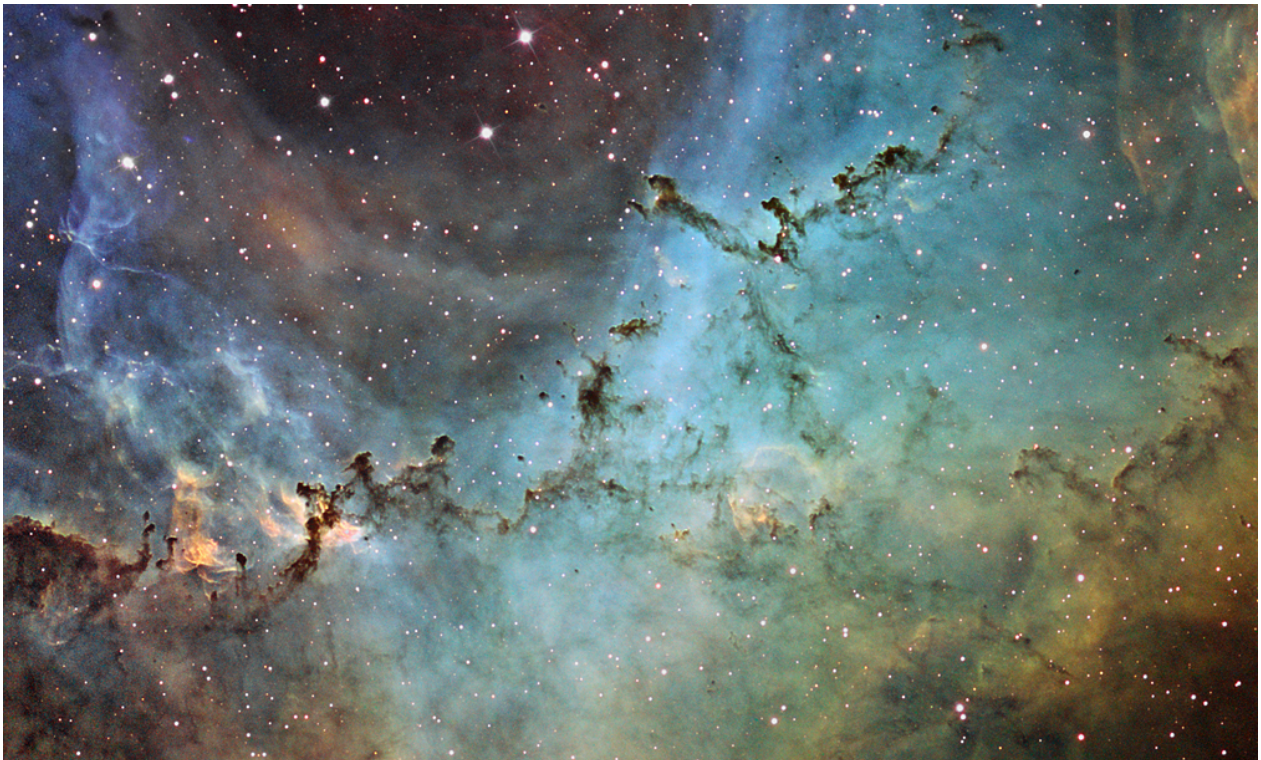


DOCUMENTATION

AutoRank: Automatic tracer ranking pipeline

October 28, 2020



1 Overview

AutoRank searches for the best observable tracers of any physical variable, based on grids of astrochemical models. It can be applied to any existing model grid. It is presented in [Bron et al. \(2020\)](#). See this paper for a description of the method.

Its purpose is the following:

- You are interested in an unobservable quantity (e.g. the ionization fraction of the medium in [Bron et al. 2020](#))
- You want to find out which are the best observable tracers of this quantity among a (possibly large) number of observables
- You have a grid of astrophysical models in which both the observable quantities and the target unobservable quantities are computed (the target quantity might also be a parameter of the model)
- The grid explores the range of physical conditions you expect to be present in the medium you intend to observe (or use observations of)

AutoRank will then rank all possible tracer ratios according to how well they allow to estimate the target quantity, and provide for each of the best ratios a Random Forest model able to predict the target quantity from the chosen ratio.

2 Requirements

AutoRank is provided as a Python 3 script. It requires the following python modules:

- os
- matplotlib
- numpy
- sklearn
- corner
- treelite¹ (optional, necessary only to export the Random Forest models in a reusable format).

3 Use

To run AutoRank, open a terminal window in the directory where `AutoRank.py` is located and type:

```
python3 AutoRank.py
```

AutoRank is provided with an example model grid, and can be run "as is" as soon as downloaded to test it.

¹On Mac OS, treelite requires installing libomp through the command "brew install libomp".

The configuration of the script is done in the `AutoRank.py` file, at the beginning. The configuration parameters are described below.

3.1 Path to the model grid file

The results of the model grid you wish to use should be gathered in a single ASCII datafile, with one line per model and one column per quantity (both computed quantities and model parameters can be mixed in any order). If comment lines are present at the beginning of the file, they should start with the character '#'. The values in the data file should not be logarithms of the physical values as the script will convert them to log10 values automatically.

The path to the model grid data file needs to be provided in the variable `model_grid_data_file`.

All outputs will be produced in an output directory named
`"Outputs_{name_of_your_model_grid_file}"`.

3.2 Specifying the observable quantities and the target quantity

To specify the variables that you want to consider as observable (potential tracers of the quantity of interest), set the variable `obs_quantities_col_num` to contain a list of the column numbers of the observable variables (in the model grid data file). The column numbers are counted starting from 0. Then specify the names of the chosen observable quantities in the variable `obs_quantity_names` (list of strings). Latex syntax is accepted (see the commented example line in the `AutoRank.py` file).

To specify the target variable, set the variable `target_col_num` to contain the column number of the target variables (in the model grid data file). The column numbers are counted starting from 0. Then specify the name of the target variable in the variable `target_name` (string). Latex syntax is accepted (see the commented example line in the `AutoRank.py` file).

3.3 Random Forest hyperparameter tuning

The Random Forest models used in AutoRank depend of a few hyperparameters. The two relevant hyperparameters in our case are the number of tree in the forest and the maximum tree depth. By default, AutoRank will automatically tune the hyperparameters value (see Appendix A in [Bron et al. 2020](#) for a discussion of the method to select the hyperparameter values). The chosen values will be displayed in the terminal during the run.

In addition, a figure providing diagnostics of the hyperparameter selection procedure is produced (`Hyperparameter_tuning_diagnostics.pdf` in the output directory). The procedure (a partial grid optimization) is the following (see Appendix A in [Bron et al. 2020](#) for more detail) :

- A first rough ranking of the ratios is performed using default values of the RF hyperparameters (100 trees, and no limit on the maximum depth).
- Only the best and worst ratio from this first ranking are used to optimize the hyperparameter values.

- A grid search is performed, exploring ranges of values of the number of trees and the maximum tree depth, and estimating the fit performance through the R^2 coefficient for each combination (the out-of-bag estimates are used as proxies for the generalization error).
- A compromise is sought between the best performance for the two selected ratios. We seek values that give R^2 within 0.01 of the optimum for both ratios, and favor lower number of trees and lower maximum tree depth if possible.

The diagnostics figure shows the R^2 maps (as functions of the number of trees and maximum tree depth) for the best and worst ratios determined in the preliminary ranking. The red contours show the region where the R^2 is within 0.01 of the optimum for each ratio, while the green contour shows the region where both are within 0.01 of the optimum (the red contours can be partly hidden beneath the green contour or be outside of the displayed region).

If the automatic selection of the hyperparameter values seems unsatisfactory from the inspection of this figure, you can manually fix values by setting `fix_RF_hyperparams` to `True` and choosing values for `N_trees` and `Max_depth`.

Alternatively, you can change the grid across which the optimization is made by changing the values of `n_est_grid` and `max_depth_grid`.

4 Outputs

The script will display a figure presenting the ranking of the best ratios (for the purpose of tracing the target quantity). The complete ranking table is saved both as an ASCII data file (`ranking_table.dat`) and as a latex table (`ranking_table.tex`).

For each of the ten best ratios, a figure is saved presenting a scatterplot of the target quantity as a function of the considered ratio, with a curve corresponding to the best RF model overplotted.

Finally, if the `treelite` Python module is installed, the trained RF models for each of the ten best ratios are provided as compilable libraries reusable on any system/architecture. You must first select in the variable `platform` the type of system on which you intend to reuse the RF model (not necessarily the same as the system on which you are running `Autorank.py`). Once `Autorank.py` has run, for each of the ten best ratios, a `.zip` archive is provided in the output folder containing automatically generated source files and makefiles.

To reuse the RF models, first copy this archive to the machine where you intend to reuse the RF model, then unzip the archive and compile its content by running the command `make`. This produces a shared library file (extension `".dylib"`, `".so"` or `".dll"` depending on the type of system you selected).

To load and use these models from a python script, you can adapt the following code :

```
import treelite_runtime
model = treelite_runtime.Predictor(filename, verbose=True)
predictions = model.predict(treelite_runtime.Batch.from_numpy2d(your_data_points))
```

where `filename` should be the compiled library file corresponding to the RF model you want to load (extension `".dylib"`, `".so"` or `".dll"`), and `your_data_points` the observed values (as a 2D

numpy array) of the ratio from which you want to predict the target quantity. **The provided input values should be \log_{10} of the values, and the predictions will be in \log_{10} of the target variable.**

These compiled library files can also be used from C code. See the documentation here :

<https://treelite.readthedocs.io/en/latest/tutorials/deploy.html#option-2-deploy-predicton-code-only>

Bibliography

Bron, E., Roueff, E., Gerin, M., et al. 2020, A&A