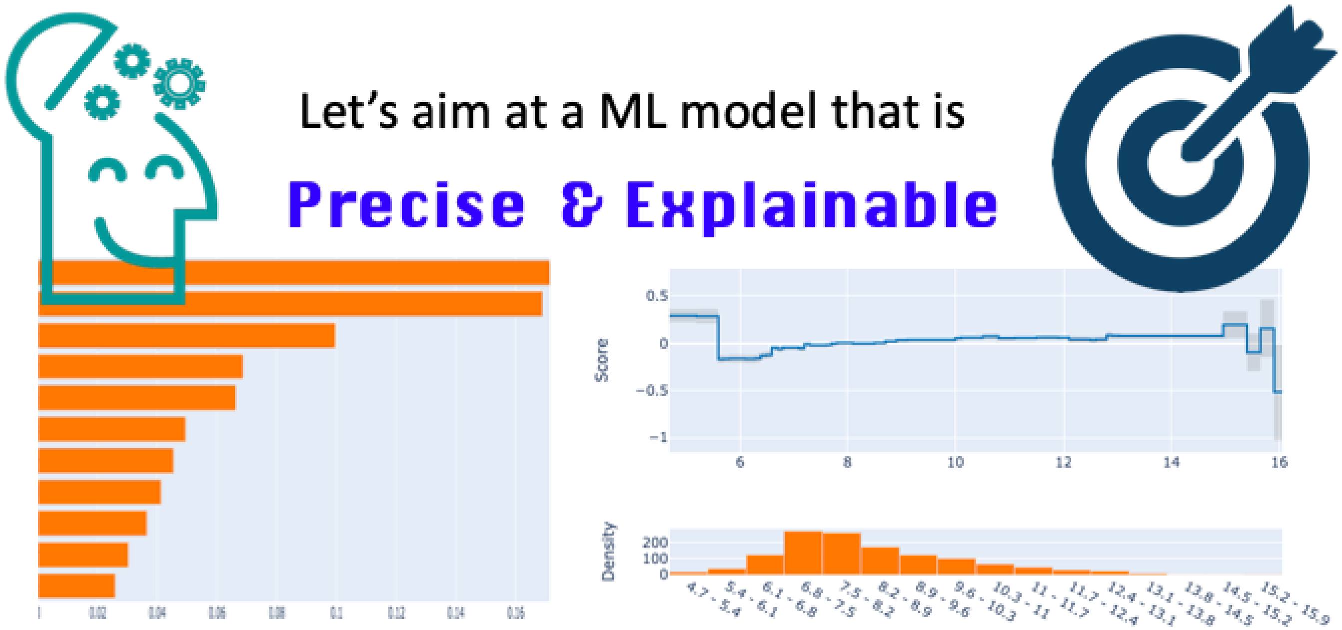
Explain Your Model with Microsoft's InterpretML

Dr. Dataman Following
Feb 27 · 8 min read ★



Model interpretability has become the main theme in the machine learning community. Many innovations have burgeoned. The **InterpretML** module, developed by a team in Microsoft Inc., offers prediction accuracy, model interpretability, and aims at serving as a unified API. Its Github receives active updates. I have written a series of articles on model interpretability, including “Explain Your Model with the SHAP Values”, “Explain Any Models with the SHAP Values — Use the KernelExplainer” and “Explain Your Model with LIME”. I am going to devote this article to introduce you to InterpretML.

In this article, I will provide a gentle mathematical background then show you how to conduct the modeling. You can also jump to the modeling part, then come back to review the mathematical background.

The InterpretML Package

Several salient features worth mentioning here. First, the Microsoft team aims at developing the package to be an ultimate unified framework API, like scikit-learn uniform API that includes all algorithms. Second, the package leverages many libraries like Plotly, LIME, SHAP, SALib so it is already compatible with other modules. Third, the package offers a new interpretability algorithm called *Explainable Boosting Machine (EBM)*, which is based on Generalized Additive Models (GAMs).

GAM is also used in Facebook's open-source "Prophet" module. See "Business Forecasting with Facebook's Prophet".

Model Interpretability Does Not Mean Causality

It is important to point out the model interpretability does not imply causality. To prove causality, you need different techniques. In the "**identify causality**" series of articles, I demonstrate econometric techniques that identify causality. Those articles cover the following techniques: Regression Discontinuity (see "Identify Causality by Regression Discontinuity"), Difference in differences (DiD)(see "Identify Causality by Difference in Differences"), Fixed-effects Models (See "Identify Causality by Fixed-Effects Models"), and Randomized Controlled Trial with Factorial Design (see "Design of Experiments for Your Change Management").

Understand Generalized Additive Models (GAM)

Generalized additive models were originally invented by Trevor Hastie and Robert Tibshirani in 1986. Although GAM does not receive sufficient popularity yet as random forest or gradient boosting in the data science community, it is certainly a powerful and yet simple technique. The idea of GAM is intuitive:

- *Relationships* between the individual predictors and the dependent variable follow smooth patterns that can be linear or nonlinear. Figure (A) illustrates the relationship between *x1* and *y* can be nonlinear.
- *Additive:* these smooth relationships can be estimated simultaneously then added up.

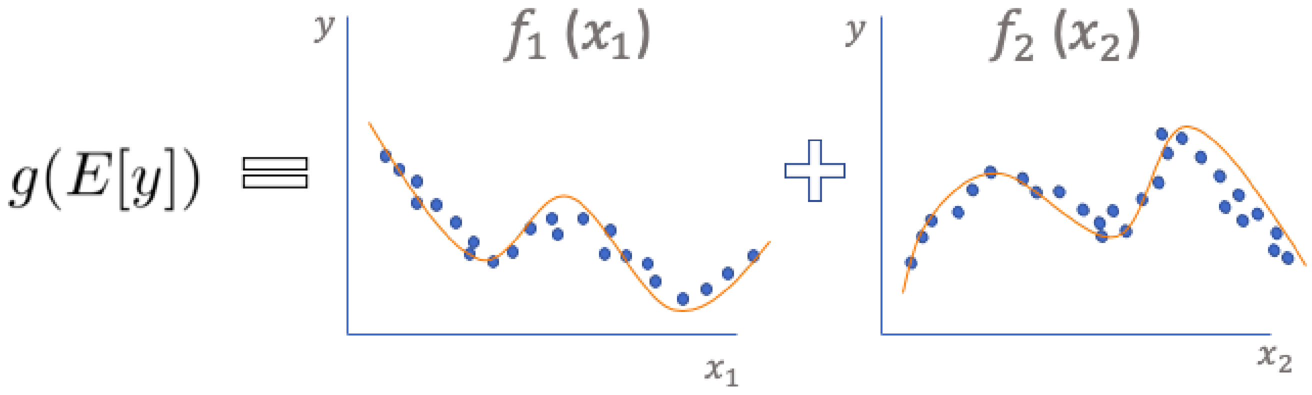


Figure (A)

In Figure (A) the *E(Y)* denotes the expected value. The *link function g()* links the expected value to the predictor variables. The function *f()* is called the smooth or *nonparametric* function. (*Nonparametric* means that the shape of predictor functions is solely determined by the data. In contrast, *parametric* means the shape of predictor functions are defined by a certain function and parameters.) When the function f() becomes linear, GAM reduces to GLM. GLM is easy to interpret, so is GAM.

You may be alarmed by the risks of overfitting because GAM uses smooth functions to fit the data non-linearly. How does GAM overcome this challenge? GAM adds an extra penalty for each smooth term. Typical regularization techniques including LASSO, Ridge or Elastic Net are used. Boosting also performs regularization as part of fitting.

If we summarize the case for GAM, we can say:

- it is easy to interpret.
- it is more flexible in fitting the data, and
- it regularizes the predictor functions to avoid overfitting.

Add the Interaction Terms to GAM for Better Prediction Accuracy

Although a GAM is easy to interpret, its accuracy is significantly less than more complex models that permit interactions. In the seminar paper "Accurate Intelligible Models with Pairwise Interactions" by Lou *et. al.* (KDD-2013), they add interaction terms to the standard GAMs and call it *GA2M* — Generalized Additive Models plus Interactions. As a result, GA2M greatly increases the prediction accuracy but still preserves its nice interpretability.

$$\textbf{GAM:}\quad g(E[y]) = \beta_0 + \sum f_j(x_j)$$

$$\textbf{GA}^2\textbf{M:}\quad g(E[y]) = \beta_0 + \sum f_j(x_j) + \overbrace{\sum f_{i_j}(x_i, x_j)}^{\text{Pairwise interaction terms}}$$

The Explainability Boosting Machine (EBM)

Although the pairwise interaction terms in GA2M increase accuracy, it is extremely time-consuming and CPU-hungry. How does EBM solve the computational problem? First, it learns each smooth function *f()* using machine learning techniques such as bagging and gradient boosting (that's the name Boosting in EBM). Second, each feature is tested against all other features like a round-robin tournament. In this way, it can find the best feature function *f()* for each feature and shows how each feature contributes to the model's prediction for the problem. Third, EBM develops the GA2M algorithm in C++ and Python and takes advantage of joblib to provide multi-core and multi-machine parallelization.

InterpretML — A One-Stop Shop

In a modeling project, you explore the data, train the models, compare the model performance, then examine the predictions globally and locally — you get it all in the InterpretML module. It is a one-stop shop and easy to use. However, I want to remind you no machine can replace the creativity of feature engineering. Check "A Data Scientist's Toolkit to Encode Categorical Variables to Numeric", "Avoid These Deadly Modeling Mistakes that May Cost You a Career", "Feature Engineering for Healthcare Fraud Detection", and "Feature Engineering for Credit Card Fraud Detection". Or you can bookmark "Dataman Learning Paths — Build Your Skills, Drive Your Career" for all articles.

Let me show you in the following (A) — (F) steps.

- (A) **Explore** the Data
- (B) **Train** the Explainable Boosting Machine (EBM)

- (C) **Performance:** How Does the EBM Model Perform?
- (D) **Global** Interpretability — What the Model Says for All Data
- (E) **Local** Interpretability — What the Model Says for Individual Data
- (F) **Dashboard:** Put All in a Dashboard — This is the Best

First do `pip install -U interpret` to install the module.

I will use the same red wine quality data so you can compare SHAP, LIME, and InterpretML, as I have been doing in "Explain Your Model with the SHAP Values", "Explain Any Models with the SHAP Values — Use the KernelExplainer" or "Explain Your Model with LIME". The target value of this dataset is the quality rating from low to high (0–10). The input variables are the content of each wine sample including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates and alcohol. There are 1,599 wine samples.

```
import pandas as pd
import numpy as np
np.random.seed(0)
df = pd.read_csv('/winequality-red.csv') # Load the data
from sklearn.model_selection import train_test_split
```

```
Y = df['quality'] # The target variable is 'quality'
X =  df[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar','chlorides', 'free
X_featurenames = X.columns
# Split the data into train and test data:
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
```

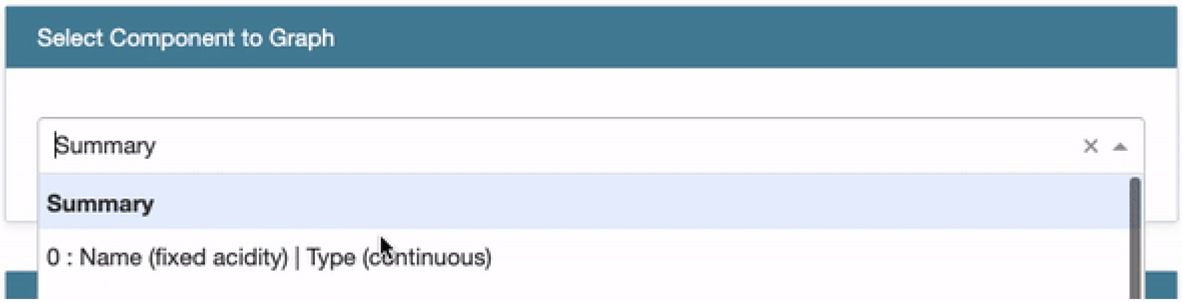
load hosted with ♡ by GitHub view raw

(A) Explore the Data

```
from interpret import show
from interpret.data import Marginal
marginal = Marginal().explain_data(X_train, Y_train, name = 'Train Data')
show(marginal)
```

explore hosted with ♡ by GitHub view raw

The outcome is a drop-down menu for the "Summary" and each variable. Click the "Summary", it presents the histogram of the target variable.



Choose the first variable "Fixed Acidity". It shows the Pearson Correlation with the target variable, followed by the histogram of "Fixed Acidity" in blue color, and the histogram of the target variable in red color.

(B) Train the Explainable Boosting Machine (EBM)

Besides building the EBM model, I also build a linear regression and a regression tree model for comparison. The `ExplainableBoostingREgressor()`

uses all the default hyper-parameters as shown in the output. You can specify any of the hyper-parameters.

```
from interpret.glassbox import ExplainableBoostingRegressor, LinearRegression, RegressionTree

lr = LinearRegression(random_state=seed)
lr.fit(X_train, Y_train)

rt = RegressionTree(random_state=seed)
rt.fit(X_train, Y_train)

ebm = ExplainableBoostingRegressor(random_state=seed)
ebm.fit(X_train, Y_train)
# For Classifier, use ebm = ExplainableBoostingClassifier()

```

EBM hosted with ♡ by GitHub view raw

```
ExplainableBoostingRegressor(binning_strategy='uniform', data_n_episodes=2000,
                             early_stopping_run_length=50,
                             early_stopping_tolerance=1e-05,
                             feature_names=['fixed acidity', 'volatile acidity',
                                            'citric acid', 'residual sugar',
                                            'chlorides', 'free sulfur dioxide',
                                            'total sulfur dioxide', 'density',
                                            'pH', 'sulphates', 'alcohol'],
                             feature_step_n_inner_bags=0,
                             feature_types=['c...ntinuous',
                                            'continuous', 'continuous',
                                            'continuous', 'continuous',
                                            'continuous', 'continuous',
                                            'continuous', 'continuous',
                                            'continuous'],
```

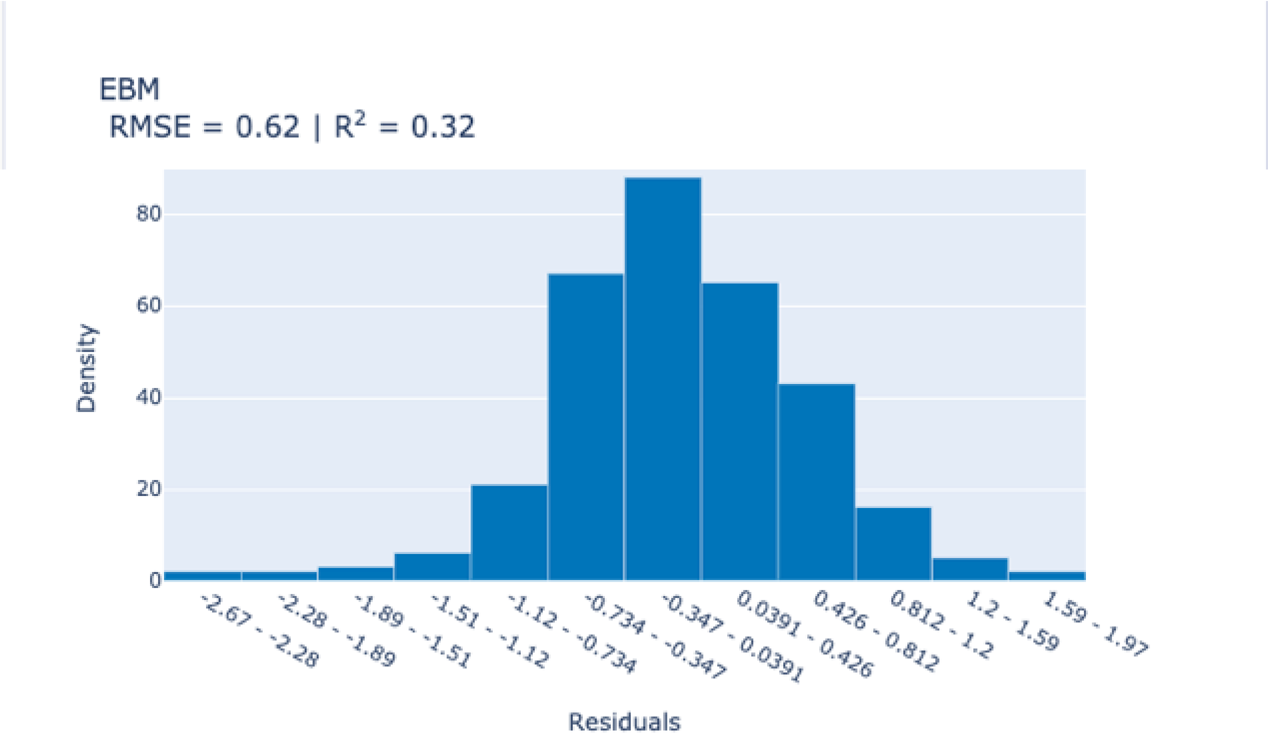
```
holdout_size=0.15, holdout_split=0.15,
interactions=0, learning_rate=0.01,
main_attr='all', max_tree_splits=2,
min_cases_for_splits=2, n_estimators=16, n_jobs=-2,
random_state=1234, schema=None, scoring=None,
training_step_episodes=1)
```

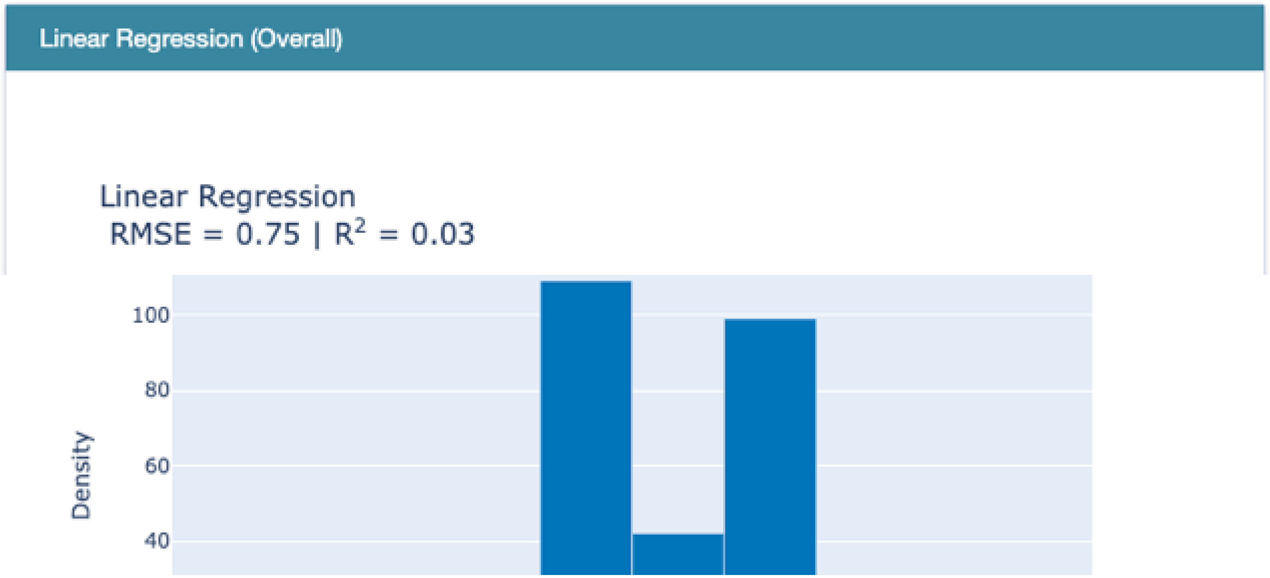
(C) How Does the EBM Model Perform?

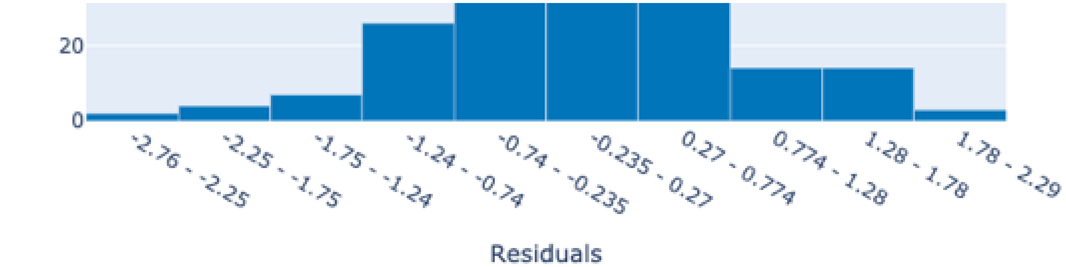
Use `RegressionPerf()` to assess the performance of each model on the test data. The R-squared value of EBM is 0.32 which outperforms those of linear regression model and regression tree model.

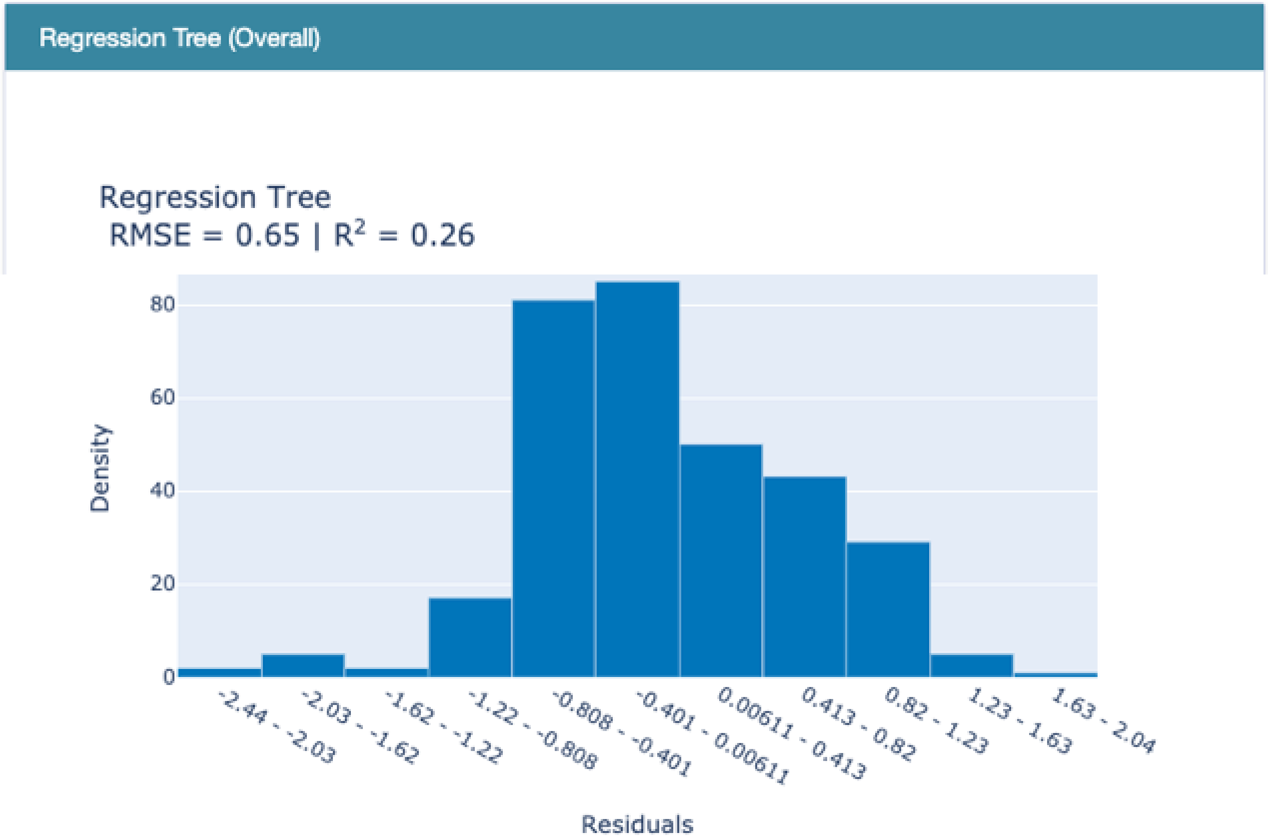
```
from interpret import show
from interpret.perf import RegressionPerf

ebm_perf = RegressionPerf(ebm.predict).explain_perf(X_test, Y_test, name='EBM')
lr_perf = RegressionPerf(lr.predict).explain_perf(X_test, Y_test, name='Linear Regression')
rt_perf = RegressionPerf(rt.predict).explain_perf(X_test, Y_test, name='Regression Tree')
show(ebm_perf)
show(lr_perf)
show(rt_perf)
```

perf hosted with ♡ by GitHub view raw


EBM
RMSE = 0.62 | R2 = 0.32
Density
80
60
40
20
0
-2.67 - -2.28
-2.28 - -1.89
-1.89 - -1.51
-1.51 - -1.12
-1.12 - -0.734
-0.734 - -0.347
-0.347 - 0.0391
0.0391 - 0.426
0.426 - 0.812
0.812 - 1.2
1.2 - 1.59
1.59 - 1.97
Residuals


Linear Regression (Overall)
Linear Regression
RMSE = 0.75 | R2 = 0.03
Density
100
80
60
40



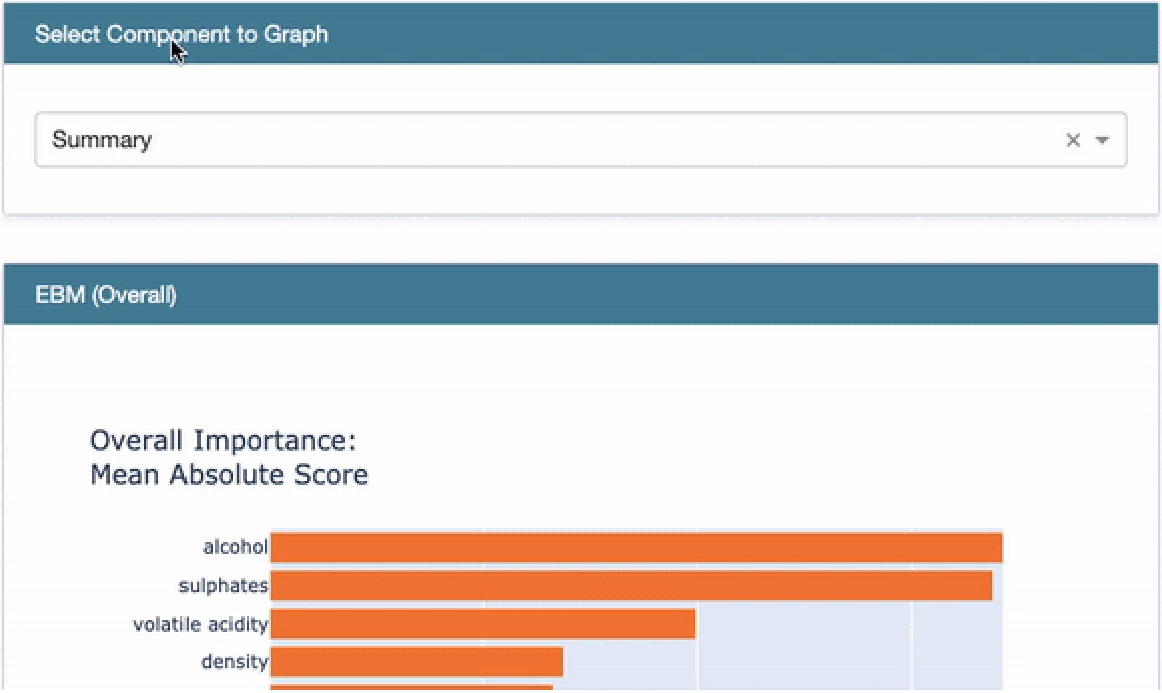


(D) Global Interpretability — What the Model Says for All Data

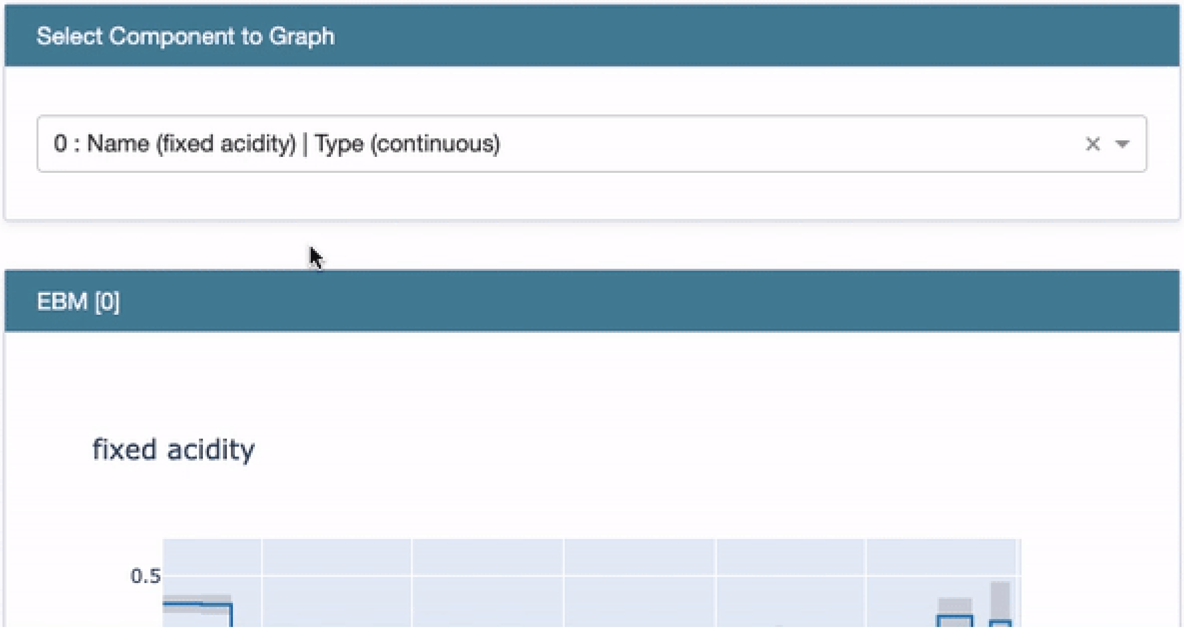
```
ebm_global = ebm.explain_global(name='EBM')
show(ebm_global)
```

global hosted with ♡ by GitHub view raw

Choose "Summary" to show the overall variable importance ranked in descending order (orange color).



Choose the first variable "Fixed Acidity". Two plots show up: the Partial Dependent Plot (PDP) and the histogram of "Fixed Acidity". The histogram indicates most of the values are between 6.0 to 10.0. The PDP presents the marginal effect of the feature on the predicted outcome of a machine learning model (J. H. Friedman 2001). It tells whether the relationship between the target and a feature is linear, monotonic or more complex. In this example the PDP shows there is a very mild linear and positive trend between "Fixed Acidity" and the target variable when "Fixed Acidity" is between 6.0 to 10.0.



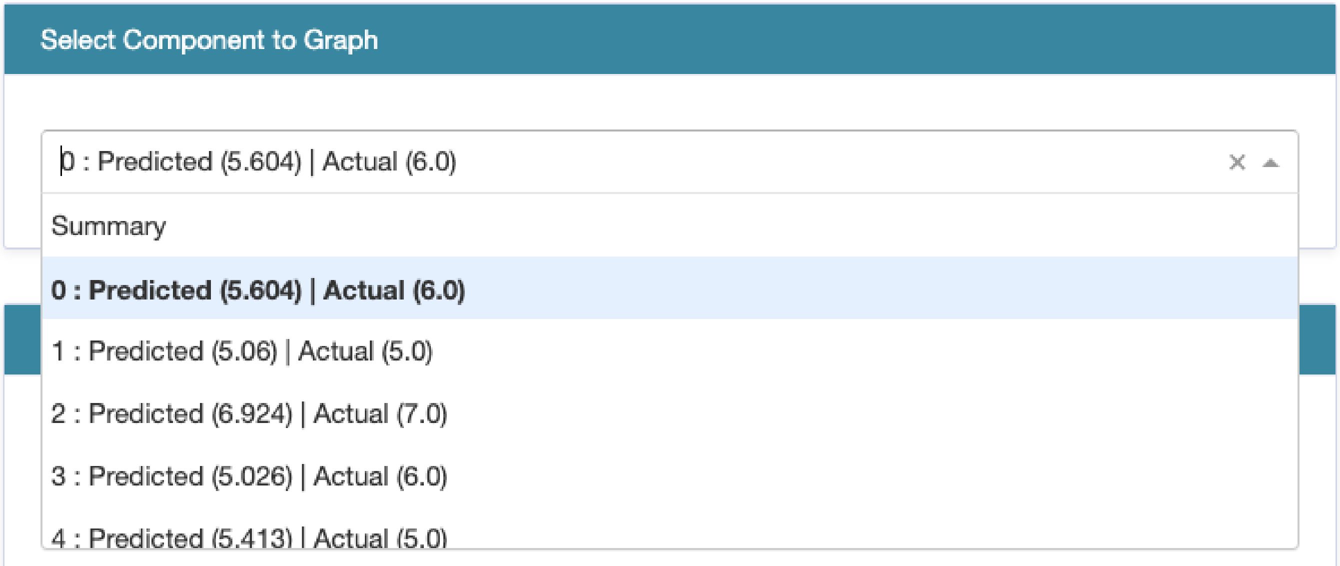
(E) Local Interpretability — What the Model Says for Individual Data

Let's take the first five observations.

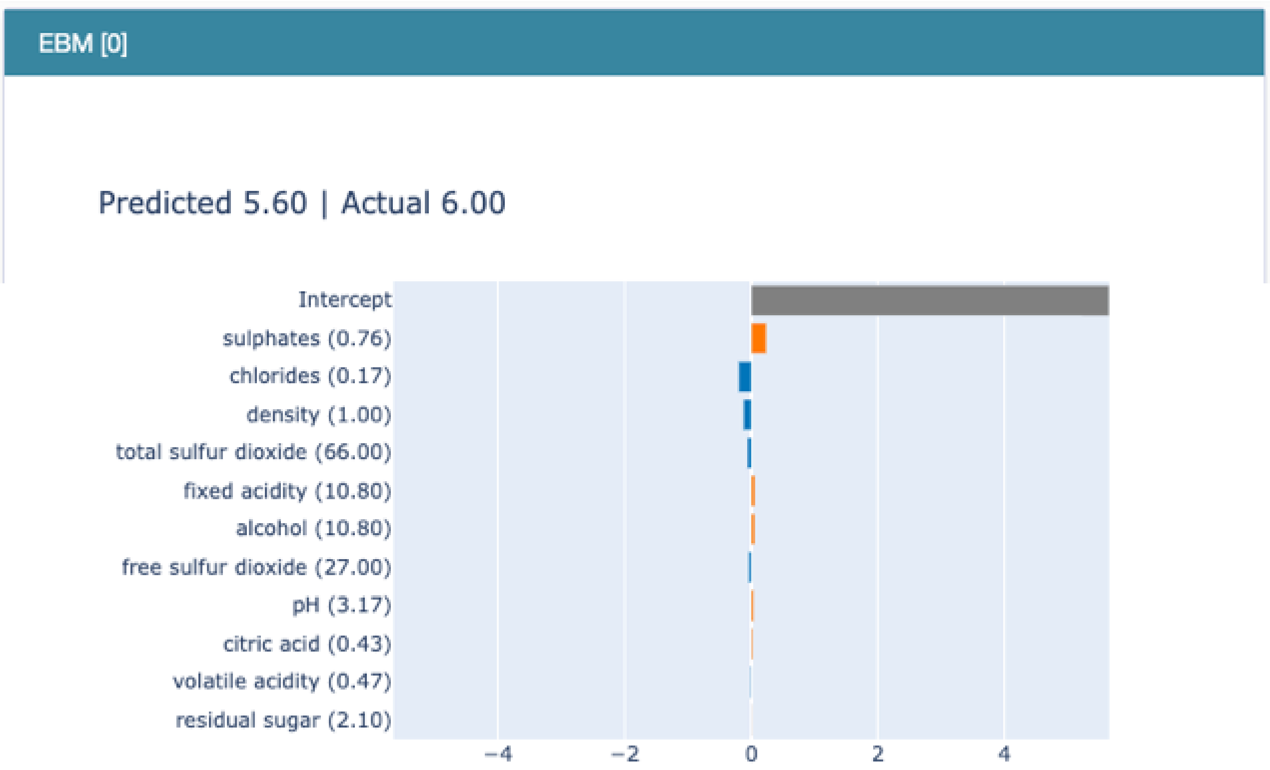
```
ebm_local = ebm.explain_local(X_test[:5], Y_test[:5], name='EBM')
show(ebm_local)
```

local hosted with ♡ by GitHub view raw

The drop-down menu lists the predicted value and the actual value for each record.



We choose the first record. The value of "Sulphates" is 0.76, and that of "Chlorides" is 0.17, and so on. The contributions of all variables for this record are ranked in descending order as below. "Sulphates" positively contributes to the target "quality", while "Chlorides", "Density", etc. negatively contributes to the target. Because EBM is a GAM-like model, the prediction is the sum of all the coefficients.



(F) Put All in a Dashboard — This is the Best

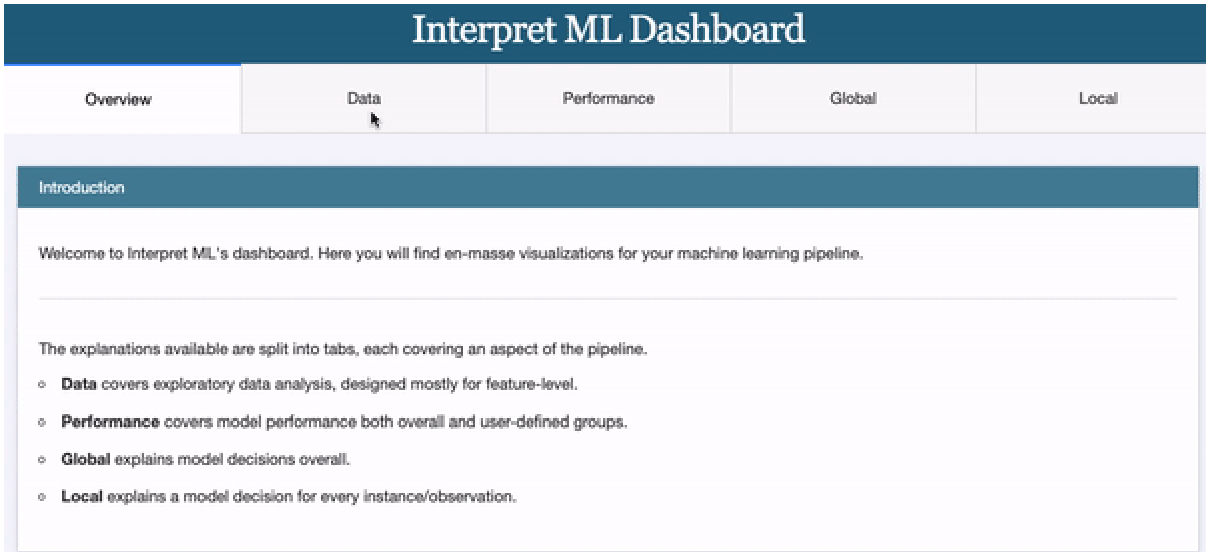
All of the above can be put together in an elegant dashboard. Simply use a list to contain all the elements in the `show()` function:

```
show([marginal, lr_global, lr_perf, rt_global, rt_perf, ebm_perf, ebm_global, ebm_local])
```

dashboard hosted with ♡ by GitHub view raw

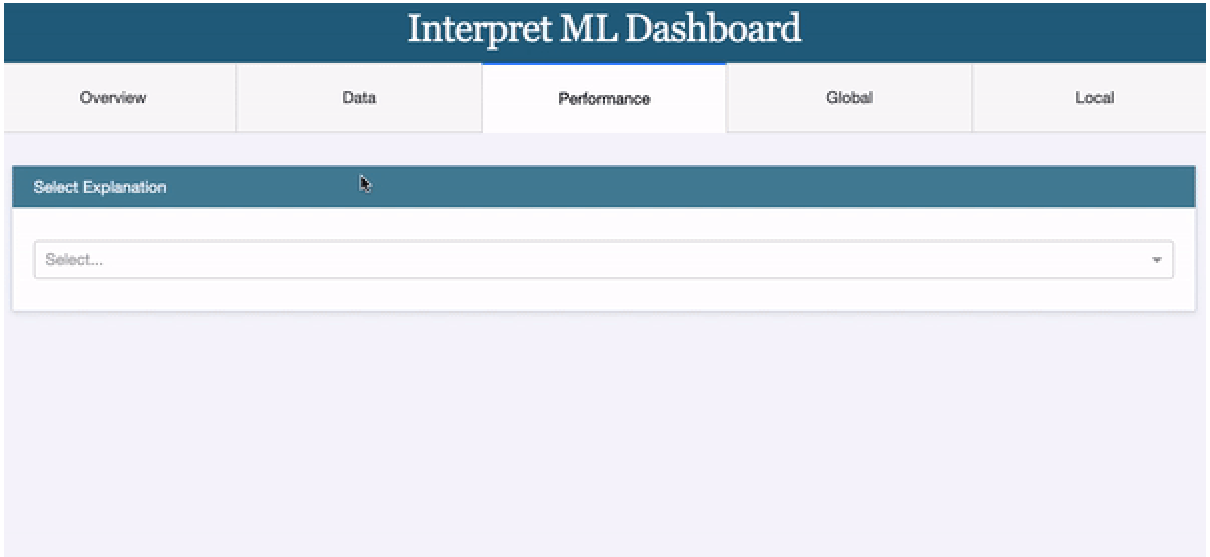
The dashboard's title is "Interpret ML Dashboard". It has five tabs. The first tab "Overview" is an introductory page. The second tab "Data" presents the same plots as described above in the "(A) Explore the Data" section.

The "Data" Tab:



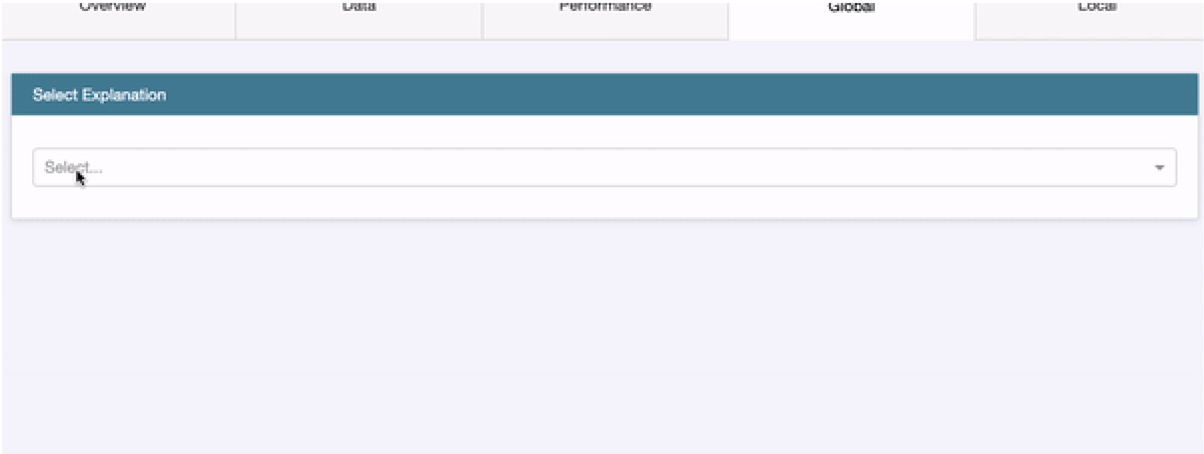
The "Performance" Tab:

The third tab "Performance" presents the same plots as described above in the "(C) How Does the EBM Model Perform" section.



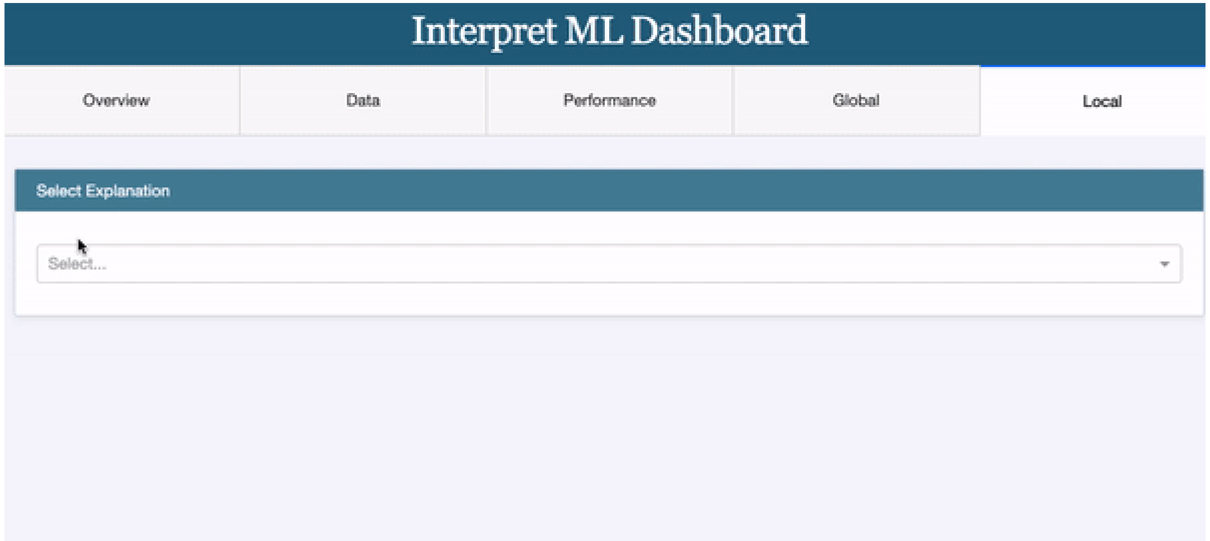
The "Global" Tab:

The fourth tab "Global" presents the same plots as described above in the "(D) Global Interpretability" section.



The "Local" Tab:

The fifth tab "Local" presents the same plots as described above in the "(E) Local Interpretability" section.



For your convenience, I put all the code lines in one block:

Data Science Machine Learning Python