rapa

FOXO Technologies

Nov 04, 2022

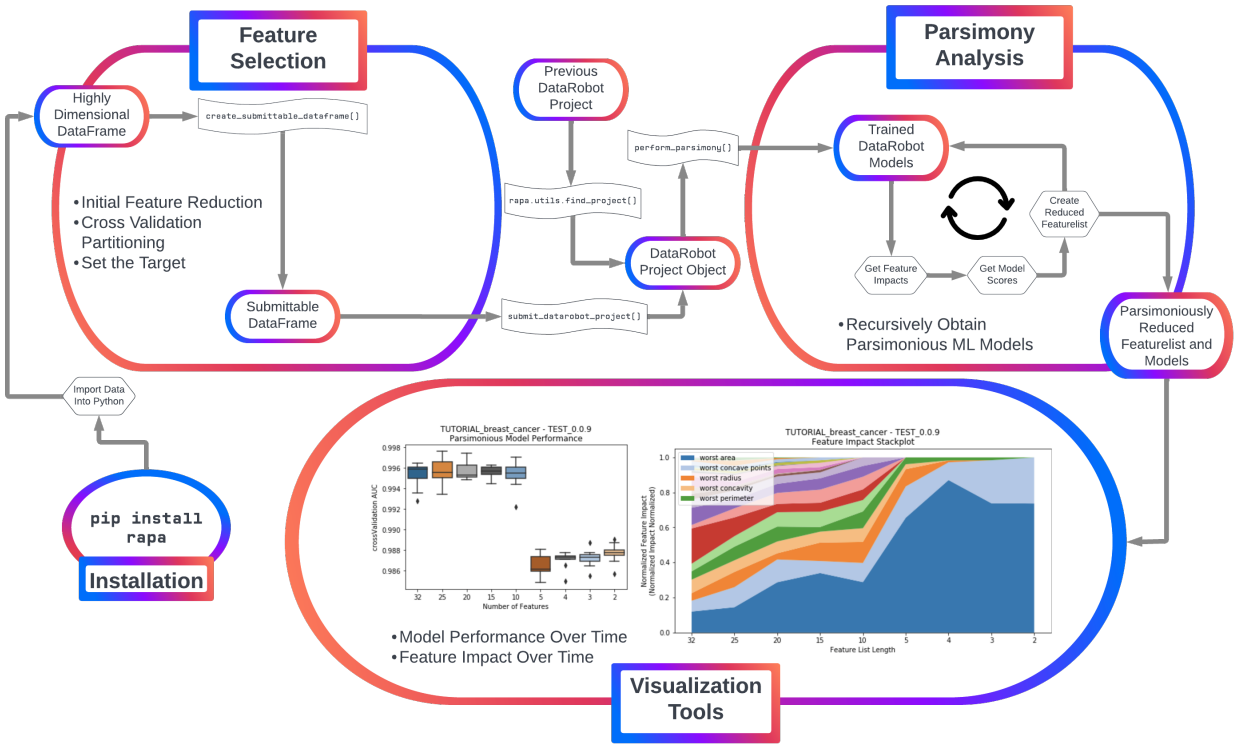
CONTENTS:

CHAPTER

ONE

ROBUST AUTOMATED PARSIMONY ANALYSIS (RAPA)

`rapa` provides a robust, freely usable and shareable tool for creating and analyzing more accurate machine learning (ML) models with fewer features in Python. View documentation on ReadTheDocs.



`rapa` is currently developed on top of DataRobot's Python API to use DataRobot as a "model-running engine", with plans to include open source software such as `scikit-learn`, `tensorflow`, or `pytorch` in the future. Install using pip!

- Getting Started
- Primary Features
  - Initial Feature Filtering
    - With Previous DataRobot Project
    - Submitting a New Project With RAPA
  - Automated Parsimony Analysis

* Visualization

1.1 Getting Started

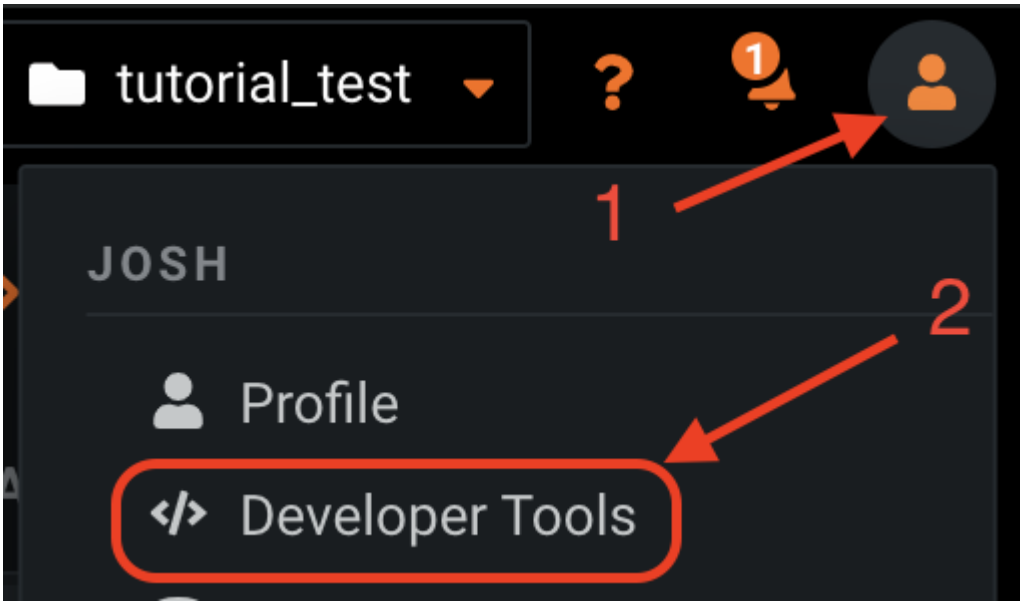
1.1.1 Installation

```
pip install rapa
```

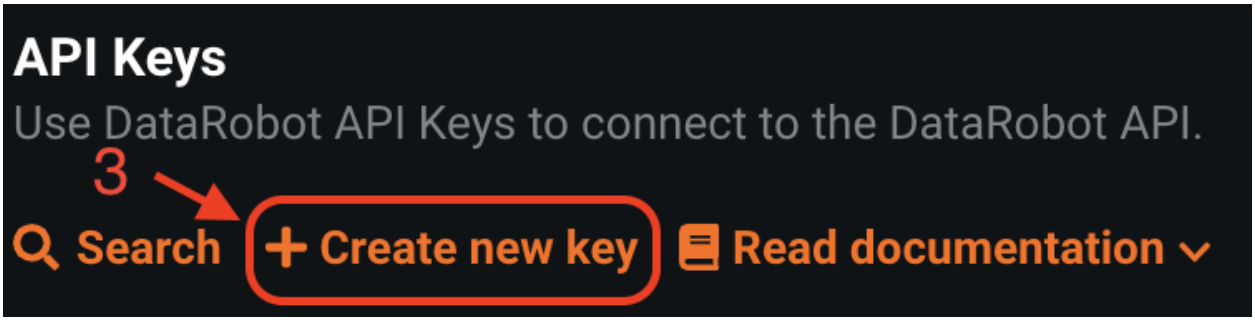
1.1.2 Initializing the DataRobot API

Majority of `rapa`'s utility comes from the DataRobot auto-ML platform. To utilize DataRobot through Python, an API key is required. Acquire an API key from app.datarobot.com after logging into an account. (More information about DataRobot's API keys)

First, log in and find the developer tools tab.



Then create an API key for access to the API with Python.



This API key lets anyone who has it access your DataRobot projects, so never share it with anyone.

To avoid sharing your API accidentally by uploading a notebook to github, it is suggested to use the `rapa` function to read in a pickled dictionary for the API key or using `datarobot`'s configuration for authentication.

Once having obtained an API key, use `rapa` or `datarobot` to initialize the API connection.

Using `rapa`, first create the pickled dictionary containting an API key.

```
# DO NOT UPLOAD THIS CODE WITH THE API KEY FILLED OUT
# save a pickled dictionary for datarobot api initialization in a new folder named 'data'
import os
import pickle

api_dict = {'tutorial':'APIKEYHERE'}
if 'data' in os.listdir('.'):
    print('data folder already exists, skipping folder creation...')
else:
    print('Creating data folder in the current directory.')
    os.mkdir('data')

if 'dr-tokens.pkl' in os.listdir('data'):
    print('dr-tokens.pkl already exists.')
else:
    with open('data/dr-tokens.pkl', 'wb') as handle:
        pickle.dump(api_dict, handle)
```

Then use `rapa` to initialize the API connection!

```
# Use the pickled dictionary to initialize the DataRobot API
import rapa

rapa.utils.initialize_dr_api('tutorial')
```

`rapa.utils.initialize_dr_api` takes 3 arguments: token_key - the dictionary key used to store the API key as a value, file_path - the pickled dataframe file path (default: data/dr-tokens.pkl), endpoint - and the endpoint (default:https://app.datarobot.com/api/v2).

1.2 Primary Features

Currently, `rapa` provides two primary features:

1. Initial feature filtering to reduce a feature list down to a size that DataRobot can receive as input.

2. Automated parsimony analysis using feature importance metrics directly tied to the feature's impact on accurate models (permutation importance).

1.3 Initial Feature Filtering

Automated machine learning is easily applicable to samples with fewer features, as the time and resources required reduces significantly as the number of initial features decreases. Additionally, DataRobot's automated ML platform only accepts projects that have up to 20,000 features per sample.

For feature selection, `rapa` uses `sklearn`'s `f_classif` or `f_regression` to reduce the number of features. This provides an ANOVA F-statistic for each sample, which is then used to select the features with the highest F-statistics.

```
# first, create a rapa classification object
rapa_classif = rapa.Project.Classification()

# then provide the original data for feature selection
sdf = rapa_classif.create_submittable_dataframe(input_data_df=input,
                                                target_name='target_column',
                                                n_features=2000)
```

When calling `create_submittable_dataframe`, the provided `input_data_df` should have all of the features as well as the target as columns, and samples as the index.

If the number of features is reduced, then there should be no missing values.

1.4 Automated Parsimony Analysis

To start automated parsimony analysis using Datarobot, a DataRobot project with a target and uploaded data must already be created.

- Use an existing project
- Create a new project using `rapa`

1.4.1 Use a previously created DataRobot project:

To use a previously created DataRobot project, you must have access to the project with the account that provided the API key.

- First, initialize the API connection with an API key that provides access to the project of interest.

```
rapa.utils.initialize_dr_api('tutorial')
```

- Then, provide either a **project id** or unique **project name** to `rapa.utils.find_project` and get a `datarobot.models.Project` object for further analysis.

```
project = rapa.utils.find_project('PROJECT_OF_INTEREST')
```

1.4.2 Create and submit data for a new DataRobot project using `rapa`:

When creating a new DataRobot project, the API key used should be from an account which the project will be created. Additionally, the data for training will be submitted, and the target will be provided and selected with the API.

- First, initialize the API connection with an API key that provides access to the account where the project will be created.

```
rapa.utils.initialize_dr_api('tutorial')
```

- Load the data for machine learning using `pandas`

```
# load data (make sure features are columns, and samples are rows)

from sklearn import datasets # data used in this tutorial
import pandas as pd # used for easy data management

# loads the dataset (as a dictionary)
breast_cancer_dataset = datasets.load_breast_cancer()

# puts features and targets from the dataset into a dataframe
breast_cancer_df = pd.DataFrame(data=breast_cancer_dataset['data'], columns=breast_
↪cancer_dataset['feature_names'])
breast_cancer_df['benign'] = breast_cancer_dataset['target']
```

- Create a `rapa` object for either classification or regression (this example is a classification problem)

```
# Creates a rapa classifcation object
bc_classification = rapa.Project.Classification()
```

- Make a DataRobot submittable dataframe using `create_submittable_dataframe`

```
# creates a datarobot submittable dataframe with cross validation folds stratified for␣
↪the target (benign)
sub_df = bc_classification.create_submittable_dataframe(breast_cancer_df, target_name=
↪'benign')
```

`rapa`'s `create_submittable_dataframe` takes the number of features to initially filter to.

If filtering features, either the `sklearn` function `sklearn.feature_selection.f_classif` or `sklearn.feature_selection.f_regression` is used depending on the `rapa` instance that is called. In the case of this example, the function is being called by a `Project.Classification` object, so `f_classif` will be used.

Additionally, `create_submittable_dataframe` can take a random state as an argument. When changing the random state, the features that are filtered can sometimes change drastically. This is because the average ANOVA F score over the cross-validation folds is calculated for selecting the features, and the random state changes which samples are in each cross-validation fold.

- Finally, submit the 'submittable dataframe' to DataRobot as a project

```
# submits a project to datarobot using our dataframe, target, and project name.
project = bc_classification.submit_datarobot_project(input_data_df=sub_df, target_name=
↪'benign', project_name='TUTORIAL_breast_cancer')
```

This will run DataRobot's autopilot feature on the data submitted.

1.4.3 After obtaining a DataRobot Project

Once a DataRobot project object is loaded into Python, the parsimonious model analysis can begin.

Using an initialized `rapa` object (`Project.Classification` or `Project.Regression`), call the `perform_parsimony` function. This function returns None.

```
# perform parsimony on the breast-cancer classification data
# use a featurelist prefix `TEST`
# start with the `Informative Features` featurelist provided by datarobot
# use a feature range starting with 25 features, down to 1
# have 5 `lives`, so if the models do not become more accurate, it will stop feature␣
↪reduction
# try and reduce overfitting with a cross-validation average mean error limit of 0.8
# graph feature performance over time, as well as model performance
bc_classification.perform_parsimony(project=project,
                                    featurelist_prefix='TEST',
                                    starting_featurelist_name='Informative Features',
                                    feature_range=[25, 20, 15, 10, 5, 4, 3, 2, 1],
                                    lives=5,
                                    cv_average_mean_error_limit=.8,
                                    to_graph=['feature_performance', 'models'])
```

While running `perform_parsimony`, `rapa` is checking job status with DataRobot. This is displayed to the user as printed statements while running the function. Additionally, if the `progress_bar` argument is True, the `tqdm` progress bar will display updates in text.

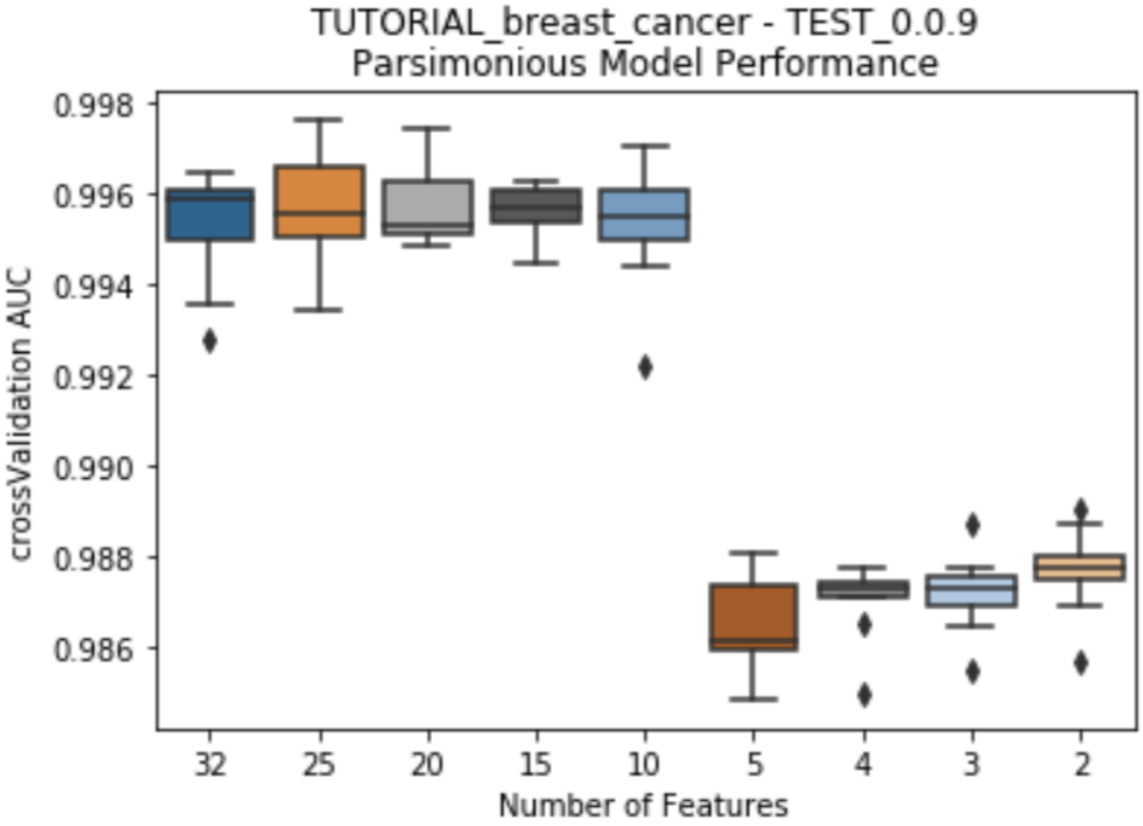
The `perform_parsimony` function takes, at minimum, a list of desired featurelist sizes (`feature_range`) and a DataRobot project (`project`). Additional arguments allow for choosing the featurelist to begin parsimonious feature reduction (`starting_featurelist`), what prefix to use for `rapa` reduced featurelists (`featurelist_prefix`), what metric to use for deciding the 'best' models (`metric`), which visuals to present (`to_graph`), etc. To get in-depth descriptions of each argument, visit the documentation for `perform_parsimony`.

1.5 Visualization

1.5.1 Model Performance

To present to the user the trade-off between the size of Feature List and the model performance for each Feature List, a series of boxplots can be plotted. The y-axis uses the chosen measurment of accuracy for the models (AUC, R-squared, etc.), while the x-axis has the featurelist sizes decreasing from left to right. Choose to plot either after each feature reduction during parsimony analysis (provide the argument `to_graph=['models']` to `perform_parsimony`), or use the function `rapa.utils.parsimony_performance_boxplot` and provide a project and the featurelist prefix used.

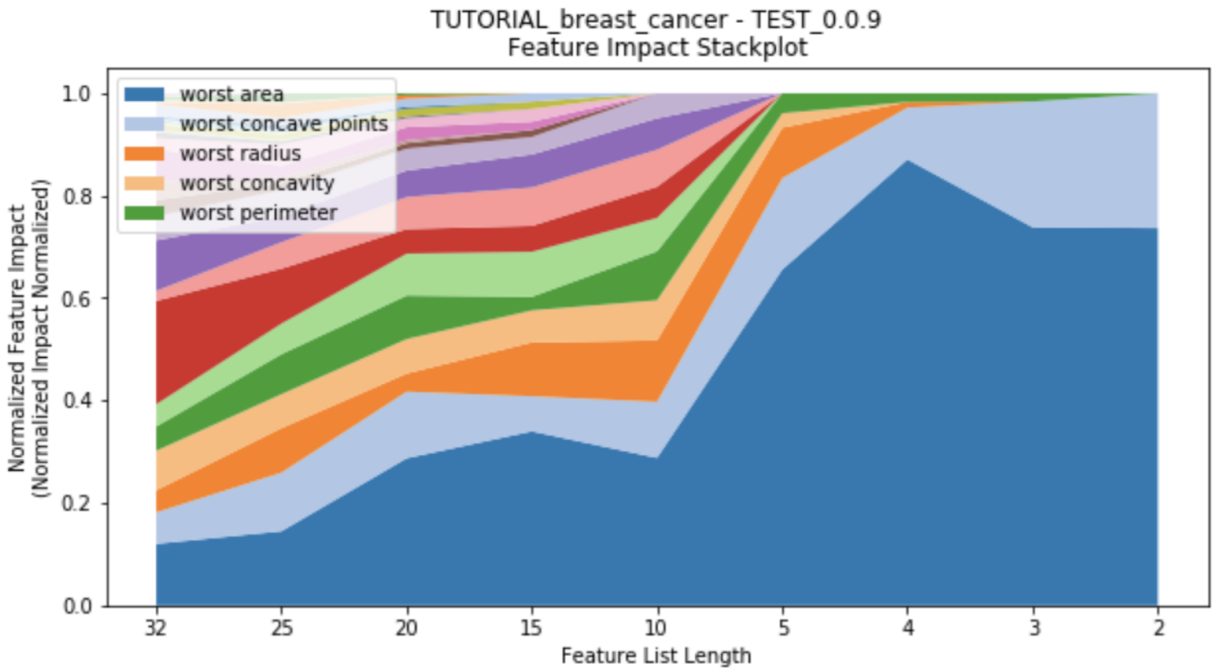
```
rapa.utils.parsimony_performance_boxplot(project=project,
                                         starting_featurelist='Informative Features')
```



1.5.2 Feature Impact Evolution

While the number of features decreases, each feature's impact changes as well. Features which had previously had high impact on the models with many other features may no longer have significance once more features are removed. This suggests towards the multi-variate nature of feature impact and it's ability to create parsimonious models. A stackplot using height in the y-axis to represent impact provides insight into the evolution of each feature's impact as the number of features decreases. Choose to plot either after each feature reduction during parsimony analysis (provide the argument `to_graph=['feature_performance']` to `perform_parsimony`), or use the function `rapa.utils.feature_performance_stackplot` and provide a project and the featurelist prefix used.

```
rapa.utils.feature_performance_stackplot(project=project,
                                         starting_featurelist='Informative Features')
```



1.6 Additional Tutorial

In addition to this readme, there is a tutorial for using `rapa` with DataRobot and readily available data from `sklearn` that is currently demonstrated in general_tutorial.ipynb, which is also in the documentation.

1.7 Plans

Although the current implementation of these features will be based on basic techniques such as linear feature filters and recursive feature elimination, we plan to rapidly improve these features by integrating state-of-the-art techniques from the academic literature.

CHAPTER

TWO

API REFERENCE

2.1 Classes

class rapa.Project.**Classification**(*project: Optional[Project] = None*)

Bases: *RAPABase*

RAPA class meant for classification problems.

class rapa.Project.**Regression**(*project: Optional[Project] = None*)

Bases: *RAPABase*

RAPA class meant for regression problems.

class rapa.base.**RAPABase**

Bases: object

The base of regression and classification RAPA analysis

POSSIBLE_TARGET_TYPES = ['ALL', 'ANOMALY', 'BINARY', 'MULTICLASS', 'MULTILABEL', 'REGRESSION', 'UNSTRUCTURED']

- _classification = None # Set by child classes
- target_type = None # Set at initialization
- project = None # Set at initialization or with 'perform_parsimony()'

create_submittable_dataframe(*input_data_df: DataFrame*, *target_name: str*, *n_features: int = 19990*, *n_splits: int = 6*, *filter_function: Optional[Callable[[DataFrame, ndarray], List[ndarray]]] = None*, *random_state: Optional[int] = None*) → DataFrame

Prepares the input data for submission as either a regression or classification problem on DataRobot.

Creates pre-determined k-fold cross-validation splits and filters the feature set down to a size that DataRobot can receive as input, if necessary. TODO: private function submit_datarobot_project explanation

Parameters

target_name: str
Name of the prediction target column in *input_data_df*.

n_features: int, optional (default: 19990)
The number of features to reduce the feature set in *input_data_df* down to. DataRobot's maximum feature set size is 20,000. If *n_features* has the same number of features as the *input_data_df*, NaN values are allowed because no feature filtering will ocurr

n_splits: int, optional (default: 6)
The number of cross-validation splits to create. One of the splits will be retained as a holdout split, so by default this function sets up the dataset for 5-fold cross-validation with a holdout. NOTE: *CV Fold 0* is the holdout set by default.

filter_function: callable, optional (default: None)
The function used to calculate the importance of each feature in the initial filtering step that reduces the feature set down to *max_features*.

This filter function must take a feature matrix as the first input and the target array as the second input, then return two separate arrays containing the feature importance of each feature and the P-value for that correlation, in that order.

When None, the filter function is determined by child class. If an instance of *RAPAClassif()*, sklearn.feature_selection.f_classif is used. If *RAPARegress()*, sklearn.feature_selection.f_regression is used. See scikit-learn's f_classif function for an example: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html

random_state: int, optional (default: None)
The random number generator seed for RAPA. Use this parameter to make sure that RAPA will give you the same results each time you run it on the same input data set with that seed.

Returns

pre-determined k-fold cross-validation splits, and was filtered down to 'max_features' size using the 'filter_function'

perform_parsimony(*feature_range: List[Union[float, int]]*, *project: Optional[Union[Project, str]] = None*, *starting_featurelist_name: str = 'Informative Features'*, *featurelist_prefix: str = 'RAPA Reduced to'*, *mode: str = 'auto'*, *lives: Optional[int] = None*, *cv_average_mean_error_limit: Optional[float] = None*, *feature_impact_metric: str = 'median'*, *progress_bar: bool = True*, *to_graph: Optional[List[str]] = None*, *metric: Optional[str] = None*, *verbose: bool = True*)

Performs parsimony analysis by repetatively extracting feature importance from DataRobot models and creating new models with reduced features (smaller feature lists). # TODO take a look at featurelist_prefix for running multiple RAPA

NOTICE: Feature impact scores are only gathered from models that have had their **cross-validation accuracy** tested!

Parameters

or a list containing floats representing desired featurelist percentages (of the original featurelist size)

project: datarobot.Project | str, optional (default = None)
Either a datarobot project, or a string of it's id or name. If None, uses the project that was provided to create the rapa class

starting_featurelist: str, optional (default = 'Informative Features')
The name or id of the featurelist that rapa will start pasimony analysis with

featurelist_prefix: str, optional (default = 'RAPA Reduced to')
The desired prefix for the featurelists that rapa creates in datarobot. Each featurelist will start with the prefix, include a space, and then end with the number of features in that featurelist

mode: str (enum), optional (default: datarobot.AUTOPILOT_MODE.FULL_AUTO)
The modeling mode to start the DataRobot project in. Options:

- datarobot.AUTOPILOT_MODE.FULL_AUTO
- datarobot.AUTOPILOT_MODE.QUICK
- datarobot.AUTOPILOT_MODE.MANUAL
- datarobot.AUTOPILOT_MODE.COMPREHENSIVE: Runs all blueprints in the repository (warning: this may be extremely slow).

lives: int, optional (default = None)
The number of times allowed for reducing the featurelist and obtaining a worse model. By default, 'lives' are off, and the entire 'feature_range' will be ran, but if supplied a number >= 0, then that is the number of 'lives' there are.

Ex: lives = 0, feature_range = [100, 90, 80, 50] RAPA finds that after making all the models for the length 80 featurelist, the 'best' model was created with the length 90 featurelist, so it stops and doesn't make a featurelist of length 50.

Similar to datarobot's Feature Importance Rank Ensembling for advanced feature selection (FIRE) package's 'lifes' https://www.datarobot.com/blog/using-feature-importance-rank-ensembling-fire-for-advanced-feature-selection/

cv_average_mean_error_limit: float, optional (default = None)
The limit of cross validation mean error to help avoid overfitting. By default, the limit is off, and the each 'feature_range' will be ran. Limit exists only if supplied a number >= 0.0

Ex: 'feature_range' = 2.5, feature_range = [100, 90, 80, 50]
RAPA finds that the average AUC for each CV fold is [.8, .6, .9, .5] respectfully, the mean of these is 0.7. The average error is += 0.15. If 0.15 >= cv_average_mean_error_limit, the training stops.

feature_impact_metric: str, optional (default = 'median')

How RAPA will decide each feature's importance over every model in a feature list
Options: * median * mean * cumulative

progress_bar: bool, optional (default = True)
If True, a simple progres bar displaying complete and incomplete featurelists. If False, provides updates in stdout Ex: current worker count, current featurelist, etc.

to_graph: List[str], optional (default = None)

A list of keys choosing which graphs to produce. Possible Keys:

- 'models': *seaborn* boxplot with model performances with provided metric
- 'feature_performance': *matplotlib.pyplot* stackplot of feature performances

metric: str, optional (default = None)
The metric used for scoring models, when finding the 'best' model, and when plotting model performance

When None, the metric is determined by what class inherits from base. For instance, a *RAPAClassif* instance's default is 'AUC', and *RAPARegress* is 'R Squared'

verbose: bool, optional (default = True)
If True, prints updates from DataRobot and rapa during parsimonious feature rduction

Returns

`submit_datarobot_project`(*input_data_df: DataFrame*, *target_name: str*, *project_name: str*, *target_type: Optional[str] = None*, *worker_count: int = -1*, *metric: Optional[str] = None*, *mode: str = 'auto'*, *random_state: Optional[int] = None*) → Project

Submits the input data to DataRobot as a new modeling project.

It is suggested to prepare the *input_data_df* using the 'create_submittable_dataframe' function first with an instance of either RAPAClassif or RAPARegress.

Parameters

target_name: str
Name of the prediction target column in *input_data_df*.

project_name: str
Name of the project in DataRobot.

target_type: str (enum)
Indicator to DataRobot of whether the new modeling project should be a binary classification, multiclass classification, or regression project.

Options:

- datarobot.TARGET_TYPE.BINARY
- datarobot.TARGET_TYPE.REGRESSION
- datarobot.TARGET_TYPE.MULTICLASS

worker_count: int, optional (default: -1)
The number of worker engines to assign to the DataRobot project. By default, -1 tells DataRobot to use all available worker engines.

metric: str, optional (default: None)
Name of the metric to use for evaluating models. You can query the metrics available for the target by way of Project.get_metrics. If none is specified, then the default recommended by DataRobot is used.

mode: str (enum), optional (default: datarobot.AUTOPILOT_MODE.FULL_AUTO)
The modeling mode to start the DataRobot project in.

Options:

- datarobot.AUTOPILOT_MODE.FULL_AUTO
- datarobot.AUTOPILOT_MODE.QUICK
- datarobot.AUTOPILOT_MODE.MANUAL
- datarobot.AUTOPILOT_MODE.COMPREHENSIVE: Runs all blueprints in the repository (this may be extremely slow).

random_state: int, optional (default: None)
The random number generator seed for DataRobot. Use this parameter to make sure that DataRobot will give you the same results each time you run it on the same input data set with that seed.

Returns

2.2 Utility Functions

rapa.utils.**feature_performance_stackplot**(*project: Project*, *featurelist_prefix: str = 'RAPA Reduced to'*, *starting_featurelist: Optional[str] = None*, *feature_impact_metric: str = 'median'*, *metric: Optional[str] = None*, *vlines: bool = False*)

Utilizes *matplotlib.pyplot.stackplot* to show feature performance during parsimony analysis.

Parameters

featurelist_prefix: str, optional (default = 'RAPA Reduced to')
The desired prefix for the featurelists that will be used for plotting feature performance. Each featurelist will start with the prefix, include a space, and then end with the number of features in that featurelist

starting_featurelist: str, optional (default = None)
The starting featurelist used for parsimony analysis. If None, only the featurelists with the desired prefix in *featurelist_prefix* will be plotted

feature_impact_metric: str, optional (default = mean)
Which metric to use when finding the most representative feature importance of all models in the featurelist

Options:

- median
- mean
- cumulative

metric: str, optional (default = 'AUC' or 'RMSE') [classification and regression]
Which metric to use when finding feature importance of each model

vlines: bool, optional (default = False)
Whether to add vertical lines at the featurelist lengths or not, False by default

Returns

rapa.utils.**find_project**(*project: str*) → Project

Uses the DataRobot api to find a current project.

Uses datarobot.Project.get() and dr.Project.list() to test if 'project' is either an id or possibly a name of a project in DataRobot, then returns the project found.

Parameters

Returns

first/only project returned by searching by project name. Returns None if the list is empty.

rapa.utils.**get_best_model**(*project: Project*, *featurelist_prefix: Optional[str] = None*, *starred: bool = False*, *metric: Optional[str] = None*, *fold: str = 'crossValidation'*, *highest: Optional[bool] = None*) → Model

Attempts to find the 'best' model in a datarobot by searching cross validation scores of all the models in a supplied project. # TODO make dictionary for minimize/maximize

CURRENTLY SUPPORTS METRICS WHERE HIGHER = BETTER

Warning: Actually finding the 'best' model takes more than averageing cross validation scores, and it is suggested that the 'best' model is decided and starred in DataRobot. (Make sure 'starred = True' if starring the 'best' model)

Note: Some models may not have scores for the supplied fold because they were not run. These models are ignored by this function. Make sure all models of interest have scores for the fold being provided if those models should be considered.

Parameters

featurelist_prefix: str, optional (default = None)
The desired featurelist prefix used to search in for models using specific rapa featurelists

starred: bool, optional (default = False)
If True, return the starred model. If there are more than one starred models, then warn the user and return the 'best' one

metric: str, optional (default = 'AUC' or 'RMSE') [classification and regression]
What model metric to use when finding the 'best'

fold: str, optional (default = 'crossValidation')

The fold of data used in DataRobot. Options are as follows:
['validation', 'crossValidation', 'holdout', 'training', 'backtestingScores', 'backtesting']

highest: bool, optional (default for classification = True, default for regression = False)
Whether to take the highest value (highest = True), or the lowest value (highest = False). Change this when assumed switch is

Returns

from the provided datarobot project

rapa.utils.**get_featurelist**(*featurelist: str*, *project: Project*) → Featurelist

Uses the DataRobot api to search for a desired featurelist.

Uses datarobot.Project.get_featurelists() to retrieve all the featurelists in the project. Then, it searches the list for id's, and if it doesn't find any, it searches the list again for names. Returns the first project it finds.

Parameters

project: datarobot.Project
The project that is being searched for the featurelist

Returns

rapa.utils.**get_starred_model**(*project: Project*, *metric: Optional[str] = None*, *featurelist_prefix: Optional[str] = None*) → Model

Alias for rapa.utils.get_best_model() but makes starred = True

rapa.utils.**initialize_dr_api**(*token_key: Optional[str] = None*, *file_path: str = 'data/dr-tokens.pkl'*, *endpoint: str = 'https://app.datarobot.com/api/v2'*)

Initializes the DataRobot API with a pickled dictionary created by the user.

Accesses a file that should be a pickled dictionary. This dictionary has the API token as the value to the provided token_key. Ex: {token_key: 'API_TOKEN'}

Parameters

file_path: str, optional (default = 'data/dr-tokens.pkl')
Path to the pickled dictionary containing the API token

endpoint: str, optional (default = 'https://app.datarobot.com/api/v2')
The endpoint is usually the URL you would use to log into the DataRobot Web User Interface

rapa.utils.**parsimony_performance_boxplot**(*project: Project*, *featurelist_prefix: str = 'RAPA Reduced to'*, *starting_featurelist: Optional[str] = None*, *metric: Optional[str] = None*, *split: str = 'crossValidation'*, *featurelist_lengths: Optional[list] = None*)

Uses *seaborn*'s *boxplot* function to plot featurelist size vs performance for all models that use that featurelist prefix. There is a different boxplot for each featurelist length. # TODO warn about multiple prefixes, try to use new prefixes

Paremeters

featurelist_prefix: str, optional (default = 'RAPA Reduced to')
The desired prefix for the featurelists that will be used for plotting parsimony performance. Each featurelist will start with the prefix, include a space, and then end with the number of features in that featurelist

starting_featurelist: str, optional (default = None)
The starting featurelist used for parsimony analysis. If None, only the featurelists with the desired prefix in *featurelist_prefix* will be plotted

metric: str, optional (default = 'AUC' or 'RMSE') [classification and regression]
The metric used for plotting accuracy of models

split: str, optional (default = 'crossValidation')
What split's performance to take from. Can be: ['crossValidation', 'holdout'] TODO: i think it can be more, double check

featurelist_lengths: list, optional (default = None)
A list of featurelist lengths to plot

Returns

CHAPTER

THREE

RAPA WALKTHROUGH

This tutorial is meant to cover multiple RAPA use cases, including starting from scratch or using a previous DataRobot project.

3.1 Overview

1. Initialize the DataRobot API
    - Save a pickled dictionary for DataRobot API Initialization
    - Use the pickled dictionary to initialize the DataRobot API
    - **(Optional)**: Skip this if the DataRobot API is previously initialized
2. Submit data as a project to DataRobot
    - Create a submittable `pandas` dataframe
    - Submit the data using RAPA
    - **(Optional)**: If parsimonious feature reduction is required on an existing project, it is possible to load the project instead of creating a new one.
3. Perform parsimonious feature reduction

[Link to the documentation]

[1]:
```
import rapa
print(rapa.version.__version__)
```

```
0.1.0
```

3.1.1 1. Initialize DataRobot the API

[2]:
```
import pickle
import os
```

On the DataRobot website , find the developer tools and retrieve an API key. Once you have a key, make sure to run the next block of code with your api key replacing the value in the dictionary. Here is a detailed article on DataRobot API keys.

Make sure to remove code creating the pickled dataframe and the pickled dataframe itself from any public documents, such as GitHub.

[3]:
```
# save a pickled dictionary for datarobot api initialization
api_dict = {'tutorial':'APIKEYHERE'}
if 'data' in os.listdir('.'):
    print('data folder already exists, skipping folder creation...')
else:
    print('Creating data folder in the current directory.')
    os.mkdir('data')

if 'dr-tokens.pkl' in os.listdir('data'):
    print('dr-tokens.pkl already exists.')
else:
    with open('data/dr-tokens.pkl', 'wb') as handle:
        pickle.dump(api_dict, handle)
```

```
data folder already exists, skipping folder creation...
dr-tokens.pkl already exists.
```

[4]:
```
# Use the pickled dictionary to initialize the DataRobot API
rapa.utils.initialize_dr_api('tutorial')
```

```
DataRobot API initiated with endpoint 'https://app.datarobot.com/api/v2'
```

The majority of this tutorial uses the DataRobot API, so if the API is not initialized, it will not run.

3.1.2 2. Submit data as a project to DataRobot

This tutorial uses the Breast cancer wisconsin (diagnostic) dataset as an easily accessible example set for `rapa`, as it easily loaded with `sklearn`.

This breast cancer dataset has **30 features** extracted from digitized images of aspirated breast mass cells. A few features are the mean radius of the cells, the mean texture, mean perimater The **target** is whether the cells are from a malignant or benign tumor, with 1 indicating benign and 0 indicating malignant. There are 357 benign and 212 malignant samples, making 569 samples total.

[5]:
```
from sklearn import datasets # data used in this tutorial
import pandas as pd # used for easy data management
```

[6]:
```
# loads the dataset (as a dictionary)
breast_cancer_dataset = datasets.load_breast_cancer()
```

[7]:
```
# puts features and targets from the dataset into a dataframe
breast_cancer_df = pd.DataFrame(data=breast_cancer_dataset['data'], columns=breast_
↪cancer_dataset['feature_names'])
breast_cancer_df['benign'] = breast_cancer_dataset['target']
print(breast_cancer_df.shape)
breast_cancer_df.head()
```

```
(569, 31)
```

[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	

(continues on next page)

(continued from previous page)

```
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  benign
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

When using `rapa` to create a project on DataRobot, the number of features is reduced using on of the sklearn functions `sklearn.feature_selection.f_classif`, or `sklearn.feature_selection.f_regress` depending on the `rapa` instance that is called. In this tutorial's case, the data is a binary classification problem, so we have to create an instance of the `Project.Classification` class.

As of now, `rapa` only supports classification and regression problems on DataRobot. Additionally, `rapa` has only been tested on tabular data.

[8]:
```
# Creates a rapa classifcation object
bc_classification = rapa.Project.Classification()
```

[9]:
```
# creates a datarobot submittable dataframe with cross validation folds stratified for␣
↪the target (benign)
sub_df = bc_classification.create_submittable_dataframe(breast_cancer_df, target_name=
↪'benign')
print(sub_df.shape)
sub_df.head()
```

```
(569, 32)
```

[9]:
```
   benign  partition  mean radius  mean texture  mean perimeter  mean area  \
0       0  CV Fold 4        17.99         10.38          122.80     1001.0
1       0  CV Fold 3        20.57         17.77          132.90     1326.0
2       0  CV Fold 1        19.69         21.25          130.00     1203.0
3       0  CV Fold 0        11.42         20.38           77.58      386.1
4       0  CV Fold 2        20.29         14.34          135.10     1297.0

   mean smoothness  mean compactness  mean concavity  mean concave points  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  worst radius  worst texture  worst perimeter  worst area  \
0  ...         25.38          17.33           184.60      2019.0
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

[10]:
```
# submits a project to datarobot using our dataframe, target, and project name.
project = bc_classification.submit_datarobot_project(input_data_df=sub_df, target_name=
↪'benign', project_name='TUTORIAL_breast_cancer')
project
```

[10]:
```
Project(TUTORIAL_breast_cancer)
```

[11]:
```
# if the project already exists, the `rapa.utils.find_project` function can be used to␣
↪search for a project
project = rapa.utils.find_project("TUTORIAL_breast_cancer")
project
```

[11]:
```
Project(TUTORIAL_breast_cancer)
```

3.1.3 3. Perform parsimonious feature reduction

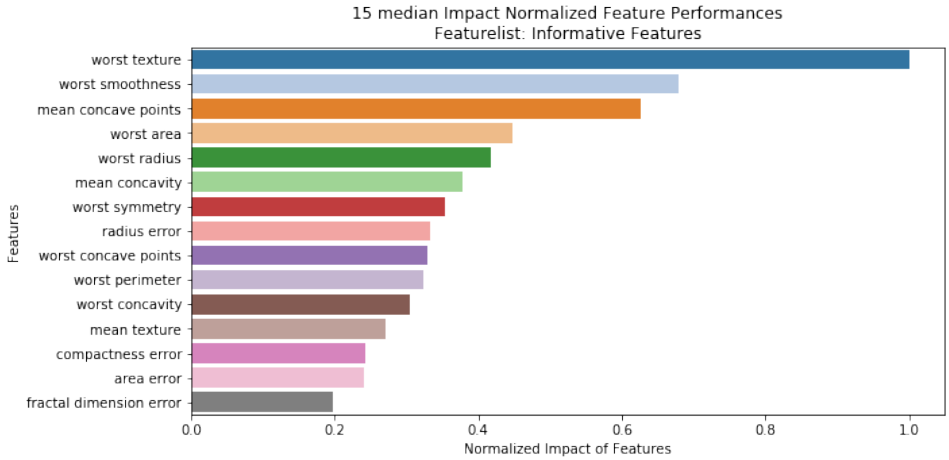
`rapa`'s main function is `perform_parsimony`. Requiring a *feature_range* and a *project*, this function recursively removes features by their relative feature impact scores across all models in a featurelist, creating a new featurelist and set of models with DataRobot each iteration.

- **feature_range:** a list of desired featurelist lengths as integers (Ex: [25, 20, 15, 10, 5, 4, 3, 2, 1]), or of desired featurelist sizes (Ex: [0.9, 0.7, 0.5, 0.3, 0.1]). This tells `rapa` how many features remain after each iteration of feature reduction.
- **project:** either a `datarobot` project, or a string of it's id or name. `rapa.utils.find_project` can be used to find a project already existing in DataRobot, or `submit_datarobot_project` can be used to submit a new project.
- **featurelist_prefix:** provides `datarobot` with a prefix that will be used for all the featurelists created by the `perform_parsimony` function. If running `rapa` multiple times in one DataRobot project, make sure to change the **featurelist_prefix** each time to avoid confusion.
- **starting_featurelist_name:** the name of the featurelist you would like to start parsimonious reduction from. It defaults to 'Informative Features', but can be changed to any featurelist name that exists within the project.
- **lives:** number of times allowed for reducing the featurelist and obtaining a worse model. By default, 'lives' are off, and the entire 'feature_range' will be ran, but if supplied a number >= 0, then that is the number of 'lives' there are. (Ex: lives = 0, feature_range = [100, 90, 80, 50] RAPA finds that after making all the models for the length 80 featurelist, the 'best' model was created with the length 90 featurelist, so it stops and doesn't make a featurelist of length 50.) This is similar to DataRobot's Feature Importance Rank Ensembling for advanced feature selection (FIRE) package's 'lifes'.
- **cv_average_mean_error_limit**: limit of cross validation mean error to help avoid overfitting. By default, the limit is off, and the each 'feature_range' will be ran. Limit exists only if supplied a number >= 0.0.
- **to_graph**: a list of keys choosing which graphs to produce. Current graphs are *feature_performance* and *models*. *feature_performance* graphs a stackplot of feature impacts across many featurelists, showing the change in impact over different featurelist lengths. *models* plots `seaborn` boxplots of some metric of accuracy for each featurelist length. These plots are created after each iteration.

Additional arguments and their effects can be found in the API documentation, or within the functions.

[12]:
```
bc_classification.perform_parsimony(project=project,
                                    featurelist_prefix='TEST_' + str(rapa.version.__
↪version__),
                                    starting_featurelist_name='Informative Features',
                                    feature_range=[25, 20, 15, 10, 5, 4, 3, 2, 1],
                                    lives=5,
                                    cv_average_mean_error_limit=.8,
                                    to_graph=['feature_performance', 'models'])
```

```
---------- Informative Features (30) ----------
Informative Features: Waiting for previous jobs to complete...
Previous job(s) remaining (0))
Informative Features: Waiting for feature impact...
Feature Impact job(s) remaining (0))
Feature Impact: (105.26s)
Graphing feature performance...
```

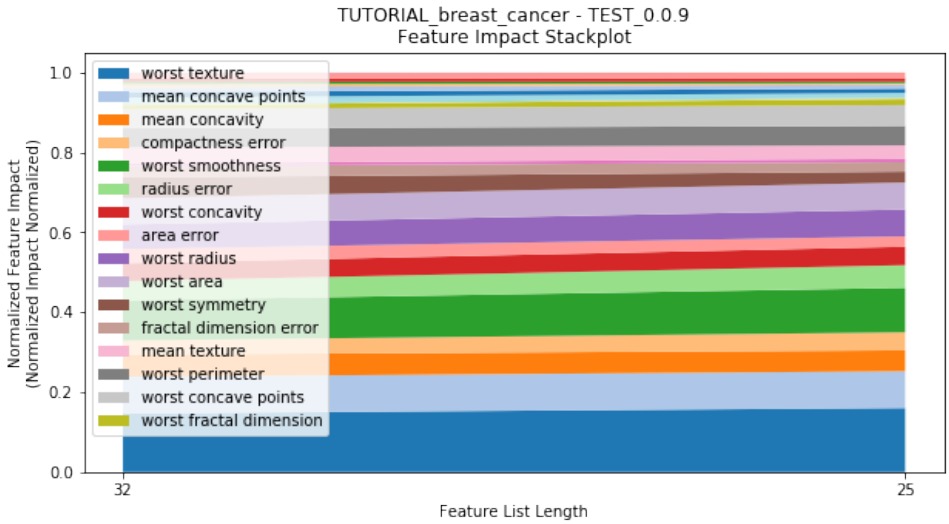


nbsphinx-code-borderwhite

```
DataRobot job(s) remaining (0)
Project: TUTORIAL_breast_cancer | Featurelist Prefix: TEST_0.0.9 | Feature Range: [25,␣
↪20, 15, 10, 5, 4, 3, 2, 1]
Feature Importance Metric: median | Model Performance Metric: AUC
Lives: 5
CV Mean Error Limit: 0.8
```

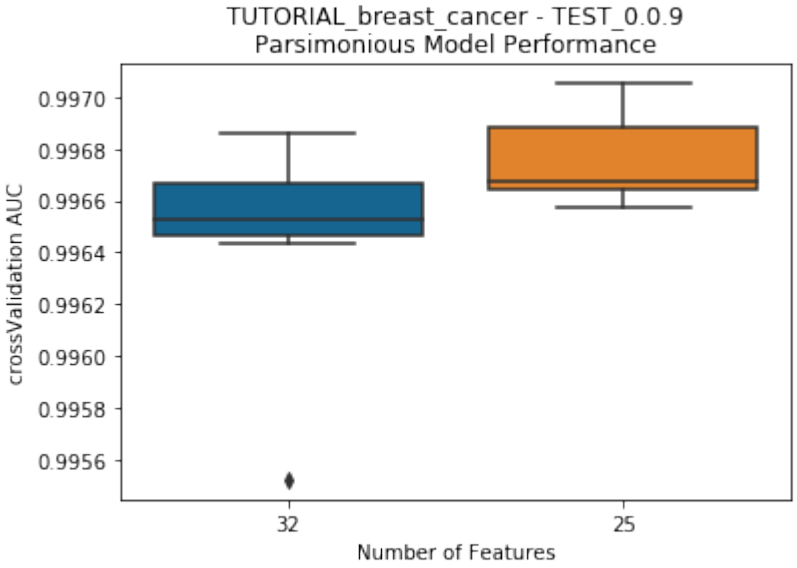
```
  0%|          | 0/9 [00:00<?, ?it/s]
```

```
---------- TEST_0.0.9 (25) ----------
Autopilot: 249.98s
Feature Impact job(s) remaining (0)
Feature Impact: 152.84s
Waiting for DataRobot: 10.71s
```



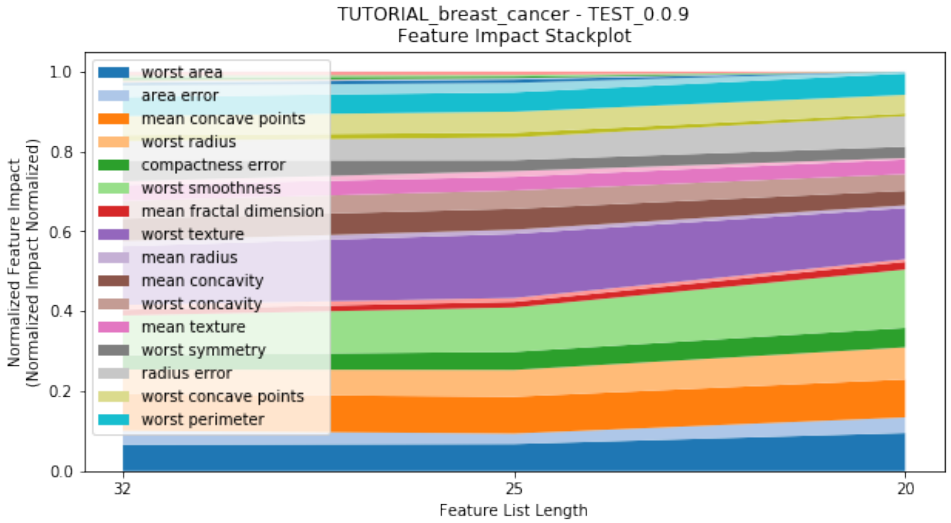
nbsphinx-code-borderwhite

```
Performance Stackplot: 11.99s
```



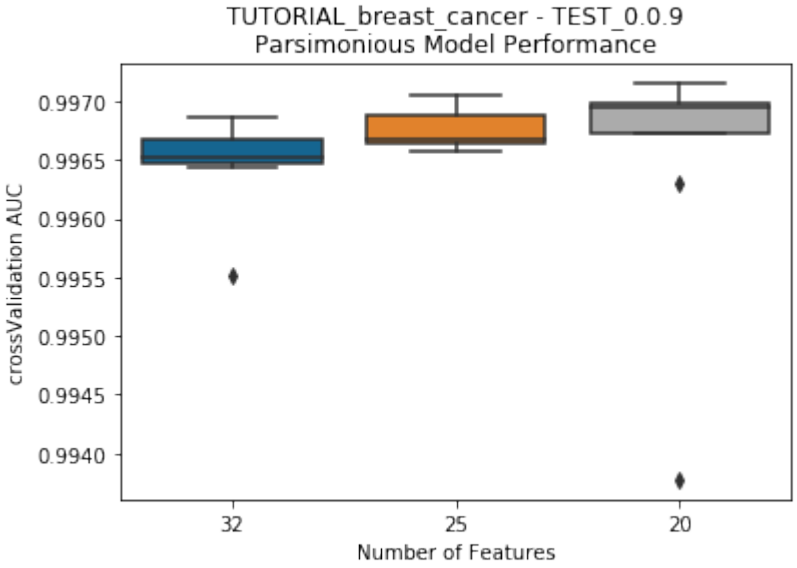
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 0.94s
Checking lives: 5.58s
Lives left: 5 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.10s
CV Error From the Mean: 0.005385306929239613 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (20) ----------
Autopilot: 270.22s
Feature Impact job(s) remaining (0)
Feature Impact: 78.82s
Waiting for DataRobot: 10.67s
```



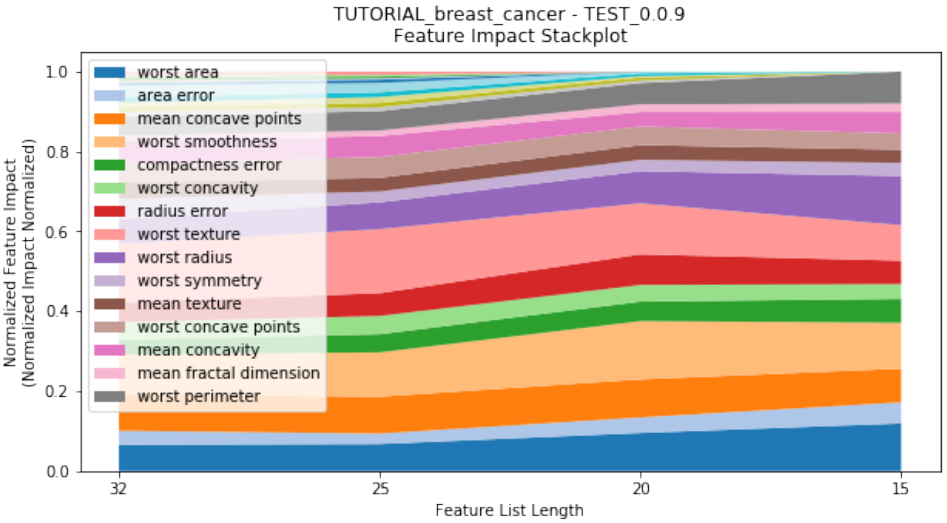
nbsphinx-code-borderwhite

```
Performance Stackplot: 18.91s
```



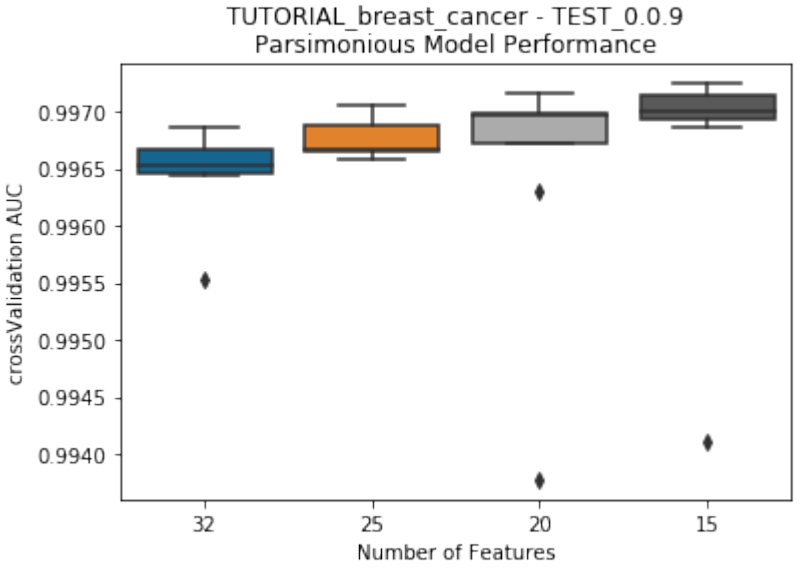
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 1.17s
Checking lives: 11.45s
Lives left: 5 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.19s
CV Error From the Mean: 0.005228428328360267 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (15) ----------
Autopilot: 230.22s
Feature Impact job(s) remaining (0)
Feature Impact: 68.98s
Waiting for DataRobot: 10.58s
```



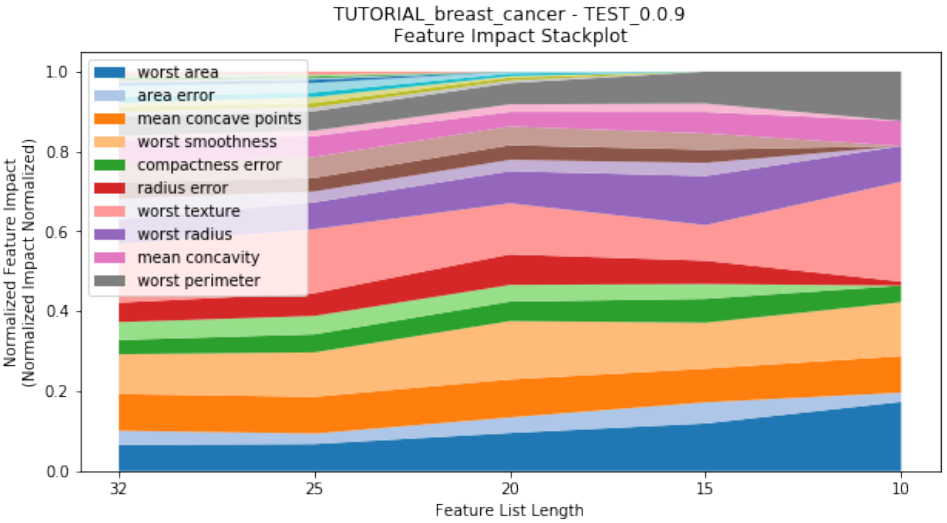
nbsphinx-code-borderwhite

```
Performance Stackplot: 25.13s
```



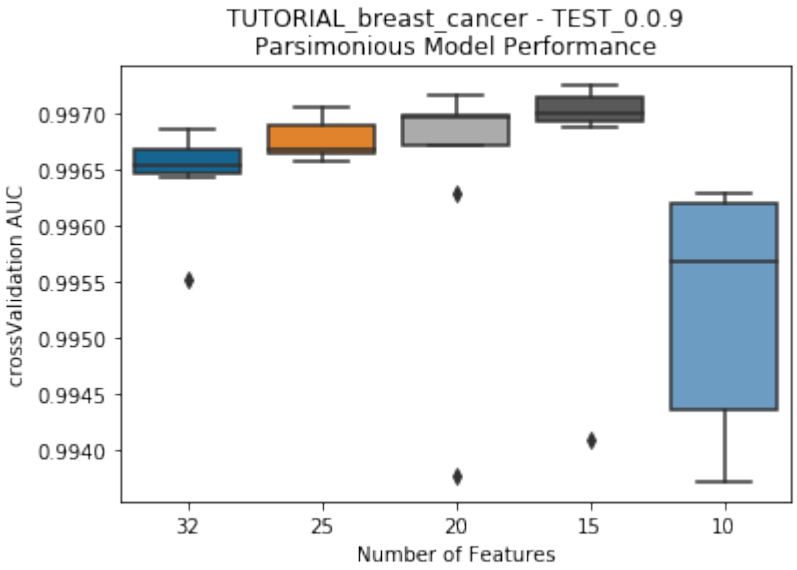
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 1.35s
Checking lives: 16.93s
Lives left: 5 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.44s
CV Error From the Mean: 0.005057377745942271 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (10) ----------
Autopilot: 472.73s
Feature Impact job(s) remaining (0)
Feature Impact: 73.14s
Waiting for DataRobot: 10.95s
```



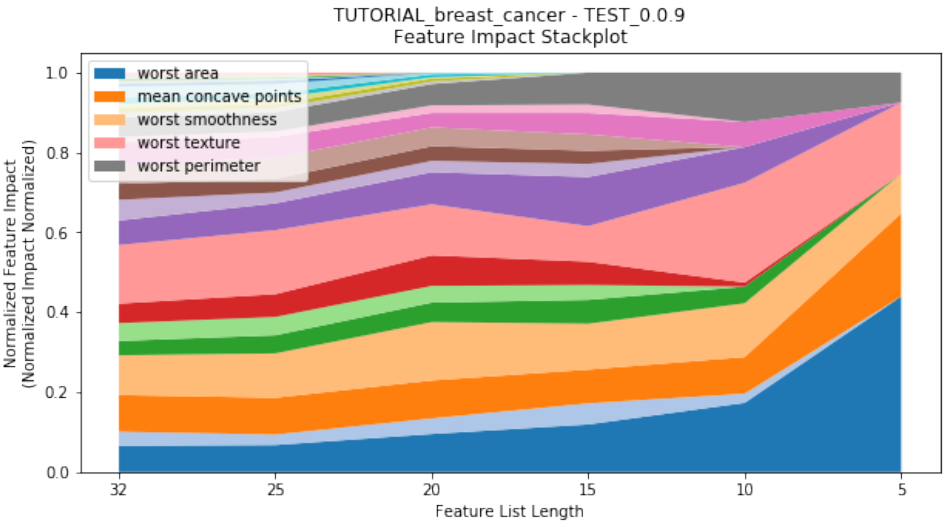
nbsphinx-code-borderwhite

```
Performance Stackplot: 33.12s
```



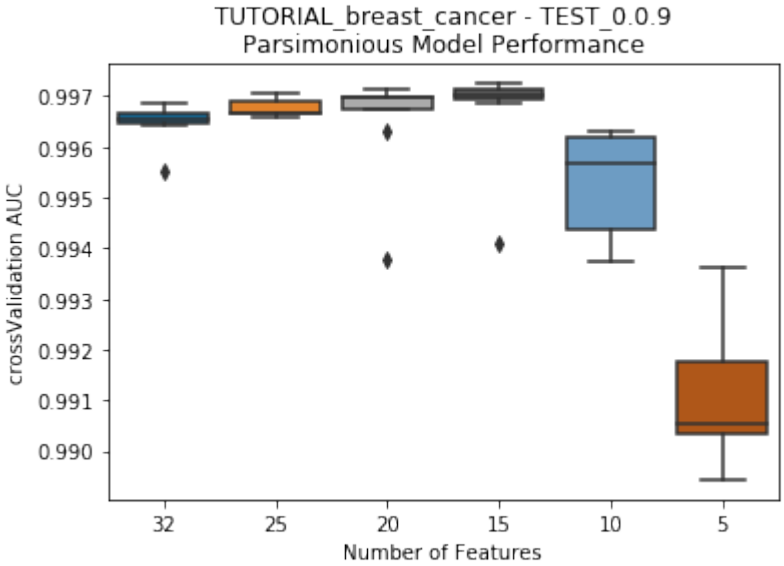
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 1.70s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '4'
Checking lives: 23.74s
Lives left: 4 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.21s
CV Error From the Mean: 0.004886720473612955 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (5) ----------
Autopilot: 230.10s
Feature Impact job(s) remaining (0)
Feature Impact: 85.14s
Waiting for DataRobot: 10.64s
```



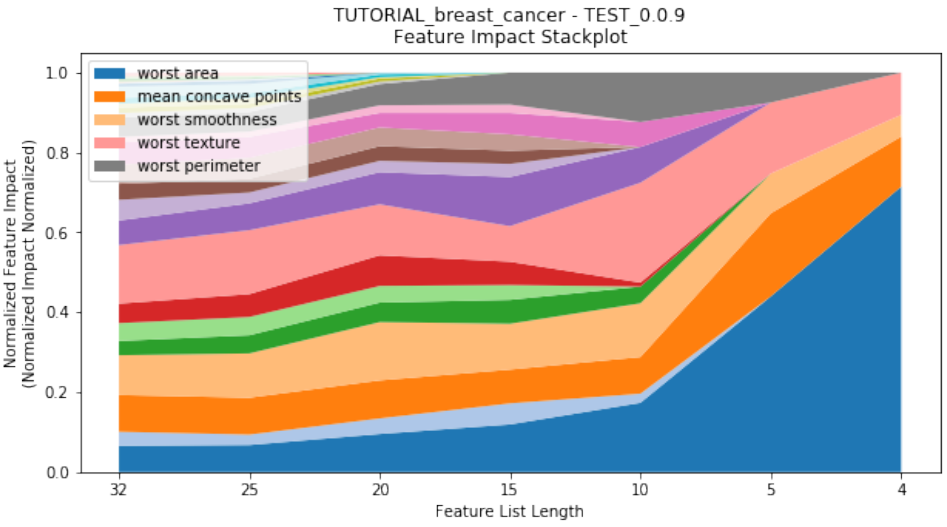
nbsphinx-code-borderwhite

```
Performance Stackplot: 36.20s
```



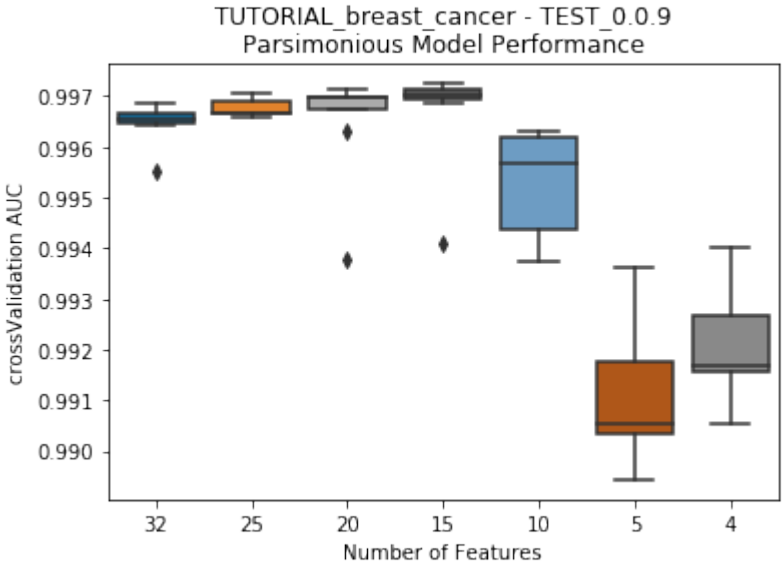
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 2.38s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '3'
Checking lives: 27.89s
Lives left: 3 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.43s
CV Error From the Mean: 0.009681340693827197 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (4) ----------
Autopilot: 249.77s
Feature Impact job(s) remaining (0)
Feature Impact: 69.58s
Waiting for DataRobot: 10.57s
```



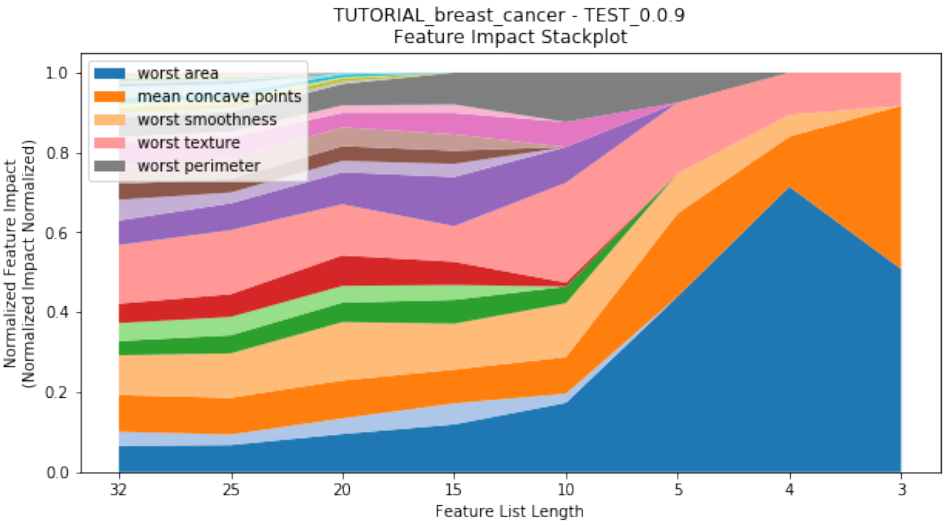
nbsphinx-code-borderwhite

```
Performance Stackplot: 40.44s
```



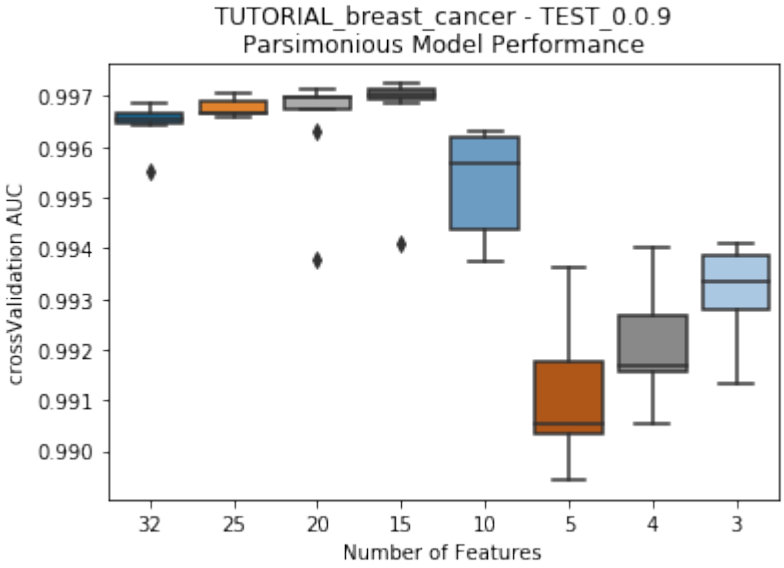
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 2.22s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '2'
Checking lives: 32.98s
Lives left: 2 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.81s
CV Error From the Mean: 0.00870168512405422 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (3) ----------
Autopilot: 249.92s
Feature Impact job(s) remaining (0)
Feature Impact: 84.92s
Waiting for DataRobot: 10.33s
```



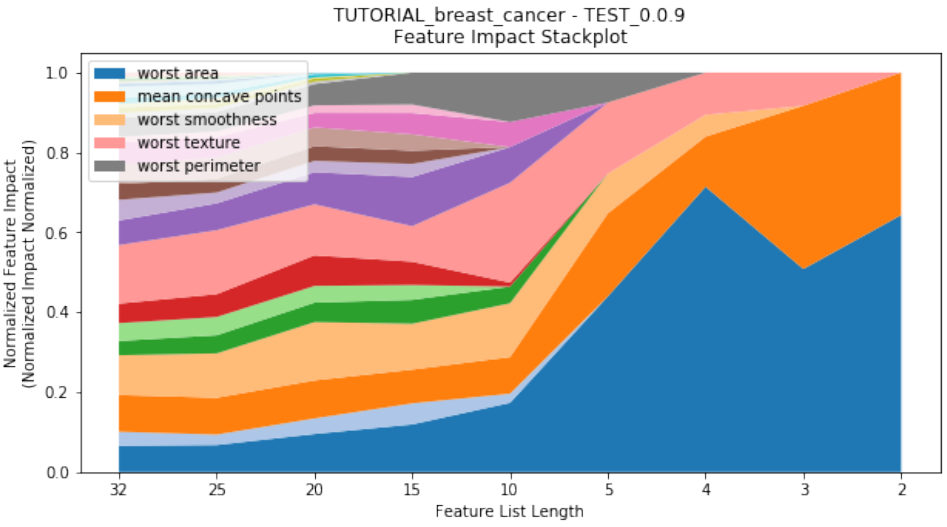
nbsphinx-code-borderwhite

```
Performance Stackplot: 50.59s
```



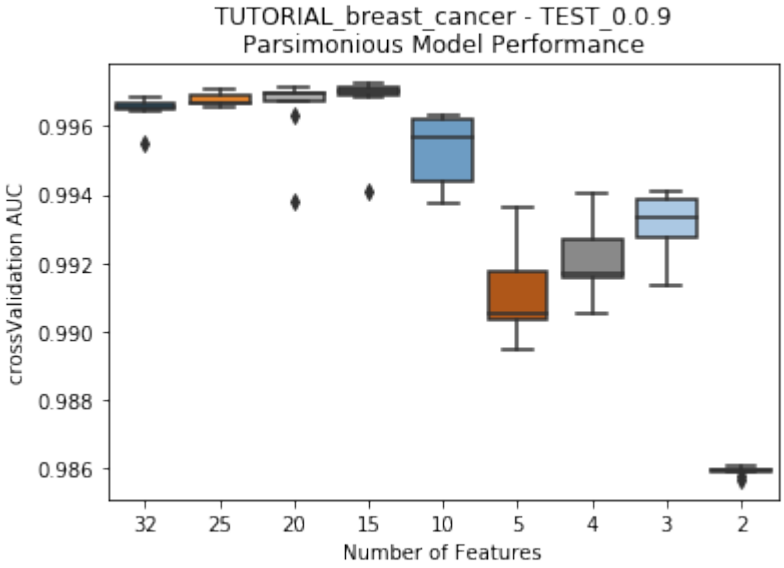
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 2.77s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '1'
Checking lives: 40.55s
Lives left: 1 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.25s
CV Error From the Mean: 0.005374831693523058 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (2) ----------
Autopilot: 230.24s
Feature Impact job(s) remaining (0)
Feature Impact: 80.29s
Waiting for DataRobot: 10.61s
```



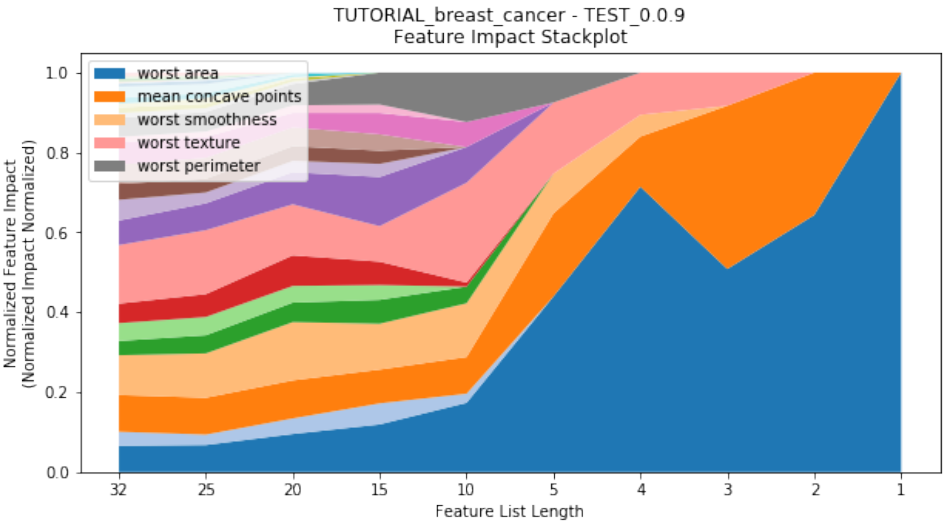
nbsphinx-code-borderwhite

```
Performance Stackplot: 54.69s
```



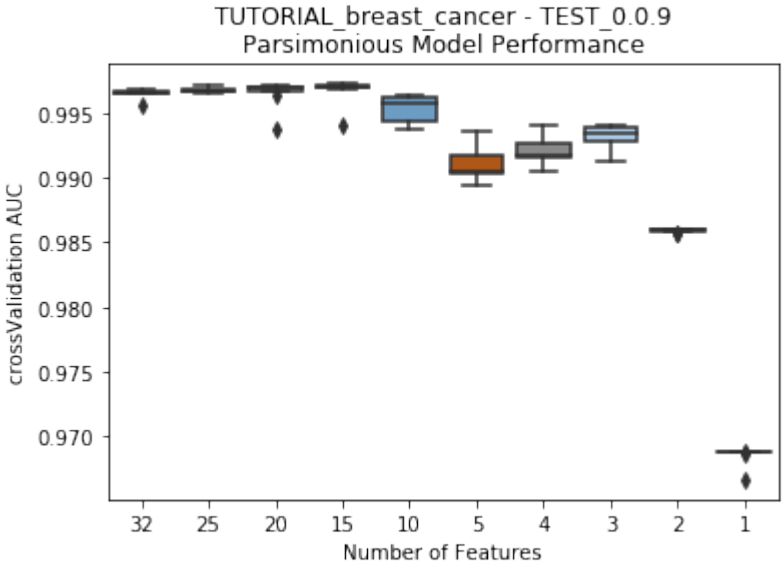
nbsphinx-code-borderwhite

```
Model Performance Boxplot: 2.78s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '0'
Checking lives: 43.40s
Lives left: 0 | Previous Model Best Score: 0.996862 | Current Best Model Score: 0.
↪9970519999999999
Mean Error Limit: 1.06s
CV Error From the Mean: 0.007301264001862837 | CV Mean Error Limit: 0.8 | CV Model␣
↪Performance Metric: AUC
---------- TEST_0.0.9 (1) ----------
Autopilot: 189.18s
Feature Impact job(s) remaining (0)
Feature Impact: 64.54s
Waiting for DataRobot: 10.16s
```



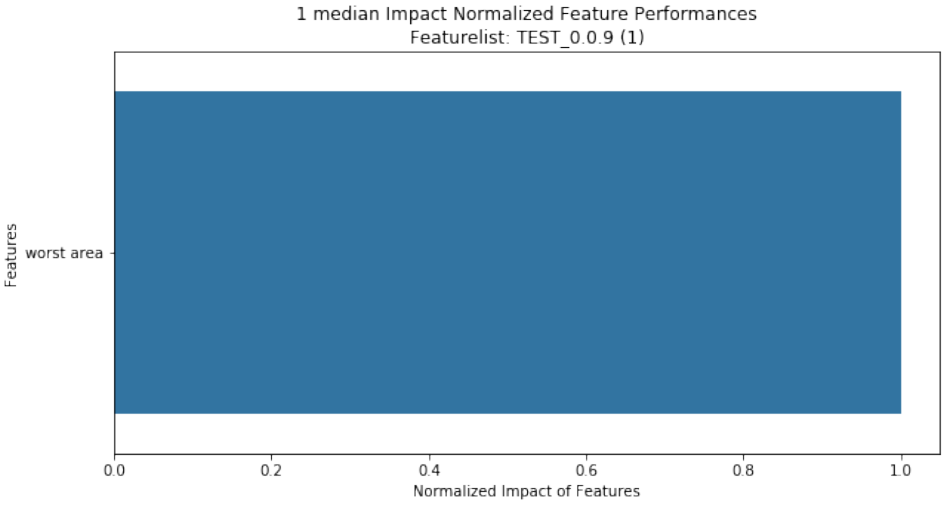
nbsphinx-code-borderwhite

```
Performance Stackplot: 57.28s
```



nbsphinx-code-borderwhite

```
Model Performance Boxplot: 2.91s
Current model performance: '0.997242'. Previous best model performance: '0.997242'
No change in the best model, so a life was lost.
Lives remaining: '-1'
Checking lives: 47.33s
Ran out of lives.
Best model: 'Model('Elastic-Net Classifier (L2 / Binomial Deviance)')'
Accuracy (AUC):'0.997242'
```



nbsphinx-code-borderwhite

```
Finished Parsimony Analysis in 4211.28s.
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX