# PML development documentation

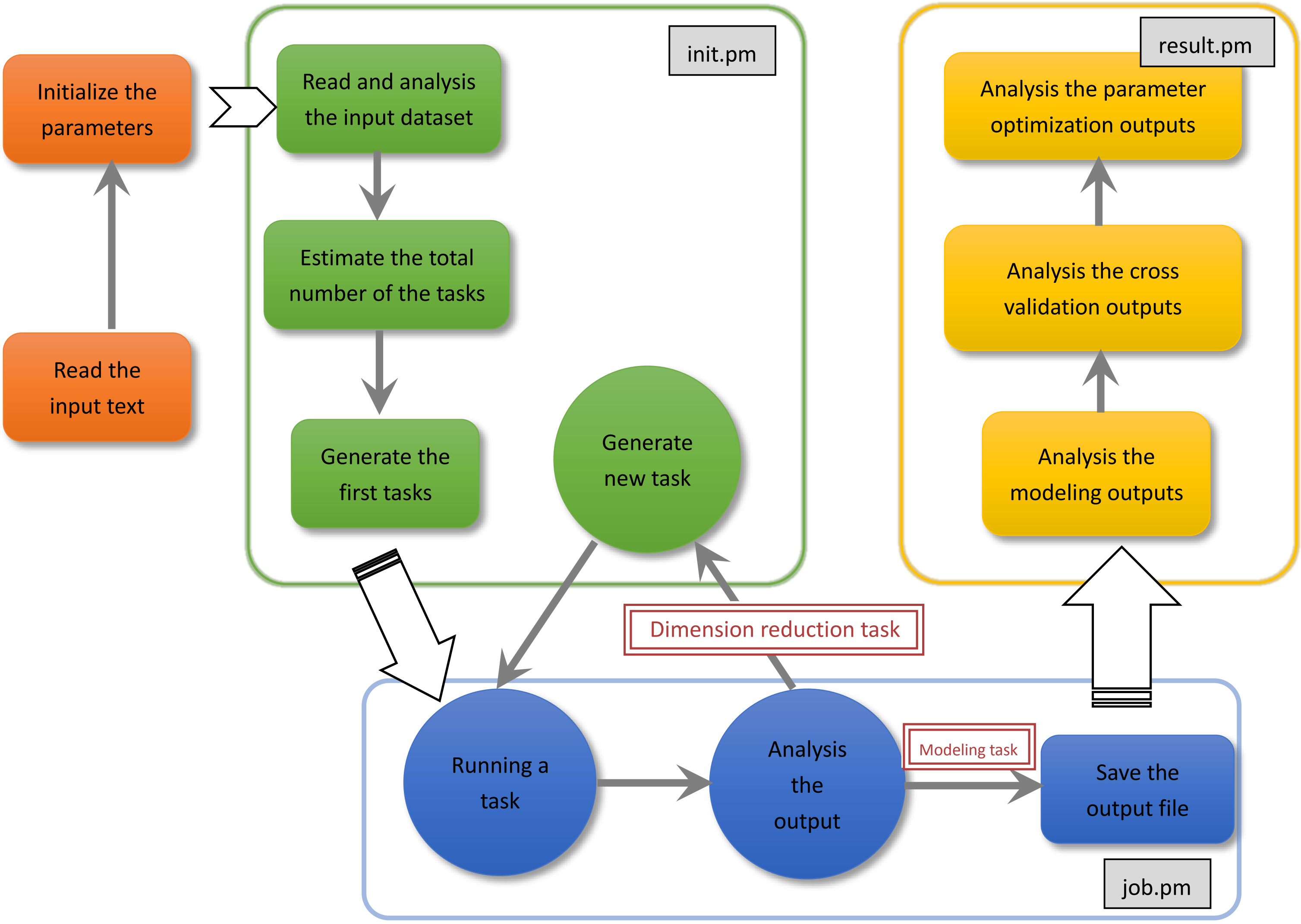
This documentation records the information of the design of PML, including the descriptions of the modules.

# The design of PML

This chapter records some designs of PML, including the whole workflow and some detailed functions.

## The workflow of PML

The basic workflow of PML is as follows:

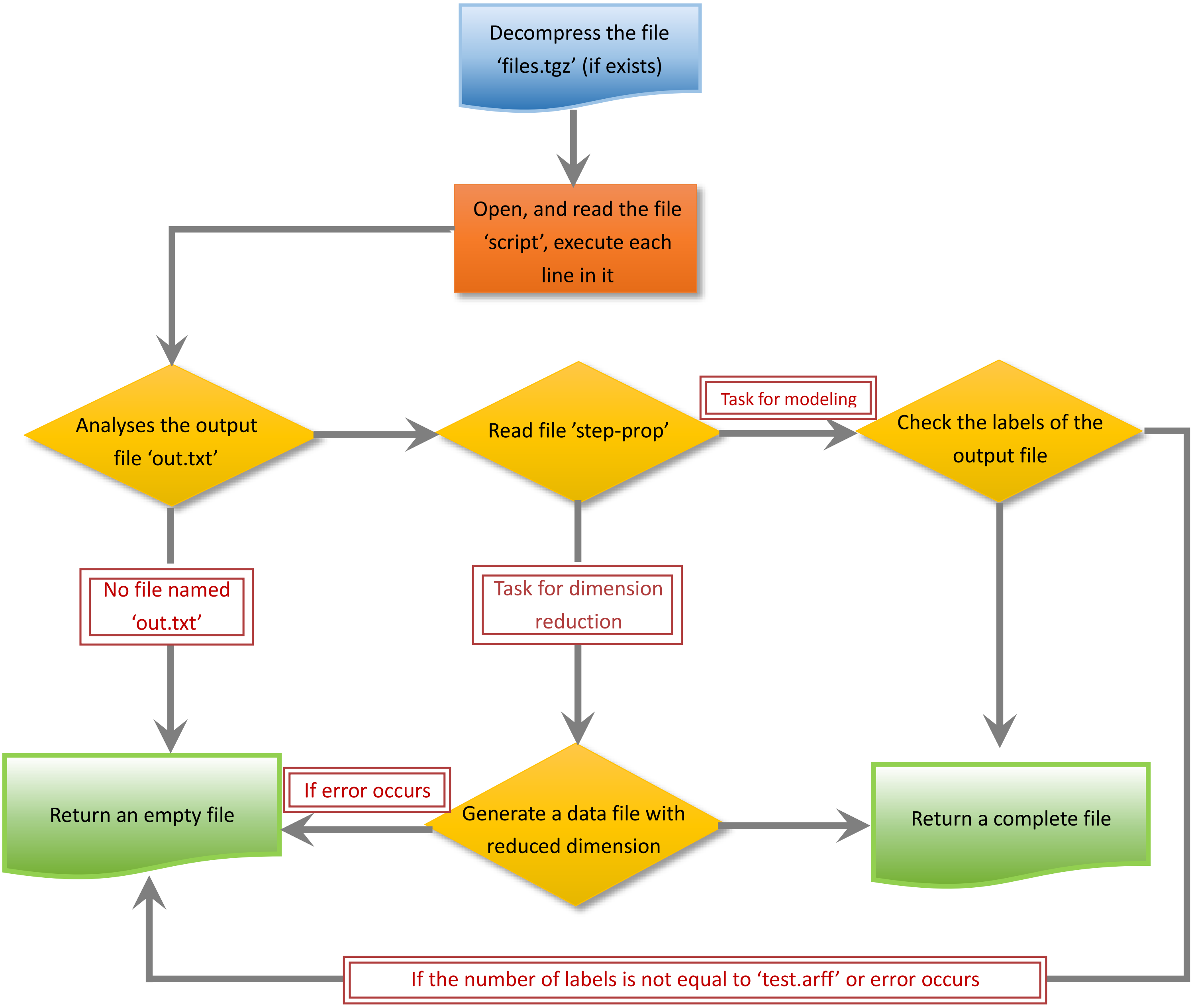

Initialize the parameters
Read the input text
init.pm
Read and analysis the input dataset
Estimate the total number of the tasks
Generate the first tasks
Generate new task
result.pm
Analysis the parameter optimization outputs
Analysis the cross validation outputs
Analysis the modeling outputs
Dimension reduction task
Modeling task
Running a task
Analysis the output
Save the output file
job.pm

This figure reveals that the two modules, init.pm and job.pm, need to work with each other, while the module result.pm is independent.

## The consistent of a task

To support the recover mechanism, PML mainly uses file to record the status and related properties of a task.

A task of PML needs three kinds of files: script (or compressed file), property file, data file(s). The script file records the command lines which need to be executed, and PML will use the 'system()' function to execute them. The property file needs to be named as 'step_prop' and contains the information of dimension reduction. The data file has two patterns of name scheme: 'data.arff' for cluster and variable selection, 'train.arff' and 'test.arff' for modeling and prediction.
This is the workflow of a task:


Decompress the file 'files.tgz' (if exists)
Open, and read the file 'script', execute each line in it
Analyses the output file 'out.txt'
Read file 'step-prop'
Task for modeling
Check the labels of the output file
No file named 'out.txt'
Task for dimension reduction
Return an empty file
If error occurs
Generate a data file with reduced dimension
Return a complete file
If the number of labels is not equal to 'test.arff' or error occurs

# The work file distribute

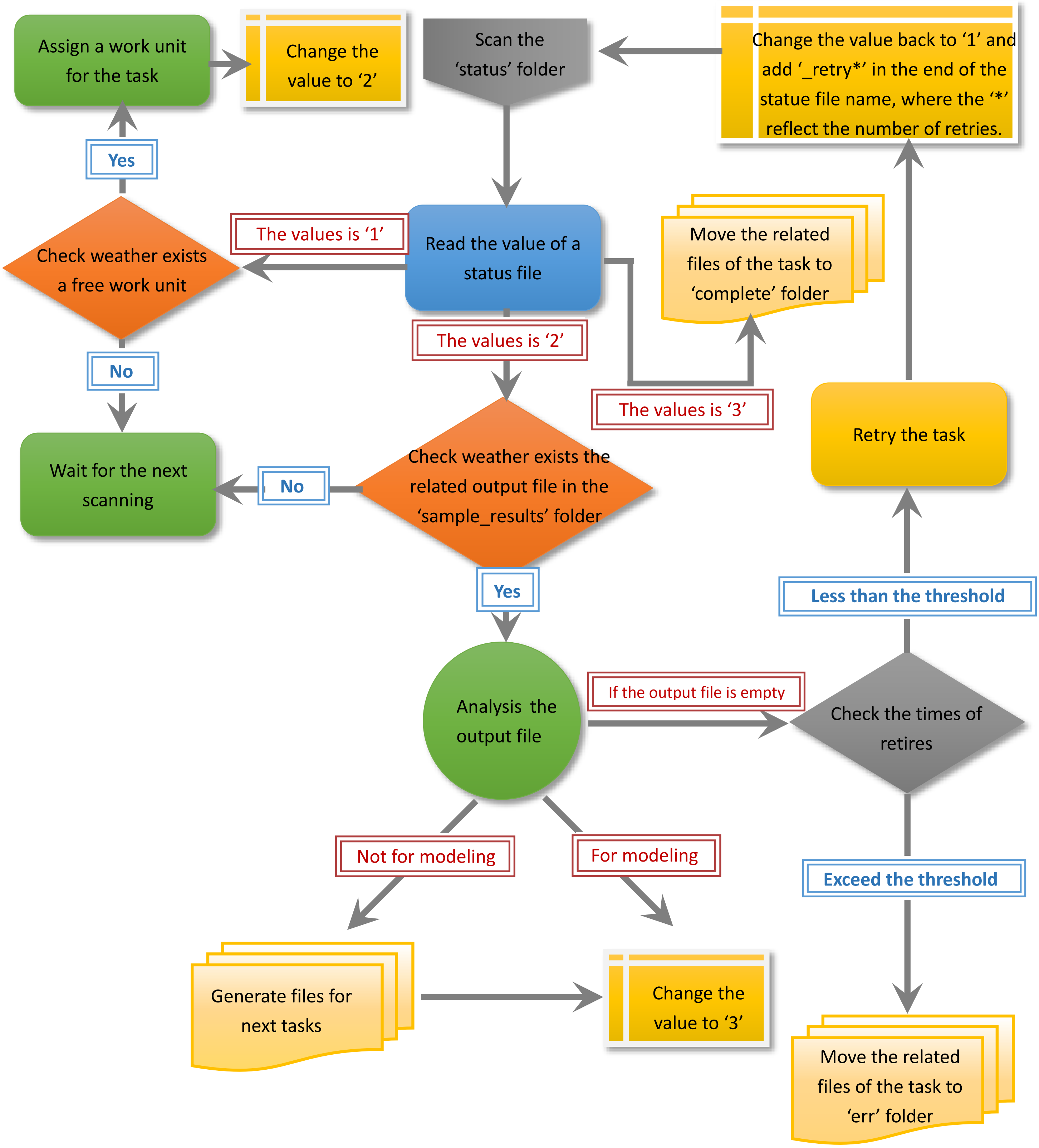
PML will create a work folder for an experiment with the experiment's name. This work folder is in the **<pml_root>/pml/results** folder because that the results files also will be generated there. There are some subfolders in this work directory:
orgdata: Archive the input data file
data: Archive the data files for workflow
stepprops: Archive the files which contain the properties of the tasks
scripts: Archive the script or compressed execute files for the tasks.
jobprops: Archive the property files of the tasks for the result analysis
status: Archive the statue files for the tasks. (the statue file will be described at workflow control)
complete/err: Archive the completed/crashed tasks for the result analysis
results: The HTML format output files which generated from the result analysis

Besides, the 'wu' folder will be created in **<pml_root>/pml** for task execution, and the 'samlple_result' in **<pml_root>/** for archiving the output files temporarily.

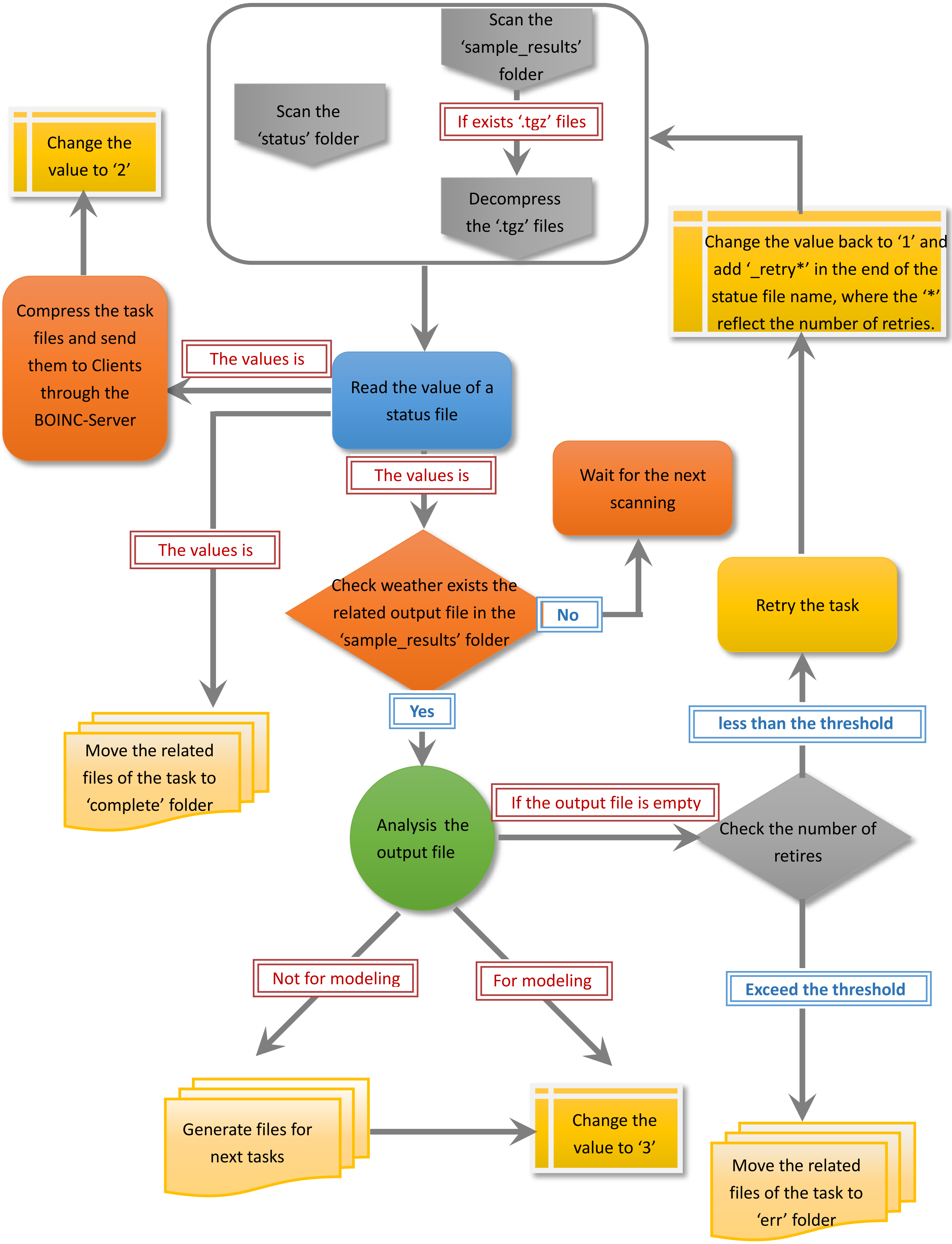
# Workflow control

PML achieves parallel through monitoring the files in status folder. A statue file would contain three kind of values: '1' means the related files (script, property, and data files) of a task have generated and the task is waiting for execution. '2' means the task is running. '3' means the task has completed and is available for the result analysis.

The workflow control of PML-Desktop is a little different from the one of PML-Server. This is the sketch of the workflow control of PML-Desktop:


Assign a work unit for the task
Change the value to '2'
Scan the 'status' folder
Change the value back to '1' and add '_retry*' in the end of the statue file name, where the '*' reflect the number of retries.
Yes
Check weather exists a free work unit
The values is '1'
Read the value of a status file
Move the related files of the task to 'complete' folder
No
The values is '2'
The values is '3'
Wait for the next scanning
Retry the task
No
Check weather exists the related output file in the 'sample_results' folder
Yes
Less than the threshold
Analysis the output file
If the output file is empty
Check the times of retires
Not for modeling
For modeling
Exceed the threshold
Generate files for next tasks
Change the value to '3'
Move the related files of the task to 'err' folder

On the other hand, this is the sketch of the workflow control of PML-Server:


Scan the 'status' folder
Scan the 'sample_results' folder
If exists '.tgz' files
Decompress the '.tgz' files
Change the value to '2'
Compress the task files and send them to Clients through the BOINC-Server
The values is
Read the value of a status file
The values is
The values is
Change the value back to '1' and add '_retry*' in the end of the statue file name, where the '*' reflect the number of retries.
Wait for the next scanning
Check weather exists the related output file in the 'sample_results' folder
No
Yes
Retry the task
less than the threshold
Move the related files of the task to 'complete' folder
Analysis the output file
If the output file is empty
Check the number of retires
Not for modeling
For modeling
Exceed the threshold
Generate files for next tasks
Change the value to '3'
Move the related files of the task to 'err' folder

The differences are concentrated on the way of task assignment and execution. PML-Desktop would copy the files to the subfolder of a work unit directly, but this way is not efficient for PML-Server because of the network transmission. Thus, the tasks would be combined into some large tasks to reduce the time cost of the network transmission.

## Naming scheme for the tasks

With PML, each experiment needs to have a unique name. To be convenient for explanation, the name is set as 'PML_example'.

The naming scheme of PML is simply:

$$\text{TaskName} = \begin{cases} InitName + AddName & if\ the\ task\ does\ not\ have\ a\ parent\ task \\ ParentTaskName + AddName & if\ the\ task\ has\ a\ parent\ task \end{cases}$$

Where the $ParentTask$ means the task that could provide an output for the new task. The $InitName$ is generated by adding the string '_train' to the end of the experiment name. In this example, the $InitName$ is 'PML_example_train'.

The naming schemes of the $AddName$ for cluster and variable selection are similar:

| AddName | note |
|---|---|
| _clu_*a*_*b* | Cluster tasks without parameter optimization |
| _clu_*a.c*_*b* | Cluster tasks with parameter optimization |
| _fea_*a*_*b* | Variable selection tasks without parameter optimization |
| _fea_*a.c*_*b* | Variable selection tasks with parameter optimization |

The '*a*' is the sequence number of the used methods (correspond to the CLUSTER_ARG and FEATURE_SELECT_ARG). The '*b*' is the sequence number of the reduced dimensions (correspond to the CLUSTER_OUT_INSTANCES and FEATURE_OUT_FEATURES). The '*c*' means the sequence number of the changed parameters (or parameter combinations).

The naming scheme of the $AddName$ for modeling is as follows:

| AddName | note |
|---|---|
| _tt_*a*_outer_*o* | Modeling task without parameter optimization and inner validation |
| _tt_*a.c*_outer_*o* | Modeling task without inner validation |
| _tt_*a*_inner_*i*_outer_*o* | Modeling task without parameter optimization |
| _tt_*a.c*_inner_*i*_outer_*o* | Modeling task with parameter optimization and inner validation |

The '*a*' and '*c*' are the sequence number of the used methods (TT_ARG) and the changed parameters (TT_ARG_OPT). The '*i*' and 'o' are the sequence number of the inner and outer cross-validation.

There are some examples of the task names:

PML_example_train_clu_0.2_1

PML_example_train_clu_1.1_0_fea_3_1

PML_example_train_clu_1.1_0_fea_3_1_tt_2.1_inner_3_outer_2

All the files in the subfolders (data, stepprops, scripts, jobprops, status and results) comply with the naming scheme. Therefore, PML could generate a tree for the workflow and find the input/output dataset in the output analysis.

# Some detailed functions

## Fault tolerant

As mentioned in Workflow control and The consistent of a task, a task would generate a non-empty file ‘out.txt’, and then PML would check the file and translate it for the next operations. If the file ‘out.txt’ is absent, the size of it is zero, or the content of it does not fit any one of the templates, PML would retry the task at first and record the times of retry. If the times exceed the preset threshold, the related files of the task would be moved to the ‘err’ subfolder, and the related information would be provided in the correlated HTML output page for users to check or retry it.

## Recovery

With the statue files, PML could recover and continue an unexpectedly terminated experiment.

For PML-Server, the tasks are managed as the work units of BOINC-Server, thus no adjustment is needed. However, for PML-Desktop, the tasks are executed in the **<pml_roots>/wus** folder, and a hash table, which is used to manage the computing resource, needs to be regenerated. Thus, all the statue files that had the value ‘2’ would be changed back to ‘1’ and the folder **<pml_roots>/wus** would be reset. This means that for those tasks, which were being executed when the process of PML-Desktop was terminated, need the restarts after the recovery. However, other tasks are not affected.

## The random number generator

PML uses the LCG (linear congruent generators) to generate the random sequence:

$$x_{n+1} = a \times x_n + c \; mod \; M$$

where

$$x_0 = seed$$

and

$$\begin{cases} M = 2^L \\ a = 8 \times \left\lfloor \frac{M}{64} \times \pi \right\rfloor + 5 \\ c = 2 \times \left\lfloor \frac{M}{2} \times 0.211324865 \right\rfloor + 1 \\ L = min\{\, l \mid where \; 2^l > SeqLength\} \end{cases}$$

The $SeqLength$means the length of the random sequence, and the values of a and c are used according to [Kunth, 1969]. With the values, the cycle of this LCG is M, and the numbers larger than the $SeqLength$would be ignored.

# Modules of PML

This chapter demonstrates the functions with the usages in the modules of PML. PML has three modules, pml::init, pml::job, pml::result, and users could found them at **<pml_root>/pml/src/pml**. To use them, users need to copy the folder into some path included in the environment variable **PATH** or add the path into **@INC** manually. For example, this line could be useful:

**BEGIN{ push @INC,'./pml/src'}**

Then the codes of PERL would be valid:

use pml::init;
use pml::job;
use pml::result;

And some globe variables are needed for most functions:

**our $name;**
**our $prog_dir = './pml';**

Where:

$name : The name of the experiment
$prog_dir : Where the work folder of PML located, default is './pml'

## init.pm

This module includes functions about initializations and task generations and could be found at **<pml_root>/pml/src/pml/init.pm**. It could be utilized with this form:

**use pml::init;**

Note that the path of the folder should be included in the environment variable **@INC**, and these functions as follows could be utilized:

### analysis_data

**Usage:**

analysis_data($input_name , $output_name);

**Description:**

Analysis the ARFF format data, return the details.

$input_name: The name of the data need to be analysis.
$output_name: The name of the file which record the details of the file.

**Return:**

An ARRAY:
number of instances, number of attributes, is for classify, if have missed value

## creat_floders

**Usage:**
creat_folders();

**Description:**
Create required folders for new experiment.

**Return:**
None

## creat_job_clu

**Usage:**
creat_job_clu( $script_name , $experiment_name , $max_memory , [@cluster_arg] , [@cluster_arg_grid] , [@cluster_out_instances] );

$script_name: The name of the script
$experiment_name: The name of the experiment
$max_memory: The allowed use of memory. (Only valid for weka tasks)
@cluster_arg: Cluster methods.
@cluster_arg_grid: The parameter optimization information.
@cluster_out_instances: The instances after cluster

**Description:**
Analysis the input of users, generate related script and status files.

The naming scheme is
**$script_cname_clu*argnum*.*optnum*_*outnum***
Where the ***argnum***, ***optnum*** and ***outnum*** means the index of @cluster_arg, @cluster_arg_grid, @cluster_out_instances.

Generated scripts could be found at **pml/result/$experiment/scripts**. Besides, the scripts will be moved to folder **complete** or **err** during the process of tasks executing.

**Return:**
none

**Use functions:**

| | | |
|---|---|---|
| creat_job_clu_3rd | alalysis_grid_parm | replace_grid_parm |

| | | |
|---|---|---|
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | |

# creat_job_fea

**Usage:**

creat_job_fea( $script_name , $experiment_name , $max_memory , [@feature_select_arg] , [@feature_arg_grid] , [@feature_out_features] );

$script_name: The name of the script
$experiment_name: The name of the experiment
$max_memory: The allowed use of memory. (Only valid for weka tasks)
@feature_select_arg: Variable selection methods.
@feature_arg_grid: The parameter optimization information.
@feature_out_features: The out variables after variable selection

**Description:**

Analysis the input of users, generate related script and status files.

The naming scheme is
**$script_cname_fea*argnum*.*optnum*_*outnum***
where the ***argnum***, ***optnum*** and ***outnum*** means the index of @feature_select_arg, @feature_arg_grid, @feature_out_features.

Generated scripts could be found at **pml/result/$experiment/scripts**. Besides, the scripts will be moved to folder **complete** or **err** during the process of tasks executing.

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| creat_job_fea_3rd | alalysis_grid_parm | replace_grid_parm |
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | |

# creat_job_tt

**Usage:**

creat_job_tt( $script_name , $experiment_name , $max_memory , [@tt_arg] , [@tt_arg_grid] , $inner_folds , $outer_folds );

$script_name: The name of the script

$experiment_name: The name of the experiment
$max_memory: The allowed use of memory. (Only valid for weka tasks)
@tt_arg: Modeling (train test) methods.
@tt_arg_grid: The parameter optimization information.
$inner_folds: The number of inner folds
$outer_folds: The number of outer folds

**Description:**
Analysis the input of users, generate related script and status files.

The naming scheme is
**$script_cname_tt*argnum.optnum*_inner_*innum*_outer_*outnum***
where the ***argnum***, ***optnum*** means the index of @tt_arg, @tt_arg_grid. And the ***innum*** and ***outnum*** means the index of inner folds and outer folds.

Generated scripts could be found at **pml/result/$experiment/scripts**. Besides, the scripts will be moved to folder **complete** or **err** during the process of tasks executing.

**Return:**
none

**Use functions:**

| | | |
|---|---|---|
| creat_job_tt_3rd | alalysis_grid_parm | replace_grid_parm |
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | |

## data_copy

**Usage:**
data_copy( $file_in , $file_out );

$file_in: The path of source file.
$file_out: The target path.
**Description:**
Copy the ARFF data file and escape the lines only with /\s+/

This function only used for reduce the error when modifying the ARFF file.

**Return:**
none

## del_job_num

**Usage:**

del_job_num ( $job_name , $hash_tasks_l );

$job_name: The name of task with error
$hash_tasks_l: A reference of hash index which generated by get_job_num

**Description:**

PML could estimate the number of tasks need to be executed. However, if a task failed, which means the related tasks would be canceled, this function could calculate the number of canceled tasks.

**Return:**

The number of canceled tasks.

## get_threads_number

**Usage:**

get_threads_number();

**Description:**

Get the number of threads of one machine.

Note that the number of threads might be multiple of CPU cores in terms of Intel hyper-threading technology.

**Return:**

The number of threads.

## get_job_num

**Usage:**

my %hash_tasks;
get_job_num(\%hash_tasks);

**Description:**

Analysis the input of users and estimate the number of tasks.

The hash %hash_tasks would be recorded with the detail of jobs after finishing this function and could be used for function del_job_num

**Return:**

The number of tasks.

**Use functions:**
get_grid_num

## init_wu

**Usage:**
init_wu();

**Description:**
Create the folders for parallel computing of PML-desktop.

**Return:**
None

## reset_desktop

**Usage:**
reset_desktop( $experiment_name )

$experimen _name: The name of experiment.

**Description:**
Clean the related files of specified experiment of PML-desktop.

Use function rmtree.

**Return:**
None

## reset_server

**Usage:**
our $name = “experiment_name”;
reset_server();

**Description:**
Clean the related files and records of specified experiment of PML-server.

**Return:**
None

## rmtree

**Usage:**
rmtree($folder_path);

$folder_path: The path of the folder which expected to be deleted.

**Description:**
Delete specified folder.

**Return:**
None

# Private functions

These functions in **pml::init** could not be utilized directly by the function name, but they act as components of the module. If a private function need to be utilized, users could add the **pml::init::** in front of the function or add a line:
**use pml::init qw(function_name);**

### alalysis_grid_parm

**Usage:**
analysis_grid_parm($string);

**Description:**
Analysis the input of parameter optimization, generate the details and return them.

For example, if the $string is "4:6", then the return ARRAY would be (4 , 5 , 6).

**Return:**
An ARRAY of the details of the changed parameters.

### alalysis_grid_parm_3rd

**Usage:**
analysis_grid_parm_3rd($string);

**Description:**

Analysis the input of parameter optimization for 3rd program, generate the details and return them.

**Return:**

An ARRAY of the details of the changed parameters.

**Use functions:**

| | | |
|---|---|---|
| alalysis_grid_parm | get_grid_position | |

## compress_dir

**Usage:**

our $compress_level = 9;

compress_dir($method_name , $method_type);

**Description:**

Compress the specified method for next step. PML would find the method from pml/config/$method type/$method_name

**Return:**

None

## creat_job_clu_3rd

**Usage:**

creat_job_clu_3rd( $script_name , $experiment_name , $max_memory , [@cluster_arg] , [@cluster_arg_grid] , [@cluster_out_instances] );

$script_name: The name of the script

$experiment_name: The name of the experiment

$max_memory: The allowed use of memory. (Only valid for weka tasks)

@cluster_arg: Cluster methods.

@cluster_arg_grid: The parameter optimization information.

@cluster_out_instances: The instances after cluster

**Description:**

If the tasks is 3rd program which added by users, this function would be utilized.

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| split_grid | alalysis_grid_parm_3rd | replace_grid_parm_3rd |

| | | |
|---|---|---|
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | dir_copy |
| recover_file | compress_dir | write_status |
| detail_back | rmtree | |

## creat_job_fea_3rd

**Usage:**

creat_job_fea_3rd( $script_name , $experiment_name , $max_memory , [@feature_select_arg] , [@feature_arg_grid] , [@feature_out_features] );

$script_name: The name of the script
$experiment_name: The name of the experiment
$max_memory: The allowed use of memory. (Only valid for weka tasks)
@feature_select_arg: Variable selection methods.
@feature_arg_grid: The parameter optimization information.
@feature_out_features: The out variables after variable selection

**Description:**

If the tasks is 3rd program which added by users, this function would be utilized.

The naming scheme is
**$script_cname_fea*argnum.optnum_outnum***
where the ***argnum***, ***optnum*** and ***outnum*** means the index of @feature_select_arg, @feature_arg_grid, @feature_out_features.

Generated scripts could be found at **pml/result/$experiment/scripts**. Besides, the scripts will be moved to folder **complete** or **err** during the process of tasks executing.

**Return:**

none

**Use functions:**

| | | |
|---|---|---|
| split_grid | alalysis_grid_parm_3rd | replace_grid_parm_3rd |
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | dir_copy |
| recover_file | compress_dir | write_status |
| detail_back | rmtree | |

## creat_job_tt_3rd

**Usage:**

creat_job_tt_3rd( $script_name , $experiment_name , $max_memory , [@tt_arg] , [@tt_arg_grid] , $inner_folds , $outer_folds );

$script_name: The name of the script
$experiment_name: The name of the experiment
$max_memory: The allowed use of memory. (Only valid for weka tasks)
@tt_arg: Modeling (train test) methods.
@tt_arg_grid: The parameter optimization information.
$inner_folds: The number of inner folds
$outer_folds: The number of outer folds

**Description:**

If the tasks is 3rd program which added by users, this function would be utilized.

The naming scheme is
**$script_cname_tt*argnum*.*optnum*_inner_*innum*_outer_*outnum***
where the ***argnum***, ***optnum*** means the index of @tt_arg, @tt_arg_grid. And the ***innum*** and ***outnum*** means the index of inner folds and outer folds.

Generated scripts could be found at **pml/result/$experiment/scripts**. Besides, the scripts will be moved to folder **complete** or **err** during the process of tasks executing.

**Return:**

none

**Use functions:**

| | | |
|---|---|---|
| split_grid | alalysis_grid_parm_3rd | replace_grid_parm_3rd |
| write_script_isweka | write_script_prop | print_step_prop |
| write_scripts_and_status | get_grid_position | dir_copy |
| recover_file | compress_dir | write_status |
| detail_back | rmtree | |

## dir_copy

**Usage:**

dircopy($dir_source , $dir_target);

**Description:**

Copy the folder.

**Return:**
None

## detail_back

**Usage:**
detail_back($org_path , $target_path , \@grid_parms)

@grid_parms: ARRAY contains the information of changed files, and generated by function alalysis_grid_parm_3rd

**Description:**
Copy specified files.

During the process of parameter optimization for 3rd program, some files might be changed, thus these files need to be recovered. This function do this job.

**Return:**
None

## get_grid_num

**Usage:**
get_grid_num($method_num, $method_type);

$method_num: The number of utilized method
$method_type: clu, fea or tt

**Description:**
Estimate the total number of changed values of specified method.

**Return:**
The number of changed values

**Use functions:**

| | | |
|---|---|---|
| alalysis_grid_parm | alalysis_grid_parm_3rd | |

## get_grid_position

**Usage:**

get_grid_position($num , @seq);

**Description:**
Get the number of tasks in the changed parameters of one method. For example, if a method has 3 parameters, A B and C, need to be changed, such as -A 1:3 -B 1:5 -C YES;NO. Then there would be 30 scripts need to be generated. This function maps each combination of changed parameters to the sequence number 1-30. There the value 1 mapped with '-A 1 -B 1 -C 1', 10 with '-A 1 -B 4 -C YES'.
**Return:**
An ARRAY of the position of each changed parameters. The length is the number of changed parameters of this method.

## modify_waffles_out_num

**Usage:**
modify_waffles_out_num ($method_type , $method_name , $out_num);

**Description:**
Modify the cluster/varselect output infomation for the script in order to get the correct output dimension.

**Return:**
None

## print_step_prop

**Usage:**
print_step_prop($file_name , @detail);

@detail: The detail of each task.

**Description:**
Generate a file to help PML executing tasks in parallel.

**Return:**
None

## recover_file

**Usage:**
recover_file ($method_type , $method_name , $file_name)

$method_type: clu, fea or tt
$method_name: the name of the method
$file_name: the name of the file which need to be recover

**Description:**

This function action similar with function detail_back. The difference is that this function could figure out and recover the files with same names in the platform subfolders.

**Return:**

None

## replace_grid_parm

**Usage:**

replace_grid_parm($para_name , $para_value , @script_line);

$para_name: The name of the parameter which need to be replace
$para_value: The value of the parameter which need to be replace
@script_line: The separated line which contains the parameter.

**Description:**

Replace the value of specified parameter in the related line.

**Return:**

The changed @script_line

## replace_grid_parm_3rd

**Usage:**

replace_grid_parm_3rd($method_type, $method_name, $file_name, $org_str, $p, $num, $change)

$method_type: clu, fea or tt
$method_name: The name of the 3rd method
$file_name: The name of the file which contain the needed parameter.
$org_str: The string used to be matched in the file
$p: The pattern to separate the line.
$num: To specify the separated elements.
$change: Change the specified element.

**Description:**

Replace the value of specified parameter by modify the related file.

More details could be found at the section **parameter optimization** in the document **PML_manul**.

**Return:**
None

## split_grid

**Usage:**
split_grid($p , $line);

$p: The pattern to separate the line.
$line: The line of the optimization information written by users.

**Description:**
Separate the line into elements by methods for the next operations.

**Return:**
An ARRAY with separated information of methods.

## uncompressdir

**Usage:**
uncompressdir ($file_name);

**Description:**
Decompress the .tgz file.

**Return:**
None

## init_waffles_options

**Usage:**
init_waffles_options ( );

**Description:**
#This function is for PML to modify some parameters of waffles

**Return:**
None

## write_script_isweka

**Usage:**
write_script_isweka($file_name , $isweka);

$isweka: 1 if is, 0 else

**Description:**
Write the information into specified file for result analysis.

**Return:**
None

## write_script_prop

**Usage:**
write_script_prop($file_name , @details)

**Description:**
Write the information into specified file for result analysis.

**Return:**
None

## write_scripts_and_status

**Usage:**
write_scripts_and_status($experiment_name , $out_name , @script_details);

**Description:**
Generate the script file and status file for task executing.

**Return:**
None

## write_status

**Usage:**
write_status( $out_name , @script_details);

**Description:**

Only generate the status file.

**Return:**
None

# job.pm

This module includes functions about initializations and task generations and could be found at **pml/src/pml/job.pm**. It could be utilized with this form:
**use pml::job;**
Note that the path of the folder should be included in the environment variable **@INC**. And these functions as follows could be utilized:

## analysis_data_j

**Usage:**
analysis_data_j($input_name , $output_name);

**Description:**
The same as the analysis_data in pml::init but this function could return the classes if the data is for classify.

**Return:**
An ARRAY, the first 4 elements are:
Number of instances, number of attributes, if has missed value, is for classify

The rest elements are the classes of the data if the data is for classify.

## data_for_tt

**Usage:**
our $inner_folds = 5;
our $outer_folds = 5;
data_for_tt( $data_name);

$inner_folds: The number of inner folds. If valued as 0, no data file for inner folds would be generated.
$outer_folds: The number of outer folds.
$data_name: The name of data that need to be separate.

**Description:**
Generate the data for n-fold cross.

**Return:**

None.

**Use functions:**

| | | |
|---|---|---|
| analysis_data | randperm | get_train_test_list |
| print_one_fold | get_train_list_for_inner | |

## data_for_independent

**Usage:**

data_for_independent( $outfile_name)

$outfile_name: The name of the output data file

**Description:**

Generate the dataset for independent test.

The training dataset is the prewierious data

Filter the attributes of test dataset if necessary

**Return:**

None.

## file_process_out

**Usage:**

our $name = “experiment_name”;

file_process_out( $task_name, @steps);

$task_name: The name of the finished task.

@steps: The sequence of the process. Like: (‘clu’ , ‘fea’ , ‘tt’)

**Description:**

Analysis the output file of a finished task, then decide mark the task as complete or error.

For complete tasks, generate the related n-fold data files if needed.

**Return:**

0 if the task marked as complete, 1 if not.

**Use functions:**

| | | |
|---|---|---|
| analysis_data | isarff | data_for_tt |

# get_train_test_list

**Usage:**

my @sequence_counter

get_train_test_list($fold_count, \@sequence_counter, \@each_fold, $num_folds, \@fold_test_files, \@fold_train_files, \@rand_sequence, \@data_detail, $outer_fold_count)

$fold_count: Specify the fold in the n-fold need to be generated.
@sequence_counter: Reference of the records of the start position of the data. For example, if the data has 4 class, then the @sequence_conter would be initialized with (0 , 0 , 0 , 0) and added by the instances used by each fold.
@each_fold: Reference of the number of instances of each classed in one fold.
$num_folds: The number of folds.
@fold_test_files: An ARRAY with the test files. This parameter is for specifying the n-fold data by users, PML would generate data by random number if @fold_test_files and @fold_train_files are empty.
@fold_train_files: An ARRAY with the train files. This parameter is for specifying the n-fold data by users, PML would generate data by random number if @fold_test_files and @fold_train_files are empty.
@rand_sequence: An pseudorandom number ARRAY generated by function randperm.
@data_detail: The ARRAY generated by function analysis_data_j.
$outer_fold_count: If the data is for inner fold, this parameter is needed to specify the related outer fold.

**Description:**
Generate the training and test sequence for specified fold of the data.

Note that if the data has two or more classes and @fold_test_files and @fold_train_files are empty, the generated list would maintain the proportion of each class. For example, if the data has 3 classes with 20, 30 and 50 instances, and 10 folds would be generated. Then for each fold, the instances for each class would become 2, 3 and 5.
**Return:**
An ARRAY with two hash tables:
my @out = get_train_test_list( … );
my %train_list = @{$out[0]};
my %test_list = @{$out[1]};

# init_pml_job

**Usage:**

our $name = ‘experiment_name’;
init_pml_job();

**Description:**

Initialize some global variables for this module.

**Return:**

None

# job_creator_server

**Usage:**

job_creator_server()

**Description:**

Create the task for PML-server by scaning the status folder
This creator could compress 1 or more tasks as one unit to execute

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| sfile_analysis | get_boinc_path | compress_tasks |
| create_boinc_wu | | |

# job_creator_desktop

**Usage:**

job_creator($task_name , $wu)

$wu: The number of work file where the task would be executed.

**Description:**

Create computing task for PML-desktop

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| sfile_analysis | | |

## job_execute_desktop

**Usage:**

job_execute_desktop($wu , $task_name , $main_path)

$wu: The number of work file where the task would be executed.

$main_path: The output file of this task would be move to $main_path/sample_result

**Description:**

Execute the created task for PML-desktop, copy the output file to the specified folder. Then clean the files in the work folder.

**Return:**

None

## print_one_fold

**Usage:**

print_one_fold($data_org_name, $add_name, $is4class, \@data_detail, \@train_test_list);

$data_org_name: The name of the original data.

$add_name: The string needed to be add to change the $data_org_name.

$is4class: If the data is for classify

@data_detail: An ARRAY generated by the function analysis_data_j.

@train_test_list: An ARRAY generated by the function get_train_test_list.

**Description:**

Print the training and test data of one fold into related files.

**Return:**

None

## randperm

**Usage:**

randperm($length , $seed);

$length: The length of the rand numbers.

$seed: The random seed. Default is 1.

**Description:**
Generate the pseudorandom sequence from 1 to $length.

**Return:**
An ARRAY with the random numbers.

# Private functions

These functions in **pml::job** could not be utilized directly by the function name, but they act as components of the module. If a private function need to be utilized, users could add the **pml::job::** in front of the function or add a line:
**use pml::job qw(function_name);**

## analysis_data

**Usage:**
analysis_data($input_name , $output_name);

**Description:**
The same as the analysis_data in pml::init but this function could return the classes if the data is for classify.

**Return:**
An ARRAY, the first 4 elements are:
Number of instances, number of attributes, if has missed value, is for classify

The rest elements are the classes of the data if the data is for classify.

## create_boinc_wu

**Usage:**
create_boinc_wu($file_name);

$file_name: The name of the compressed .tgz task file

**Description:**
Create a boinc workunit by cmd interface.

**Return:**

1 if successful

## get_boinc_path

**Usage:**
get_boinc_path ($file_name);

$file_name: The name of the file which need to be copy

**Description:**
Get the target path of BOINC download folder for the input name.

**Return:**
The path of the target folder

## compress_tasks

**Usage:**
compress_tasks($hash_tar_file_p , $tar_name);

$hash_tar_file_p: The reference of the hash table %hash_tar_file
$tar_name: The name of the compressed .tgz file

**Description:**
Compress the task files into one file.

**Return:**
None

## find_position

**Usage:**
find_position($val , @arr)

$val: The value need to be located
@arr: The ARRAY that $val in.

**Description:**
Get the position of specified value in the related ARRAY.

**Return:**
The first position in the ARRAY if the value is located, or return -1.

## get_lsa

**Usage:**
get_lsa(\@lsa_vector_files , $label);

@lsa_vector_files: The files contain the lsa vectors generated by weka
$labes: The label with the related line.

**Description:**
Get an instance with the values from the output files of latent semantic analysis.

**Return:**
ARRAY of values for an instance

## get_pca

**Usage:**
get_pca(\@attributes_name,\@eigen_m,\%hash_var_names,@input_attribute);

@attributes_name: The names of the attributes in the data
@eigen_m: Eigen matrix (2_dimention ARRAY)
%hash_var_names: The reference from the names of attributes to the related Eigen vectors.
@input_attribute: The attribute in the original data.

**Description:**
Calculate the principle components of the related attribute through the use of Eigen matrix.

**Return:**
ARRAY with principle components

## get_train_list_for_inner

**Usage:**
get_train_list_for_inner($data_name , $label , $inner_folds_count , $instance_num);

$data_name: The name of output data
$label: Specify the label of the data.
$inner_folds_count: The number of the inner fold
$instance_num: the number of the instances.

**Description:**
Generate the train list for inner fold which is related with the outer fold.

This function is only for the situation that the output files are generated by files but the files for inner folds need to be generated by random number.

**Return:**
An ARRAY with the sequence numbers for the train list

## isarff

**Usage:**
isarff( $file_name);

**Description:**
Judge if the data file is ARFF format.

**Return:**
1 if true, 0 if false

## process_clu

**Usage:**
process_clu($out_name , $org_name , $out_prop , $seed);

$out_name: The name of output data file
$org_name: The name of the data file which would be clustered.
$out_prop: The proportion of the instances of the output file
$seed: Random seed

**Description:**
Reduce the dimension of the instances.

The instanced would be reduced by the proportion of the classes. For example, if a data has 3 classes and the instances are 10, 20 and 30, the proportion is 0.1 (10$), then the output data would contain the instances 1, 2 and 3 for related classes.

Besides, if $out_prop is more than 1, PML would calculate the proportion through dividing the $out_prop by the number of instance of the original data.

**Return:**
None

**Use functions:**

| | | |
|---|---|---|
| find_position | randperm | |

## process_fea

**Usage:**

process_fea($out_name , $org_name , $out_num , $nfold_threshold);

$out_name: The name of output data file
$org_name: The name of the data file which would reduce variables.
$out_num: The number of variables after variable selection.
$nfold_threshold: The threshold in selecting variables (only valid in n-fold selection)

**Description:**

Reduce the dimension of the variables (features).

This function will detected the output file and decide the way of dimension reduction for different types of variable selection methods.

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| get_lsa | get_pca | |

## sfile_analysis

**Usage:**

sfile_analysis($task_name);

**Description:**

Analysis the type of the task.

**Return:**

ARRAY with two element:
$isweka: Whether the task would executed with weka. (1 or 0)
$istt: Whether the task would model data. (1 or 0)

# result.pm

This module includes functions about initializations and task generations and could be found at

pml/src/pml/result.pm. It could be utilized with this form:
use pml::result;
Note that the path of the folder should be included in the environment variable @INC. And these functions as follows could be utilized:

# analysis_out_files

**Usage:**
analysis_out_files();

**Description:**
Analysis the results of all the tasks, then generate the output html files, related trees and tables.

**Return:**
None

**Use functions:**

| | | |
|---|---|---|
| get_related_name | get_nf_name | get_gd_name |
| get_parent_name | get_relate_file | show_out_files_progress |
| analysis_nfold | analysis_grid | replace_out_name |
| print_matrix_trans | | |

# init_pml_result

**Usage:**
init_pml_result();

**Description:**
Initialize some global variables for this module.

**Return:**
None

# show_process_err

**Usage:**
show_process_err($task_name , $del_tasks , \$complete_tasks , \$total_tasks);

$del_tasks: Generated by function del_job_num.
$complete_jobs: The number of complete tasks.
$total_tasks: The number of the total tasks.

**Description:**

If a task has crashed, this function could generate a message about the name of failed task and the reduced number of tasks.

**Return:**

None

## show_out_files_progress

**Usage:**

show_out_files_progress($complete_tasks , $total_tasks);

**Description:**

Show a progress bar of the current progress.

**Return:**

None

# Private functions

These functions in **pml::result** could not be utilized directly by the function name, but they act as components of the module. If a private function need to be utilized, users could add the **pml::result::** in front of the function or add a line:
**use pml::result qw(function_name);**

### analysis_nfold

**Usage:**

analysis_nfold(\%is_complete , \%nodes , \%parents , \%nfold_nodes);

%is_complete: A hash tables records the whether task was completed or failed
%nodes: A hash tables records the serial of tasks in the output tree
%parents: A hash tables records the parent nodes of each nodes.
%nfold_nodes: A hash tables records the node of nfold.

**Description:**

Generate the analysis of n-fold cross validation based on the output files of function get_relate_file.

**Return:**

A matrix (2-dimension ARRAY) contain the best methods with the best evaluation criterions values.

**Use functions:**

| | | |
|---|---|---|
| get_gd_name | replace_out_name | get_script_detail |
| init_best_accs | read_html_list | updata_confusion_matrix |
| mean_acc | get_overall | mean_acc_overall |
| print_acc | update_best_acc | show_out_files_progress |
| print_acc_all_body | print_acc_all_head | read_html_list_nf |

## analysis_grid

**Usage:**

analysis_grid(\%is_complete , \%nodes , \%parents , \%grid_nodes , \%related_names);

%is_complete: A hash tables records the whether task was completed or failed
%nodes: A hash tables records the serial of tasks in the output tree
%parents: A hash tables records the parent nodes of each nodes.
%grid_nodes: A hash tables records the node of parameter optimization analysis.

**Description:**

Generate the analysis of n-fold cross validation based on the output files of function analysis_nfold

**Return:**

none

## cal_acc_c

**Usage:**

cal_acc_c([@org_list] , [@out_list] , [%labels_hash] , [@labels]);

@org_list: The labels of the data.
@out_list: The predicted labels.
%labels_hash: The sequences of the instances with labels.
@ labels: The labels of classes.
All of the input parameters are generated by function read_out_file.

**Description:**

Calculate the TPR, FPR, ACC, SPC, PPV, NPV, FDR, MCC, F1 through the modeling output

**Return:**

The values of the evaluation criterions, related confusion matrix and labels.

**Use functions:**

| | | |
|---|---|---|
| get_acc_single | analysis_grid | |

## cal_acc_r

**Usage:**
cal_acc_r([@org_list] , [@out_list] );

@org_list: The labels of the data.
@out_list: The predicted labels.

**Description:**
Calculate the CC, MAE, RMSE, RAE, RRSE through the modeling output

**Return:**
The values of the evaluation criterions

## get_acc_single

**Usage:**
get_acc_single([@confu_matrix] , $label_count);

@confu_matrix: The confusion matrix of the predicted task
$label_count: The serial number of the label which is used to generated the outputs.

**Description:**
Get the TPR, FPR, ACC, SPC, PPV, NPV, FDR, MCC, F1 for the specified label.

**Return:**
The values of the evaluation criterions

## get_data_name

**Usage:**
get_data_name($task_name);

**Description:**
Get the related test data names of specified modeling task.

**Return:**
The related test data name

## get_gd_name

**Usage:**
get_gd_name($nf_name);

$nf_name: The name of n-fold validation node

**Description:**
Get the name of the node of parameter optimization from the name of n-fold validation node

**Return:**
The name of the node of parameter optimization

## get_labels

**Usage:**
get_labels($data_name);

**Description:**
Get the labels of the specified data.

**Return:**
ARRAY of labels.

## get_nf_name

**Usage:**
get_nf_name($node_name);

$node_name: The name of modeling node

**Description:**
Get the name of the node of n-fold validation from the name of mdeling node

**Return:**
The name of the node of n-fold validation

## get_org_list

**Usage:**
get_org_list( $data_name );

**Description:**
Get the labels of each instances in the specified data file.

**Return:**
ARRAY of lables

## get_out_list

**Usage:**
get_out_list( $out_file_name );

**Description:**
Get the labels of each instances in the specified modeling output file.

**Return:**
ARRAY of predicted lables

## get_parent_name

**Usage:**
get_parent_name($node_name);

$node_name: The name of node

**Description:**
Get the name of the parent node from the input node name

**Return:**
The name of the parent node

## get_relate_file

**Usage:**
get_relate_file($task_name , \%is_complete )

%is_complete: A hash tables records the whether task was completed or failed

**Description:**
Analysis the output of the task, generate html file to record the related information.

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| replace_out_name | get_nf_name | get_script_detail |
| get_tt_name | get_parent_name | retry_detail |
| read_out_file | cal_acc_r | cal_acc_c |
| print_matrix | print_conf | print_acc |

## get_related_name

**Usage:**

our (%hash_clu_num , %hash_num_clu , %hash_fea_num ,
    %hash_num_fea , %hash_tt_num , %hash_num_tt);
get_related_name(\%hash_clu_num , \%hash_num_clu , \%hash_fea_num , \%hash_num_fea , \%hash_tt_num , \%hash_num_tt)

%hash_clu_num: The name of cluster methods with the serial numbers
%hash_num_clu: The serial numbers with the name of cluster methods
%hash_fea_num: The name of variable selection methods with the serial numbers
%hash_num_fea: The serial numbers with the name of variable selection methods
%hash_tt_num: The name of modeling methods with the serial numbers
%hash_num_tt: The serial numbers with the name of modeling methods

**Description:**

Add information of the serial numbers and related method names into the hash indexes.

**Return:**

None

## get_tt_name

**Usage:**

get_tt_name($task_name);

**Description:**

Get the related names of the files which contain the information of training/test data.

**Return:**

The two names of training data information file and test data information file.

## get_script_detail

**Usage:**

get_script_detail($prop_file)

$prop_file: The file contains the information of related task

**Description:**
Get the information from the file which was generated during the process of scripts generations. Then format the information into HTML format.

**Return:**
The formatted information ARRAY.

## get_overall

**Usage:**
get_overall( @confu_matrix );

@confu_matrix: The confusion matrix of the predicted task

**Description:**
Get the overall information for n-fold classify tasks.

Overall means combine all the instances of n-fold tasks then analysis them as one predict task.

**Return:**
The values of the evaluation criterions

**Use functions:**

| | | |
|---|---|---|
| get_overall_single | | |

## get_overall_single

**Usage:**
get_overall_single($label_count , @confu_matrix);

@confu_matrix: The confusion matrix of the predicted task
$label_count: The serial number of the label which is used to generated the outputs.

**Description:**
Get the TPR, FPR, ACC, SPC, PPV, NPV, FDR, MCC, F1 for the specified label of overall analysis.

**Return:**
The values of the evaluation criterions

## init_best_accs

**Usage:**

my @best_accs;
init_best_accs(\@best_accs , $isoverall);

@best_accs: The records of the best methods for each evaluation criterion
$isoverall: Whether to initialize @best_accs for overall analysis

**Description:**

Initialize a matrix (2-dimention ARRAY) to record the best methods for each evaluation criterion

**Return:**

None

## mean_acc

**Usage:**

mean_acc(@accs);

@accs: The values of the evaluation criterions about specified n-fold results.

**Description:**

Calculate the mean values of each label with values of the evaluation criterions in n-fold results. Then get the related average values for all the labels.

**Return:**

A matrix (2-dimention ARRAY) contains the mean evaluation criterions for labels and the average of them in the last row.

**Use functions:**

| | | |
|---|---|---|
| round | | |

## mean_acc_overall

**Usage:**

mean_acc(@overall_accs);

@overall_accs: The values of the evaluation criterions about specified overall accs.

**Description:**

Calculate the average values for all the labels of the overall matrix.

**Return:**
The @overall_accs add with a row with the average values.

**Use functions:**

| | | |
|---|---|---|
| round | | |

## modify_sub_tables

**Usage:**
modify_sub_tables($file_handle , $subname , $statue , $istext , $out_accs_p);

$file_handle: The file handle for the outfile
$subname: The name of the specified criterion (such ACC or MCC)
$statue: 'head' / 'body' / 'end'
$istext: if the value is 1, print the TEXT format
$out_accs_p: The reference of the @out_accs**Description:**

This function is only for parameter optimaztion Create a sortable table and print some outputs into it

**Return:**
None

## print_acc

**Usage:**
print_acc(@acc);

@acc: The matrix (2-dimention ARRAY) with values of the evaluation criterions.

**Description:**
Format the matrix into HTML format

**Return:**
The formatted ARRAY

**Use functions:**

| | | |
|---|---|---|
| round | | |

## print_acc_all_body

**Usage:**

print_acc_all_body(@accs , $isoverall);

@accs: The matrix (2-dimention ARRAY) with values of the evaluation criterions.

$isoverall: Whether the @accs is for overall analysis

**Description:**

Add the specified matrix into the sortable table.

This table is designed to record all the tasks’ evaluation criterions.

**Return:**

None

**Use functions:**

| | | |
|---|---|---|
| round | | |

## print_acc_all_head

**Usage:**

print_acc_all_head($isoverall);

$isoverall: Whether to initialize this table with overall analysis.

**Description:**

Generate the file acc_all.html with some configurations.

The table contained in this file is designed to record all the tasks’ evaluation criterions.

**Return:**

None

## print_conf

**Usage:**

print_conf(@confu_matrix);

@confu_matrix: The confusion matrix.

**Description:**

Format the confusion matrix into HTML format

**Return:**
The formatted ARRAY

## print_matrix

**Usage:**
print_matrix(@accs);

@accs: The matrix (2-dimention ARRAY) with values of the evaluation criterions.

**Description:**
Format the matrix into HTML format

**Return:**
The formatted ARRAY.

## print_matrix_trans

**Usage:**
print_matrix_trans(@accs);

@accs: The matrix (2-dimention ARRAY) with values of the evaluation criterions.

**Description:**
Format the transpose of the matrix into HTML format

**Return:**
The formatted ARRAY.

## read_html_list

**Usage:**
read_html_list($file)

**Description:**
Read the matrix in the .html file which is generated by function get_relate_file

**Return:**
Matrix (2-dimention ARRAY) of the evaluation criterions.

## read_html_list_nf

**Usage:**

read_html_list_nf( $nf_file );

**Description:**

Read the average values of evaluation criterions about outer folds in the file generated by function analysis_nfold.

**Return:**

Matrix (2-dimention ARRAY) of the evaluation criterions about the outer folds of the specified n-fold validation.

## read_out_file

**Usage:**

read_out_file($task_name)

**Description:**

Read the output file of the task, get the related labels and information for analysis.

**Return:**

4 references of ARRAY:<br>
@org_list: The labels of the data.
@out_list: The predicted labels.
%labels_hash: The sequences of the instances with labels.
@labels: The labels of classes.

**Use functions:**

| | | |
|---|---|---|
| get_data_name | get_labels | get_org_list |
| get_out_list | | |

## replace_job_type

**Usage:**

replace_job_type( $task_name );

**Description:**

Convert the types of tasks for human readable.

Convert the clu, fea, grid, nfold, outer, inner to Cluster, VarSelection, ParaOptimization, CrossValidation, ModelingOut, ModelingIn

**Return:**

The converted type.

## replace_out_name

**Usage:**

replace_out_name($task_name , $condition);

$taks_name: The name of task generated by PML

$condition: 0, 1, 2

**Description:**

Convert the task name for human readable.

The names of tasks are designed for easily identified by machine, but not for human. This function is designed for convert the name to a readable one for human.

**Return:**

The converted name which depend on the value of $condition:

$condition = 0 : Default, just return the readable name

$condition = 1 : Return the shortest name for the data process tree

$condition = 2 : Return the readable name with its type which generated by function replace_job_type

**Use functions:**

| | | |
|---|---|---|
| replace_job_type | | |

## retry_detail

**Usage:**

retry_detail($step , $isweka);

$step: clu, fea or tt

$isweka: if the related task used weka

**Description:**

Return some prompt for users to retry a task.

**Return:**

ARRAY with strings of HTML format

## round

**Usage:**

round($in , $d);

$in: The input number
$d: The number of decimal places to remain

**Description:**
Remain specified number decimal places, and the number of the last decimal place would be rounded.

**Return:**
The modified number

## update_confusion_matrix

**Usage:**
update_confusion_matrix($file , \@confu_matrix);

$file: The name of file which is generated by function get_relate_file
@confu_matrix: Confusion matrix.

**Description:**
Update the confusion matrix during the overall analysis.

**Return:**
None

## update_best_acc

**Usage:**
update_best_acc(\@best_accs , \@acc , $method_name , $is_overall);

@best_accs: Records of the best methods for each evaluation criterion
@acc: The evaluation criterion of a prediction
$method_name: The name of the used method
$is_overall: Valued as 1 if the @acc and @best_acc is for overall analysis

**Description:**
Compare the @acc with @best_acc and update the records if some evaluation criterions in @acc are better.

**Return:**
None

## Debug environment

PML was developed by Eclipse SDK 4.2.2 with the plugin EPIC 0.6.53 on Ubuntu 12.04. Users could check any line of the PML by the bug tracker of Eclipse. Note that some functions are called in parallel, thus the global variable $core_number (controlled by the parameter CORE_NUM) need to be set as 1 when debugging.

# The processor_single script

Independent with the modules, the script processor_single_script is utilized as a component of task executor in parallel. The script could execute a task, analysis the output file and then either generate the data for the related tasks or copy the predicted output to the specified folder.

Because of the different rules of the change with working directory with Windows and Linux (Section 'Current working directory' in http://perldoc.perl.org/threads.html ), this script is utilized by function job_execute_desktop through command line interface.

The functions in the script have been included in the modules, thus in this section the functions just listed here with the links to the related one in modules.

| | | |
|---|---|---|
| analysis_data | dir_copy | find_position |
| get_labels | get_lsa | get_org_list |
| get_out_list | get_pca | isarff |
| process_clu | process_fea | randperm |
| uncompressdir | isarff | |