howto-run

August 15, 2014

1 A how-to to running image-photo-z

A guide to installation and running the package with the main file

1.1 Installation:

Installing the dependencies and cloning the repository is enough to set up the environment to run the package. For installing the dependencies, the setup-deps script provided in the root folder of the package should work on a system that can use apt-get. You need administrator priviledges to run this script. Run it as follows on the system shell.

```
In [6]: cd image-photo-z/

/home/aloo/image-photo-z

In []: ./setup-deps.sh
```

If for some reason you are not able to or do not wish to run the setup script, here is a list of dependencies (in order of their requirement for the pipeline) that need to be installed. Unless mentioned, it is best to install dependencies in their default locations using the standard installer provided or the standard package management for your system Python >= v2.7. python-dev, the development package, needs to be installed as well. Numpy >= v1.6.1 Scipy >= v0.14.0 Astropy >= v0.3.2 Montage Image Mosaic Software >= v3.3. Best installed by downloading and compiling the source package with GNU Make. montage_wrapper >= v0.9.7 SExtractor >= v2.19.5. In my experience, gives trouble if installed via any source other than the rpm package on the Astromatic website. Scikit-learn >= v0.15.0b1 MLZ >= v0.0.1, best installed via pip. The procedure for compiling source packages using make is the following, on the system shell:

```
In []: cd [package-name]          # This is the root directory for the source package.

In []: cmake . [OR] ./configure # This depends on the package. If it contains a configure
                                 # file, use ./configure.

In []: make                      # This compiles the package.

In []: sudo make install         # Optional, need admin priviledges for this. Places
                                 # binaries in their default positions.
```

For installing via rpm you need the rpm package management software. Once you have that, you do

```
In []: sudo rpm -i [package-name].rpm
```

Installing via pip is accomplished as follows:

```
In []: sudo pip install [package-name]
```

1.2 Creating the config file

The config file is the set of input parameters to the code. Most parameter names are self-explanatory. Examples are provided in the config.cfg.template file provided in the root folder. They are explained here for completeness:
For yes/no parameters, specify 'yes' or 'no' without the quotes. This is **case-sensitive**. For an explanation of what the files described below are, refer to the "Getting to know the pipeline" document.

- IMAGE_PHOTOZ_PATH: This is the path to your local image-photo-z installation. My value for this is /home/aloo/image-photo-z.
- USE_MPI: Set this to yes if you want to use MPI for the computation.
- N_PROCESSORS: Set this to the number of processors you want to use, if using MPI.
- TRAINING_CATALOG: The object catalog file for training objects. This file must be in a specified format as shown in the example catalog file one_square_degree.csv provided with the package. For more information on generating this file see the pipeline description.
- TESTING_CATALOG: The object catalog file for testing objects. This is similar to the earlier file.
- TRAINING_CATALOG_PROCESSED: The name of the processed catalog file generated by the program.
- TESTING_CATALOG_PROCESSED: The name of the processed catalog file generated by the program.
- BANDS: Filters to be used. Specify these as comma-separated without spaces.
- REMAKE_CATALOGS: Specify whether or not to regenerate the catalogs by querying the SDSS SAS.
- REGENERATE_PIXEL_DATA: Specify whether or not to (re)generate pixel data vectors.
- LOG_INDEPENDENTLY: Specify whether or not to generate a list (logfile) of run-camcol-field combinations independent of image downloading.
- LOGFILE: Name of the logfile used to record run-camcol-field combinations in the catalog.
- LOCAL_IMAGES: Specify if the images are stored locally or not. Alternatively, select whether or not to download images (again).
- TRAINING_IMAGES_DIR: If images are local, this is the directory in which training images are stored. The images need to be stored as [run]-[camcol]-[field]-[band].fits (3813-5-24-u.fits, for example) in this directory.
- TESTING_IMAGES_DIR: Similar to the above for testing images.
- PROCESSING_DIR: The directory in which all processing is to be done. Images per frame are moved into this directory, all data is extracted from the images. Temporary files generated by the process are also stored here.
- TRAINING_CLASSIFIED_DATA_DIR: The code generates classified training data into this directory. Depending on which source types (galaxies, stars, ..) are used, the data is stored in subfolders inside this folder.
- TESTING_CLASSIFIED_DATA_DIR: Similar to the above, with testing data.
- TRAINING_DATA_FILE: Data collected from all sources is collected into this file for training.
- TESTING_DATA_FILE: Similar to the above for testing.
- TRAINING_TARGET_FILE: If using kNN, the redshifts are stored in this file before training.

- TESTING_TARGET_FILE: Similar to the above, for testing.
- TESTING_PREDICTION_FILE: If using kNN, immediately post testing, the output for respective predictions are stored in this file.
- TRAIN_AND_TEST: Enable or disable training and testing.
- PROBLEM_TYPE: Select whether you want to do classification or regression.
- USE_GALAXIES: Enable or disable using galaxy pixels.
- USE_STARS: Enable or disable using star pixels.
- USE_QSOS: Enable or disable using QSO pixels.
- USE_BACKGROUND: Enable or disable using background pixels.
- NUMBER_NEIGHBORS: If using kNN, this sets the number of neighbors to be used.
- KNN_OUTPUT_FILE: If using kNN, this sets the final output filename from which final plots may be generated.
- KNN_ROUND_OFF_CLASSIFICATION: If using kNN, this sets whether or not to round off the classification result.
- CLEAN_AFTER_DONE: Set to yes if only the output file and the downloaded images are to be kept after the pipeline finishes.
- CLEAN_ON_INTERRUPT: Set to yes if generated temporary files and data is to be erased when the process is interrupted.
- REMOVE_IMAGES_AFTER_DONE: Set this to yes if images are to be deleted after the program finishes.
- REMOVE_IMAGES_ON_INTERRUPT: Set this to yes if images are to be deleted when the process is interrupted.
- REMOVE_INTERMEDIATE_IMAGES: Set this to yes if intermediate images (like registered and error images) are to be deleted or no if they are to be moved to another location specified by INTERMEDIATE*FILES.
- INTERMEDIATE_TRAINING_FILES: Directory to store intermediate files if REMOVE_INTERMEDIATE_IMAGES is set to no.
- TIME: Set whether or not to time the process. *Not implemented yet.*
- TIME_DETAIL: Select the level of timing detail required.
- TIME_FILE: File to dump timing information in.

Refer to MLZ documentation for the meaning of MLZ config options. These go straight to the MLZ inputfile.

Note that in case of directories, do **NOT** put a '/' at the end of the parameter. The code does this automatically. Also, do **NOT** delete any entry from the config file, even if it is irrelevant to the configuration. This is done to prevent possible KeyErrors. Do **NOT** put spaces in any of the parameters.

1.3 Running the pipeline:

Using the main file with an appropriate config file is the most straightforward way to run the pipeline end-to-end. Here, you have two options, depending on whether you want to or not to use MPI.

This assumes you have the training and testing catalogs in the required format. Refer to the pipeline description for how to generate these.

1) Not using MPI: If you cannot or do not wish to use MPI, the executable main.py is to be run on the system shell, as follows:

```
In [4]: cd image-photo-z

/home/aloo/image-photo-z
```

Note that the path after the cd should be the path to *your local image-photo-z installation*. Mine is installed at /home/aloo/image-photo-z, so its where it is. Once this is done, just run the main on the shell as follows:

```
In []: python main.py
```

This runs the code with ./config.cfg as the config file. To specify a different config file run

```
In []: python main.py [config-file-name]
```

This *should* get the code running. If you do get any errors at this stage, **do report them**.

2) Using MPI: The file main-mpi.py needs to be run with mpirun in this case. Make sure that the N_PROCESSORS variable is set to the number of processors you are willing to allocate to the computation in the config file.

```
In []: mpirun -np [N_PROCESSORS] python main-mpi.py
```

To specify a custom config file:

```
In []: mpirun -np [N_PROCESSORS] python main-mpi.py [config-file-name]
```

The run script in the root folder is already set to run this. To avoid the hassle of typing all this, you can run this instead:

```
In []: ./run
```

having made sure that N_PROCESSORS is set. This takes config.cfg as the config file.

The code *does not* print anything out on the terminal except for the generating training data stage.