Automated test generation for `ctsa`

@hgfernan

February 13, 2025

Contents

Abstract

A routine for the automated test generation for the statistical library `ctsa` is outlined. It envolves the generation of simple tasks of model fitting and prediction using `ctsa`, compared with equivalent code in the Python libraries `pmdarima` and `statsmodels`, and in the R library `forecast.`

1 Motivation

Why using a test database and automatic generation of tests, instead of the handcrafted tests ?

2 Introduction

Overview of the project

3 Tables

Since the automated test generation is based on a database, here follows a description of the database to be used.

It has a mixed relational and document architecture: the main tables follow a conventional relational structure, but the parameters and test results are stored as JSON values. That's for pragmatic reasons: the parameters and test results are varied and have different structures. That could be easily mapped to a relational database structure, but it would be too laborious and cumbersome.

For instance: an $ARIMA$ model will have three basic parameters, while a $SARIMA$ model will have 7 basic parameters.

As such, test parameters and results will be stored as JSON objects encoded as strings, and dealt with by classes specialized in their content. There will be a class for each one of the models, that will be able to unpack and allow the use of the information in those JSON values.

A short description of the data base structure displayed in Figure 1 can be:

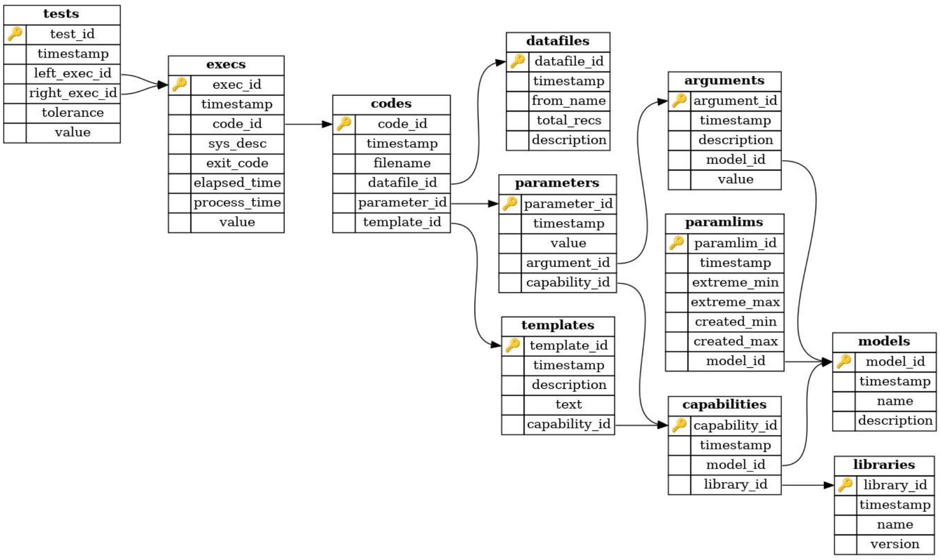

Figure 1: Database used for automated test generation

1. The most central table is `models`m that contains the name of all statistical models in use.

   It is related to 3 other tables: `arguments` (item 5), `capabilities` (item 3), and `paramlims` (item 9);

2. The second central table is `libraries`. It contains the names of the libraries that shall be used in the tests.

   All libraries in use – that is `ctsa`, `forecast`, `pmdarima`, and `statsmodels` – are available in just one language, be it C, Python or R. Therefore there's currently no need to add a language field in this table.

   Another possible lack of information will be the partial support of a statistical model in one or more of the libraries, but for now its solution will be postponed;

3. Another important table is `capabilities`, that's a relationship between `models` (item 1) and `libraries` (item 2); that is: it informs what statistical models are supported by each library.

   This table guides the creation of code `templates` (item 7) for each language and library, as well as the adaptation of statistical `arguments` (item 5) for each library, in the form of `parameters` (item 6);

4. The table `datafiles` contains a list of data files specially prepared for the tests: they contain only a time series, in the fields `index` and `value`, to ease the creation of tests. As such they don't have a specific name: they're named with the `datafile_id` and the extension "`.csv`". For instance, the 1st file registered in this table will be named "`0001.csv`".

   The table field `from_name` relates this adapted data file to the original data where the data was obtained;

5. The table `arguments` contains values of elements of the statistical `models` (item 1), without any concern to its implementation in the `libraries` (item 2);

6. The table `parameters` contains the adaptation of the table `arguments` (item 5) to each implementation of one of the `models`) (item 1) of the codes of the `libraries` (item 2).

   That adaptation is done via the field `value`.

   This table refers the `capabilities` (item 3) to

   It is used to create a relate `models` and `libraries`;

7. The table `templates` contains codes templates to each one of the library `capabilities` (item 3) – that is all the statistical `models` (item 1) that all `libraries` (item 2) implement.

   A template is conceptually an open structure with holes where the `parameters` (item 6) and the `datafiles` (item 4) will be filled in. A template is also a fragment of source code where just these two informations are missing;

8. The table `codes` contains the source codes generated by filling in each one of the `templates` (item 7) with one of `datafiles` (item 4) and one of `parameters` (item 6).

   The table `codes` a tiny of the table `templates`, but it's an important one, since makes more concrete this filling of a template.

   Another point to make matters more clear is that the name of a code file (contained in the field `filename`) won't be simply the transcription of the field `code_id` and an extension, but will refer to the field `name` of the corresponding row in `model` (item 1), of the `name` of used row in `libraries`, the `parameter_id` of the parameter used, and the `datafile_id` of the data file used in it.

   That is: the simple naming used in `datafiles` (item 4) won't be used here.

   For instance: if a `codes` row with `code_id` of 15, uses a C code template of library `ctsa`, its filename would "`0015.c`". But if it uses the *AR* model implementation of `ctsa`, uses the 1st row of parameters, and the `datafile_id` of 23, its extended name will be "`ar_ctsa_p0001_d0023.c`"

9. The table `paramlims` contains the valid range of each parameter of a given model from the table `models` (item 1), from the value in the field `extreme_min` to the value `extreme_max`, inclusive.

   The execution of all codes – as contained in the table `execs` (item 10) – can take some time; therefore the parameters really used to create code files for all libraries will be kept in the closed interval of the field pair [`created_min`, `created_max`];

10. The table `execs` contains a history of the executions of a given code. Its only link is with the table `codes` (item 8).

    The fields in `execs` document the execution of a given code, identified by its `code_id`. The field `sys_desc` should contain the description of the machine where code was run, `exit_code` is the value returned by the execution of the code. The time of execution is measured by `elapsed_time` (the difference between the time the execution was started and the time it was finished), and `process_time` is the time of the CPU that was effetively used.

    The field `value` contains both the parameters used in to start the program, and the result it obtained. It contains the information presented as the summary of the fitting of the model to the data, and the result of the forecasting of a few time steps, both as the value obtained, and its deviation.

    That information is formatted as a JSON encoded string, due to the variety of parameters of each statistical model, as well as the variety of their measures of fitting.

11. Finally, the table `tests` is a comparison between two different executions of the same data, aka `datafile_id`, the same statistical model and the same model arguments; that is the same `argument_id`. One is called `lef_exec_id` and the other is `right_exec_id`.

    A test allows two different versions of the same library to be compared for the same model and arguments. Or two different libraries with the same model and arguments.

    The comparison can use the statistical or the results of the physical execution, like the process time or the exit code.

4 Software design

High level specification of the softwares and their use.

5 Examples by hand

Worked examples of the database and software use.

6 Implementation

A shiplog reporting details of the system development, and possible deviations from the specification in the section Software design.