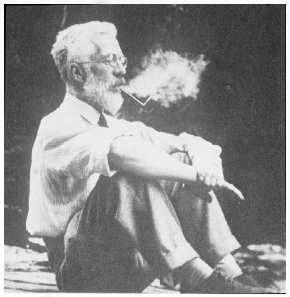
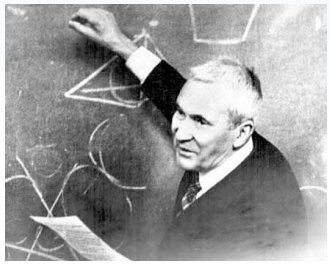
Applied ML

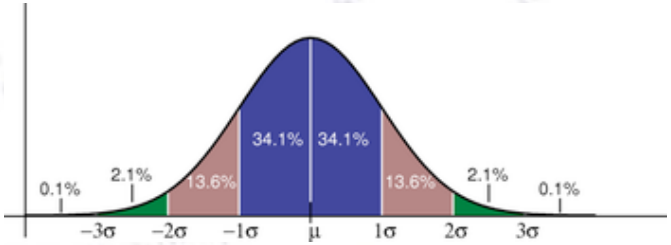
Input Feature Ranking - SHAP values





Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense - Machine Learning is a sharpening of it!"

ML as a science

While Machine Learning is fantastic, it is a black box, and thus unsatisfactory both regarding understanding it, and as a science in itself.

"As a data scientist, I can predict what is likely to happen, but I cannot explain why it is going to happen. I can predict when someone is likely to attrite, or respond to a promotion, or commit fraud, or pick the pink button over the blue button, but I cannot tell you why that's going to happen. And I believe that the inability to explain why something is going to happen is why I struggle to call 'data science' a science."

[Bill Schmarzo, Author of "Big Data: Understanding How Data Powers Big Business"]

ML as a science

While Machine Learning is fantastic, it is a black box, and thus unsatisfactory both regarding understanding it, and as a science in itself.

"As a data scientist, I can predict what is likely to happen, but I cannot explain why it is going to happen. I can predict when someone is likely to attrite, or respond to a promotion, or commit fraud, or pick the pink button over the blue button, but I cannot tell you why that's going to happen. And I believe that the inability to explain why something is going to happen is why I struggle to call 'data science' a science."

[Bill Schmarzo, Author of "Big Data: Understanding How Data Powers Big Business"]

However, there are ways of "opening the box", and the most common one is to find out, which input features are important and which are not.

For more info, see:

Interpretable Machine Learning

A Guide for Making Black Box Models Explainable.

Christoph Molnar

Input Feature Ranking

It is of course useful to know, which of your input features/variables are useful, and which are not. Thus a **ranking of the features** is desired.

This is not only possible, but actually a general nice feature of ML and feature ranking:

It works as an automation of the detective work behind finding relations.

In principle, one could obtain a variables ranking by testing **all combinations** of variables. But that is not feasible in most situation (N features > 5-7)…

Most algorithms have a build-in input feature ranking, which is based on various approximations.

A very simple idea (next slide) that works quite well is **"permutation importance"**.

Input Feature Ranking

There are many different ways of ranking input features. Three (simple) implementations into XGBoost are:

- **Weight**. The number of times a feature is used to split the data across all trees.

Input Feature Ranking

There are many different ways of ranking input features. Three (simple) implementations into XGBoost are:

- **Weight**. The number of times a feature is used to split the data across all trees.
- **Cover**. The number of times a feature is used to split the data across all trees weighted by the number of training data points that go through those splits.

Input Feature Ranking

There are many different ways of ranking input features. Three (simple) implementations into XGBoost are:

- **Weight**. The number of times a feature is used to split the data across all trees.
- **Cover**. The number of times a feature is used to split the data across all trees weighted by the number of training data points that go through those splits.
- **Gain**. The average training loss reduction gained when using a feature for splitting.

These have different pro's and con's.

Personally, I much like the idea of…

"permutation invariance"

Permutation Importance

Permutation Importance

One of the most used methods is “permutation importance” (below quoting Christoph M.: "Interpretable ML" chapter 5.5). The idea is really simple:

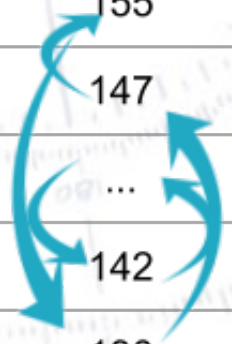
We measure the importance of a feature **by calculating the increase in the model’s loss function after permuting the feature**.
A feature is “important” if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.
A feature is “unimportant” if shuffling its values leaves the model error unchanged, because the model thus ignored the feature for the prediction.

Permutation Importance

One of the most used methods is “permutation importance” (below quoting Christoph M.: "Interpretable ML" chapter 5.5). The idea is really simple:

We measure the importance of a feature **by calculating the increase in the model’s loss function after permuting the feature**.
A feature is “important” if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.
A feature is “unimportant” if shuffling its values leaves the model error unchanged, because the model thus ignored the feature for the prediction.

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24



Permutation Importance

Input: Trained model f, feature matrix X, target vector y, loss function L(y,f).

[Fisher, Rudin, and Dominici (2018)]

- Estimate the original model error $e_{orig} = L(y, f(X))$
- For each feature $j = 1,\ldots,p$ do:
  - Generate feature matrix $X_{perm,j}$ by permuting feature j in the data X. This breaks the association between feature j and true outcome y.
  - Estimate error $e_{perm,j} = L(Y,f(X_{perm,j}))$ based on the predictions of $X_{perm,j}$.
  - Calculate permutation feature importance $FI_j = e_{perm,j} / e_{orig}$ (or $e_{perm,j} - e_{orig}$).
- Sort features by descending FI_j.

X_A	X_B	X_C	Y
xa1	xb1	xc1	y1
xa2	xb2	xc2	y2
xa3	xb3	xc3	y3
xa4	xb4	xc4	y4
xa5	xb5	xc5	y5
xa6	xb6	xc6	y6

Note: Permutation Importance calculations are computationally fast. (why?)

Feature importance with Neural Networks (Towards Data Science)

SHAP Values

SHAP Values

SHAP is a technique for deconstructing a machine learning model's predictions into a sum of contributions from each of its input variables.

Shapley values

Shapley values is a concept from **corporative game theory**, where they are used to provide a possible answer to the question:

"How important is each player to the overall cooperation,
and what payoff can each player reasonably expect?"

The Shapley values are considered "fair", as they are the only distribution with the following properties:

- **Efficiency**: Sum of Shapley values of all agents equals value of grand coalition.
- **Linearity**: If two coalition games described by v and w are combined, then the distributed gains should correspond to the gains derived from the sum of v and w.
- **Null player**: The Shapley value of a null player is zero.
- **Stand alone test**: If v is sub/super additive, then $\varphi_i(v) \leq / \geq v(\{i\})$, where ϕ is the Shapley value for agent i, and v is the worth function (of a coalition). Also called "Monotonicity": A consistently more contributing feature much a get higher v.
- **Anonymity**: Labelling of agents doesn't play a role in assignment of their gains.
- **Marginalism**: Function uses only marginal contributions of player i as arguments.

From such values, one can determine which variables contribute to a final result. And summing the values, one can get an overall idea of which variables are important.

Shapley value calculation

Consider a set **N** (of n players) and a (characteristic or worth) function **v** that maps any subset of players to real numbers:

$$v \ : \ 2^N \rightarrow \mathbb{R}, \quad v(\emptyset) = 0$$

If S is a coalition of players, then v(S) yields the total expected sum of payoffs the members of S can obtain by cooperation.

Shapley value calculation

Consider a set **N** (of n players) and a (characteristic or worth) function **v** that maps any subset of players to real numbers:

$$v \ : \ 2^N \rightarrow \mathbb{R}, \quad v(\emptyset) = 0$$

If S is a coalition of players, then v(S) yields the total expected sum of payoffs the members of S can obtain by cooperation.

The Shapley values are calculated as:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)]$$

The formula can be understood, if we imagine a coalition being formed one actor at a time, with each actor demanding their contribution **v(S ∪ {i}) − v(S)** as a fair compensation. For each actor we then take the average of this contribution over the possible different permutations in which the coalition can be formed (sampled in real life code!). That gives the SHAP value.

Shapley value calculation

Consider a set **N** (of n players) and a (characteristic or worth) function **v** that maps any subset of players to real numbers:

$$v \ : \ 2^N \to \mathbb{R}, \quad v(\emptyset) = 0$$

If S is a coalition of players, then v(S) yields the total expected sum of payoffs the members of S can obtain by cooperation.

The Shapley values can also be calculated as:

$$\varphi_i(v) = \frac{1}{n!} \sum_R \left[v(P_i^R \cup \{i\}) - v(P_i^R) \right]$$

where the sum ranges over all n! orders R of the players and P_i^R is the set of players in N which precede i in the order R. This has the interpretation:

$$\varphi_i(v) = \frac{1}{N_{\text{players}}} \sum_{C \setminus i} \frac{\text{marginal contribution of } i \text{ to coalition C}}{\text{number of coalitions excluding } i \text{ of this size}}$$

Shapley value calculation example

Example 1:

Two friends (F1 and F2) make a business: **Payoff 600$** (i.e. v(F1,F2) = 600).

If F1 or F2 did not participate, payoff would be 0$ (i.e. v(F1) = v(F2) = 0).

Result: F1 and F2 each gets 300$.

Shapley value calculation example

Example 1:

Two friends (F1 and F2) make a business: **Payoff 600$** (i.e. v(F1,F2) = 600).
If F1 or F2 did not participate, payoff would be 0$ (i.e. v(F1) = v(F2) = 0).
Result: F1 and F2 each gets 300$.

Example 2:

Two friends (F1 and F2) make a business: **Payoff 600$** (i.e. v(F1,F2) = 600).
If F1 did not participate, payoff would be 0$ (i.e. v(F1) = 0).
If F2 did not participate, payoff would be 200$ (i.e. v(F2) = 200).
Cases:
F1 1st gets 0$. With F2 also they get 600$. F2's marginal contribution: 600$.
F2 1st gets 200$. With F1 also they get 600$. F1's marginal contribution: 400$.
Result:
F1 should have: 0.5 x 0$ + 0.5 x 400$ = 200$
F2 should have: 0.5 x 200$ + 0.5 x 600$ = 400$

Note that the number of cases quickly expands!

SHAP Values

A great approximation was developed by Scott Lundberg with **SHAP values**:
SHAP (SHapley Additive exPlanations):
https://github.com/slundberg/shap

This algorithm provides - **for each entry** - a ranking of the input variables, i.e. a sort of explanation for the result.

One can also sum of the SHAP values over all entries, and then get the overall ranking of feature variables. **They are based on Shapley values**.

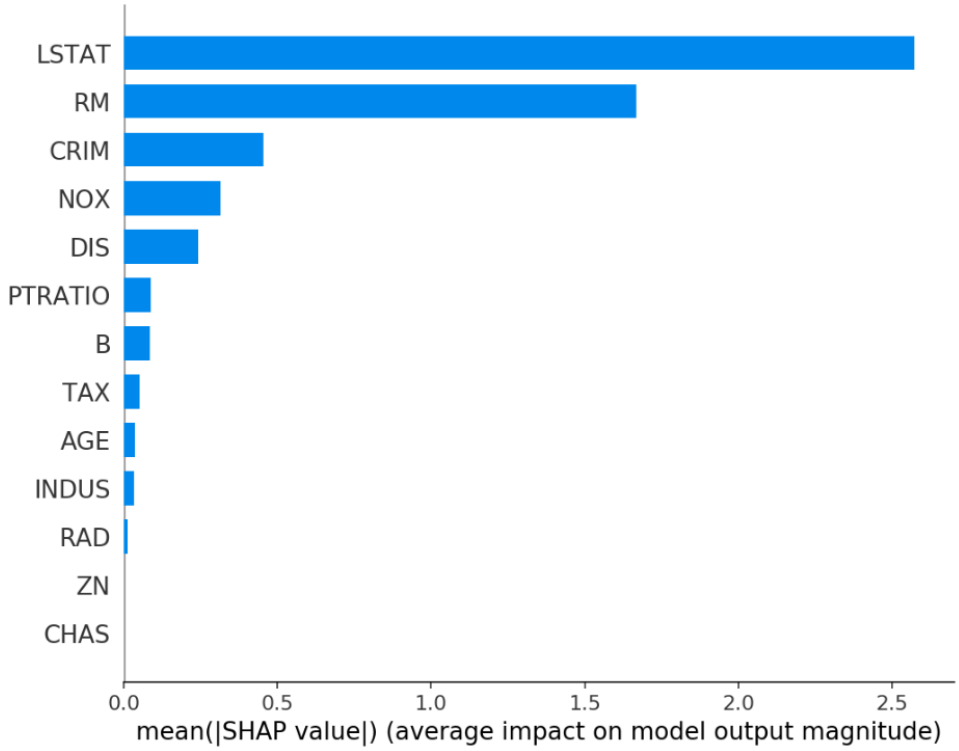
Note: SHAP values are computationally “heavy”.

Input Feature Ranking

Here is an example from SHAP's github site.

Clearly, LSTAT and RM are the best variables (whatever they are!).

```
shap.summary_plot(shap_values, X, plot_type="bar")
```

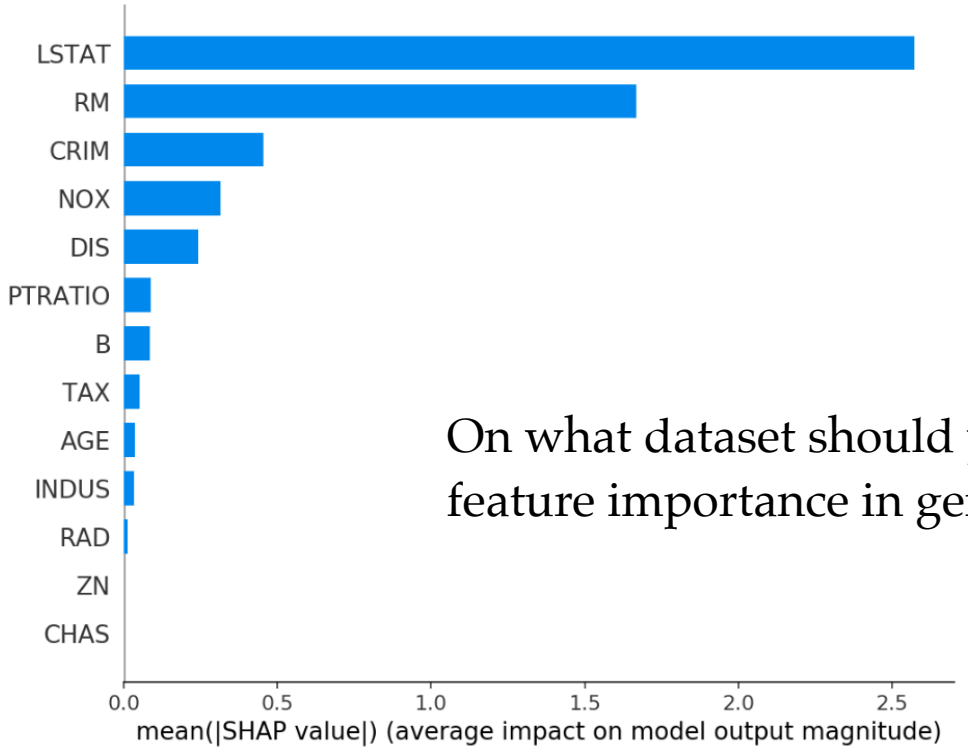


Input Feature Ranking

Here is an example from SHAP's github site.

Clearly, LSTAT and RM are the best variables (whatever they are!).

```
shap.summary_plot(shap_values, X, plot_type="bar")
```



On what dataset should you calculate feature importance in general?

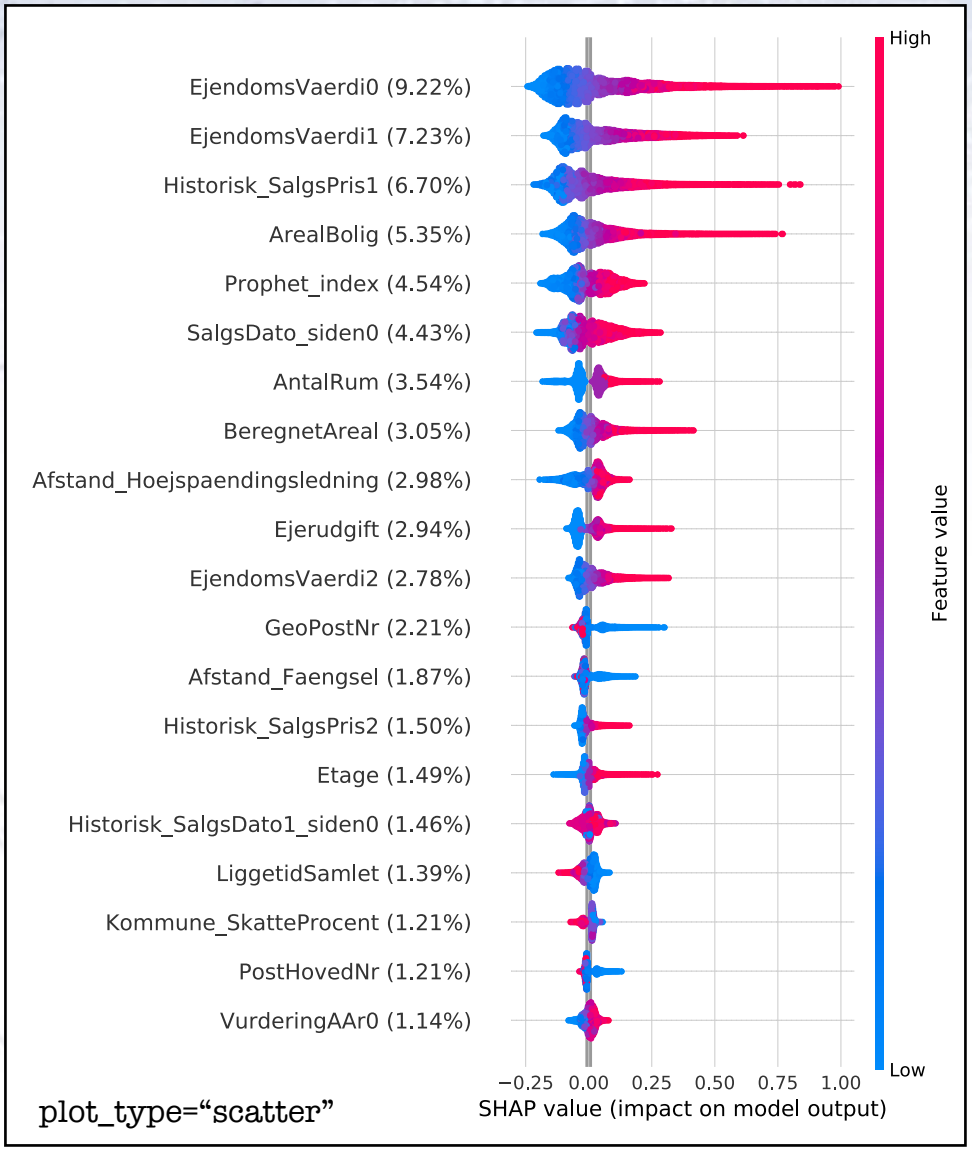
Ranking of feature values

The impact of individual input (feature) values can be evaluated using SHAP (Shapley) values.

These are shown on the plot, where the percentage is proportional to the sum of the SHAP values.

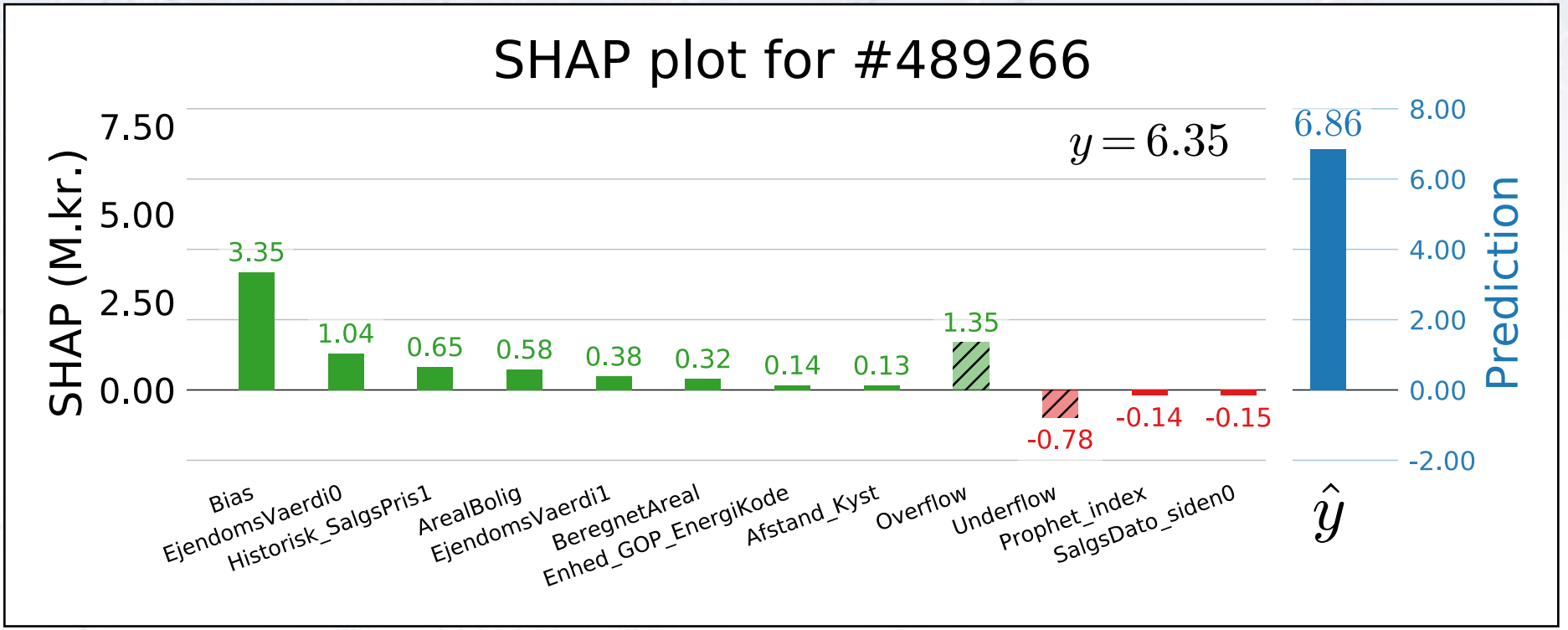
Not surprisingly, the evaluation of a housing sales price depends mostly on the previous evaluations, followed by area and number of rooms.

Location is already included, and thus postal codes are not that important.



Individuel estimates

Shapley-values also gives the possibility to see the reason behind **individuel estimates**. Below is an example, illustrating this point.



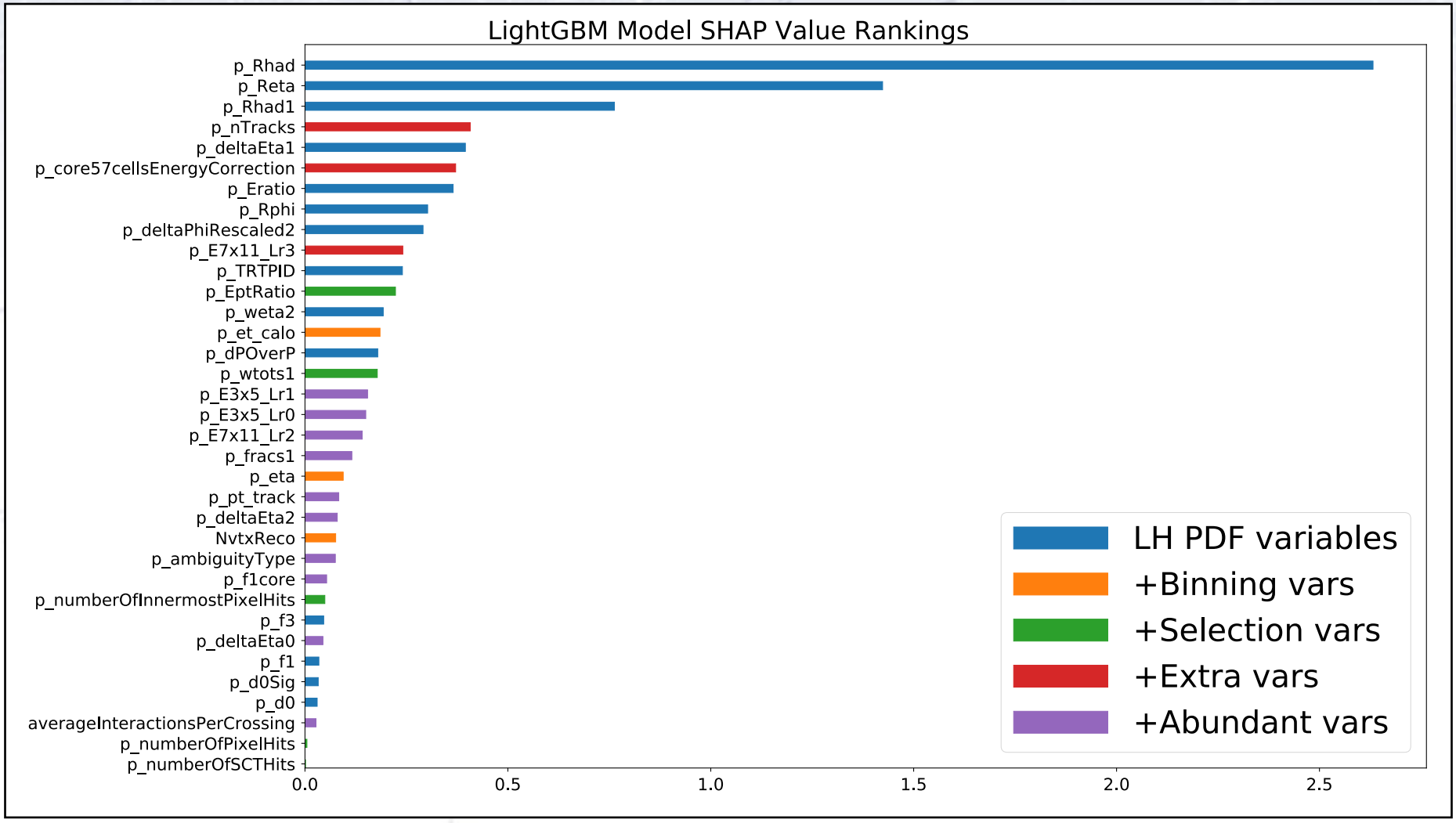
Above is shown which factors that influences the final estimate of the sales price (and how much). The estimate is the sum of the contributions (here 6.86 MKr.).

This is a fantastic tool to get insight into the ML workings!!!

Example of usage

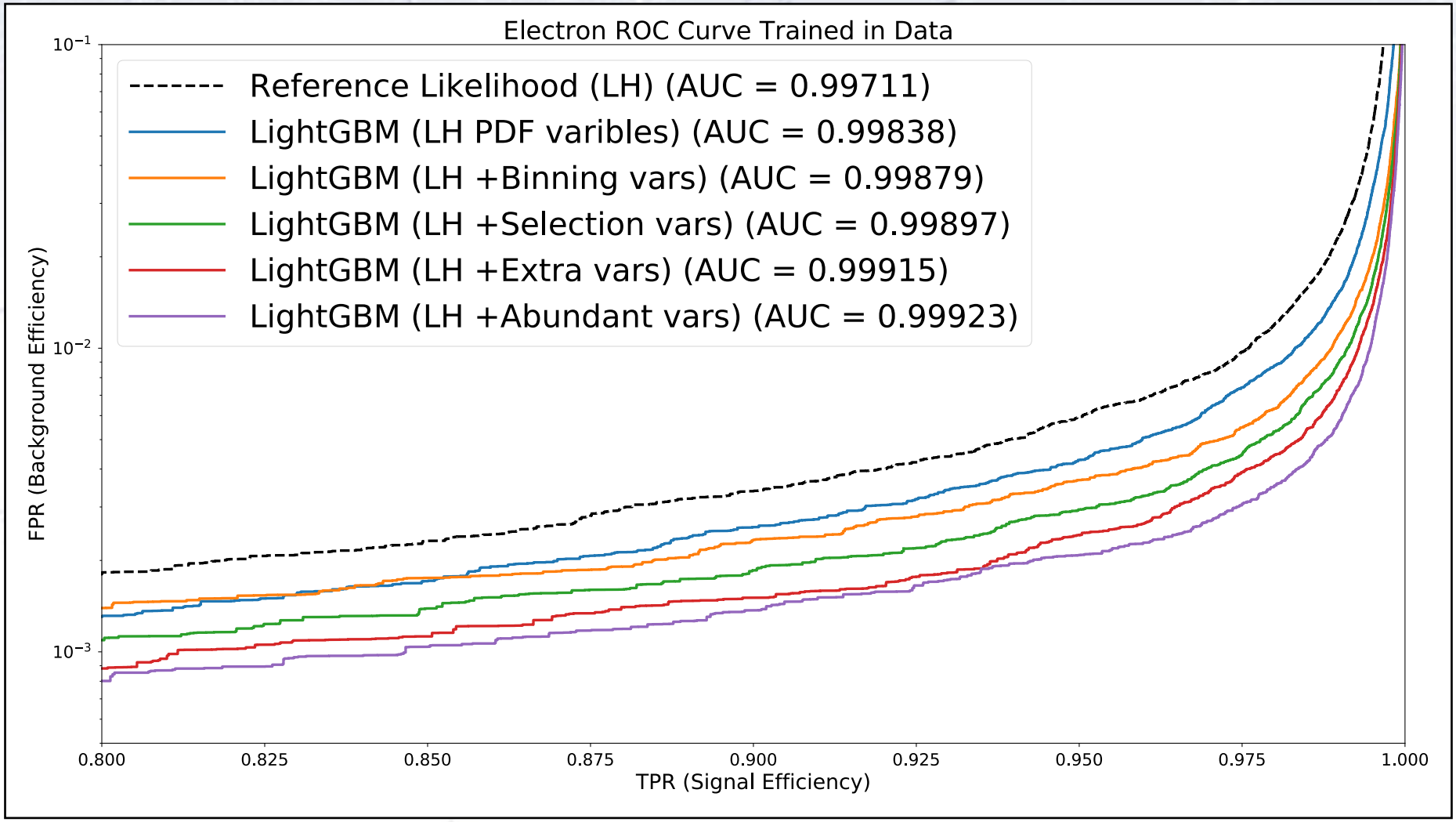
Input Feature Ranking

Here is an example from particle physics. The blue variables were "known", but with SHAP we discovered three new quite good variables in data.



Input Feature Ranking

We could of course just add all variables, but want to stay simple, and training the models, we see that the three extra variables gives most of gain.



Questions on feature rankings

Do you understand the difference in the output between permutation importance (PI) and Shapley values (SHAP)?

Would you be able to explain to others (non-scientists) the concept of a feature ranking? Of SHAP values?

What should you do, if you happen to get (very) different rankings from e.g. permutation importance (PI) and Shapley values (SHAP)?

Can you think of other uses of feature ranking?