avocado

Release 0.1

Apr 26, 2021

Contents:

Photometric Classification of Astronomical Transients and Variables With Biased Spectroscopic Samples

CHAPTER 1

About

Avocado is a general photometric classification code that is designed to produce classifications of arbitrary astronomical transients and variable objects. This code is designed from the ground up to address the problem of biased spectroscopic samples. It does so by generating many lightcurves from each object in the original spectroscopic sample at a variety of redshifts and with many different observing conditions. The "augmented" samples of lightcurves that are generated are much more representative of the full datasets than the original spectroscopic samples.

CHAPTER 2

Installation

2.1 Requirements

Avocado has been tested on Python 3.7. It is not compatible with Python 2. Avocado depends on the following packages:

- astropy
- george (available through the conda-forge channel on conda)
- lightgbm
- matplotlib
- numpy
- pandas
- pytables (pytables on conda, tables on pip)
- requests
- scikit-learn
- scipy
- tqdm

We recommend using anaconda to set up a python environment with these packages and using the conda-forge channel.

2.2 Installation

Avocado can be downloaded from github using the following command:

```
git clone https://github.com/kboone/avocado.git
```

It can then be installed as follows:

```
cd avocado
python setup.py install
```

Along with installing the avocado module, this package also provides a set of scripts that can be used to process datasets on the command line for a variety of common tasks.

CHAPTER 3

Usage

Avocado is designed to be a general purpose photometric classification code that can be used for different surveys with implementations of different classifiers. An example of how avocado can be applied to the PLAsTiCC dataset using a LightGBM classifier can be seen *here*.

3.1 Classifying the PLAsTiCC dataset

The Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC) was a challenge released in 2018 to develop photometric classification methods for upcoming data from LSST. The authors of this challenge produced a realistic dataset for 3-years of LSST observations and simulated followup spectroscopic observations. The resulting dataset contains light curves for 3,492,888 astronomical objects, but only 7,846 of these objects have spectroscopic followup to confirm their types. The authors released this blinded dataset through the Kaggle platform and challenged the community to develop new methods for photometric classification.

An early version of avocado won this challenge, achieving the best score on a weighted log-loss metric of 1,094 classifiers submitted to this challenge. In this document, we show how to reproduce avocado's classifications for the PLAsTiCC dataset.

All time estimates are for running on a machine with the following specifications:

- CentOS 6.5
- Intel(R) Xeon(R) CPU E3-1270
- 32 GB RAM

3.1.1 Setup

Installating avocado

First, install avocado and all of its dependencies following the *installation instructions*. This will install a set of scripts that can be used to interact with avocado datasets.

Setting up a working directory

Create and move to a new working directory for avocado. All of the datasets, classifiers and predictions will be stored in this directory. Every script listed after this should be run from the base of this working directory. For example:

```
mkdir ~/plasticc
cd ~/plasticc
```

Downloading the PLAsTiCC dataset

A script is included in avocado to download the PLAsTiCC dataset from zenodo. This will download the dataset and preprocess it to get it into the format used internally by avocado. Running this script takes ~30 minutes.

```
avocado_download_plasticc
```

3.1.2 Augmenting the PLAsTiCC dataset

The avocado_augment script is included to augment datasets. To generate an augmented dataset with the name "plasticc_augment", run the following command in the working directory:

```
avocado_augment plasticc_train plasticc_augment
```

This will take several hours to run. Optionally, if a SGE grid system is available, the augmentation can be split to run in the grid system across several jobs with the following command:

```
avocado_augment_submit plasticc_train plasticc_augment --num_jobs 100 --qsub_
↪arguments '-q all.q'
```

This will split the augmenting procedure into 100 jobs, and submit them to the queue 'all.q'. Job files and output will be stored in the jobs directory of the working directory. Modify –qsub_arguments as appropriate for your system, or similar scripts can be created for other job systems.

3.1.3 Featurizing the datasets

Featurizing datasets is a slow process and takes ~100 hours for the full PLAsTiCC dataset. It is highly recommended to run these jobs in parallel.

To featurize sequentially:

```
avocado_featurize plasticc_train
avocado_featurize plasticc_test --num_chunks 500
avocado_featurize plasticc_augment
```

To submit featurize jobs to an SGE queue:

```
avocado_featurize_submit plasticc_train --qsub_arguments '-q all.q'
avocado_featurize_submit plasticc_test --qsub_arguments '-q all.q' --num_jobs 500
avocado_featurize_submit plasticc_augment --qsub_arguments '-q all.q'
```

3.1.4 Training the classifier

Several different classifiers can be trained using the same augmented dataset. To train a standard classifier with flat weights named "flat_weight", run:

```
avocado_train_classifier plasticc_augment flat_weight
```

This will take approximately 30 minutes.

3.1.5 Generating predictions

To generate predictions for the full dataset with our "flat_weight" classifier, run:

```
avocado_predict plasticc_test flat_weight
```

This will take approximately 1 hour to run.

3.1.6 (optional) Converting predictions to the kaggle format

The predictions generated by avocado will be saved in an HDF5 file by default. These can be converted to a CSV file used by kaggle with the following command:

```
avocado_convert_kaggle plasticc_test flat_weight
```

3.1.7 (optional) Training a redshift-weighted classifier

As shown in Boone (2019), a redshift-weighted classifier can be used to generate predictions that are independent of the redshift distribution and rates in the training sample. This is especially important for augmented datasets where the exact form of augmentation will otherwise leak into the classification. To train and generate predictions with a redshift-weighted classifier, run the following commands:

```
avocado_train_classifier plasticc_augment redshift_weight --object_weighting redshift
avocado_predict plasticc_test redshift_weight
```

3.1.8 (optional) Training classifiers on biased samples

In Boone (2019), we illustrate the bias of a classically trained classifier when the redshift distributions of the training samples are modified. To reproduce these results, run the following commands:

```
avocado_train_classifier plasticc_augment flat_weight_bias_high --simulate_plasticc_
↪bias high_redshift
avocado_train_classifier plasticc_augment flat_weight_bias_low --simulate_plasticc_
↪bias low_redshift
avocado_train_classifier plasticc_augment redshift_weight_bias_high --object_
↪weighting redshift --simulate_plasticc_bias high_redshift
avocado_train_classifier plasticc_augment redshift_weight_bias_low --object_weighting␣
↪redshift --simulate_plasticc_bias low_redshift

avocado_predict plasticc_test flat_weight_bias_high
avocado_predict plasticc_test flat_weight_bias_low
avocado_predict plasticc_test redshift_weight_bias_high
avocado_predict plasticc_test redshift_weight_bias_low
```

3.1.9 (optional) Reproducing the figures in Boone 2019

A Jupyter notebook that was used to produce all of the figures in Boone (2019) is included with avocado. It can be found on github. To run this notebook, copy it to the working directory after running all of the previous steps in this

document, and open it using Jupyter. Note that the augmentation procedure is not deterministic, so the results will vary slightly between runs. The plots of augmented light curves will need to be adjusted to select objects in the new augmented sample.

3.2 Reference / API

3.2.1 Datasets

Loading/saving a dataset

`Dataset`(name, metadata[, observations, ...])	A dataset of many astronomical objects.
`Dataset.load`(name[, metadata_only, chunk, ...])	Load a dataset that has been saved in HDF5 format in the data directory.
`Dataset.from_objects`(name, objects, **kwargs)	Load a dataset from a list of AstronomicalObject instances.
`Dataset.path`	Return the path to where this dataset should lie on disk
`Dataset.write`([overwrite])	Write the dataset out to disk.

avocado.Dataset

class avocado.**Dataset**(*name*, *metadata*, *observations=None*, *objects=None*, *chunk=None*, *num_chunks=None*, *object_class=<class 'avocado.astronomical_object.AstronomicalObject'>*)

A dataset of many astronomical objects.

Parameters

- **name** (*str*) – Name of the dataset. This will be used to determine the filenames of various outputs such as computed features and predictions.
- **metadata** (*pandas.DataFrame*) – DataFrame where each row is the metadata for an object in the dataset. See `AstronomicalObject` for details.
- **observations** (*pandas.DataFrame*) – Observations of all of the objects' light curves. See `AstronomicalObject` for details.
- **objects** (*list*) – A list of `AstronomicalObject` instances. Either this or observations can be specified but not both.
- **chunk** (*int (optional)*) – If the dataset was loaded in chunks, this indicates the chunk number.
- **num_chunks** (*int (optional)*) – If the dataset was loaded in chunks, this is the total number of chunks used.

__init__(*name*, *metadata*, *observations=None*, *objects=None*, *chunk=None*, *num_chunks=None*, *object_class=<class 'avocado.astronomical_object.AstronomicalObject'>*)

Create a new Dataset from a set of metadata and observations

Methods

__init__(name, metadata[, observations, ...])	Create a new Dataset from a set of metadata and observations
extract_raw_features(featurizer[, keep_models])	Extract raw features from the dataset.
from_objects(name, objects, **kwargs)	Load a dataset from a list of AstronomicalObject instances.
get_bands()	Return a list of all of the bands in the dataset.
get_models_path([tag])	Return the path to where the models for this dataset should lie on disk
get_object([index, object_class, object_id])	Parse keywords to pull a specific object out of the dataset
get_predictions_path([classifier])	Return the path to where the predicitons for this dataset for a given classifier should lie on disk.
get_raw_features_path([tag])	Return the path to where the raw features for this dataset should lie on disk
label_folds([num_folds, random_state])	Separate the dataset into groups for k-folding
load(name[, metadata_only, chunk, ...])	Load a dataset that has been saved in HDF5 format in the data directory.
load_predictions([classifier])	Load the predictions for a classifier from disk.
load_raw_features([tag])	Load the raw features from disk.
plot_interactive()	Make an interactive plot of the light curves in the dataset.
plot_light_curve(*args, **kwargs)	Plot the light curve for an object in the dataset.
predict(classifier)	Generate predictions using a classifier.
read_object(object_id[, object_class])	Read an object with a given object_id.
select_features(featurizer)	Select features from the dataset for classification.
write([overwrite])	Write the dataset out to disk.
write_models([tag])	Write the models of the light curves to disk.
write_predictions([classifier])	Write predictions for this classifier to disk.
write_raw_features([tag])	Write the raw features out to disk.

Attributes

path	Return the path to where this dataset should lie on disk

avocado.Dataset.load

classmethod Dataset.**load**(*name*, *metadata_only=False*, *chunk=None*, *num_chunks=None*, *object_class=<class 'avocado.astronomical_object.AstronomicalObject'>*, ***kwargs*)

Load a dataset that has been saved in HDF5 format in the data directory.

For an example of how to create such a dataset, see *scripts/download_plasticc.py*.

The dataset can optionally be loaded in chunks. To do this, pass chunk and num_chunks to this method. See *read_dataframes* for details.

Parameters

- **name** (*str*) – The name of the dataset to load
- **metadata_only** (*bool (optional)*) – If False (default), the observations are loaded. Otherwise, only the metadata is loaded. This is useful for very large datasets.

- **chunk** (*int (optional)*) – If set, load the dataset in chunks. chunk specifies the chunk number to load. This is a zero-based index.
- **num_chunks** (*int (optional)*) – The total number of chunks to use.
- ****kwargs** – Additional arguments to *read_dataframes*

Returns **dataset** – The loaded dataset.

Return type *Dataset*

avocado.Dataset.from_objects

classmethod Dataset.**from_objects**(*name*, *objects*, ***kwargs*)

Load a dataset from a list of AstronomicalObject instances.

Parameters

- **objects** (*list*) – A list of AstronomicalObject instances.
- **name** (*str*) – The name of the dataset.
- ****kwargs** – Additional arguments to pass to Dataset()

Returns **dataset** – The loaded dataset.

Return type *Dataset*

avocado.Dataset.path

Dataset.**path**

Return the path to where this dataset should lie on disk

avocado.Dataset.write

Dataset.**write**(*overwrite=False*, ***kwargs*)

Write the dataset out to disk.

The dataset will be stored in the data directory using the dataset's name.

Parameters ****kwargs** – Additional arguments to be passed to *utils.write_dataframe*

Retrieving objects from the dataset

Dataset.get_object([index, object_class, ...])	Parse keywords to pull a specific object out of the dataset

avocado.Dataset.get_object

Dataset.**get_object**(*index=None*, *object_class=None*, *object_id=None*)

Parse keywords to pull a specific object out of the dataset

Parameters

- **index** (*int (optional)*) – The index of the object in the dataset in the range [0, num_objects-1]. If a specific object_class is specified, then the index only counts objects of that class.

- **object_class** (*int or str (optional)*) – Filter for objects of a specific class. If this is specified, then index must also be specified.
- **object_id** (*str (optional)*) – Retrieve an object with this specific object_id. If index or object_class is specified, then object_id cannot also be specified.

Returns **astronomical_object** – The object that was retrieved.

Return type *AstronomicalObject*

Plotting lightcurves of objects in the dataset

Dataset.plot_light_curve(*args, **kwargs)	Plot the light curve for an object in the dataset.
Dataset.plot_interactive()	Make an interactive plot of the light curves in the dataset.

avocado.Dataset.plot_light_curve

Dataset.**plot_light_curve**(**args*, ***kwargs*)

Plot the light curve for an object in the dataset.

See *get_object* for the various keywords that can be used to choose the object. Additional keywords are passed to *AstronomicalObject.plot()*

avocado.Dataset.plot_interactive

Dataset.**plot_interactive**()

Make an interactive plot of the light curves in the dataset.

This requires the ipywidgets package to be set up, and has only been tested in jupyter-lab.

Extracting features from objects in the dataset

Dataset.extract_raw_features(featurizer[, ...])	Extract raw features from the dataset.
Dataset.get_raw_features_path([tag])	Return the path to where the raw features for this dataset should lie on disk
Dataset.write_raw_features([tag])	Write the raw features out to disk.
Dataset.load_raw_features([tag])	Load the raw features from disk.
Dataset.select_features(featurizer)	Select features from the dataset for classification.

avocado.Dataset.extract_raw_features

Dataset.**extract_raw_features**(*featurizer*, *keep_models=False*)

Extract raw features from the dataset.

The raw features are saved as *self.raw_features*.

Parameters

- **featurizer** (*Featurizer*) – The featurizer that will be used to calculate the features.
- **keep_models** (*bool*) – If true, the models used for the features are kept and stored as Dataset.models. Note that not all featurizers support this.

Returns **raw_features** – The extracted raw features.

Return type pandas.DataFrame

avocado.Dataset.get_raw_features_path

Dataset.**get_raw_features_path**(*tag=None*)

Return the path to where the raw features for this dataset should lie on disk

Parameters **tag** (*str (optional)*) – The version of the raw features to use. By default, this will use settings['features_tag'].

avocado.Dataset.write_raw_features

Dataset.**write_raw_features**(*tag=None*, ***kwargs*)

Write the raw features out to disk.

The features will be stored in the features directory using the dataset's name and the given features tag.

Parameters

- **tag** (*str (optional)*) – The tag for this version of the features. By default, this will use settings['features_tag'].
- ****kwargs** – Additional arguments to be passed to *utils.write_dataframe*

avocado.Dataset.load_raw_features

Dataset.**load_raw_features**(*tag=None*, ***kwargs*)

Load the raw features from disk.

Parameters **tag** (*str (optional)*) – The version of the raw features to use. By default, this will use settings['features_tag'].

Returns **raw_features** – The extracted raw features.

Return type pandas.DataFrame

avocado.Dataset.select_features

Dataset.**select_features**(*featurizer*)

Select features from the dataset for classification.

This method assumes that the raw features have already been extracted for this dataset and are available with *self.raw_features*. Use *extract_raw_features* to calculate these from the data directly, or *load_features* to recover features that were previously stored on disk.

The features are saved as *self.features*.

Parameters **featurizer** (*Featurizer*) – The featurizer that will be used to select the features.

Returns **features** – The selected features.

Return type pandas.DataFrame

Classifying objects in the dataset

Dataset.predict(classifier)	Generate predictions using a classifier.

Continued on next page

Table 7 – continued from previous page

Dataset.get_predictions_path([classifier])	Return the path to where the predicitons for this dataset for a given classifier should lie on disk.
Dataset.write_predictions([classifier])	Write predictions for this classifier to disk.
Dataset.load_predictions([classifier])	Load the predictions for a classifier from disk.
Dataset.label_folds([num_folds, random_state])	Separate the dataset into groups for k-folding

avocado.Dataset.predict

Dataset.**predict**(*classifier*)

Generate predictions using a classifier.

Parameters **classifier** (*Classifier*) – The classifier to use.

Returns **predictions** – A pandas Series with the predictions for each class.

Return type pandas.DataFrame

avocado.Dataset.get_predictions_path

Dataset.**get_predictions_path**(*classifier=None*)

Return the path to where the predicitons for this dataset for a given classifier should lie on disk.

Parameters **classifier** (str or *Classifier* (optional)) – The classifier to load predictions from. This can be either an instance of a *Classifier*, or the name of a classifier. By default, the stored classifier is used.

avocado.Dataset.write_predictions

Dataset.**write_predictions**(*classifier=None*, ***kwargs*)

Write predictions for this classifier to disk.

The predictions will be stored in the predictions directory using both the dataset and classifier's names.

Parameters

- **classifier** (str or *Classifier* (optional)) – The classifier to load predictions from. This can be either an instance of a *Classifier*, or the name of a classifier. By default, the stored classifier is used.
- ****kwargs** – Additional arguments to be passed to *utils.write_dataframe*

avocado.Dataset.load_predictions

Dataset.**load_predictions**(*classifier=None*, ***kwargs*)

Load the predictions for a classifier from disk.

Parameters **classifier** (str or *Classifier* (optional)) – The classifier to load predictions from. This can be either an instance of a *Classifier*, or the name of a classifier. By default, the stored classifier is used.

Returns **predictions** – A pandas Series with the predictions for each class.

Return type pandas.DataFrame

avocado.Dataset.label_folds

Dataset.**label_folds**(*num_folds=None*, *random_state=None*)

Separate the dataset into groups for k-folding

This is only applicable to training datasets that have assigned classes.

If the dataset is an augmented dataset, we ensure that the augmentations of the same object stay in the same fold.

Parameters

- **num_folds** (*int (optional)*) – The number of folds to use. Default: settings['num_folds']
- **random_state** (*int (optional)*) – The random number initializer to use for splitting the folds. Default: settings['fold_random_state'].

Returns **fold_indices** – A pandas Series where each element is an integer representing the assigned fold for each object.

Return type *pandas.Series*

3.2.2 Astronomical objects

AstronomicalObject(metadata, observations)	An astronomical object, with metadata and a lightcurve.
AstronomicalObject.bands	Return a list of bands that this object has observations in
AstronomicalObject.subtract_background()	Subtract the background levels from each band.
AstronomicalObject.preprocess_observations([...])	Apply preprocessing to the observations.
AstronomicalObject.fit_gaussian_process([...])	Fit a Gaussian Process model to the light curve.
AstronomicalObject.get_default_gaussian_process()	Get the default Gaussian Process.
AstronomicalObject.predict_gaussian_process(...)	Predict the Gaussian process in a given set of bands and at a given set of times.
AstronomicalObject.plot_light_curve([...])	Plot the object's light curve
AstronomicalObject.print_metadata()	Print out the object's metadata in a nice format.

avocado.AstronomicalObject

class avocado.**AstronomicalObject**(*metadata*, *observations*)

An astronomical object, with metadata and a lightcurve.

An astronomical object has both metadata describing its global properties, and observations of its light curve.

Parameters

- **metadata** (*dict-like*) – Metadata for this object. This is represented using a dict internally, and must be able to be cast to a dict. Any keys and information are allowed. Various functions assume that the following keys exist in the metadata:
  - object_id: A unique ID for the object. This will be stored as a string internally.
  - galactic: Whether or not the object is in the Milky Way galaxy or not.

- host_photoz: The photometric redshift of the object's host galaxy.
- host_photoz_error: The error on the photometric redshift of the object's host galaxy.
- host_specz: The spectroscopic redshift of the object's host galaxy.

For training data objects, the following keys are assumed to exist in the metadata: - redshift: The true redshift of the object. - class: The true class label of the object.

- **observations** (*pandas.DataFrame*) – Observations of the object's light curve. This should be a pandas DataFrame with at least the following columns:
  - time: The time of each observation.
  - band: The band used for the observation.
  - flux: The measured flux value of the observation.
  - flux_error: The flux measurement uncertainty of the observation.

__init__ (*metadata*, *observations*)
Create a new AstronomicalObject

Methods

__init__(metadata, observations)	Create a new AstronomicalObject
fit_gaussian_process([fix_scale, verbose, ...])	Fit a Gaussian Process model to the light curve.
get_default_gaussian_process()	Get the default Gaussian Process.
plot_light_curve([show_gp, verbose, axis])	Plot the object's light curve
predict_gaussian_process(bands, times[, ...])	Predict the Gaussian process in a given set of bands and at a given set of times.
preprocess_observations([subtract_background])	Apply preprocessing to the observations.
print_metadata()	Print out the object's metadata in a nice format.
subtract_background()	Subtract the background levels from each band.

Attributes

bands	Return a list of bands that this object has observations in

avocado.AstronomicalObject.bands

AstronomicalObject.**bands**
Return a list of bands that this object has observations in

Returns **bands** – A list of bands, ordered by their central wavelength.

Return type numpy.array

avocado.AstronomicalObject.subtract_background

AstronomicalObject.**subtract_background**()
Subtract the background levels from each band.

The background levels are estimated using a biweight location estimator. This estimator will calculate a robust estimate of the background level for objects that have short-lived light curves, and it will return something like the median flux level for periodic or continuous light curves.

Returns **subtracted_observations** – A modified version of the observations DataFrame with the background level removed.

Return type pandas.DataFrame

avocado.AstronomicalObject.preprocess_observations

`AstronomicalObject.`**`preprocess_observations`**(*subtract_background=True*, ***kwargs*)

Apply preprocessing to the observations.

This function is intended to be used to transform the raw observations table into one that can actually be used for classification. For now, all that this step does is apply background subtraction.

Parameters

- **`subtract_background`** (*`bool (optional)`*) – If True (the default), a background subtraction routine is applied to the lightcurve before fitting the GP. Otherwise, the flux values are used as-is.
- **`kwargs`** (*`dict`*) – Additional keyword arguments. These are ignored. We allow additional keyword arguments so that the various functions that call this one can be called with the same arguments, even if they don't actually use them.

Returns **preprocessed_observations** – The preprocessed observations that can be used for further analyses.

Return type pandas.DataFrame

avocado.AstronomicalObject.fit_gaussian_process

`AstronomicalObject.`**`fit_gaussian_process`**(*fix_scale=False*, *verbose=False*, *guess_length_scale=20.0*, ***preprocessing_kwargs*)

Fit a Gaussian Process model to the light curve.

We use a 2-dimensional Matern kernel to model the transient. The kernel width in the wavelength direction is fixed. We fit for the kernel width in the time direction as different transients evolve on very different time scales.

Parameters

- **`fix_scale`** (*`bool (optional)`*) – If True, the scale is fixed to an initial estimate. If False (default), the scale is a free fit parameter.
- **`verbose`** (*`bool (optional)`*) – If True, output additional debugging information.
- **`guess_length_scale`** (*`float (optional)`*) – The initial length scale to use for the fit. The default is 20 days.
- **`preprocessing_kwargs`** (*`kwargs (optional)`*) – Additional preprocessing arguments that are passed to *preprocess_observations*.

Returns

- **gaussian_process** (*function*) – A Gaussian process conditioned on the object's lightcurve. This is a wrapper around the george *predict* method with the object flux fixed.

- **gp_observations** (*pandas.DataFrame*) – The processed observations that the GP was fit to. This could have effects such as background subtraction applied to it.
- **gp_fit_parameters** (*list*) – A list of the resulting GP fit parameters.

avocado.AstronomicalObject.get_default_gaussian_process

`AstronomicalObject.`**`get_default_gaussian_process`**`()`

Get the default Gaussian Process.

This method calls fit_gaussian_process with the default arguments and caches its output so that multiple calls only require fitting the GP a single time.

avocado.AstronomicalObject.predict_gaussian_process

`AstronomicalObject.`**`predict_gaussian_process`**`(`*bands*, *times*, *uncertainties=True*, *fitted_gp=None*, ***gp_kwargs*`)`

Predict the Gaussian process in a given set of bands and at a given set of times.

Parameters

- **bands** (*list (str)*) – bands to predict the Gaussian process in.
- **times** (*list or numpy.array of floats*) – times to evaluate the Gaussian process at.
- **uncertainties** (*bool (optional)*) – If True (default), the GP uncertainties are computed and returned along with the mean prediction. If False, only the mean prediction is returned.
- **fitted_gp** (*function (optional)*) – By default, this function will perform the GP fit before doing predictions. If the GP fit has already been done, then the fitted GP function (returned by fit_gaussian_process) can be passed here instead to skip redoing the fit.
- **gp_kwargs** (*kwargs (optional)*) – Additional arguments that are passed to *fit_gaussian_process*.

Returns

- **predictions** (*numpy.array*) – A 2-dimensional array with shape (len(bands), len(times)) containing the Gaussian process mean flux predictions.
- **prediction_uncertainties** (*numpy.array*) – Only returned if uncertainties is True. This is an array with the same shape as predictions containing the Gaussian process uncertainty for the predictions.

avocado.AstronomicalObject.plot_light_curve

`AstronomicalObject.`**`plot_light_curve`**`(`*show_gp=True*, *verbose=False*, *axis=None*, ***kwargs*`)`

Plot the object's light curve

Parameters

- **show_gp** (*bool (optional)*) – If True (default), the Gaussian process prediction is plotted along with the raw data.
- **verbose** (*bool (optional)*) – If True, print detailed information about the light curve and GP fit.

- **axis** (*matplotlib.axes.Axes* (optional)) – The matplotlib axis to plot to. If None, a new figure will be created.
- **kwargs** (*kwargs (optional)*) – Additional arguments. If show_gp is True, these are passed to *fit_gaussian_process*. Otherwise, these are passed to *preprocess_observations*.

avocado.AstronomicalObject.print_metadata

AstronomicalObject.**print_metadata**()
Print out the object's metadata in a nice format.

3.2.3 Dataset augmentation

Augmentor API

Augmentor(**cosmology_kwargs)	Class used to augment a dataset.
Augmentor.augment_object(reference_object[, ...])	Generate an augmented version of an object.
Augmentor.augment_dataset(augment_name, ...)	Generate augmented versions of all objects in a dataset.

avocado.Augmentor

class avocado.**Augmentor**(***cosmology_kwargs*)
Class used to augment a dataset.

This class takes *AstronomicalObject* instances as input and generates new *AstronomicalObject* instances with the following transformations applied:

- Drop random observations.
- Drop large blocks of observations.
- For galactic observations, adjust the brightness (= distance).
- For extragalactic observations, adjust the redshift.
- Add noise.

When changing the redshift, we use the host_specz measurement as the redshift of the reference object. While in simulations we might know the true redshift, that isn't the case for real experiments.

The augmentor needs to have some reasonable idea of the properties of the survey that it is being applied to. If there is a large dataset that the classifier will be used on, then that dataset can be used directly to estimate the properties of the survey.

This class needs to be subclassed to implement survey specific methods. These methods are:

- *Augmentor._augment_metadata()*
- Either *Augmentor._choose_sampling_times()* or *Augmentor._choose_target_observation_count()*
- *Augmentor._simulate_light_curve_uncertainties()*
- *Augmentor._simulate_detection()*

Parameters **cosmology_kwargs** (*kwargs (optional)*) – Optional parameters to modify the cosmology assumed in the augmentation procedure. These kwargs will be passed to astropy.cosmology.FlatLambdaCDM.

__init__(***cosmology_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**cosmology_kwargs)	Initialize self.
augment_dataset(augment_name, dataset, ...)	Generate augmented versions of all objects in a dataset.
augment_object(reference_object[, force_success])	Generate an augmented version of an object.

avocado.Augmentor.augment_object

Augmentor.**augment_object**(*reference_object*, *force_success=True*)
Generate an augmented version of an object.

Parameters

- **reference_object** (*AstronomicalObject*) – The object to use as a reference for the augmentation.
- **force_success** (*bool*) – If True, then if we fail to generate an augmented light curve for a specific set of augmented parameters, we choose a different set of augmented parameters until we eventually get an augmented light curve. This is useful for debugging/interactive work, but when actually augmenting a dataset there is a massive speed up to ignoring bad light curves without a major change in classification performance.

Returns **augmented_object** – The augmented object. If force_success is False, this can be None.

Return type *AstronomicalObject*

avocado.Augmentor.augment_dataset

Augmentor.**augment_dataset**(*augment_name*, *dataset*, *num_augments*, *include_reference=True*)
Generate augmented versions of all objects in a dataset.

Parameters

- **augment_name** (*str*) – The name of the augmented dataset.
- **dataset** (*Dataset*) – The dataset to use as a reference for the augmentation.
- **num_augments** (*int*) – The number of times to use each object in the dataset as a reference for augmentation. Note that augmentation sometimes fails, so this is the number of tries, not the number of sucesses.
- **include_reference** (*bool (optional)*) – If True (default), the reference objects are included in the new augmented dataset. Otherwise they are dropped.

Returns **augmented_dataset** – The augmented dataset.

Return type *Dataset*

Augmentor Implementations

plasticc.PlasticcAugmentor()	Implementation of an Augmentor for the PLAsTiCC dataset

avocado.plasticc.PlasticcAugmentor

class avocado.plasticc.**PlasticcAugmentor**

Implementation of an Augmentor for the PLAsTiCC dataset

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
augment_dataset(augment_name, dataset, ...)	Generate augmented versions of all objects in a dataset.
augment_object(reference_object[, force_success])	Generate an augmented version of an object.

Augmentor methods to implement in subclasses

Augmentor._augment_metadata(reference_object)	Generate new metadata for the augmented object.
Augmentor._choose_sampling_times(...[, ...])	Choose the times at which to sample for a new augmented object.
Augmentor._choose_target_observation_count(...)	Choose the target number of observations for a new augmented light curve.
Augmentor._simulate_light_curve_uncertainty(...)	Simulate the observation-related noise for a light curve.
Augmentor._simulate_detection(observations, ...)	Simulate the detection process for a light curve.

avocado.Augmentor._augment_metadata

Augmentor.**_augment_metadata**(*reference_object*)

Generate new metadata for the augmented object.

This method needs to be implemented in survey-specific subclasses of this class.

Parameters **reference_object** (*AstronomicalObject*) – The object to use as a reference for the augmentation.

Returns **augmented_metadata** – The augmented metadata

Return type dict

avocado.Augmentor._choose_sampling_times

Augmentor.**_choose_sampling_times**(*reference_object*, *augmented_metadata*, *max_time_shift=50*, *block_width=250*, *window_padding=100*, *drop_fraction=0.1*)

Choose the times at which to sample for a new augmented object.

This method should really be survey specific, but a default implementation is included here that works reasonably well for generic light curves. If you are implementing a survey specific version of this method, you only need to have the reference_object and augmented_metadata parameters. The other parameters are different knobs for this method.

This implementation of _choose_sampling_times requires that the method _choose_target_observation_count() be defined that returns how many observations we should attempt to have for the new light curve. If a different implementation of _choose_sampling_times is used, that method may not be required.

Parameters

- **reference_object** (*AstronomicalObject*) – The object to use as a reference for the augmentation.
- **augmented_metadata** (*dict*) – The augmented metadata
- **max_time_shift** (*float (optional)*) – The new sampling times will be shifted by up to this amount relative to the original ones.
- **block_width** (*float (optional)*) – A block of observations with a width specified by this parameter will be dropped.
- **window_padding** (*float (optional)*) – Observations outside of a window bounded by the first and last observations in the reference objects light curve with a padding specified by this parameter will be dropped.
- **drop_fraction** (*float (optional)*) – This fraction of observations will always be dropped when creating the augmented light curve.

Returns

sampling_times – A pandas Dataframe that has the following columns:

- time : the times of the simulated observations.
- band : the bands of the simulated observations.
- reference_time : the times in the reference light curve that correspond to the times of the simulated observations.

Return type pandas Dataframe

avocado.Augmentor._choose_target_observation_count

Augmentor.**_choose_target_observation_count**(*augmented_metadata*)

Choose the target number of observations for a new augmented light curve.

This method needs to be implemented in survey-specific subclasses of this class if using the default implementation of *_choose_sampling_times*.

Parameters **augmented_metadata** (*dict*) – The augmented metadata

Returns **target_observation_count** – The target number of observations in the new light curve.

Return type int

avocado.Augmentor._simulate_light_curve_uncertainties

Augmentor.**_simulate_light_curve_uncertainties**(*observations*, *augmented_metadata*)

Simulate the observation-related noise for a light curve.

This method needs to be implemented in survey-specific subclasses of this class. It should simulate the observation uncertainties for the light curve.

Parameters

- **observations** (*pandas.DataFrame*) – The augmented observations that have been sampled from a Gaussian Process. These observations have model flux uncertainties listed that should be included in the final uncertainties.
- **augmented_metadata** (*dict*) – The augmented metadata

Returns **observations** – The observations with uncertainties added.

Return type pandas.DataFrame

avocado.Augmentor._simulate_detection

Augmentor.**_simulate_detection**(*observations*, *augmented_metadata*)

Simulate the detection process for a light curve.

This method needs to be implemented in survey-specific subclasses of this class. It should simulate whether each observation is detected as a point-source by the survey and set the "detected" flag in the observations DataFrame. It should also return whether or not the light curve passes a base set of criterion to be included in the sample that this classifier will be applied to.

Parameters

- **observations** (*pandas.DataFrame*) – The augmented observations that have been sampled from a Gaussian Process.
- **augmented_metadata** (*dict*) – The augmented metadata

Returns

- **observations** (*pandas.DataFrame*) – The observations with the detected flag set.
- **pass_detection** (*bool*) – Whether or not the full light curve passes the detection thresholds used for the full sample.

3.2.4 Classification

Classifier API

Classifier(name)	Classifier used to classify the different objects in a dataset.
Classifier.train(dataset)	Train the classifier on a dataset
Classifier.predict(dataset)	Generate predictions for a dataset
Classifier.path	Get the path to where a classifier should be stored on disk
Classifier.write([overwrite])	Write a trained classifier to disk
Classifier.load(name)	Load a classifier that was previously saved to disk

avocado.Classifier

class avocado.**Classifier**(*name*)

Classifier used to classify the different objects in a dataset.

Parameters **name** (*str*) – The name of the classifier.

__init__(*name*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(name)	Initialize self.
load(name)	Load a classifier that was previously saved to disk
predict(dataset)	Generate predictions for a dataset
train(dataset)	Train the classifier on a dataset
write([overwrite])	Write a trained classifier to disk

Attributes

path	Get the path to where a classifier should be stored on disk

avocado.Classifier.train

Classifier.**train**(*dataset*)
Train the classifier on a dataset

This needs to be implemented in subclasses.

Parameters **dataset** (*Dataset*) – The dataset to use for training.

avocado.Classifier.predict

Classifier.**predict**(*dataset*)
Generate predictions for a dataset

This needs to be implemented in subclasses.

Parameters **dataset** (*Dataset*) – The dataset to generate predictions for.

Returns **predictions** – A pandas Series with the predictions for each class.

Return type pandas.DataFrame

avocado.Classifier.path

Classifier.**path**
Get the path to where a classifier should be stored on disk

avocado.Classifier.write

Classifier.**write**(*overwrite=False*)
Write a trained classifier to disk

Parameters

- **name** (*str*) – A unique name used to identify the classifier.

- **overwrite** (*bool (optional)*) – If a classifier with the same name already exists on disk and this is True, overwrite it. Otherwise, raise an AvocadoException.

avocado.Classifier.load

classmethod Classifier.**load**(*name*)

Load a classifier that was previously saved to disk

Parameters **name** (*str*) – A unique name used to identify the classifier to load.

Classifier Implementations

LightGBMClassifier(name, featurizer[, ...])	Feature based classifier using LightGBM to classify objects.

avocado.LightGBMClassifier

class avocado.**LightGBMClassifier**(*name*, *featurizer*, *class_weights=None*, *weighting_function=<function evaluate_weights_flat>*)

Feature based classifier using LightGBM to classify objects.

This uses a weighted multi-class logarithmic loss that normalizes for the total counts of each class. This classifier is optimized for the metric used in the PLAsTiCC Kaggle challenge.

Parameters

- **featurizer** (*Featurizer*) – The featurizer to use to select features for classification.
- **class_weights** (*dict (optional)*) – Weights to use for each class. If not set, equal weights are assumed for each class.
- **weighting_function** (*function (optional)*) – Function to use to evaluate weights. By default, *evaluate_weights_flat* is used which normalizes the weights for each class so that their overall weight matches the one set by class_weights. Within each class, *evaluate_weights_flat* gives all objects equal weights. Any weights function can be used here as long as it has the same signature as *evaluate_weights_flat*.

__init__(*name*, *featurizer*, *class_weights=None*, *weighting_function=<function evaluate_weights_flat>*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(name, featurizer[, class_weights, ...])	Initialize self.
load(name)	Load a classifier that was previously saved to disk
predict(dataset)	Generate predictions for a dataset
train(dataset[, num_folds, random_state])	Train the classifier on a dataset
write([overwrite])	Write a trained classifier to disk

Attributes

path	Get the path to where a classifier should be stored on disk

Weights and metrics

evaluate_weights_flat(dataset[, class_weights])	Evaluate the weights to use for classification on a dataset.
evaluate_weights_redshift(dataset[, ...])	Evaluate redshift-weighted weights to use to generate a rates-independent classifier.
weighted_multi_logloss(true_classes, predictions)	Evaluate a weighted multi-class logloss function.

3.2.5 Feature extraction

Featurizer API

Featurizer	Class used to extract features from objects.
Featurizer.extract_raw_features(...[, ...])	Extract raw features from an object
Featurizer.select_features(raw_features)	Select features to use for classification
Featurizer.extract_features(astronomical_object)	Extract features from an object.

avocado.Featurizer

class avocado.**Featurizer**

Class used to extract features from objects.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

extract_features(astronomical_object)	Extract features from an object.
extract_raw_features(astronomical_object[, ...])	Extract raw features from an object
select_features(raw_features)	Select features to use for classification

avocado.Featurizer.extract_raw_features

Featurizer.**extract_raw_features**(*astronomical_object*, *return_model=False*)

Extract raw features from an object

Featurizing is slow, so the idea here is to extract a lot of different things, and then in *select_features* these features are postprocessed to select the ones that are actually fed into the classifier. This allows for rapid iteration of training on different feature sets. Note that the features produced by this method are often unsuitable for classification, and may include data leaks. Make sure that you understand what features can be used for real classification before making any changes.

For now, there is no generic featurizer, so this must be implemented in survey-specific subclasses.

Parameters

- **astronomical_object** (*AstronomicalObject*) – The astronomical object to featurize.
- **return_model** (*bool*) – If true, the light curve model is also returned. Defaults to False.

Returns

- **raw_features** (*dict*) – The raw extracted features for this object.
- **model** (*dict (optional)*) – A dictionary with the light curve model in each band. This is only returned if return_model is set to True.

avocado.Featurizer.select_features

Featurizer.**select_features**(*raw_features*)

Select features to use for classification

This method should take a DataFrame or dictionary of raw features, produced by *featurize*, and output a list of processed features that can be fed to a classifier.

Parameters **raw_features** (*pandas.DataFrame or dict*) – The raw features extracted using *featurize*.

Returns **features** – The processed features that can be fed to a classifier.

Return type pandas.DataFrame or dict

avocado.Featurizer.extract_features

Featurizer.**extract_features**(*astronomical_object*)

Extract features from an object.

This method extracts raw features with *extract_raw_features*, and then selects the ones that should be used for classification with *select_features*. This method is just a wrapper around those two methods and is intended to be used as a shortcut for feature extraction on individual objects.

Parameters **astronomical_object** (*AstronomicalObject*) – The astronomical object to featurize.

Returns **features** – The processed features that can be fed to a classifier.

Return type pandas.DataFrame or dict

Featurizer Implementations

plasticc.PlasticcFeaturizer	Class used to extract features for the PLAsTiCC dataset.

avocado.plasticc.PlasticcFeaturizer

class avocado.plasticc.**PlasticcFeaturizer**

Class used to extract features for the PLAsTiCC dataset.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

extract_features(astronomical_object)	Extract features from an object.
extract_raw_features(astronomical_object[, ...])	Extract raw features from an object
select_features(raw_features)	Select features to use for classification

3.2.6 Exceptions

AvocadoException	The base class for all exceptions raised in avocado.

avocado.AvocadoException

exception avocado.**AvocadoException**
The base class for all exceptions raised in avocado.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

E

F

G

L

P

S

T

W