trelawney Documentation

Release 0.3.1

Skander Kamoun

Feb 18, 2020

Contents:

CHAPTER 1

Installation

1.1 Stable release

To install trelawney, run this command in your terminal:

```
$ pip install trelawney
```

This is the preferred method to install trelawney, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for trelawney can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/skanderkam/trelawney
```

Or download the tarball:

```
$ curl -OJL https://github.com/skanderkam/trelawney/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 2

trelawney

2.1 trelawney package

2.1.1 Submodules

2.1.2 trelawney.base_explainer module

module that provides the base explainer class from which all future explainers will inherit

class trelawney.base_explainer.**BaseExplainer**

Bases: abc.ABC

the base explainer class. this is an abstract class so you will need to define some behaviors when implementing your new explainer. In order to do so, override:

- the *fit* method that defines how (if needed) the explainer should be fited
- the *feature_importance* method that extracts the relative importance of each feature on a dataset globally
- the *explain_local* method that extracts the relative impact of each feature on the final decisionfor every sample in a dataset

explain_filtered_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], cols: List[str], n_cols: Optional[int] = None*) → List[Dict[str, float]]

same as *explain_local* but applying a filter on each explanation on the features

explain_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → List[Dict[str, float]]

explains each individual predictions made on x_explain. BEWARE this is usually quite slow on large datasets

Parameters

- **x_explain** – the samples to explain
- **n_cols** – the number of columns to limit the explanation to

feature_importance(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → Dict[str, float]

returns a relative importance of each feature on the predictions of the model (the explainer was fitted on) for *x_explain* globally. The output will be a dict with the importance for each column/feature in *x_explain* (limited to n_cols)

if some importance are negative, this means they are negatively correlated with the output and absolute value represents the relative importance

Parameters

- **x_explain** – the dataset to explain on
- **n_cols** – the maximum number of features to return (ordered by importance)

filtered_feature_importance(*x_explain: pandas.core.frame.DataFrame, cols: Optional[List[str]], n_cols: Optional[int] = None*) → Dict[str, float]

same as *feature_importance* but applying a filter first (on the name of the column)

fit(*model: sklearn.base.BaseEstimator, x_train: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], y_train: pandas.core.frame.DataFrame*)

prepares the explainer by saving all the information it needs and fitting necessary models

Parameters

- **model** – the TRAINED model the explainer will need to shed light on
- **x_train** – the dataset the model was trained on originally
- **y_train** – the target the model was trained on originally

graph_feature_importance(*x_explain: pandas.core.frame.DataFrame, cols: Optional[List[str]] = None, n_cols: Optional[int] = None, irrelevant_cols: Optional[List[str]] = None*)

graph_local_explanation(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], cols: Optional[List[str]] = None, n_cols: Optional[int] = None, info_values: Union[pandas.core.frame.DataFrame, pandas.core.series.Series, None] = None*) → plotly.graph_objs._figure.Figure

creates a waterfall plotly figure to represent the influance of each feature on the final decision for a single prediction of the model.

You can filter the columns you want to see in your graph and limit the final number of columns you want to see. If you choose to do so the filter will be applied first and of those filtered columns at most *n_cols* will be kept

Parameters

- **x_explain** – the example of the model this must be a dataframe with a single ow
- **cols** – the columns to keep if you want to filter (if None - default) all the columns will be kept
- **n_cols** – the number of columns to limit the graph to. (if None - default) all the columns will be kept

Raises **ValueError** – if x_explain doesn't have the right shape

2.1.3 trelawney.colors module

2.1.4 trelawney.lime_explainer module

class trelawney.lime_explainer.**LimeExplainer**(*class_names: Optional[List[str]] = None*, *categorical_features: Optional[List[str]] = None*)

Bases: *trelawney.base_explainer.BaseExplainer*

Lime stands for local interpretable model-agnostic explanations and is a package based on this article. Lime will explain a single prediction of you model by crechariotsating a local approximation of your model around said prediction.'sphinx.ext.autodoc', 'sphinx.ext.viewcode']

```
>>> X = pd.DataFrame([np.array(range(100)), np.random.normal(size=100).tolist()],␣
↪index=['real', 'fake']).T
>>> y = np.array(range(100)) > 50
>>> # training the base model
>>> model = LogisticRegression().fit(X, y)
>>> # creating and fiting the explainer
>>> explainer = LimeExplainer()
>>> explainer.fit(model, X, y)
<trelawney.lime_explainer.LimeExplainer object at ...>
>>> # explaining observation
>>> explanation =  explainer.explain_local(pd.DataFrame([[5, 0.1]]))[0]
>>> abs(explanation['real']) > abs(explanation['fake'])
True
```

explain_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *n_cols: Optional[int] = None*) → List[Dict[str, float]]

explains each individual predictions made on x_explain. BEWARE this is usually quite slow on large datasets

Parameters

- **x_explain** – the samples to explain
- **n_cols** – the number of columns to limit the explanation to

feature_importance(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *n_cols: Optional[int] = None*) → Dict[str, float]

returns a relative importance of each feature on the predictions of the model (the explainer was fitted on) for *x_explain* globally. The output will be a dict with the importance for each column/feature in *x_explain* (limited to n_cols)

if some importance are negative, this means they are negatively correlated with the output and absolute value represents the relative importance

Parameters

- **x_explain** – the dataset to explain on
- **n_cols** – the maximum number of features to return (ordered by importance)

fit(*model: sklearn.base.BaseEstimator*, *x_train: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *y_train: pandas.core.frame.DataFrame*)

prepares the explainer by saving all the information it needs and fitting necessary models

Parameters

- **model** – the TRAINED model the explainer will need to shed light on
- **x_train** – the dataset the model was trained on originally

- **y_train** – the target the model was trained on originally

2.1.5 trelawney.logreg_explainer module

Module that provides the LogRegExplainer class base on the BaseExplainer class

class trelawney.logreg_explainer.**LogRegExplainer**(*class_names: Optional[List[str]] = None*, *categorical_features: Optional[List[str]] = None*)

Bases: *trelawney.base_explainer.BaseExplainer*

The LogRegExplainer class is composed of 3 methods: - fit: get the right model - feature_importance (global interpretation) - graph_odds_ratio (visualisation of the ranking of the features, based on their odds ratio)

explain_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *n_cols: Optional[int] = None*) → List[Dict[str, float]]

returns local relative importance of features for a specific observation. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

feature_importance(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *n_cols: Optional[int] = None*) → Dict[str, float]

returns the absolute value (i.e. magnitude) of the coefficient of each feature as a dict. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

fit(*model: sklearn.base.BaseEstimator*, *x_train: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray]*, *y_train: pandas.core.frame.DataFrame*)

prepares the explainer by saving all the information it needs and fitting necessary models

Parameters

- **model** – the TRAINED model the explainer will need to shed light on
- **x_train** – the dataset the model was trained on originally
- **y_train** – the target the model was trained on originally

graph_odds_ratio(*n_cols: Optional[int] = 10*, *ascending: bool = False*, *irrelevant_cols: Optional[List[str]] = None*) → pandas.core.frame.DataFrame

returns a plot of the top k features, based on the magnitude of their odds ratio. :n_cols: number of features to plot :ascending: order of the ranking of the magnitude of the coefficients

2.1.6 trelawney.shap_explainer module

2.1.7 trelawney.surrogate_explainer module

class trelawney.surrogate_explainer.**SurrogateExplainer**(*surrogate_model: sklearn.base.BaseEstimator*, *class_names: Optional[List[str]] = None*)

Bases: *trelawney.base_explainer.BaseExplainer*

A surrogate model is a substitution model used to explain the initial model. Therefore, substitution models are generally simpler than the initial ones. Here, we use single trees and logistic regressions as surrogates.

adequation_score(*metric: Union[Callable[[numpy.ndarray, numpy.ndarray], float], str] = 'auto'*)

returns an adequation score between the output of the surrogate and the output of the initial model based on the x_train set given.

explain_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → List[Dict[str, float]]
returns local relative importance of features for a specific observation. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

feature_importance(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → Dict[str, float]
returns a relative importance of each feature globally as a dict. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

fit(*model: sklearn.base.BaseEstimator, x_train: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], y_train: pandas.core.frame.DataFrame*)
prepares the explainer by saving all the information it needs and fitting necessary models

Parameters

- **model** – the TRAINED model the explainer will need to shed light on
- **x_train** – the dataset the model was trained on originally
- **y_train** – the target the model was trained on originally

plot_tree(*out_path: str = './tree_viz'*)
returns the colored plot of the decision tree and saves an Image in the wd.

2.1.8 trelawney.tree_explainer module

Module that provides the TreeExplainer class base on the Baseexplainer class

class trelawney.tree_explainer.**TreeExplainer**(*class_names: Optional[List[str]] = None*)
Bases: *trelawney.base_explainer.BaseExplainer*

The TreeExplainer class is composed of 4 methods: - fit: get the right model - feature_importance (global interpretation) - explain_local (local interpretation, WIP) - plot_tree (full tree visualisation)

explain_local(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → List[Dict[str, float]]
returns local relative importance of features for a specific observation. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

feature_importance(*x_explain: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], n_cols: Optional[int] = None*) → Dict[str, float]
returns a relative importance of each feature globally as a dict. :param x_explain: the dataset to explain on :param n_cols: the maximum number of features to return

fit(*model: sklearn.base.BaseEstimator, x_train: Union[pandas.core.series.Series, pandas.core.frame.DataFrame, numpy.ndarray], y_train: pandas.core.frame.DataFrame*)
prepares the explainer by saving all the information it needs and fitting necessary models

Parameters

- **model** – the TRAINED model the explainer will need to shed light on
- **x_train** – the dataset the model was trained on originally
- **y_train** – the target the model was trained on originally

plot_tree(*out_path: str = './tree_viz'*)
creates a png file of the tree saved in out_path

Parameters **out_path** – the path to save the png representation of the tree to

2.1.9 trelawney.trelawney module

2.1.10 Module contents

Top-level package for trelawney.

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/skanderkam/trelawney/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

3.1.4 Write Documentation

trelawney could always use more documentation, whether as part of the official trelawney docs, in docstrings, or even on the web in blog posts, articles, and such.

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/skanderkam/trelawney/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.2 Get Started!

Ready to contribute? Here's how to set up *trelawney* for local development.

1. Fork the *trelawney* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/trelawney.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv trelawney
$ cd trelawney/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 trelawney tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/skanderkam/trelawney/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ pytest tests.test_trelawney
```

3.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 4

Credits

4.1 Development Lead

- Skander Kamoun <skander.kam2@gmail.com>

4.2 Contributors

- Ludmila Exbrayat
- Amelie Meurer
- Antoine Redier
- Ines Vanagt

CHAPTER 5

History

5.1 0.1.0 (2019-10-02)

- First release on PyPI.

CHAPTER 6

trelawney

Trelawney is a general interpretability package that aims at providing a common api to use most of the modern interpretability methods to shed light on sklearn compatible models (support for Keras and XGBoost are tested).

Trelawney will try to provide you with two kind of explanation when possible:

- global explanation of the model that highlights the most importance features the model uses to make its predictions globally
- local explanation of the model that will try to shed light on why a specific model made a specific prediction

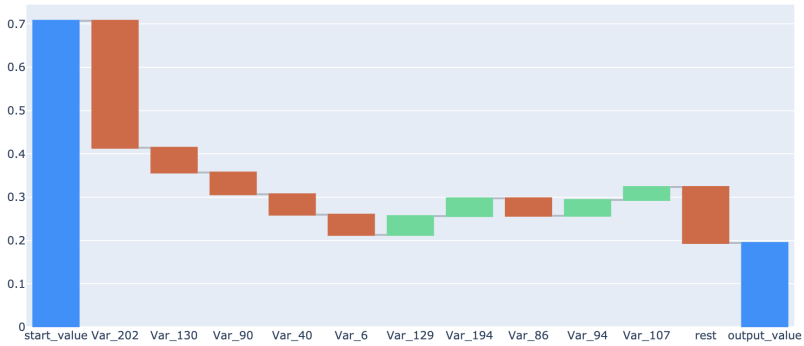
The Trelawney package is build around:

- **some model specific explainers that use the inner workings of some types of models to explain them:**
  - *LogRegExplainer* that uses the weights of the your logistic regression to produce global and local explanations of your model
  - *TreeExplainer* that uses the path of your tree (single tree model only) to produce explanations of the model
- **Some model agnostic explainers that should work with all models:**
  - *LimeExplainer* that uses the Lime package to create local explanations only (the local nature of Lime prohibits it from generating global explanations of a model
  - *ShapExplainer* that uses the SHAP package to create local and global explanations of your model
  - *SurrogateExplainer* that creates a general surogate of your model (fitted on the output of your model) using an explainable model (*DecisionTreeClassifier*,'LogisticRegression' for now). The explainer will then use the internals of the surrogate model to explain your black box model as well as informing you on how well the surrogate model explains the black box one

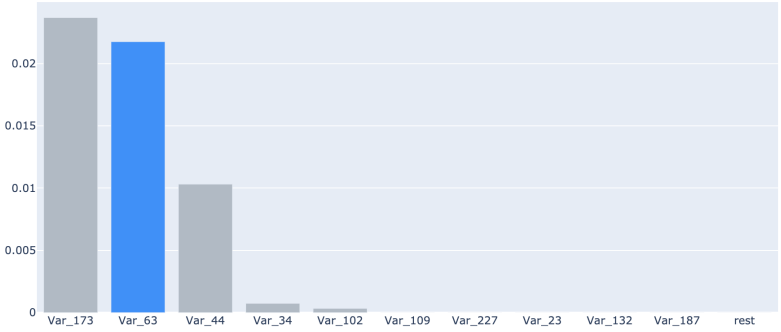
6.1 Quick Tutorial (30s to Trelawney):

Here is an example of how to use a Trelawney explainer

```
>>> model = LogisticRegression().fit(X, y)
>>> # creating and fiting the explainer
>>> explainer = ShapExplainer()
>>> explainer.fit(model, X, y)
>>> # explaining observation
>>> explanation =  explainer.explain_local(X_expain)
[
    {'var_1': 0.1, 'var_2': -0.07, ...},
    ...
    {'var_1': 0.23, 'var_2': -0.15, ...} ,
]
>>> explanation =  explainer.graph_local_explanation(X_expain.iloc[:1, :])
```




```
>>> explanation =  explainer.feature_importance(X_expain)
{'var_1': 0.5, 'var_2': 0.2, ...} ,
>>> explanation =  explainer.graph_feature_importance(X_expain)
```



6.2 FAQ

Why should I use Trelawney rather than Lime and SHAP

while you can definitally use the Lime and SHAP packages directly (they will give you more control over how to use their packages), they are very specialized packages with different APIs, graphs and vocabulary. Trelawnaey offers you a unified API, representation and vocabulary for all state of the art explanation methods so that you don't lose time adapting to each new method but just change a class and Trelawney will adapt to you.

6.3 Comming Soon

- Regressor Support (PR welcome)
- Image and text Support (PR welcome)

6.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

T