Documenting and Implementing Multiple SciKit Learn Algorithms

Julia Piscioniere, Philip Cork, Allison Hansen, Dylan Waks

CSCI 462 - Data Science Capstone

Abstract

SciKit-Learn is an open-source software that enables predictive data analysis. Built in Python on the foundations of NumPy, SciPy, and matplotlib, SciKit-Learn makes complex statistical models easily applicable to datasets in a range of contexts.

Documentation is vital to users' ability to capitalize on the capabilities of the software. Due to the nature of open-source software having different algorithms contributed by various members of the community, there are at times some inconsistent practices for documentation throughout. Here, we undertook pieces of a larger issue to standardize documentation usually relating to missing parameters within a method of a statistical model.

The statistical models we studied include Kernel Density, Mini Batch KMeans, CCA, PLS SVD, Incremental PCA, and Lasso. This process required both theoretical knowledge of the statistical model and a practical understanding of the SciKit-Learn codebase, so we have included examples of some of the statistical models invoked using sample data sets while describing our experience contributing to the documentation.

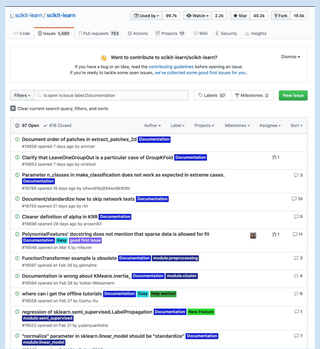
Sci-Kit Learn Features

Sci-Kit learn features numerous methods for exploring and analyzing data through machine learning processes using both Supervised Learning and Unsupervised Learning methods.

Some of the most popular tools include their myriad of Classification and Regression models as well as multiple methods for Classification or Association Mining.

Sci-Kit Learn also features tools for dimension reduction and preprocessing and visualizing data.

Contributing to Sci-Kit Learn



Sci-Kit Learn uses Github for hosting code, ensuring version control, and tracking bugs and new feature requests. Getting involved in the community is as simple as creating a Github account and joining one of the many ongoing conversations.

Sci-Kit Learn's website provides extensive resources for formatting both issues and pull request and the Issues tab on Github deploys a detailed labeling system including filters for "Good First Issues" to help first time contributors get started.

Documentation in Sci-Kit Learn

Our project focussed on improving the documentation within Sci-Kit Learn's codebase. In particular, we worked to help fill in gaps where attributes used by various models were not correctly listed in the comments of the file itself and thus were also not imported into their online support documentation via the Sci-Kit Learn API.

For our first foray into providing documentation contributions, we dove into the Bayes Regression code which was missing references to several internally used variables. After exploring the code for which attributes weren't properly listed, we settled on the list of three variables which required further explanation, namely, *x_offset*, *y_offset*, and *x_scale*.

While the x and y variables typically represent the data and dependent variables in machine learning, it became our task to dig into the functions called throughout the Bayes Regression codebase and determine how the variables were being transformed to adequately define the offset and scaled versions of the respective variables.

After cross-referencing several of the preprocessing functions to see how these attributes were being called and defined, we drafted our definitions of these attributes and submitted them as a pull request to the community with review and revision ongoing. We then worked on several other files with missing attributes, which we bundled together into a second pull request which is still to be reviewed.

```
if self.n_iter < 1:
    raise ValueError('n_iter should be greater than or equal to 1.'
                     ' Got {!r}.'.format(self.n_iter))

X, y = check_X_y(X, y, dtype=np.float64, y_numeric=True)

if sample_weight is not None:
    sample_weight = _check_sample_weight(sample_weight, X,
                                         dtype=X.dtype)

X, y, X_offset_, y_offset_, X_scale_ = self._preprocess_data(
    X, y, self.fit_intercept, self.normalize, self.copy_X,
    sample_weight=sample_weight)

if sample_weight is not None:
    # Sample weight can be implemented via a simple rescaling.
    X, y = _rescale_data(X, y, sample_weight)

self.X_offset_ = X_offset_
self.X_scale_ = X_scale_
n_samples, n_features = X.shape
```

[MRG][DOC] Issue #14312 - Ensure All Attributes are Documented: Bayesian Ridge #16417

Open — juliapiscioniere wants to merge 12 commits into scikit-learn:master from juliapiscioniere:julia_feature_branch

Conversation 19 · Commits 12 · Checks 14 · Files changed 1 · +12 −0

juliapiscioniere commented on Feb 8 · edited

Added Bayesian Ridge Regression attribute documentation for X_offset_, X_scale_, and y_offset

Reference Issues/PRs

Issue #14312

What does this implement/fix? Explain your changes.

This adds documentation for attributes for the Bayesian Ridge function. The attributes are x_offset, y_offset, and x_scale.

Any other comments?

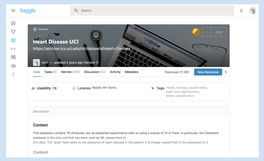
Reviewers: cmarmo, adrinjalali, jnothman

Assignees: No one assigned

Labels: module:linear_model

Principal Component Analysis Example

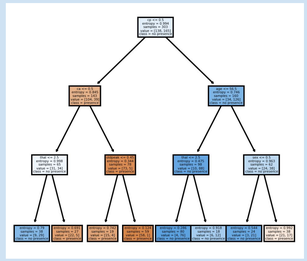
"Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD." - SciKit-learn Documentation



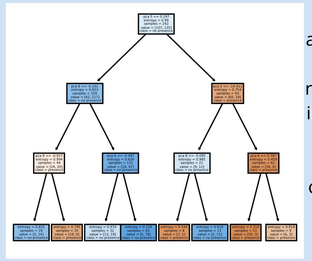
Health data is a common example of multicollinearity (high correlations between features in a dataset; one column could potentially influence another) and thus is additionally an excellent use case for Principal Component Analysis (PCA).

PCA works by taking the 14 columns in the original heart data, and reducing them in combination to 8 columns which improve the accuracy of the decision tree.

Original Tree



PCA Tree



These 8 columns are somewhat ambiguous in form but are meant to represent the most important features (ie. statistical variation) of each column in the new combinations which allows the function to increase accuracy.

Unsupervised Learning

Unsupervised learning is one type of machine learning that does not rely on labels of data, but searches for previously undetected patterns in the data set. This type of machine learning requires less human supervision, hence its name. It works by sorting data points into groups based on their probability density. Because of the lack of labels, another important aspect of unsupervised learning is its difficulty in measuring accuracy as often there is nothing to measure the groupings by.

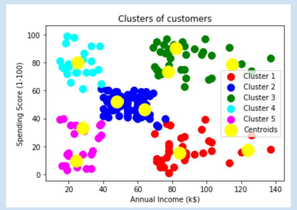
Using the Scikit Learn's Mini Batch K-Means algorithm on a data set to group its customers based off of a few traits into categories, it is easy to see how unsupervised learning can be very useful in grouping people and finding patterns.

The attributes of the data can be seen below

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39

The Mini Batch K-Mean algorithm works by grouping all data points in k different groups based off their distance to randomly chosen centroid data points. After the initial grouping, the most center point becomes the a new centroid and all the other points are then regrouped based off of the new centroids. This process repeats until there is no more change between new groupings.

In order to visualize who with works, here is a two dimensional graph of based off of only two of the attributes from the model, annual income and spending score. Using these two attributes, the algorithm found five different groupings.



The yellow dots represent an earlier attempt at groupings and are centeroids if the algorithm was searching for ten groupings. However, it is important not to over fit your data so the colors represent a much better fit data model

Supervised Learning

Supervised Learning involves separating data into training and testing subsections and using the testing data to see how well your model does when introduced to new data of the same type. The model is "supervised" because the test set has the actual answers, yielding an accurate way to test your model.

We performed Scikit Learn's Lasso Regression on the Boston Housing Dataset in order to display the performance of the model that we helped document.

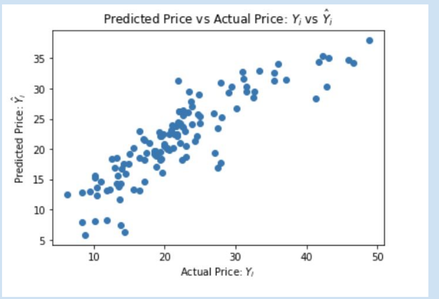
The Boston Housing Dataset is a well-known set with 13 attributes and one numerical target attribute (the price of the house). Here are some of the attributes:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0

Lasso Regression uses shrinkage to minimize the points to one central points. This is used as an alternative to linear regression when there exists collinearity in the dataset. The Housing dataset has many collinear attributes so Lasso is a great model to use on this data.

Performance

After splitting the Boston set into testing and training data, I fit the Lasso model (from scikit) to the training set with an alpha of .1. The alpha numeric is the constant that multiplies the L1 term. This was the best alpha, and when tested against the test dataset, produced an accuracy of 75.8%.



We would like to thank Dr. Jim Bowring, the Computer Science and Data Science Departments of CofC, and our fellow classmates of CSCI 462. It has been a great semester and an even better four years!