PyDePr Documentation

Release 0.14

Eric Strong

Aug 08, 2017

Contents

PyDePr ("pie-deeper") is a toolkit designed to facilitate the preprocessing and validation of machine learning models.

CHAPTER 1

1. Introduction

PyDePr ("pie-deeper") is a toolkit designed to facilitate the preprocessing and validation of machine learning models. What does that mean? Machine learning models are often specialized, requiring years of experience to correctly configure. The hardest step is often preprocessing the dataset to be analyzed. An old joke in machine learning is that 90% of the work is cleaning the dataset, while only 10% is actual data analysis.

Keeping that in mind, PyDePr is meant to ease the initial step of model-building by automatically preprocessing some very specific types of machine learning models. PyDePr is not meant to build machine learning models for you (there are plenty of Python libraries for that purpose), and it is not meant to be some sort of catch-all solution for all types of machine learning models. However, support for additional model types will be added as PyDePr is developed.

Package Organization

PyDePr is organized into several namespaces, including:

- regression
- inference
- waveform

The regression namespace handles all preprocessing and validation for regression models, including a visualization of results. The inference namespace is meant for Bayesian inference, where accumulated evidence points to a conclusion.

CHAPTER 2

2. Getting Started

Installation

If Python is already installed on your computer, PyDePr can be installed using PyPI by opening a command window and typing:

pip install pydepr

Upgrading to a new version of PyDePr can be accomplished by:

pip install pydepr –upgrade

The source code of PyDePr is hosted on GitHub at:

https://github.com/drericstrong/pydepr

Python Requirements

Required modules: matplotlib, numpy, pandas, seaborn, scipy, scikit-learn, sympy, statsmodels

PyDePr is integrated tightly with PyeDNA, with static helper functions for pulling data directly from eDNA. PyeDNA is not required for most functions in PyDePr, but it will be loaded if necessary.

A requirements.txt document is located in the GitHub repository, and all package requirements can be installed using the following line in a command window:

pip install -r requirements.txt

Python Version Support

Currently, PyDePr only supports Python 3.2+ and is not compatible with Python 2. If Python 2 support is important to you, please make a pull request at:

https://github.com/drericstrong/pydepr

The package maintainer welcomes collaboration.

Importing PyDePr

Modules in PyDePr are usually imported into a script using the following lines:

import pydepr.regression as regr

import pydepr.inference as infer

CHAPTER 3

3. Regression

PyDePr supports the construction of regression models, with built-in visual model validation. The following features are available:

- Supply data from either a pandas DataFrame or eDNA
- Automatically perform Ridge Regression, Lasso, LassoLars, and ElasticNet
- Calculate model performance metrics
- Build the model equation in a user-friendly form
- Create a series of plots for model validation and visualization

All code in this section assumes that you have imported PyDePr like:

import pydepr.regression as regr

Regression Curve

The base class in this namespace is the RegressionCurve class. You can initialize a RegressionCurve by supplying it with a model type:

curve = regr.RegressionCurve(model_type="ridge")

Possible values for the model_type parameter are "Lasso", "ElasticNet", "Ridge", and "LassoLars". The default value is "Ridge".

Next, provide the model with inputs (x data) and outputs (y data), using the **add_input** and **add_output** methods, respectively:

curve.add_input(x_data)

curve.add_output(y_data)

If x data already exists, it will merge the new data with the existing data using an outer join, by default. However, only one y variable may be specified, so the **add_output** function will overwrite all the existing y_data.

Optionally, you can specify data filters using the add_filter method:

curve.add_filter(filter_data, low_value, high_value)

The x and y data will be filtered based on when the filter_data is greater than or equal to the low_value, and less than or equal to the high_value.

Next, the "run" function will automatically create a Ridge Regression model using the x and y data:

curve.run()

Once the regression curve has been run, the model validation metrics can be found (next section).

Model Validation

PyDePr will generate a plot which can be used for validation of the regression model, using the following function:

f = curve.plot_validation()

Warning and alarm limits (based on standard deviations) can also be supplied to the plots:

f = curve.plot_validation(warn=2, alarm=3)

The validation metrics generated for the above plot can also be found directly:

metrics = curve.calculate_metrics()

Equation Building

The RegressionCurve class will automatically build equations based on the results from the Ridge Regression:

eq, corr_eq = curve.build_equation()

The first value returned will be the full regression model equation, while the second value returned will be the "corrected" equation (more explanation below).

The regular equation is of the format (for ease of import into eDNA):

Value = AX + BZ

The corrected equation is of the format:

Value = BZ - Y

CHAPTER 4

4. Inference

Inference can be accomplished using time-series evidence within this PyDePr module.

- Assign evidence to a conclusion
- Use fuzzy logic to interpolate between conclusions

Evidence

Conclusions are made using Bayesian inference based on the assigned Evidence. This class defines an Evidence input to a Conclusion. Any symptom which indicates the Conclusion may be used as an Evidence input.

The following parameters may be supplied:

- thresholds: A list of thresholds in order: [normal, warning, alarm, danger, extreme danger]
- cpt: the conditional probability table for the evidence, which specifies the marginal probability distribution of the truth of the failure mode, depending on selected state. Must be an array of length 4. For example: [0.1, 0.2, 0.3, 0.4]
- name: an optional description or identifier.
- default_state: the default state of the evidence if bad input is received. 0 defaults to the prior, 1 defaults to Normal, 2 defaults to Warning, 3 defaults to Alarm, and 4 defaults to Danger. Recommended values are either 0 or 1.

Counter Evidence

This class defines a contrary evidence input to a Conclusion. Any symptom which will excuplate the occurrence of a Conclusion can be used as a ContraryEvidence node.

The following parameters may be supplied:

- thresholds: A list of 4 thresholds in order: [normal, undetermined, abnormal, contradictory]

- priors: an array of updated priors for the Conclusion, based on the selected ContraryEvidence state. Must be length 4.
- name: an optional description or identifier.

Conclusion

This is the primary class that will be used in this model. Defines a Conclusion in terms of evidence and counter evidence.

The following parameters may be supplied:

- name: an optional description or identifier
- prior: the prior probability of occurrence
- evidence: an array of "Evidence" class nodes which define symptoms associated with the Conclusion. At least 1 must exist.
- contrary: a single "ContraryEvidence" class node which contradicts the Conclusion. A maximum of 1 may exist.

FuzzyStates

This class is not meant to be used standalone, but it provides backend functionality for the above classes.

FuzzyStates uses a fuzzy linear interpolation between the four user-defined states. Warning- the supplied values must be the "midpoints", or the thresholds at which the fuzzy membership for the state should be equal to 100%. Total membership across all four states must always be equal to 100%.

The following parameters may be supplied:

- normal: signal value under normal behavior.
- warning: signal value that begins to indicate an anomaly.
- alarm: onset of significant poor behavior.
- danger: signal strongly indicates the Conclusion.
- steps: an optional parameter for the # of steps of the range of fuzzy values. Must be at least 100.

CHAPTER 5

Change Log

Version 0.11

- Initial release
- Regression models can now pull data using PyeDNA

Version 0.12

- Release of the inference module

Version 0.13

- New and improved documentation
- Minor work on the waveform module

Version 0.14

- Significant refactoring of the regression model

CHAPTER 6

Indices and tables

- genindex
- modindex
- search