HBSGsep

Ross Fadely, rfadely@haverford.edu

1. Introduction

Found at http://github.com/rossfadely/star-galaxy-classification are `C` routines which may be used to compile the code `HBSGsep`. Details about the purpose of the code, as well as the algorithm's concept and formalism can be found in the documents portion of the above URL. It is recommended the user reads this document (Fadely, Hogg, and Willman (2012)) before proceeding.

As discussed in Fadely, Hogg, and Willman (2012), the routines provided here are a basic implementation of a hierarchical star-galaxy classification algorithm. In this light, the code provided here is meant to be conceptually clear — it is by no means written for conciseness or optimal performance. The hope, instead, is that the code may be useful for simple implementations, and inspirational for further algorithmic developments. Please feel free to contact the author(s) with questions and comments.

Please do fork this code if you would like to make improvements.

2. Compiling and Dependencies

`HBSGsep` makes use of two external packages -

GNU Scientific Library
CFITSIO.

Before attempting to compile the routine, build the above packages (you may already have done so), and modify the makefile INCLUDES and LFLAGS to point to the path locations. After doing so, typing `make` at the terminal will compile the executable. Before calling the executable (with no arguments), we first must modify a few variables to get started (Section **??**).

3. Getting Started

During runtime, `HBSGsep` accepts and returns a number of FITS and ascii files. Before compiling, one must specify the path and names of these files as well as changing a few parameters. These variables can be found at the beginning of `HBSGsep.c`. In this section we describe the variables which are *required* to be modified. Optional modifications are described in Section **??**.

3.1. Input Files

`filtersinput` – Path to a file which specifies the disk locations of the throughput response curves for the filters of the photometric data. The file format is each disk location, written as an ascii string, one per line.

`galssedinput` – Path to a file which specifies the disk locations of the galaxy SED templates. The file format is each disk location, written as an ascii string, one per line.

`starssedinput` – Path to a file which specifies the disk locations of the star SED templates. The file format is each disk location, written as an ascii string, one per line.

3.2. Data file

`datafile` – Path to the FITS table file containing the data to be classified. The table must have at least $2N_{filter}$ columns (where N_{filter} is the number of filters), but may have more. The first N_{filter} columns should contain the magnitudes for each source, while the second N_{filter} should contain the associated measurement uncertainties. The order should be the same, e.g., $\{u, g, r, i, z, u_{err}, g_{err}, r_{err}, i_{err}, z_{err}\}$ should be the column order for *ugriz* data. The code can handle sources with non-detections or missing data. For non-detections, the magnitude should be greater than 50 and the associated uncertainty should give the limiting magnitude of that filter, for that survey. If the data is missing, the magnitude should be negative. If you prefer some other format, you need to modify `calc_datavals.c`.

3.3. Output Files

`hypoutfile` – Path to one-column FITS table file which will be written by the code after each improvement in the total log likelihood. This file has $N_{star} + N_{gal} + 2$ rows, where $N_{star} + N_{gal}$ are the number of star and galaxy templates, respectively. The first N_{star} values are the hyperparameter weights for the star templates, in the order they are given by `starssedinput`. These are followed by those for the galaxy templates. Finally, the last two weights are the total prior probably that any source is any type of star or galaxy, respectively.

`lnlikeratiooutfile` – Path to one-column FITS table file which will be written by the code after optimization. The file has N_{data} lines, where each line is the loglikelihood ratio for a source, listed in the same order as the data file. Loglikelihood ratios for each source are defined as the log of Ω in Eqn. 1 of Fadely, Hogg, and Willman (2012).

`lnPtotfile` – Path to one-column ascii file which is written after the specified number of iterations (`writeiter`, see below). The output file contains the values of the loglikelihood, one per line. Useful for convergence monitoring, otherwise set to `/dev/null`.

3.4. Variables

`zmin, zmax, Nz` – Together these specify the redshift grid of the galaxy templates. Setting these gives the minimum and maximum redshift, and the number of gridpoints `Nz`. Fadely, Hogg, and Willman (2012) uses $\{0, 4, 51\}$ which gives a redshift step of 0.08. We found this suitable for our purposes.

`noisefudge` – A few percent factor η which is part of the noise model of the algorithm (see Appendix of Fadely, Hogg, and WIllman (2012)). This value can be modified to find the maximum likelihood of the data. Values between 0.04 and 0.08 seem to be optimal for COSMOS data, but may be different for different survey, template, filter combinations.

`Niter` – The maximum number of loglikelihood calls made by the GSL optimizer. By default, the optimizer is called twice, so 2 `Niter` iterations will be done.

`tol` – The tolerance for the simplex size of the downhill simplex optimizer.

4. Running tips and comments

4.1. Templates

We provide a number of star and galaxy templates at http://github.com/rossfadely/star-galaxy-classification. There are a few issues to consider when selecting the templates used in the classification. First is the number of templates. If the number of templates is very large, optimization can take a *very* long time. We have found $100 - 300$ star templates and ~ 30 galaxy templates (over 51 redshifts) works well. Another thing to consider is whether the templates used span the color-color space of the data in question – this may mean examining color-color plots beforehand. Future versions of this algorithm will learn the number (and shape) of the templates from the data itself.

4.2. Optimization

Optimization in the code proceeds using a downhill simplex algorithm. The optimization process will stop when either 2 `Niter` is reached or the simplex size falls below `tol`. We have found greater success running for a fix number of iterations, rather than specifying an arbitrary tolerance (which changes as a function of the data, templates, etc.). For the optimization in Fadely, Hogg, and WiIlman (2012), `Niter` = 50000 was found to give a fairly good optimum.

We note that for other versions of this algorithm we used a BFGS optimizer from GSL. We found this to be only slightly faster than the simplex method, due to the cost of numerical derivatives. Oddly, the BFGS did not seem to consistently give as good optimums as the one used here.

4.3. Data

We have found the results of `HBSGsep` are not overly sensitive to the sample size of the data, except in extreme cases. For large datasets, the code runs slower and may run into memory allocation issues depending on the user's machine. For very small datasets, one runs the risk of not having data which spans the range of demographics and signal-to-noise of the entire dataset. In Fadely, Hogg, and WIllman (2102), we find sample sizes between $3000 - 10000$ are adequate, but small sizes may be ok. As a check, the user should make sure the optimum loglikelihood of any smaller dataset scales linearly with the size of larger sets.

5. Optional features

5.1. Specifying initial hyperparameters

The code allows you to specify input hyperparameters to start the optimization. This is particularly useful if continuing from a previous optimization run. Setting the variable `usehypin` = 1 will read in the parameters from

`hypinfile` – Path to one column FITS table file, which specifies hyperparameter weights for star template, galaxy template, and the prior for star/galaxy. If the optional analysis flag `usehypin` = 1, this file will be read in as used as the starting point for the optimization. This file is primary in place to allow continuation from previous optimization runs, and is not required if `usehypin` = 0. See the description for `hypoutfile` for a description of the file format.

5.2. Calculate model magnitudes

By setting `calc_model_mags` = 1, model magnitudes will be calculated for the templates. One needs to specify:

`starmodmagsfile` – Path to output FITS table containing star model magnitudes. There will be N_{filter} columns for each row, which are the magnitudes in the order given by `filtersinput`. Each row contains the values for the star templates, in the order given by `starssedinput` .

`galmodmagsfile` – Path to output FITS table containing galaxy model magnitudes. This has the same column format as for `starmodmagsfile`. The rows, however, total N_{gal} times `Nz` in number. The first `Nz` rows are for the first template in `galssedinput`, the next `Nz` rows are for second first template, and so on.

Setting `Niter` = 0 will allow model magnitudes to be computed without the need to optimize hyperparameters.

5.3. ML χ^2 values

One can calculate the maximum likelihood (minimum chi-squared) for each data point, for both star and galaxy models, by setting `writeminchi2` = 1. This outputs two files to

`starchi2file` – Path to output one-column FITS table. The output file will contain the best chi-squared for *any* star model for each source. Row number corresponds to that of `datafile`.

`galchi2file` – Same as `starchi2file`, but for galaxy models.

Setting `Niter` = 0 will allow ML χ^2 values to be computed without the need to optimize hyperparameters.

5.4. Nuisance Parameters

There are a few parameters which the algorithm uses but need not be changed.

`fluxunitfactor` – A scalar factor used in the template fitting to insure the values are kept well within machine precision.

`probfrac` – The fraction of the ML star or galaxy model likelihood, which sets the tolerance for the "sparse" arrays propagated into the optimization.

`P_floor` – The smallest probability assigned to anything.

`weight_fact` – Sets initial step size of optimizer for total prior weights.

`Ncstep` – Number of steps used in marginalization over fit coefficient uncertainty.

`writeiter` – Number of iterations before update of current loglikelihood is printed to screen and to `lnPtotfile`.