# Astraea

**May 21, 2020**

# Contents

*Astraea* is a package to train Random Forest (RF) models on datasets. It provides tools to train RF classifiers and regressors as well as perform simple cross-validation tests and performance plots on the test set.

It was first developed to calculate rotation period of stars from various stellar properties provided and is intended to predict long rotation periods (e.g. that of M-dwarfs) from short TESS lightcurves (27-day lightcurves).

We provide access to trained models on stars from the catalog by McQuillian et all. (2014). User can predict whether the rotation period can be recovered and measure recoverable rotation periods for the stars in the Kepler field by using their temperatures, colors, kinematics, etc.

CHAPTER 1

# User Guide

## 1.1 Installation

From GitHub source:

```
git clone https://github.com/lyx12311/Astraea.git
cd Astraea
python setup.py install
```

## 1.2 Dependencies

The dependencies of *Astraea* are NumPy, matplotlib, pandas, Astropy, scikit-learn.

These can be installed using pip:

```
pip install numpy matplotlib pandas Astropy scikit-learn
```

## 1.3 Running tests

You can test *Astraea* from within the base directory using pytest.

```
pytest
```

## 1.4 API documentation

Astraea.**FLICKERinstall**()
Installs the FLICKER software to calcualte Flicker values from light curves. Documentation: https://flicker.readthedocs.io.

Astraea.**getFlicker**(*t*, *sig*)
Calculates the Flicker value

**Parameters**

- **t** – Time [days]
- **sig** – Flux

**Returns** Flicker ([float]): Flicker value

Astraea.**getKeplerProt**(*X_pred*)
Predict rotation period from trained models.

This function predicts rotation periods for stars in the Kepler field. The models are trained on rotation periods from McQuillian et all. (2014), Santos et all. (2019) and Garcia et all. (2014). If the models are not already downloaded, this tool will download the model which might take a couple of minues. It first passes the stars through a classifier, which identifies what stars have measureable rotation periods. Then it uses two regressor models (one with 1 estimator and another one with 100 estimators) to predict rotation periods. If column "Prot" exist, it will also output the true periods associated with the predicted periods. The light curve feature "flicker" can be calculated using software FLICKER.

**Parameters** **X_pred** (*[Pandas DataFrame]*) – DataFrame contains all variables needed, run Astraea.getTrainF() to print out requirements

**Returns**

**Containing**

**TrueProt** True rotation period (if avaliable)

**Prot_prediction_1est** Period predictions with 1 estimator

**Prot_prediction_100est** Period prediction with 100 estimators

**Return type** <pandas.DataFrame> or <pandas.Series>

Astraea.**getRvar**(*Flux*)
Calculates light curve Rvar

**Parameters** **Flux** – The light curve flux in ppm

**Returns** The variability of the light curve

**Return type** Rvar ([float])

Astraea.**getLGpeak**(*t*, *sig*, *sig_err*)
Calculates the location of the highest peak and the maximum power of the hightest peak

**Parameters**

- **t** – Time [days]
- **sig** – Flux
- **sig_err** – Flux error

**Returns** LG_Prot ([float]): The period calculated from Lomb-Scargle :LG_peaks ([float]): The maximum peak height

Astraea.**getTrainF**()
Print out needed featuers for the model in order.

Astraea.**getVs**(*df*)
Calculates tangential velocity (v_tan) and vertical velocity proximation (v_b).

**Parameters** **df** (*[Pandas DataFrame]*) – DataFrame contains columns 'parallax', 'pmra', 'pmdec', 'ra', 'dec', which are parallax, ra proper motion, dec propermotion, right ascension and declination, respectively

**Returns** v_t ([array-like]): Tangential velocity :v_b ([array-like]): Proxy for vertical velocity

Astraea.**RFclassifier**(*df*, *testF*, *modelout=False*, *traind=0.8*, *ID_on='KID'*, *X_train_ind=[]*, *X_test_ind=[]*, *target_var='Prot_flag'*, *n_estimators=100*, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*)

Train RF classifier model and predict values for cross-validation dataset.

It uses scikit-learn Random Forest classifier model. All default hyper-parameters are taken from the scikit-learn model that user can change by adding in optional inputs. More details on hyper-parameters, see https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. To use the module to train a RF model to predict rotation period, input a pandas dataFrame with column names as well as a list of attribute names.

**Parameters**

- **df** (*[Pandas DataFrame]*) – DataFrame contains all variables needed
- **testF** (*[string list]*) – List of feature names used to train
- **modelout** (*Optional [bool]*) – Whether to only output the trained model
- **traind** (*Optinal [float]*) – Fraction of data use to train, the rest will be used to perform cross-validation test (default 0.8)
- **ID_on** (*Optional [string]*) – What is the star identifier column name (default 'KID'). If specified ID column does not exist, it will just take the index as ID
- **X_train_ind** (*Optional [list]*) – List of *ID_on* for training set, if not specified, take random *traind* fraction of indexes from *ID_on* column
- **X_test_ind** (*Optional [list]*) – List of *ID_on* for testing set, if not specified, take the remaining (1-*traind*) fraction of indexes from *ID_on* column that is not in the training set (*X_train_ind*)
- **target_var** (*Optional [string]*) – Label column name (default 'Prot_flag')

**Returns**

**regr** Sklearn RF classifier model (attributes see https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

**<pandas.Series> containing**

**actrualF ([string list])** Actrual features used

**importance ([float list])** Impurity-based feature importance ordering as *actrualF*

**ID_train ([list])** List of *ID_on* used for training set

**ID_test ([list])** List of *ID_on* used for testing set

**predictp ([float list])** List of prediction on testing set

**X_test ([matrix])** Matrix used to predict label values for testing set

**y_test ([array-like])** Array of true label values of testing set

**X_train ([matrix])** Matrix used to predict label values for training set

**y_train ([array-like])** Array of true label values of training set

**Return type** <RF model>, <pandas.Series>

Astraea.**RFregressor**(*df*, *testF*, *modelout=False*, *traind=0.8*, *ID_on='KID'*, *X_train_ind=[]*, *X_test_ind=[]*, *target_var='Prot'*, *target_var_err='Prot_err'*, *chisq_out=False*, *MREout=False*, *n_estimators=100*, *criterion='mse'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *bootstrap=True*, *oob_score=False*, *n_jobs=1*, *random_state=None*, *verbose=0*, *warm_start=False*)

Train RF regression model and perform cross-validation test.

It uses scikit-learn Random Forest regressor model. All default hyper-parameters are taken from the scikit-learn model that user can change by adding in optional inputs. More details on hyper-parameters, see https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. To use the module to train a RF model to predict rotation period, input a pandas dataFrame with column names as well as a list of attribute names.

**Parameters**

- **df** (*[Pandas DataFrame]*) – DataFrame contains all variables needed
- **testF** (*[string list]*) – List of feature names used to train
- **modelout** (*Optional [bool]*) – Whether to only output the trained model
- **traind** (*Optinal [float]*) – Fraction of data use to train, the rest will be used to perform cross-validation test (default 0.8)
- **ID_on** (*Optional [string]*) – What is the star identifier column name (default 'KID'). If specified ID column does not exist, it will just take the index as ID
- **X_train_ind** (*Optional [list]*) – List of *ID_on* for training set, if not specified, take random *traind* fraction of indexes from *ID_on* column
- **X_test_ind** (*Optional [list]*) – List of *ID_on* for testing set, if not specified, take the remaining (1-*traind*) fraction of indexes from *ID_on* column that is not in the training set (*X_train_ind*)
- **target_var** (*Optional [string]*) – Label column name (default 'Prot')
- **target_var_err** (*Optional [string]*) – Label error column name (default 'Prot_err')
- **chisq_out** (*optional [bool]*) – If true, only output average chisq value
- **MREout** (*optional [bool]*) – If true, only output median relative error. If both *chisq_out* and *MREout* are true, then output only these two values

**Returns**

**regr** Sklearn RF regressor model (attributes see https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

**<pandas.Series> containing**

**actrualF ([string list])** Actrual features used

**importance ([float list])** Impurity-based feature importance ordering as *actrualF*

**ID_train ([list])** List of *ID_on* used for training set

**ID_test ([list])** List of *ID_on* used for testing set

**predictp ([float list])** List of prediction on testing set

**ave_chi ([float])** Average chisq on cross-validation (testing) set

**MRE_val ([float])** Median relative error on cross-validation (testing) set

**X_test ([matrix])** Matrix used to predict label values for testing set

**y_test ([array-like])** Array of true label values of testing set

**X_train ([matrix])** Matrix used to predict label values for training set

**y_train ([array-like])** Array of true label values of training set

**Return type** <RF model>, <pandas.Series>

Astraea.**load_RF**()

Load random forest classifier and regressors from zendo.org.

Two regressors will be laoded, one with 1 estimator and one with 100 estimators. 1 estimator minimizes bias (systematic offset) with the cost of high varience (scattering) and 100 estimators minimizes varience and maximizes bias. If user wants to predict rotation period from Kepler light curves then it is best to use model with 100 estimiators and user should use model with 1 estimator otherwise. Model trained on TESS light curves are still being developed.

Astraea.**plot_corr**(*df*, *y_vars*, *x_var='Prot'*, *logplotarg=[]*, *logarg=[]*, *MS=1*)

Plot correlations on one variable vs other variables specified by user

**Parameters** **df** – DataFrame contains all variables needed

**Returns** plots for feature correlations

**Return type** <matplotlib.plot>

Astraea.**plot_result**(*actrualF*, *importance*, *prediction*, *y_test*, *y_test_err=[]*, *topn=20*, *MS=3*, *labelName='Period'*)

Plot impurity-based feature importance as well as predicted values vs true values for a random forest model

**Parameters**

- **actrualF** (*[array-like]*) – Feature used (from function output of RFregressor())
- **importance** (*[array-like]*) – importance of the model (from function output of RFregressor())
- **prediction** (*[array-like]*) – Predicted values (from function output of RFregressor())
- **y_test** (*[array-like]*) – true values (from function output of RFregressor())
- **y_test_err** (*Optional [array-like]*) – Errors for true values (from function output of RFregressor())
- **topn** (*Optional [int]*) – How many most important features to plot
- **MS** (*Optional [int]*) – Markersize for plotting true vs predicted values
- **labelName** (*Optional [string]*) – Label name

**Returns** importance plot as well as true vs prediction plot

**Return type** <matplotlib.plot>

CHAPTER 2

# Tutorials

**Note:** This tutorial was generated from an IPython notebook that can be downloaded here.

## 2.1 Predict rotation period for Kepler stars using existing model

load trained random forest models and predict rotation periods from provided features or light curve(s).

### 2.1.1 Calculate rotation period(s) from features

Below is a tutorial to calculate rotation period(s) for single or multiple stars using the existing model. Currently, this model is only tested on stars in the Kepler field. To achieve the best results, the light curve statistics should be calculated from Kepler light curves. For any stars outside of the Kepler field, it is best to use the model with 1 estimators to minimize model bias.

```
import Astraea
import pandas as pd
import numpy as np

# print out needed features in order
TrainF_class, TrainF_reg = Astraea.getTrainF()

# load in existing testing data
KeplerTest = Astraea.load_KeplerTest()
```

```
>>> classification features are: ['LG_peaks [Lomb-Scargle peak height]', 'Rvar [ppm]',
↪ 'parallax [gaia]', 'radius_percentile_lower [gaia]', 'radius_percentile_upper␣
↪[gaia]', 'phot_g_mean_flux_over_error [gaia]', 'bp_g [gaia]']
```

(continues on next page)

(continued from previous page)

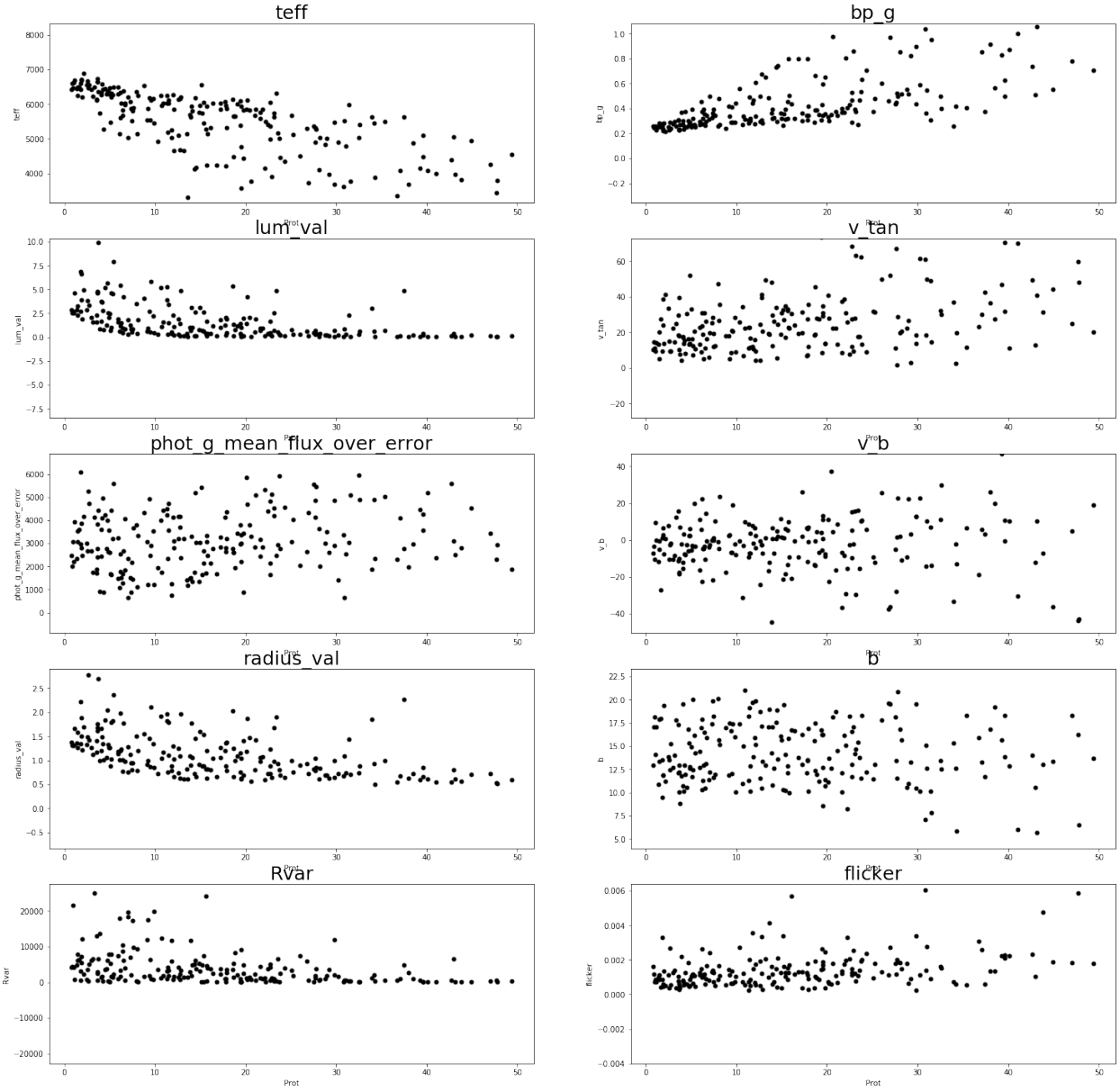
```
>>> regression features are: ['teff [gaia]','bp_g [gaia]','lum_val [gaia]','v_tan␣
↪[getVs()]','phot_g_mean_flux_over_error [gaia]','v_b [getVs()]','radius_val [gaia]',
↪'b [gaia]','Rvar [ppm]','flicker [FLICKER]']
```

If the data contains all features, first create a dictionary with required columns, then convert it into a <*pd.DataFrame*>,

```
# construct pd.DataFrame that contains all the features
starStat = {'LG_peaks': KeplerTest.LG_peaks.values, 'Rvar': KeplerTest.Rvar.values,
          'parallax': KeplerTest.parallax.values,
          'radius_percentile_lower': KeplerTest.radius_percentile_lower.values,
          'radius_val': KeplerTest.radius_val.values,
          'radius_percentile_upper': KeplerTest.radius_percentile_upper.values,
          'phot_g_mean_flux_over_error': KeplerTest.phot_g_mean_flux_over_error.
↪values,
          'bp_g': KeplerTest.bp_g.values, 'teff': KeplerTest.teff.values,
          'lum_val': KeplerTest.lum_val.values, 'v_tan':KeplerTest.v_tan.values,
          'v_b': KeplerTest.v_b.values, 'b': KeplerTest.b.values,
          'flicker':KeplerTest.flicker.values, 'Prot': KeplerTest.Prot.values,
          'Prot_err': KeplerTest.Prot_err.values}

# dictionary -> dataframe
star_data = pd.DataFrame(starStat)

# only display 6 columns
pd.set_option("display.max_columns", 6)

star_data
```

Plot correlations between features and rotation period,

```
Astraea.plot_corr(star_data,TrainF_reg,MS=10)
```




You can now feed the *<pd.DataFrame>* into the function to predict rotation periods,

```
predics = Astraea.getKeplerProt(star_data)
```

```
>>> classification features are: ['LG_peaks [Lomb-Scargle peak height]', 'Rvar [ppm]',
↪ 'parallax [gaia]', 'radius_percentile_lower [gaia]', 'radius_percentile_upper␣
↪[gaia]', 'phot_g_mean_flux_over_error [gaia]', 'bp_g [gaia]']


>>> regression features are: ['teff [gaia]','bp_g [gaia]','lum_val [gaia]','v_tan␣
↪[getVs()]','phot_g_mean_flux_over_error [gaia]','v_b [getVs()]','radius_val [gaia]',
↪'b [gaia]','Rvar [ppm]','flicker [FLICKER]']


Total 205 stars!
```

(continues on next page)

(continued from previous page)

```
Classifing 205 stars!
205.0 stars have predictable rotation periods (100.0%)
Predicting rotation periods!
Finished!
```

```
predics
```

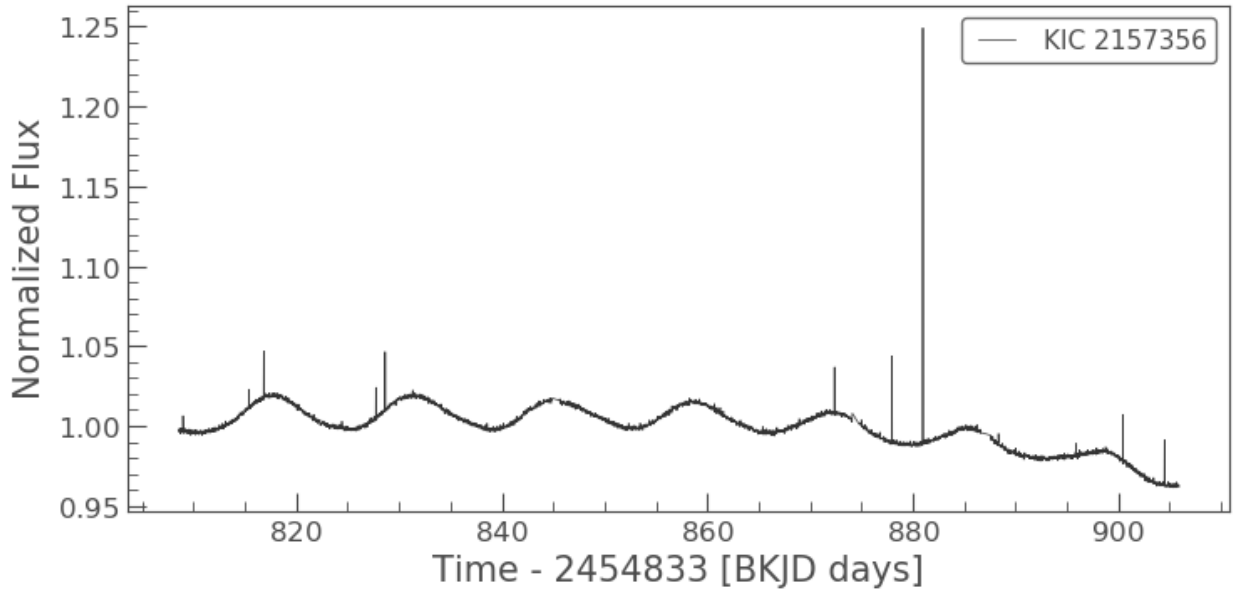
### 2.1.2 Download a light curve and calculate variablity, lomb-scargle peak and flickers needed for the trained model

Here is a basical tutorial for calculating light curve statistics needed for the trained model. Other statistics can found by cross-matching any Kepler stars with Gaia (a useful website crossmatching Kepler and Gaia by Megan Bedell https://gaia-kepler.fun). v_tan and v_b can be calculated after crossmatching Kepler with Gaia and use the function *Astraea.getVs()*.

*lightkurve* is used to download the light curve (https://docs.lightkurve.org).

```
import Astraea
import pandas as pd
import numpy as np
from lightkurve import search_targetpixelfile

# download light curve and plot it
tpf = search_targetpixelfile('KIC 2157356', quarter=9).download()
lc = tpf.to_lightcurve(aperture_mask=tpf.pipeline_mask)
lc.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a198a6b70>
```




Normalize the light curve and get time, flux and flux error,

```
t = lc.time # get time in days
sig = lc.flux*1e6/(np.median(lc.flux)-1) # get flux in ppm
sig_err = lc.flux_err/(np.median(lc.flux)-1) # get flux_err in ppm
```

Get varibility of light curve (**Rvar**)

```
Rvar = Astraea.getRvar(sig)
Rvar
```

```
42303.15312499995
```

Get Lomb-Scargle peak height (**LG_peaks**),

```
LG_Prot, LG_peaks = Astraea.getLGpeak(t,sig,sig_err)
LG_Prot, LG_peaks
```

```
(13.275704451936583, 0.20658748911841823)
```

Get flicker value (**flicker**). It will download *FLICKER* if not already installed,

```
flicker = Astraea.getFlicker(t,sig)
flicker
```

```
12609.067447067864
```

## 2.2 Train a regressor model and test its performance

Here is a simple example to train a simple regressor model and test the performance by calculating $\chi^2$, the relative median error, plotting the impurity feature importance and plotting the true vs predicted values from the cross-validation test.

### 2.2.1 Train a regressor model

Generate a <*pd.DataFrame*> of random features and labels to test Astraea.RFregressor.

Normally user will create a <*pd.DataFrame*> to include all the features and labels. Below is an example of creating a DataFrame with 20 feature columns with random numbers, named "$X0$", "$X1$", ..., "$X19$" and a label column that is a sum of all the linear combinations of the features, named "$y$" as well as randomly generated label error named "$y\_err$". Note that for this problem, the coefficients for the linear combinations are in decreasing order, so that $y = 20 * X0 + 19 * X1 + ... + 1 * X19$. By doing so, we can validate the feature importance output later on.

```
import Astraea
import pandas as pd
import numpy as np

# create random feature matrix with 20 features and 5000 total data points
X = np.random.rand(5000, 20)

# create labels from features
y = sum([X[:,i] * (20-i) for i in range(np.shape(X)[1])])

# put features and labels into one pandas dataFrame
```

(continues on next page)

(continued from previous page)

```
X_y = pd.DataFrame(np.hstack((X,np.reshape(y, (5000, 1)))),
                   columns = np.append(['X'+str(i) for i in range(np.shape(X)[1])], [
↪'y']))

# assign random errors
X_y['y_err'] = np.random.rand(5000)

# only display 9 columns
pd.set_option("display.max_columns", 9)

X_y
```

Train the regressor model with the *<pd.DataFrame>* generated above.

To use the regressor (*Astraea.RFregressor*) with default settings, input the **<*pd.DataFrame*>** combining all the features, label and label error, the **feature column names** in a list, the **label column name** and the **label error column name**. The default label column name is "Prot" and the default label error column name is "Prot_err". Here we also used 3 estimators instead of the default, which is 100 estimators, to minimize the variance. User could tune any hyper-parameters described in https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

```
# train the model with default settings
regr, regr_outs = Astraea.RFregressor(X_y, ['X'+str(i) for i in range(np.
↪shape(X)[1])],
                                      target_var='y', target_var_err='y_err', n_
↪estimators=3)
```

```
Simpliest example:
 regr,regr_outs = RFregressor(df,testF)

Fraction of data used to train: 0.8
# of Features attempt to train: 20
Features attempt to train: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9
↪', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19']
ID column not found, using index as ID!
5000 stars in dataframe!
5000 total stars used for RF!
4000 training stars!
Finished training! Making predictions!
Finished predicting! Calculating statistics!
Median Relative Error is: 0.06410001201586864
Average chi^2 is: 1016.3934151688192
Finished!
```

Print out the model statistics. Output discription see https://astraea.readthedocs.io/en/latest/user/api.html.

```
regr_outs
```

```
importance    [0.12945057256415657, 0.1158549055147204, 0.13...
actrualF      [X0, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, ...
ID_train      [2209, 56, 4923, 3246, 2584, 2332, 3790, 1088,...
ID_test       [0, 5, 11, 14, 24, 26, 27, 29, 32, 34, 44, 46,...
prediction    [101.27825467776267, 106.76937748535295, 120.9...
ave_chi2                                                1016.39
MRE                                                      0.0641
X_test        [[0.6065293441789202, 0.06295926764117177, 0.7...
```

(continues on next page)

(continued from previous page)

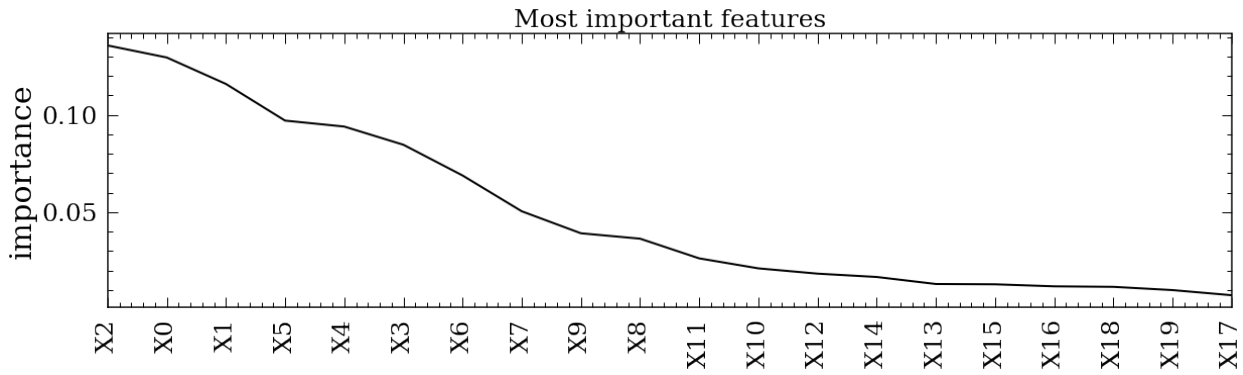
```
y_test        [98.64085252981158, 106.29389418685801, 131.77...
X_train       [[0.706831266028258, 0.14969010436464858, 0.06...
y_train       [106.74227914082933, 117.09292189496267, 141.1...
dtype: object
```

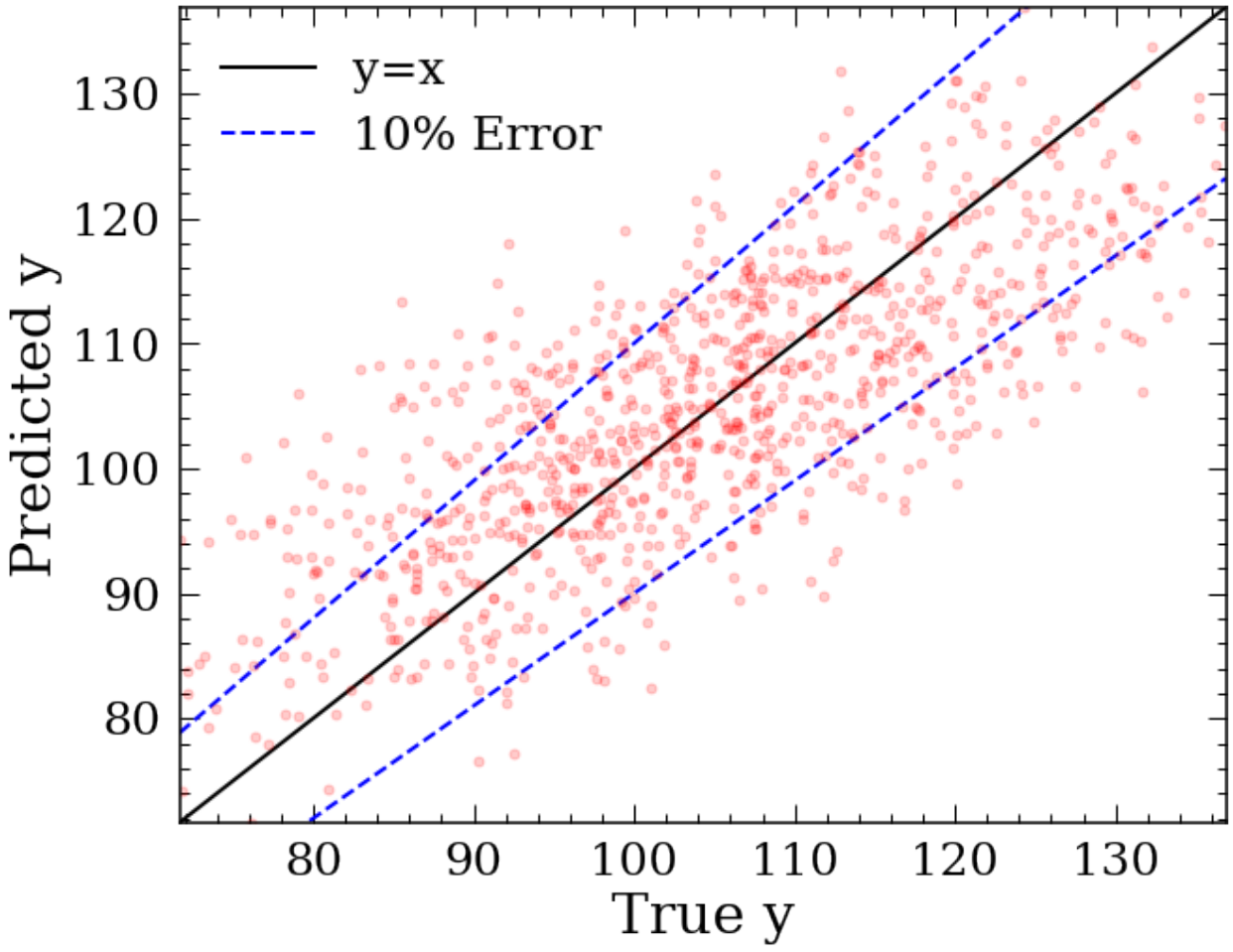
### 2.2.2 Plot the feature importances and Predicted vs True plot

User can use the function *Astraea.plot_result* to plot the basic impurity feature importance in decending order. https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e explained what impurity feature importance is and some other way of determining the importance that user can impliment.

To use this function, user could use the outputs from the *Astraea.RFregressor* function directly or specify the required inputs.

```
# plot cross-validation result
Astraea.plot_result(regr_outs['actrualF'], regr_outs['importance'], regr_outs[
↪'prediction'],
                    regr_outs['y_test'], labelName='y', MS=10)
```






### 2.2.3 Use the trained model to predict a new label value

User can now use the trained model to predict a new label value based on the trained features. To do so, simply pass the feature values in order.

```
# generate new random data
X_test_matrix = np.random.rand(5000, 20)

# put into dataframe so we can call the feature names in order
X_test = pd.DataFrame(X_test_matrix, columns = ['X'+str(i) for i in range(np.
↪shape(X)[1])])

# predict using the trained model
y_test = regr.predict(X_test[regr_outs['actrualF']])

y_test
```

```
array([ 93.65384246,  92.54149011, 103.50259242, ..., 120.4926051 ,
        86.8585833 , 102.70998368])
```

## 2.3 Train a classifier model and plot the Receiver operating characteristic (ROC) curve

Here is a example to train a classifier and plot the ROC curve.

### 2.3.1 Train a classification model

The process is very similar to training a regressor. User need to first generate a *<pd.DataFrame>* with all the features and labels and feed it to the model.

```
import Astraea
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification

# use sklearn.datasets to generate a dataset to test
X, y = make_classification(n_samples=1000, n_features=4, n_informative=2, n_
↪redundant=0,
                           random_state=0, shuffle=False)

# put features and labels into one pandas dataFrame
X_y = pd.DataFrame(np.hstack((X,np.reshape(y,(1000,1)))),
                   columns=np.append(['X'+str(i) for i in range(np.shape(X)[1])], ['y
↪']))

# print out DataFrame
X_y
```

```
# train the model with default settings
regr, regr_outs = Astraea.RFclassifier(X_y, ['X'+str(i) for i in range(np.
↪shape(X)[1])],
                                       target_var='y', n_jobs=1)
```

```
Simpliest example:
 regr,regr_outs = RFregressor(df,testF)

Fraction of data used to train: 0.8
# of Features attempt to train: 4
Features attempt to train: ['X0', 'X1', 'X2', 'X3']
ID column not found, using index as ID!
1000 stars in dataframe!
1000 total stars used for RF!
800 training stars!
Finished training! Making predictions!
Finished predicting!
Finished!
```

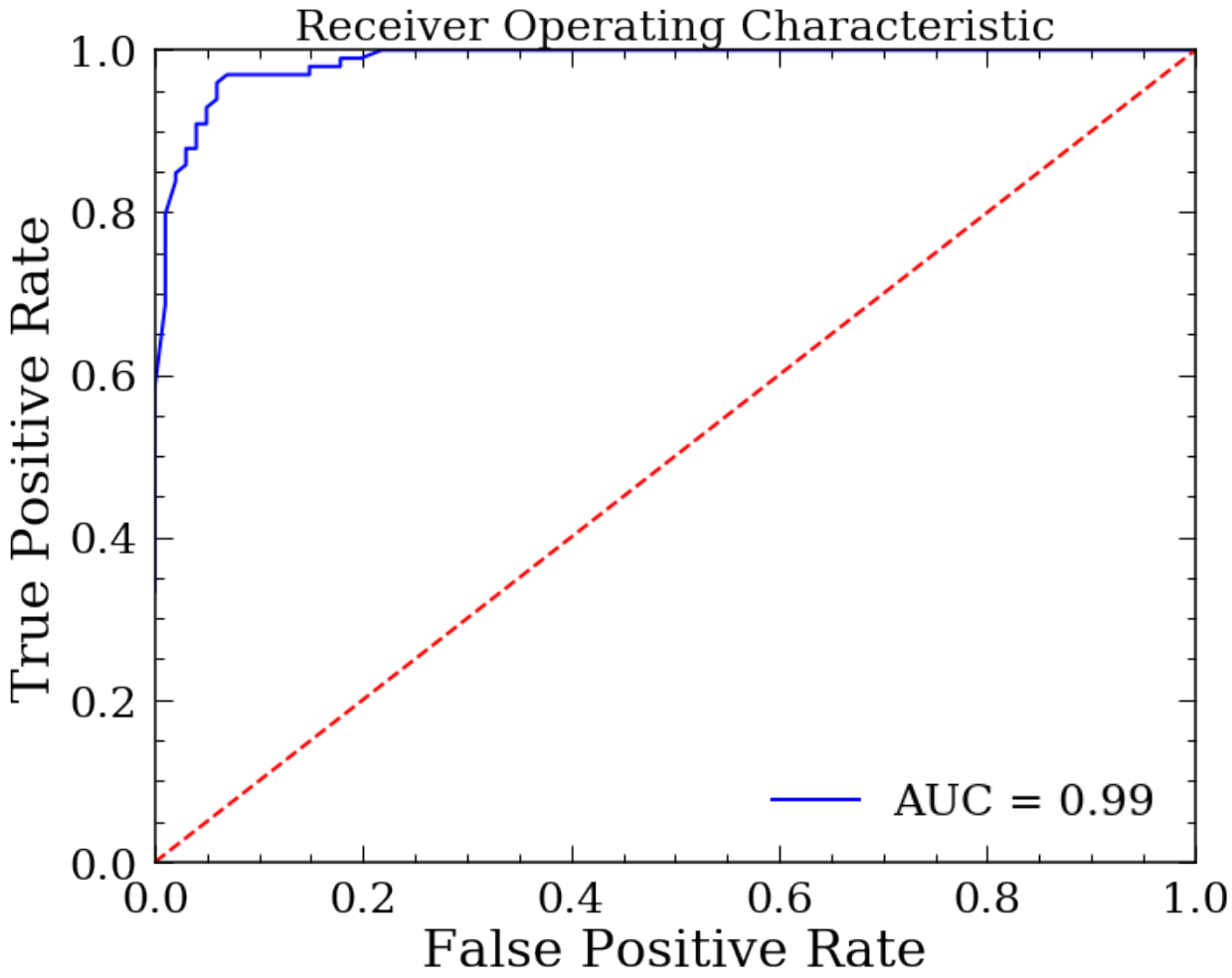
### 2.3.2 Plot the ROC curve

```
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
# predict the probability for testing set using the trained model
probs = regr.predict_proba(regr_outs.X_test)
preds = probs[:,1]

# calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, threshold = metrics.roc_curve(regr_outs.y_test, preds)

# get the accuracy
roc_auc = metrics.auc(fpr, tpr)

# plot the ROC curve
plt.figure(figsize=(10,8))
plt.title('Receiver Operating Characteristic',fontsize=25)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.tight_layout()
plt.savefig('ROC.png')
```



CHAPTER 3

# License & attribution



If you make use of this code, please cite this package and its dependencies. You can find more information about how and what to cite in the citation documentation (not yet complete).

# Python Module Index

## a



## s

# Index