.

**Directories created at execution time*

3-Main files description

auxiliary_functions.py: A python file that contains functions related to auxiliary operations or utilities.

data_request.py: A python file that contains functions related to the **automatic recognition of events** or the introduction of data by the user (**semi-automated mode**).

global_definitions.py: A python file that contains functions invoked by the other scripts that return static strings. The main purpose of this file is to make easier future modifications of the code.

metrics.py: A python file that contains functions related to the computation of the metrics (learning curves, confusion matrices and their derived indicators) of the models.

parameters_selection.py: A python file that uses the **RHOASo module** to compute the optimal parameters for each model according to the current event. The module has been improved to compute hyperparameters of bigger values than in the previous version by using a exponent value that can be set in the conf.ini.

reports_generation.py: A python file that allows to dump the information gathered during the process into pdf files using the html templates located at the directory *Auxiliary_data.*

supervised.py: A python file that allows to compute the supervised models.

unsupervised.py: A python file that allows to compute the unsupervised models.

main_train.py: A python file that allows to process the input data to generate the prediction models (**training and test phase).** The reports with the results will be stored at the correct location under *Results* directory. **In this new version:**

The training phase can be executed using mutual information function or fisher score for selecting the relevant features in addition with the improved functionality of the confusion matrices. To achieve this, the key "*main_metric***" of the conf.ini must be set according to the purpose.**

The learning curves can be generated or not, setting the parameter "*compute_lc*" to True or False in the conf.ini.

To compute hyperparameters of higher value for the models the user can specify a value for he exponent, using the key "*exponent_value***" of the conf.ini.** High exponent values increase the elapsed time to compute the hiperparameters.

The models can be penalized when producing False Positive and False Negative. **If the appearance of those items is very negative for the model, the key "***penalize_falses***" can be set to True in the conf.ini**

*main_prediction.*py: A python file that allows to process the validation/non classified data to generate a classified dataset using the models obtained in the **training and test phase** (main_train.py). It is mandatory to obtain the models before using this

script in order to predict. A csv with the predictions that is generated and the end of the process. **In this new version:**

If validation data (observations with known target) are provided setting the key "validation mode to True" or directly by the user, **a pdf with the summary of the prediction phase is generated.**

conf.ini: This file contains the basic parameters to achieve a successful execution. It is divided in several sections:

- Logs section: Parameters related to the creation of the *Logs* directory and the files that it will be generated at execution time. By default there is one log for the execution process and another log for the computation of the time in each one of the phases (training and prediction).

- Input data section: Parameters related to the to the main_train.py execution process. **It is recommended not to modify them if not sure.** There are several parameters that are fundamental:
  - *event_name_feature*: Its value must be **the name of the feature in the dataset that contains the names of the events to generate the prediction models for.** It is the only feature that **can contain textual values.** If the feature does not exist in the provided dataset the process will end.
  - *label_non_catalogued*: Numeric value of the target feature that is associated with the unknown classification (**it must be a number).**
  - *obsnumber*: This key contains the name of the feature that will be inserted at the beginning of the execution in order **to trace the samples that were incorrectly classified in training, test and prediction phases. It is recommended not to modify it.**
  - *input_files_delimiter*: The delimiter of the fields for the input dataset/s.
  - path_to_root_directory_input_files: Name of the directory that contains csv files, by default its named *Data.* **The directory must be at the same level of the .py files.**
  - user_files_directoryname: Name of the directory that contains the files to be processed that are defined in the keys *events_filename* and *user_discarded_variables_filename.*
  - *events_filename*: It contains the name of the file that gathers the events and features for the semi-automatic recognition.
  - *user_discarded_variables_filename*: It contains the name of the file that gathers the common discarded features (features discarded for all the events).
  - *maximum_number_files_to_read:* If there is more than one input dataset under *Data* directory, it can be specified the total number of csv to read. **0 value means all and other value the specified number.**
  - *maximum_number_observations_to_read*: Similar to the previous parameter but related to the events, it limits the

maximum number of observations to process for each event individually.

- *train_test_division_percentaje*: Percentage of division in the **interval [0.6, 0.9]** in train a test datasets. The default value is 0.8 (80% of the available catalogued data for the train and 20% for the test).
- *percentil_relevant_variables*: In fact this number in the **interval [0, 100]** refers to the percentile value of features that will be considered to train the models. Those features are computed at execution time using the mutual information function and the final percentile value is computed using the inserted one according to the scores derived from the mutual information function.
- *n_feat:* Percentage of features between the relevant ones selected by the percentil. **It can be used when the percentil gives a big number of features, to reduce the relevant ones in a "second" step.**
- *main_metric*: The ranking of models is computed according to the metrics derived in the execution process and the value of this parameter. **It can take the value acc by default,** if the user wants the accuracy of the models to be the predominant factor in the ranking formula, or the value **mcc** (Matthews Correlation Coefficient) that will generate a more accurate ranking when input dataset has unbalanced data.
- *feature_selection_method*: It can take the value "**mif"** when the relevant features must be selected according to the Mutual Information Function or "**fisher"** when the relevant features must be selecting according to Fisher Score.
- *default_matrix:* This parameter takes the value True, if the matrix of weights wants to be computed at execution time **(recommended option, it can be adjusted to the number of values in the target)**, or the value False, in this case the program will take the matrix from the parameter *matrix_of_weights_fp_fn.*
- *diagonal_weight:* It is the weight used for pondering the items in the main diagonal of the confusion matrix. The coefficient for pondering the FP and FN is computed dividing between the rest of the dimensions, 1 – diagonal_weight value.
- *matrix_of_weights_fp_fn*: This parameters must be specified **only and only if the user wants a customized matrix of weights** to ponder the metrics derived from the confusion matrix (False Positives, True Positives, False Negatives and True Negatives). The parameter *default_matrix* **must be set to False in order to use the customized matrix of weights**.
- *penalize_falses*: If set to **True**, the number of False Positives and False Negatives will be multiplied by the corresponding coefficient and the value will be discounted from the overall score of the current model, penalizing it.

- *exponent_value:* A positive value must be set to allow RHOASo to compute hyperparemeters of higher magnitude.
- *compute_lc* : It set to **True**, the learning curves are computed for the models; it will require more time. Learning curves are useful to solve ties between models where some of them achieve the highest score.

- Auxiliary data section: It contains the name of the *Auxiliary_data* directory under the key *auxiliary_directory_filename.*

- Ouput data section: It contains all the parameters related to the results of the training phase:
  - *output_files_delimiter:* Field delimiter for the output csv files generated at execution time.
  - *output_directory_rootname:* Name of the root directory for the results.
  - *output_directory_name_mif:* Name of the directory that stores the generated csv with the scores of the features computed using Mutual Information Function so they can be checked by the user after the process.
  - *output_directory_name_report:* Name of the directory that will gather all the reports with the results of the performance of the generated models for the current execution.
  - *output_directory_name_prediction_models:* Name of the directory where the optimal computed models (in pickle format) and the dictionary (*prediction_models_dictionary_filename*), that links them with the events, is stored.
  - *validation_data_directory_name:* Name of the directory to store the validation data or the data to predict in the prediction phase.
  - *prediction_models_dictionary_filename*: Key that stores the name of the dictionary (in pickle format) that links optimal models with events. It is generated after the first execution (if there is at least one model after training process) and it is updated after each execution.

- Prediction section: parameters related to the prediction phase are defined inside this section:
  - *target_to_predict:* It specifies the name of the feature with the target values. This feature should contain numeric values. **It must be the same target specified as parameter in the training phase** (see section *How to get started*).
  - *path_to_prediction_models_pkl:* This key contains the path of to the dictionary, generated and updated in the training phase, with the prediction models linked to the events (defined in the key *prediction_models_dictionary_filename*).

- *non_catalogued_data_csv_separator:* This key contains the delimiter of the **input dataset/s** for the **prediction phase.**
- *number_files_to_catalogue:* Maximum number of input csv files to catalogue. 0 specifies all in de directory and other positive number specifies that maximum number of files to read.
- *path_to_directory_input_files_to_catalogue:* Path to the directory that contains the input files that must be labeled (which target must be predicted) using the optimal models.

- Validation: This section is used to specify if the *main_train.py* must create a validation dataset *(validation_mode* key must be set to **True**) and which percentage of the total number of observations in the input dataset/s must be destinated to the validation step(*validation_division_percentaje*) taking real values in the interval [0,1]. **It must be taking into account that if "***validation_mode"* is set to false, no validation data will be generated in the prediction phase and the user will have to provide validation data to check the generalization capacity of the models in the prediction phase.

4-General directories and files overview

Figure 1 shows Input (black), Intermediate Output Files (blue) and Final Output (Red) for the training phase. In the prediction phase the files obtained under prediction_models directory are used predict using the available data.

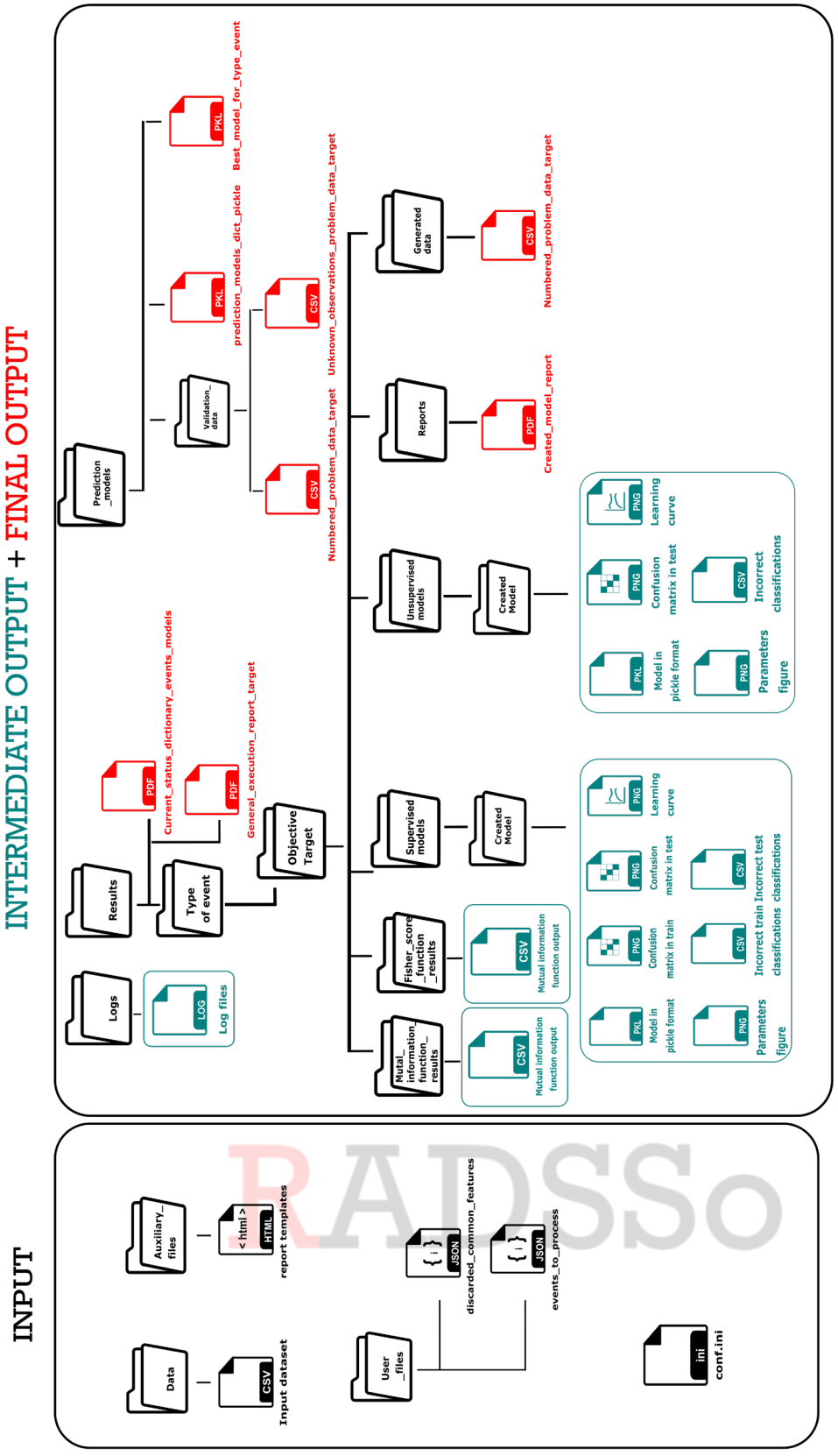


Figure 1. Input, intermediate and final output for training phase.

5-Workflow overview

1- Training Phase

The first phase in the process in the training phase using a target feature. In this phase the events can be extracted automatically (based on the feature that contains them) or from the *events_to_process.json* to solve the multi-CASH problem.

Once the events have been determined, the dataset for the current event is created and it is split in train, test and validation sub-datasets (when corresponding).

The relevant features vary from one event to another, so the prominent ones are selected event by event to configure the models according to the important ones. **The mutual information function** select these features.

The models are created with the optimal parameters using **RHOASo library** and the available models are trained and tested.

The obtained models are compared using a scoring ranking and the best one for the current event and target is stored in the *Prediction_models* directory. The intermediate output files and the reports can be checked at the correct locations (see Figure 1).

To start this phase it must be executed the file main_train.py:

2- Prediction Phase

The models obtained in the training phase can be used in the prediction phase to classify the events that have an unknown value for the target feature.

Figure 2 shows execution workflow.

Figure 2. Training and Prediction

6-How to get started

To train the models:

Input data in csv files must be place in *Data* directory (**do not create subdirectories inside *Data*** folder).

Automated way

The tool will try to create the models for the *target* (target feature to train the models) extracting the names of the events from the feature specified in the key *event_name_feature* from the *Input Data Section* of the conf.ini

Execution:

$ python main_train.py **target**

Semi-automated way

The tool will read the event from the *events_to_process.json* with the relevant features, if any, specified by the user and the discarded features, if any, specified by the user. The process is also automated but it offers more freedom to the user.

Execution:

$ python main_train.py **target semi-automated**

If several problems must be solved at the same execution, it can be used the next command to create the *events_to_process.json* through a guided process where the user can specify the problems to solve, between the available ones in the current csv, in addition to the important and discarded features for each one of them.

Execution:

$ python main_train.py **target semi-automated build_files**

To predict using the trained models:

Input data in csv files must be place in *Prediction_models/Validation_data* directory (**do not create subdirectories in Data folder)** to obtain the predictions for the events.

The predictions are made using the **features that were relevant for training the models** and using the feature **obsnumber inserted** at the beginning of the training phase in training and test datasets and in the validation dataset. **Be sure that obsnumber feature is available in the validation dataset before predicting.**

*If the prediction phase was successful there must be a *prediction_models_dict.pickle* insdie *Prediction_models* directory and the corresponding pkl files of the models related to each specific event.

The data with the predicted values of the target is stored in a csv file created in *Prediction_models* directory and a Log with the accuracy of the predictions for the event is also

generated. In addition, a **pdf with the summary of the process must be generated, for each one of the problems classified, using the corresponding model.**

Execution:

$ python **main_prediction.py**

***** the feature used as prediction target must be specified in the Conf.ini