Review of SHAP interpreting method

Contents

Introduction

AI, machine learning has been used for predicting result in many areas, and it is involving fast. However, there are still some good reason people refuse to use it. One would be interpretation issue. When people do not understand or have doubt in the predicted result, people cannot use the model no matter how accurate the model predicts. Some of models like linear regression, KNN, single decision tree is self-explainable. However other models such as Random Forest, neural network is not easy to understand by non-technical people or simply not possible to explainable information in a multi layered neural network model. We need to use some other algorithm to show people the reason behind the 'black box'. SHAP stands for Shapley Additive exPlanations, it was originally explained by Scott M. Lundberg and Su-In Lee in 'A Unified Approach to Interpreting Model Predictions'(Lundberg, Lee 2017). It is an algorithm that use a measure called shapley values to interpret the feature importance in a complex model.

This review is to prove SHAP can be a great interpreting tool to explain complex model in an easy and efficient way. The dataset we can using is Boston Housing Dataset. The variable description as below:

Table 1 Variable description in Boston Housing Dataset (PERERA 2017)

Column name	Description
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000

PTRATIO	pupil-teacher ratio by town
B	1000(Bk - 0.63) ^2 where Bk is the proportion of blacks by town
LSTAT	% lower status of the population
MEDV	Median value of owner-occupied homes in $1000's

SHAP interpretation for a Tree ensemble example

We are going to use all variables except last column to predict MEDV (median house price). We first train our model by using XBGRegressor.

```
import xgboost
!pip install shap
import shap

# train an XGBoost model
X, y = shap.datasets.boston()
model = xgboost.XGBRegressor().fit(X, y)
```

Note: Since we are not going to test the model performance, so no train/test split in the code.

Before we start use SHAP to explain the model, we need to understand what is *shapley value* and how *shapley values* is **calculated**.

The original formular given by Lundberg and Lee (2017):

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right].$$

Where F is the set of all features, $f_{sU\{i\}}$is trained with that feature present while f_s is trained with the feature was not in present. x_s represents the values of the input features in the set S.

We are going to look at an example introduced by Chu Lan (2022).

The concept of SHAP was used in game theories in 1950. So, think it like game, all the independent variables are players, the target variable are the score, each player have different contribution to the final score, the higher contribution the more importance they are. So, we need to find the contribution value and that is *shapley* value.

Secondly, to get the *shapley* value, we need to compute weighted sum of each feature 's contribution in each combination. The contribution value is the score difference between the player in the combination and not in it. Take 3 player A, B, and C for example, the weighted sum of A will be:

$$\omega_1\big(f(x_A,-,-)-f(-,-,-)\big)+$$
$$\omega_2\big(f(x_A,x_B,-)-f(-,x_B,-)\big)+$$
$$\omega_3\big(f(x_A,-,x_C)-f(-,-,x_C)\big)+$$
$$\omega_4\big(f(x_A,x_B,x_C)-f(-,x_B,x_C)\big)$$

Equation 1 Shapley value is weighted sum of score difference of all combinations (Chu 2022)

W is the weight of each combination have appearance of A.

Thirdly, we need to find the W(weight), the following formular shows how we calculate weight:

$$\omega = -\frac{(|S|-1)!(3-|S|)!}{3!}$$

Equation 2 Weight calculation (Chu 2022)

S is the number of players in the combination when A is in the game. So, S values for w1 to w4 will be 1,2,2,3 and back into the equation 2, we get the weight w1 to w4 is 1/3, 1/6, 1/6, 1/3.

Lastly, we need to find the scores for each needed combination. It's easy to find out the score when features in the combination (for example f (xA, xB, xC)), the hard part is when a feature is not in the combination (for example f(-, xB,xC)). When feature A is not in the combination, we need fix all the other feature's value (xB, xC), then put all column A values into the equation to get an average value, this value will be the expected score when feature A is not in the combination.

If we take our Boston dataset as example,

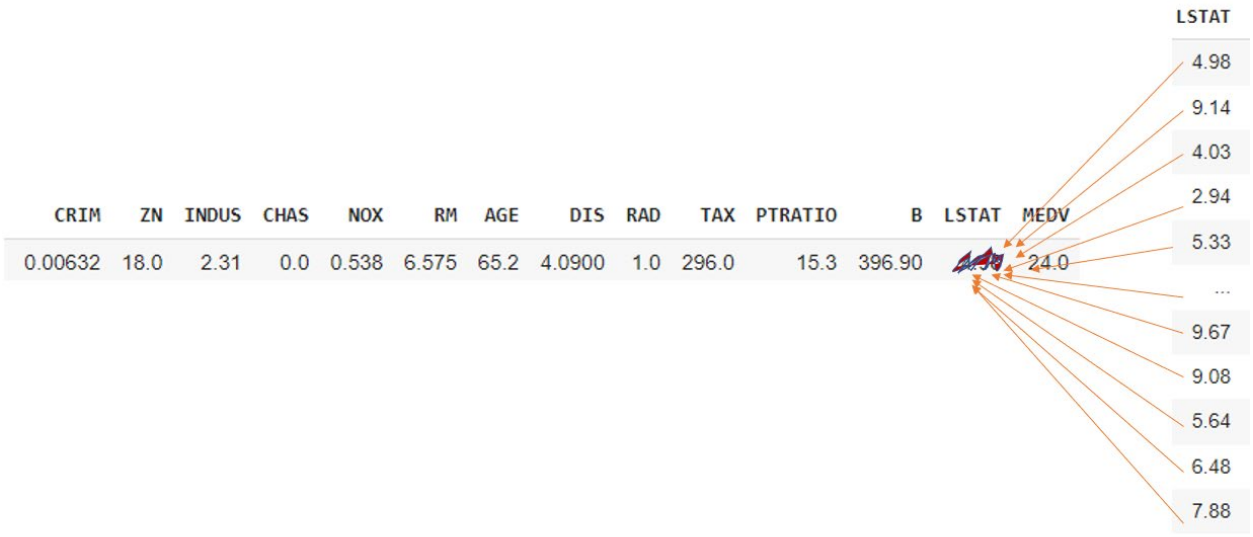

Figure 1 Calculate LSTAT score for a combination when LSTAT is not in it. Replace original value with each value in LSTAT column, get the average score (the instance used in this demonstration is just a random picked sample).

When 2 or more features are not in the combination, the calculation will be more complex, but the idea is the same. With the scores and weights ready, we can get our final shaley value for feature A.

Now we use SHAP to **explain** the first prediction:

```
explainer = shap.Explainer(model)
shap_values = explainer(X)

# visualize the first prediction's explanation
shap.plots.waterfall(shap_values[0])
```

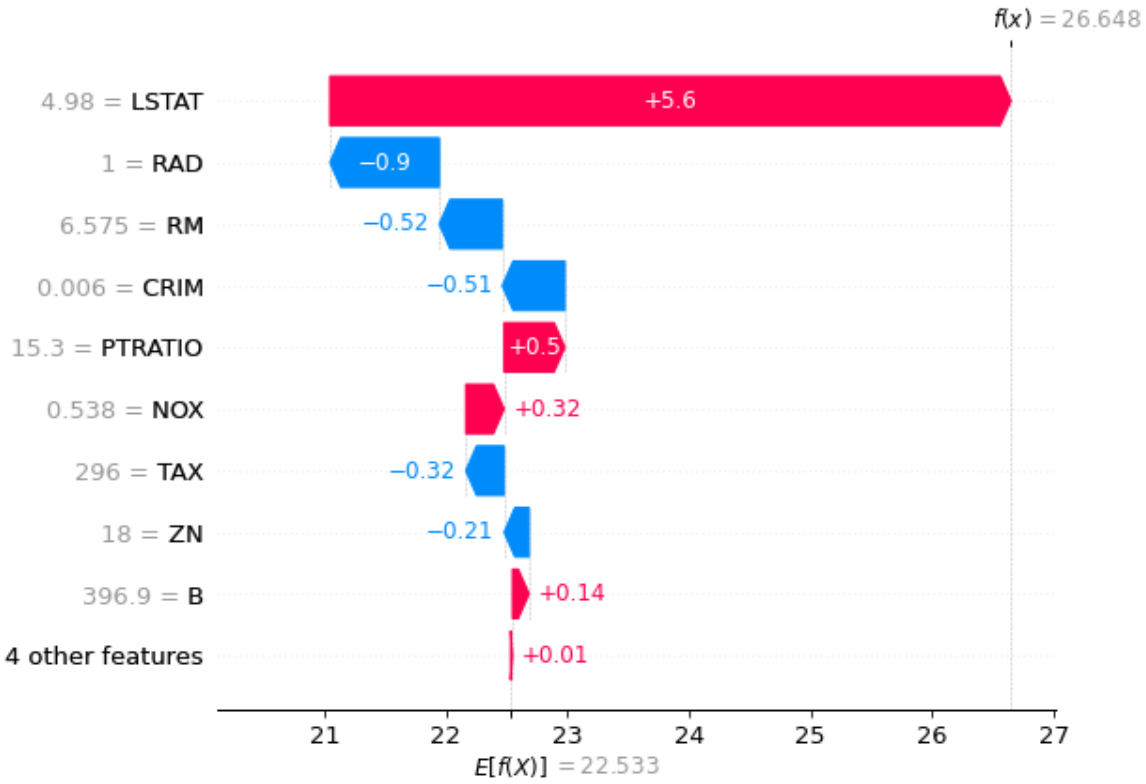

Figure 2 Waterfall plot for shapley values for Boston tree ensemble model

This waterfall plot (Figure 2) shows the features' contribution to the first prediction, red means positive contributes to the prediction value, blue means the opposite. The number down the bottom 22.533 is the base value (average output value), the number on top right 26.648 is the predicted value. The number on the left of the variable name is actual value of the variable in the predicted row. The numbers with positive/negative sign are the *shapley* values for features in this instance.

There is another way to visualize the feature contribution (Figure 3). The features have been organized horizontally to demonstrate how the base value (average output value) got pushed to the predicted value.

```
# visualize the first prediction's explanation with a force plot
shap.initjs()
shap.plots.force(shap_values[0])
```

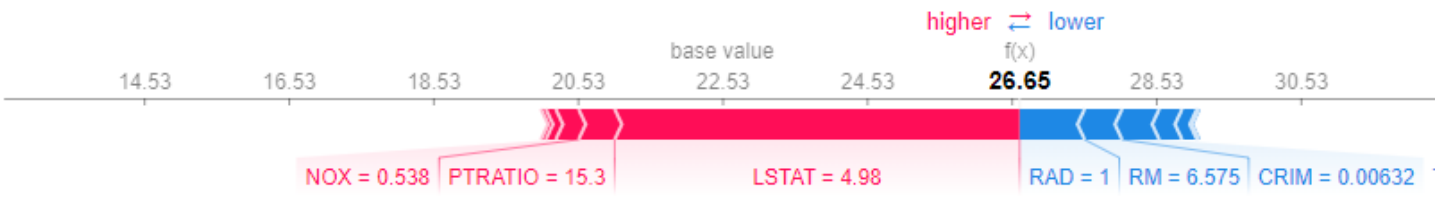

Figure 3 this stacked chart shows how features pushed the base value (average output value for a instance) to the predicted value

At this point, we only see how features impact a single instance, what about the feature contribution to all the instances? Sure, we can also form a visual for all predictions.

```
# visualize all the training set predictions
shap.initjs()
shap.plots.force(shap_values)
```

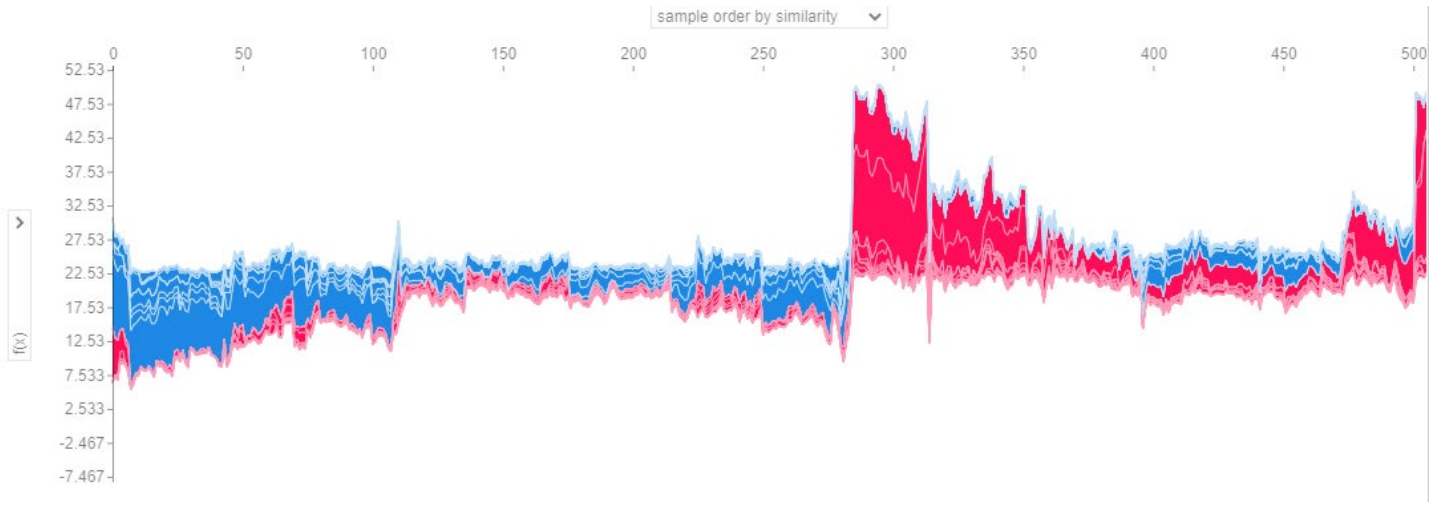

Figure 4 Stream graph shows features' contribution for all predictions.

This interactive streamgraph (figure 4) shows the individual feature's contribution (*shapley value*) for predictions in all instances. The horizontal axis are instances in the data. Vertical axis is the value of predictions (the middle value is average output value 22.53). We can move over to see feature contribution details in a specific instance.

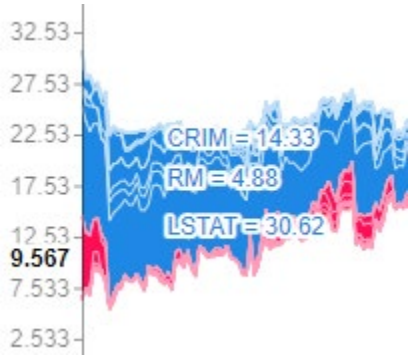

Figure 5 Mouse over to see feature contribution details for a specific instance

Additional filter on the left side of the graph gives us the option to choose only to show specific feature (as show in figure 6).

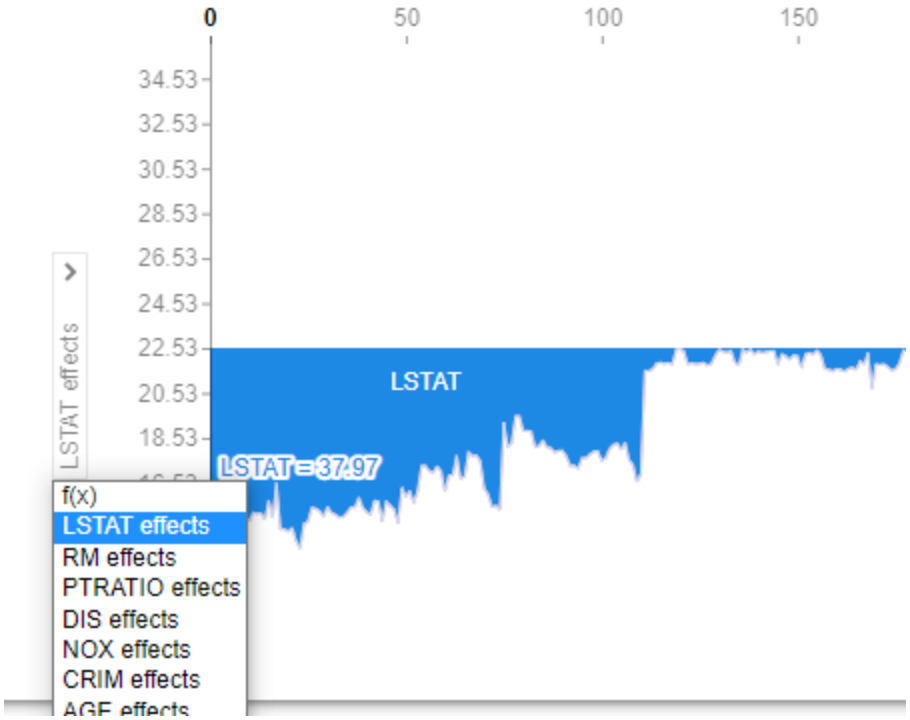

Figure 6 filter option for choosing specific feature

The filter on the top side of the graph is to choose the sorting method for showing the samples (see figure 7 below).

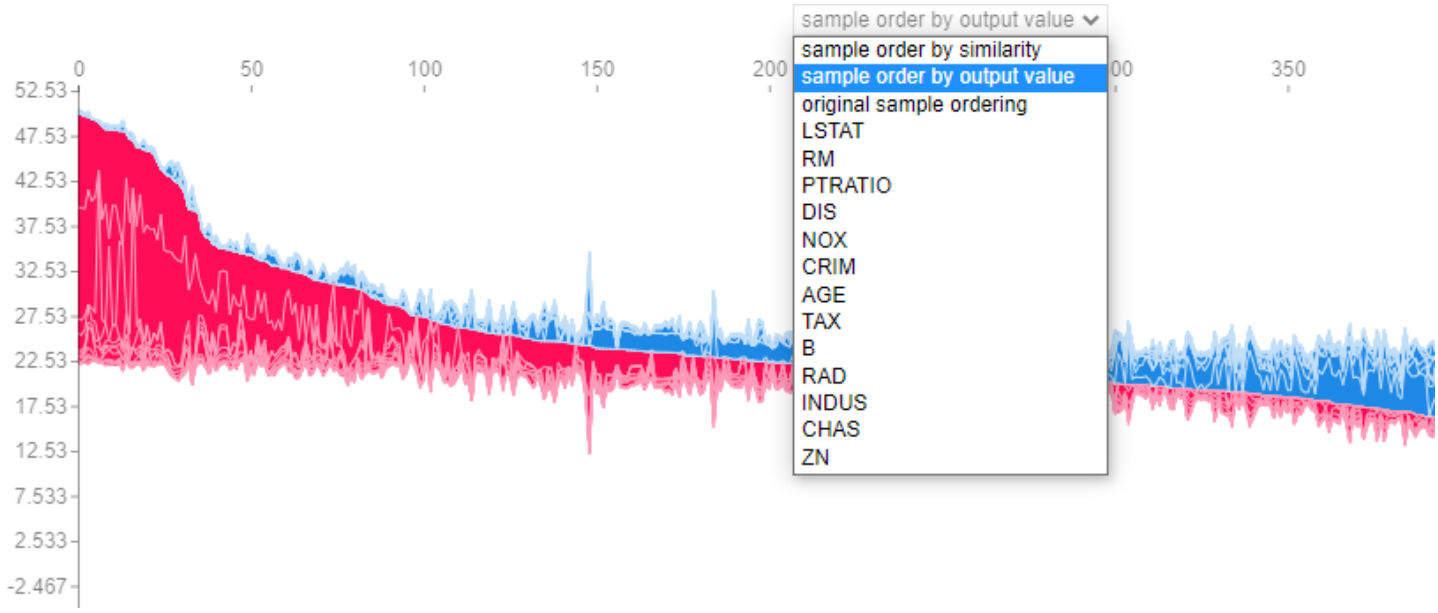

Figure 7 Fitters for sorting the sample display order.

There are other ways to show feature contributions. Figure 8 shows the feature contribution on each observation(dots) ranking from the top. Figure 9 shows the ranking for average *shapley* value.

```
# summarize the effects of all the features
shap.initjs()
shap.plots.beeswarm(shap_values)
```

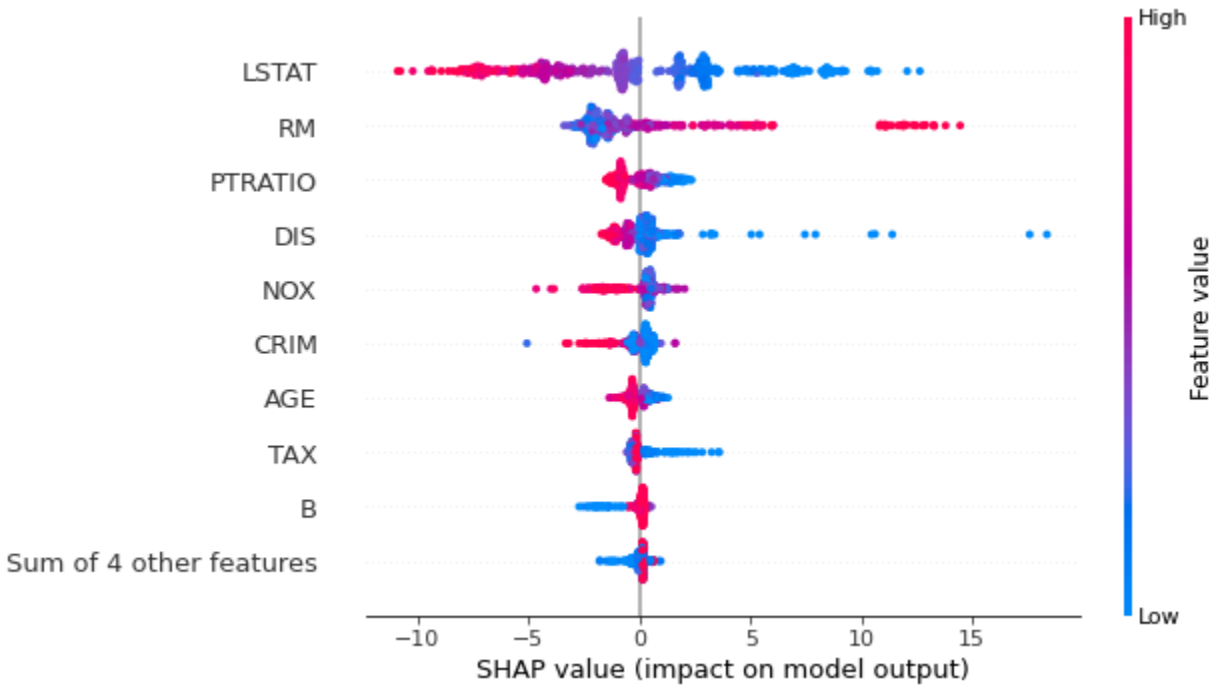

Figure 8 feature impact on all features, each dot is one observation.

```
shap.initjs()
shap.plots.bar(shap_values)
```

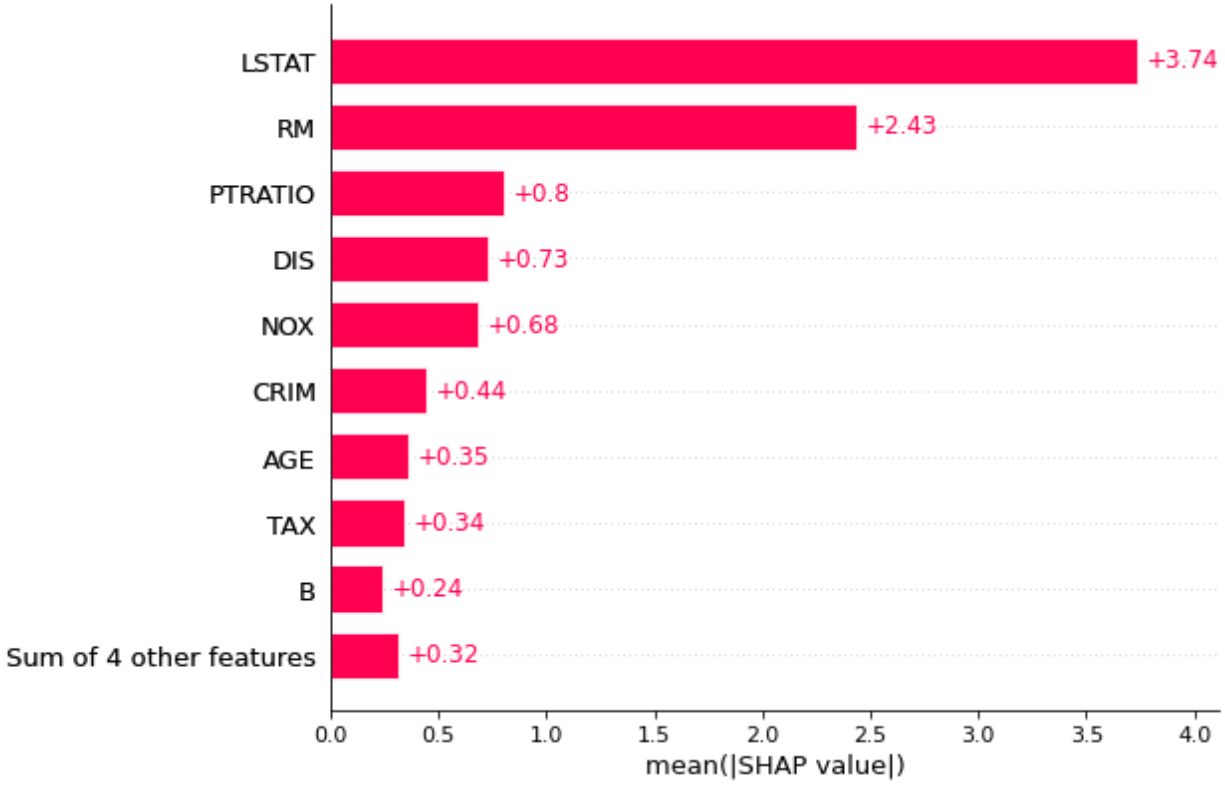

Figure 9 mean feature contribution (SHAP value) for top features

Kernel SHAP (Linear LIME + Shapley values)

The dataset we demonstrated is a small dataset, what if we encounter a large data with hundred columns and thousand rows. If we have a dataset with 100 features and 1000 rows, there will be 2^{100} combinations, for each combination with absent features, we will need another calculation $O(1000^n)$, n is the absent feature we need to consider. In this situation, approximation method needed.

For **Model-Specific** approximations, Lundberg and Lee (2017) explained the following methods:

- Linear SHAP: assume input feature independence, SHAP values can be extract from the weight coefficients of the model.
- Max SHAP: calculate the probability of a input will increase the maximum value compare to other input. This will decrease the calculation from $O(M2^M)$ to $O(M^2)$.
- Deep SHAP (DeepLIFT + Shapley values): try to use the knowledge of deep network to gain computational performance. Deep SHAP first calculate the SHAP values for smaller components of the network, then combine them to get the SHAP values for the whole network.

In this review we will use a **Model-Agnostic approximation** called **Kernel SHAP** (Linear LIME + Shapley values).

Linear LIME uses a linear model to explain a local area in the original model. Kernel SHAP recovers the shapley values from the Linear LIME, it is given by Lundberg, Lee (2017):

$$\pi_{x'}(z') = \frac{(M-1)}{(M \text{ choose } |z'|)|z'|(M-|z'|)},$$

Equation 3 Kernel SHAP recover shapley values from Linear LIME (Sukumar 2020)

where $\pi_{x'}$ is the local kernel, M is the number of features, $|Z'|$ is the number of non-zero features in the simplified input Z'.

We will still use the same dataset (Boston housing dataset) to demonstrate this method.

```
import sklearn
import numpy as np
!pip install shap
import shap
import time

X,y = shap.datasets.boston()

shap.initjs()
```

We use Neural network for the original model.

```
def print_accuracy(f):
    print("Root mean squared test error = {0}".format(np.sqrt(np.mean((f(X) - y)**2))))
    time.sleep(0.5) # to let the print get out before any progress bars
```

```
from sklearn.neural_network import MLPRegressor
nn = MLPRegressor(solver='lbfgs', hidden_layer_sizes=(8,8,8), random_state=0)
nn.fit(X,y)
```

```
print_accuracy(nn.predict)
```

```
Root mean squared test error = 4.8919544581594545
```

```
explainer = shap.KernelExplainer(nn.predict, data = shap.kmeans(X,10))
shap_values = explainer.shap_values(X)
```

Noticed the data we passed here is shap.kmeans(X,10), this is because the background dataset here is to train the surrogate model, when we have really large data, we use kmeans function to summarize the dataset.

Now we explain the first prediction:

```
shap.initjs()

shap.force_plot(explainer.expected_value,shap_values[0])
```

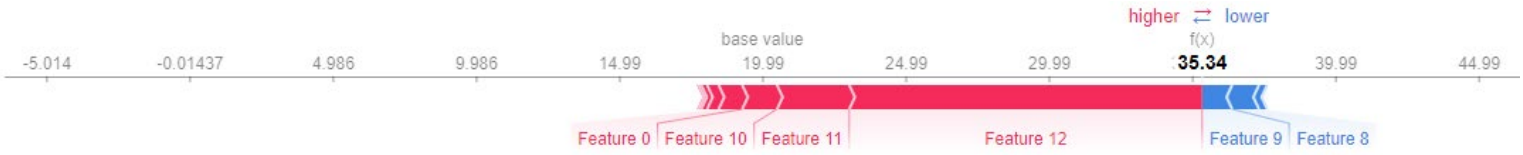

Figure 10 Kernel SHAP explainer: horizontal stacked chart shows how features pushed the base value to the prediction in the first sample

Like the default SHAP explainer, Kernel SHAP explainer shown in Figure 10, the red and blue shows feature importance, they pushed the base value to the predicted value.

```
shap.summary_plot(shap_values, X)
```

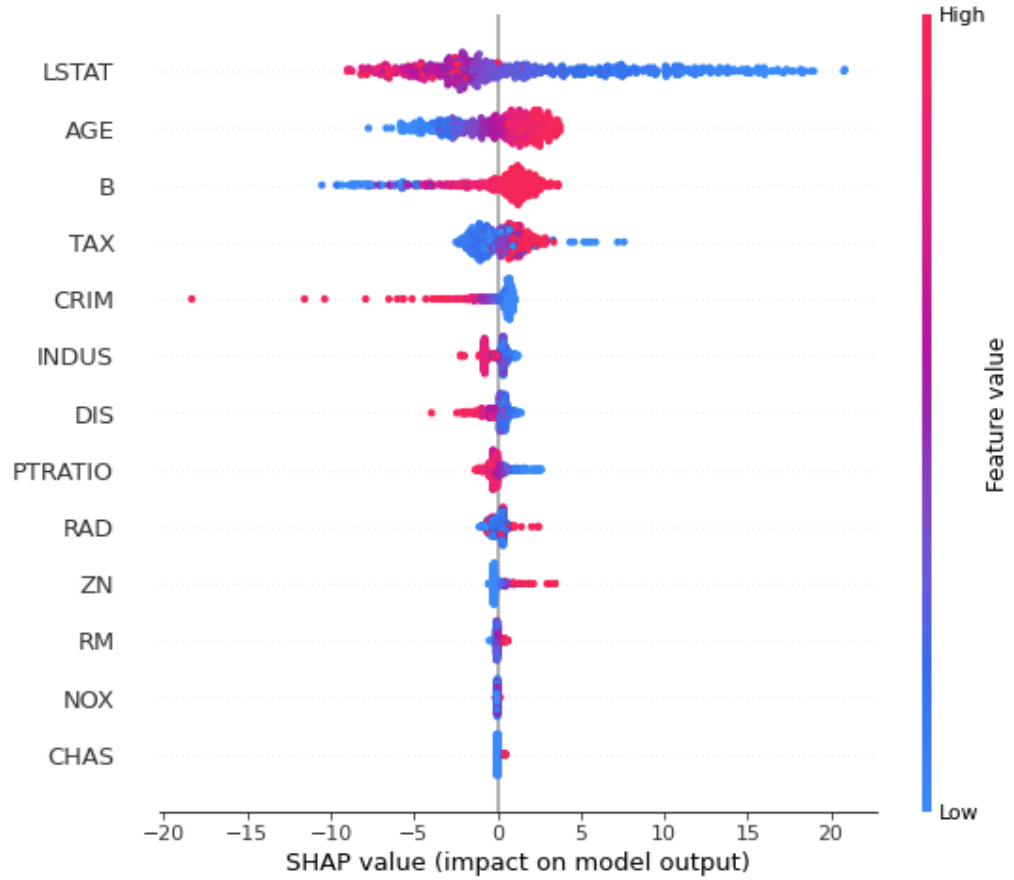

Figure 11 Summary plot for Kernel SHAP

The summary plot for Kernel SHAP (figure 11) shows that the most important feature is LSTAT (same as the default SHAP model as shown in figure 8), but the ranks for other features are different, this is because we used neural network this time.

If we plot the whole dataset, change filter option to original sample ordering, we can see that our kernel SHAP (figure 12) and default SHAP explainer (figure 13) give similar result for shapley values in all predictions.

```
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values, X)
```

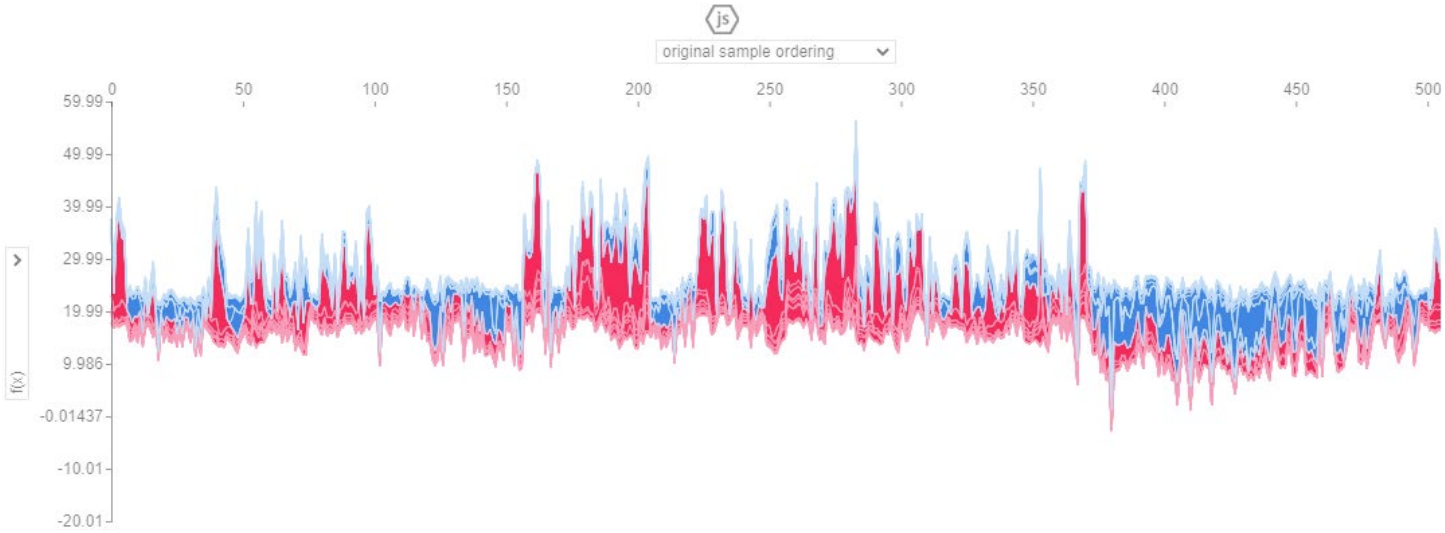

Figure 12 Kernel SHAP explainer for all predictions

```
shap.initjs()
shap.plots.force(shap_values)
```

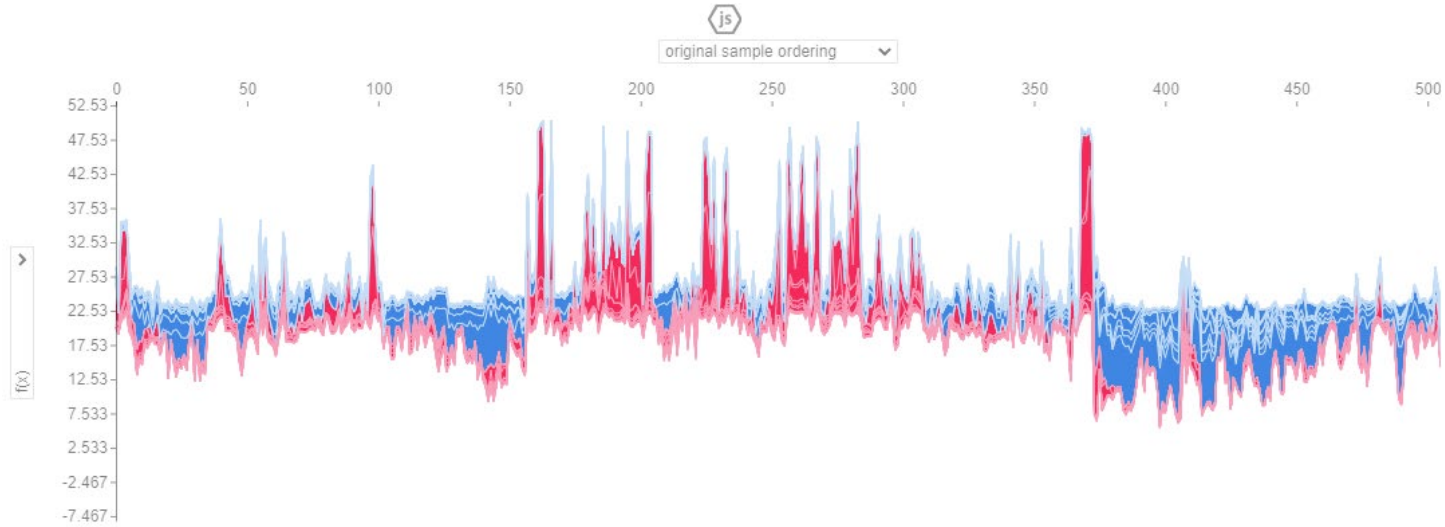

Figure 13 Default SHAP explainer for all predictions

Conclusion

SHAP provide a simple way to explain a complex mode by using Shapley value. This review report gives details on how SHAP works and how to present visualization for feature analysis. From the *shapley* value, we can see that different features are having different impact on a prediction, this is not only good for an analyst to better understand the result of the model, but also provide a way to demonstrate the meaning behind the scenes to a non- technique person. Other than SHAP explainers which are demonstrated in this review, there are more options such as *TreeExplainer*, *GradientExplainer*, *AdditiveExplainer*. SHAP also provide varieties of plots to help visualize our explanation. The more people understand a model, the more chance they will use it. Building trust between people and AI (machine leaning) is the best way to lead the society into a digital intelligent future.

Reference

Lundberg, S M, & Lee, S 2017, A Unified Approach to Interpreting Model Predictions, NeurIPS Proceedings, viewed 1 December, https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Chu, L 2022, *Model Explainability - SHAP vs. LIME vs. Permutation Feature Importance,* Towards AI, viewed 29 November 2022, <https://pub.towardsai.net/model-explainability-shap-vs-lime-vs-permutation-feature-importance-98484efba066>.

Perera, P 2017, *The Boston Housing Dataset,* Kaggle, viewed 29 November 2022, https://www.kaggle.com/code/prasadperera/the-boston-housing-dataset.

Slundberg 2017, *shap,* Github*,* viewed 29 November 2022, https://github.com/slundberg/shap.

Sukumar, R 2020, *SHAP Part 2: Kernel SHAP,* Medium, viewed 1 December, <https://medium.com/analytics-vidhya/shap-part-2-kernel-shap-3c11e7a971b1>.