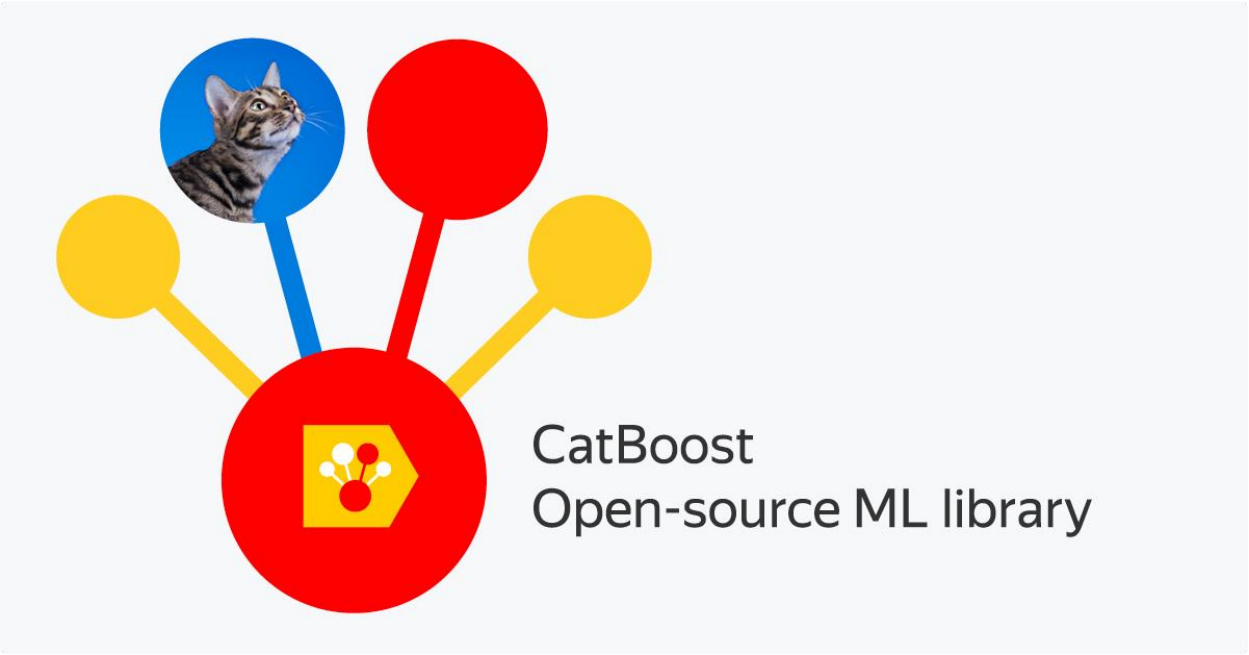
Deep Dive into Catboost Functionalities for Model Interpretation

Do we really understand what happens inside ML models we build? Let's explore.

Alvira Swalin
Follow

Jun 24 · 8 min read

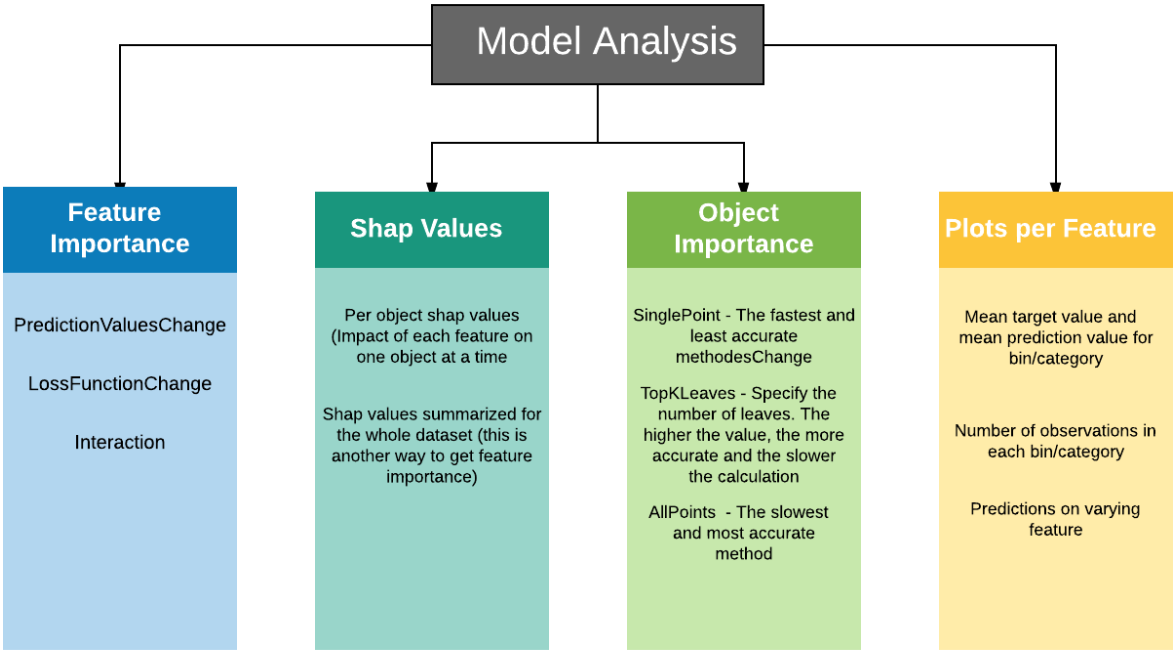


In my previous blog, we saw a comparative study of XGBoost, LightGBM & Catboost. With that analysis, we were able to conclude that catboost outperformed the other two in terms of both speed and accuracy. In this part, we will dig further into the catboost, exploring the new features that catboost provides for efficient modeling and understanding the hyperparameters.

For new readers, catboost is an open-source gradient boosting algorithm developed by Yandex team in 2017. It is a machine learning algorithm which allows users to quickly handle categorical features for a large data set and

this differentiates it from XGBoost & LightGBM. Catboost can be used to solve **regression, classification and ranking problems**.

As Data Scientists, we can easily train models and make predictions, but, we often fail to understand what's happening inside those fancy algorithms. This is one of the reasons why we see a huge difference in model performance between offline evaluation and final production. It is high time we stop treating ML as a **"black box"** and give importance to model interpretation while improving model accuracy. This will also help us in identifying data biases. In this part, we will see how catboost helps us to analyze models and improve visibility with the following functionalities:



Feature Importance

Why should you know it?
- Remove unnecessary features to simplify the model and reduce training/prediction time
- Get the most influential feature for your target value and manipulate them for business gains (eg: healthcare providers want to identify what factors are driving each patient's risk of some disease so they can directly address those risk factors with targeted medicines)

Apart from choosing the type of feature importance, we should also know which data we want to use for finding feature importance — train or test or complete dataset. There are pros and cons of choosing one over the other but in the end, you need to decide whether you want to know how much the model relies on each feature for making predictions (**use training data**) or how much the feature contributes to the performance of the model on unseen data (**use test data**). We will see later in this blog that only some methods can be used to find feature importance on data not used for training model.

If you care for the second and assuming you have all the time and resources, the crudest and most reliable way to find feature importance is to train multiple models leaving one feature at a time and compare the performance on the test set. If the performance changes a lot with respect to the baseline (performance when we use all the features) that means that the feature was important**.** But since we live in a practical world where we need to optimize both accuracy and computation time, this method is unnecessary. Here are a few smart ways in which catboost lets you find the best feature for your model:

PredictionValuesChange

For each feature, PredictionValuesChange shows how much on average the prediction changes if the feature value changes. The bigger the value of the importance the bigger on average is the change to the prediction value if this feature is changed.

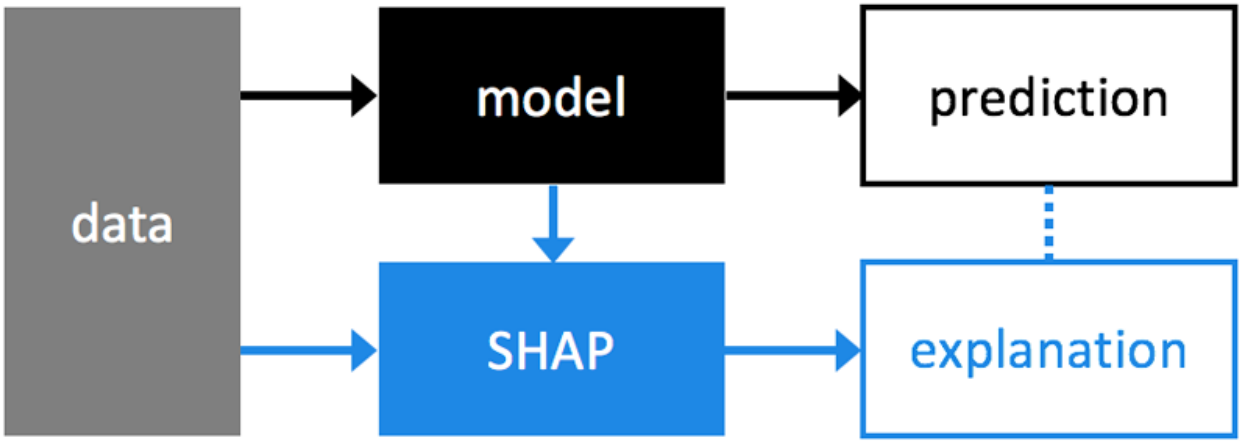
Pros: *It is cheap to compute as you don't have to do multiple training or testing and you will not be storing any extra information. You will get* ***normalized values*** *as the ouput (all the importances will add up to 100).*
Cons: *It may give misleading results for ranking objectives, it might put groupwise features into the top, even though they have a little influence on the resulting loss value.*

LossFunctionChange

To get this feature importance, catboost simply takes the difference between the metric (Loss function) obtained using the model in normal scenario (when we include the feature) and model without this feature (model is built approximately using the original model with this feature removed from all the trees in the ensemble). Higher the difference, the more important the feature is. It is not clearly mentioned in catboost docs how we find the model without feature.

Pros & Cons: *This works well for most type of problems unlike* `predictionvalueschange` *where you can get misleading results for ranking problems, at the same time, it is computationally heavy .*

Shap Values

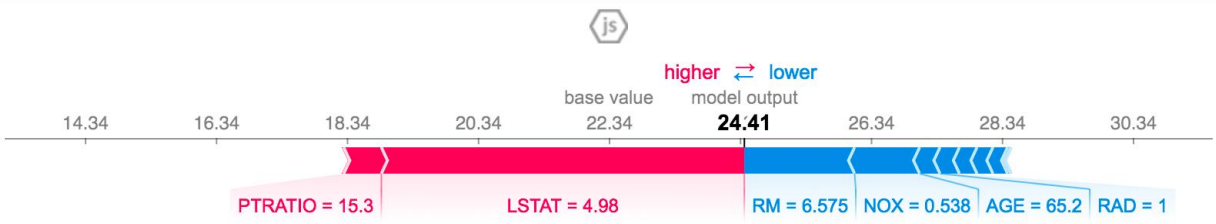


https://github.com/slundberg/shap

SHAP value breaks a prediction value into contributions from each feature. It measures the impact of a feature on a single prediction value in comparison to the baseline prediction (mean of the target value for the training dataset).

Two major use cases for shap values:

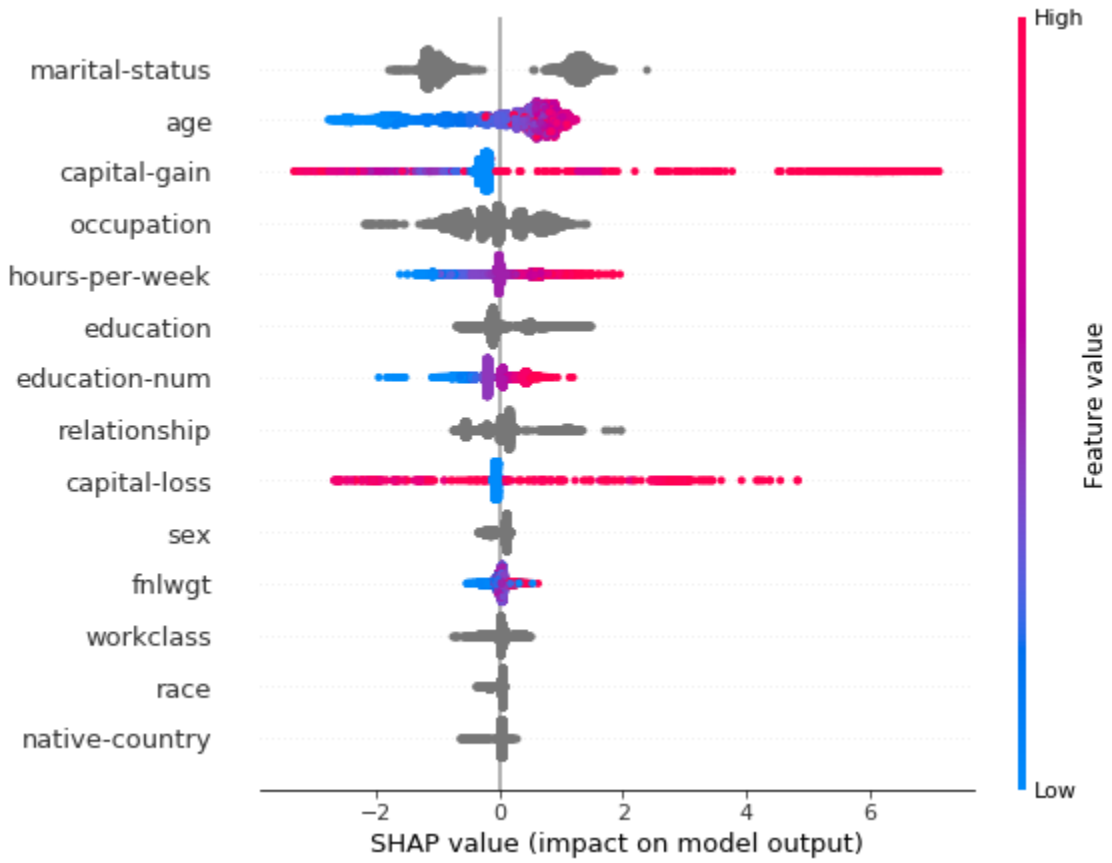
1. **object-level contributions of features**



https://github.com/slundberg/shap

2. Summary for the whole dataset (overall feature importance)

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



Though we can get accurate feature importances through shap, they are computationally more expensive than catboost inbuilt feature importance. For more details on SHAP values, please read this kernel .

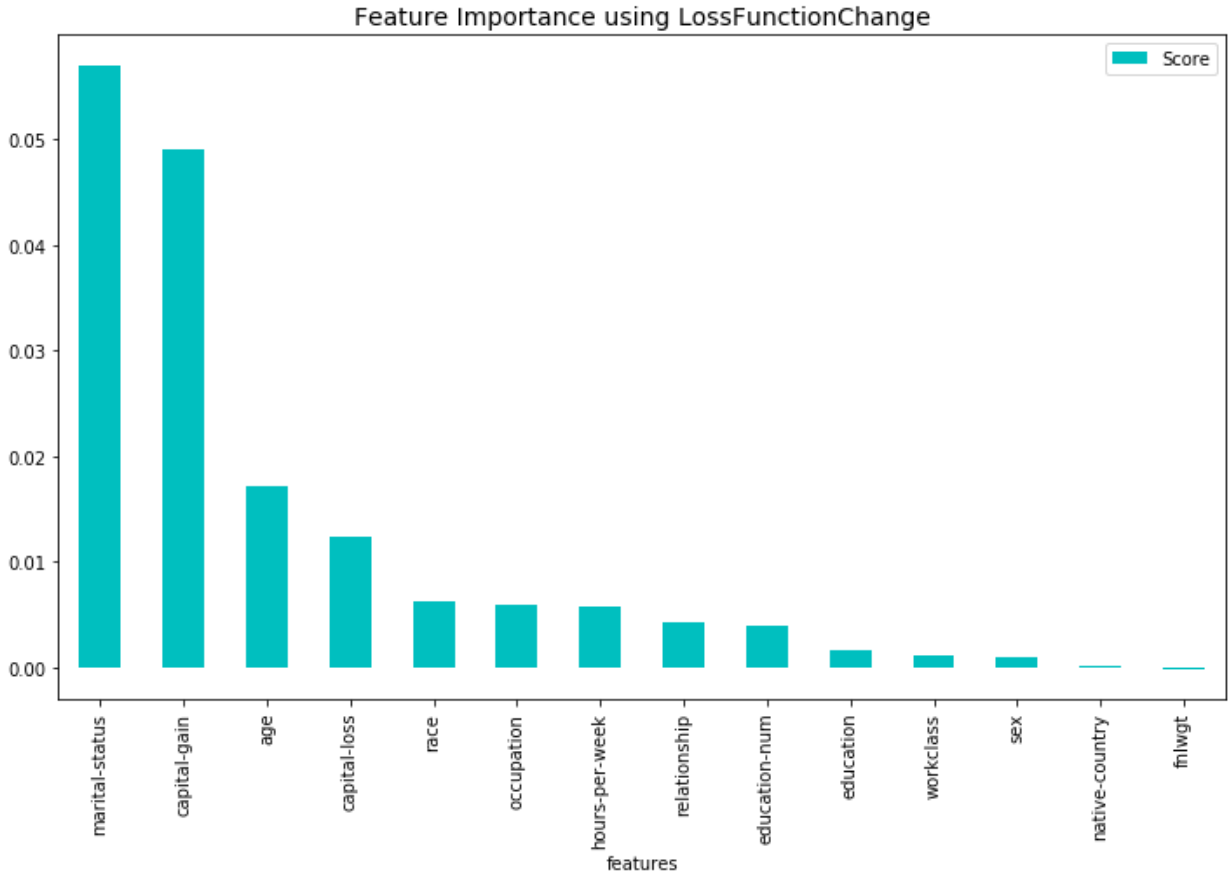
Bonus

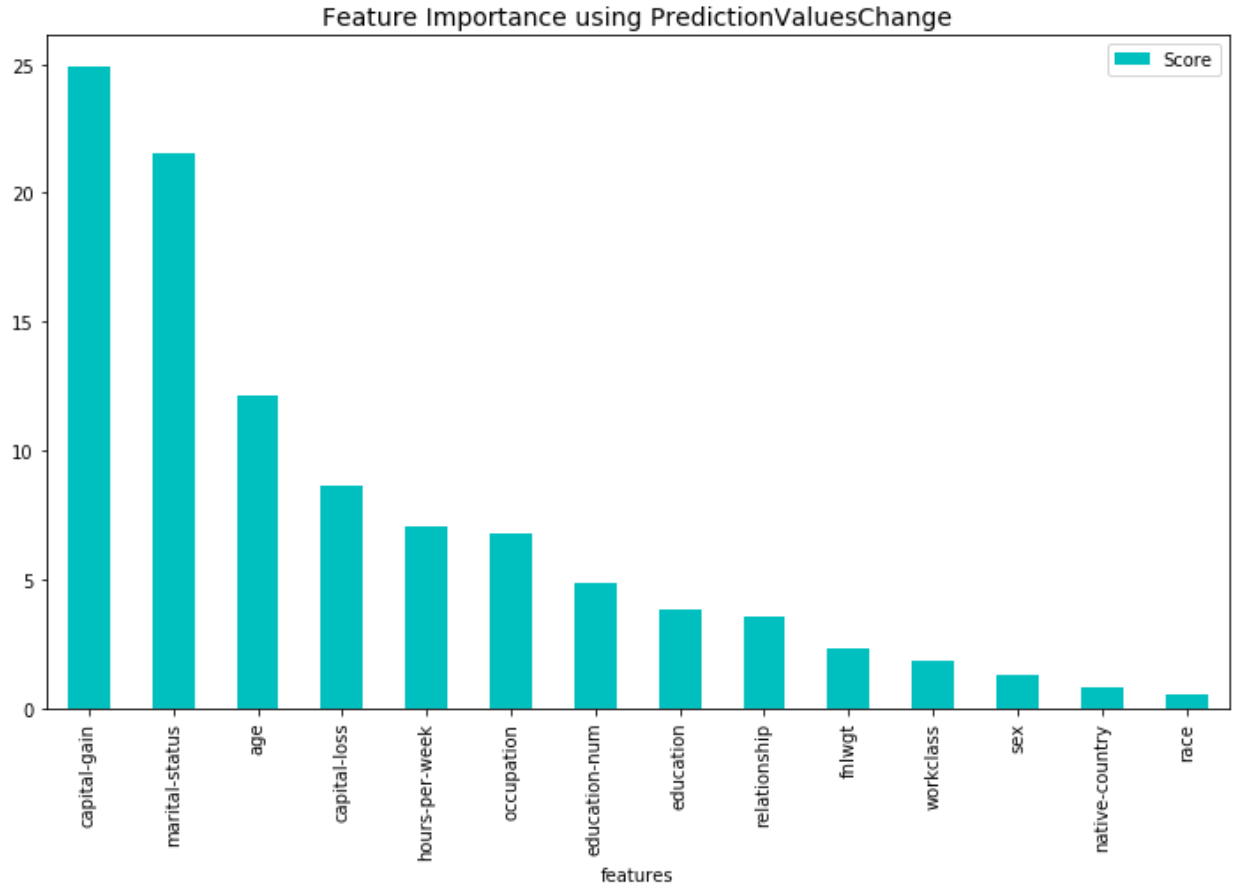
Another feature importance based on the same concept but different implementation is— **Permutation based feature importance.** Though catboost does not use this, this is purely **model-agnostic** and easy to calculate.

How are we going to choose one?

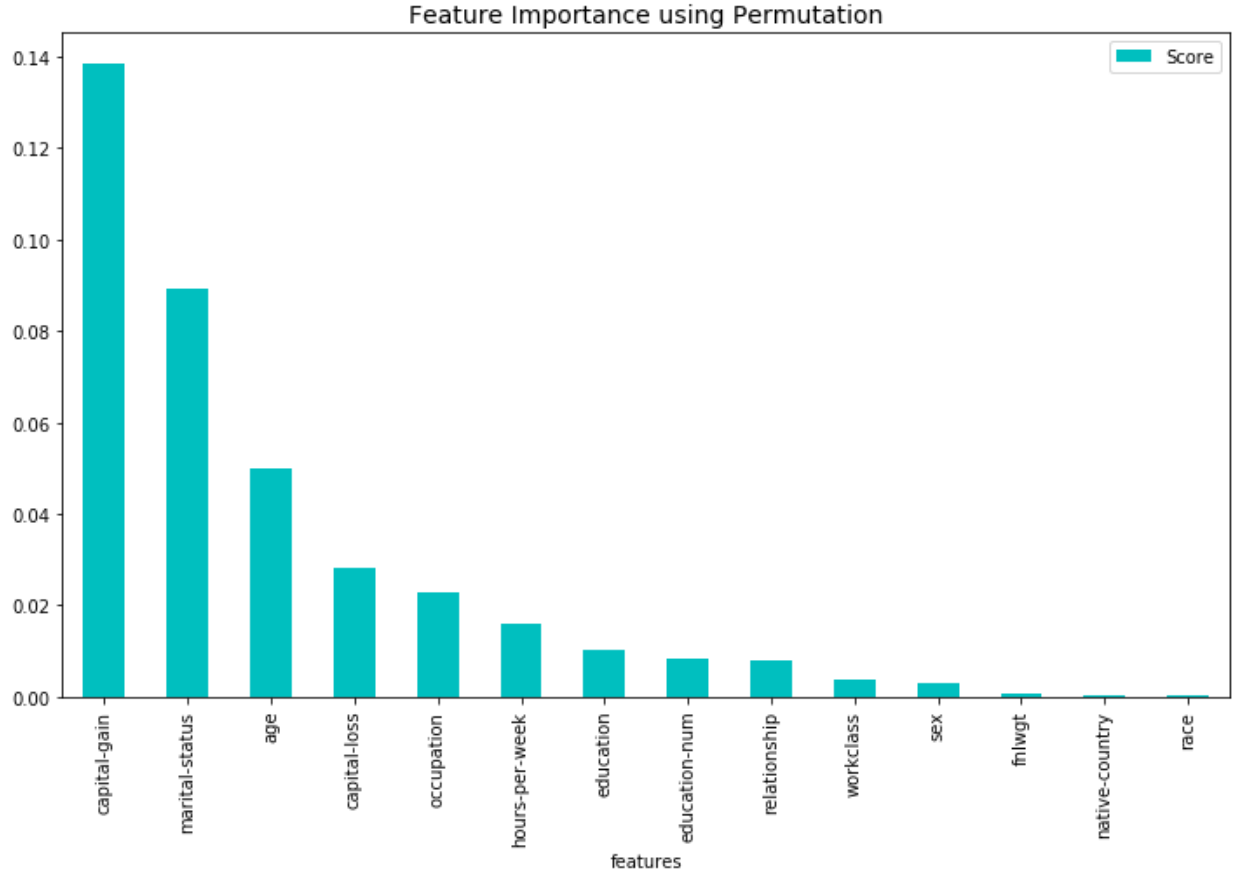
Though both `PredictionValuesChange` & `LossFunctionChange` can be used for all types of metrics, it is recommended to use `LossFunctionChange` for ranking metrics. Except for `PredictionValuesChange`, all other methods can use test data to find feature importance using models trained on train data.

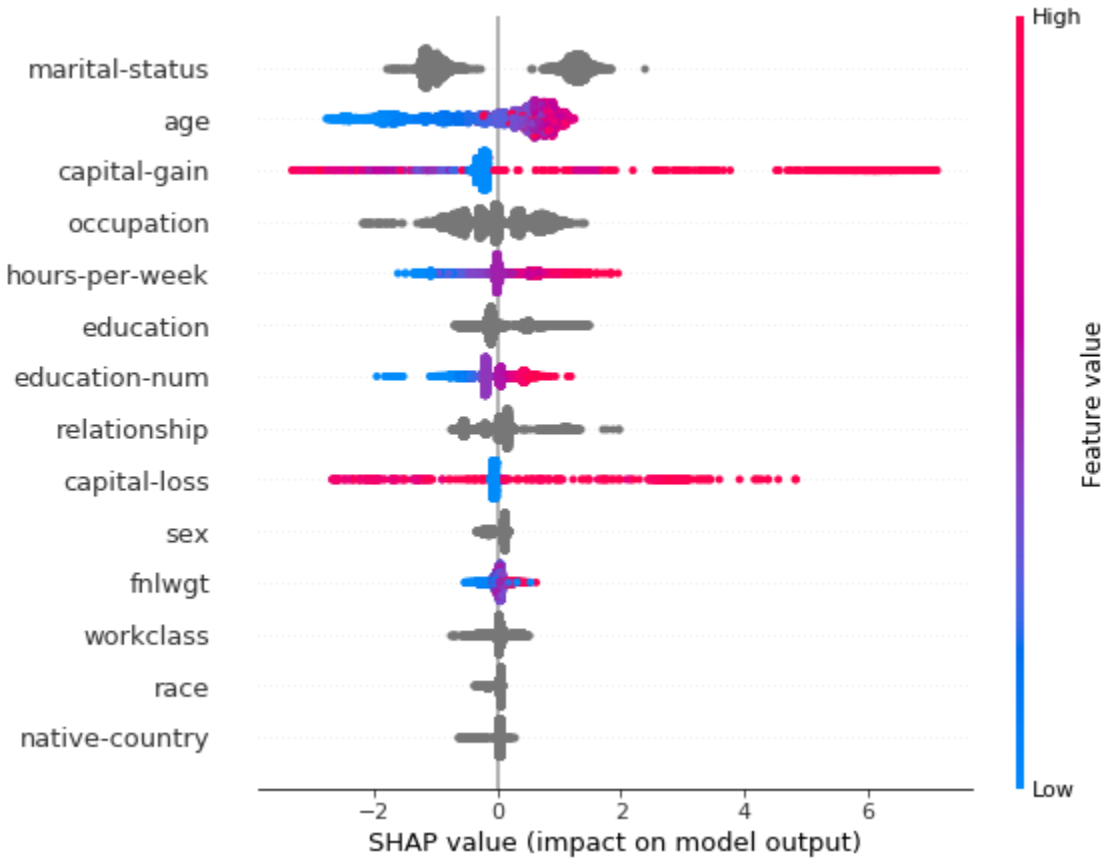
To understand the differences better, here are the results for all the methods we discussed:





Results of catboost feature imp. to predict if people will report over $50k of income from the classic [“adult” census dataset](#)(using log-loss)





From the above plots, we can see that most of the methods are agreeing on top features. it looks like `LossFunctionChange` is closest to shap(which is more reliable). However, it is not fair to compare these methods directly because `predictionvalueschange` is based on train data whereas all others are based on test data.

We should also see the time it takes to run all of them:

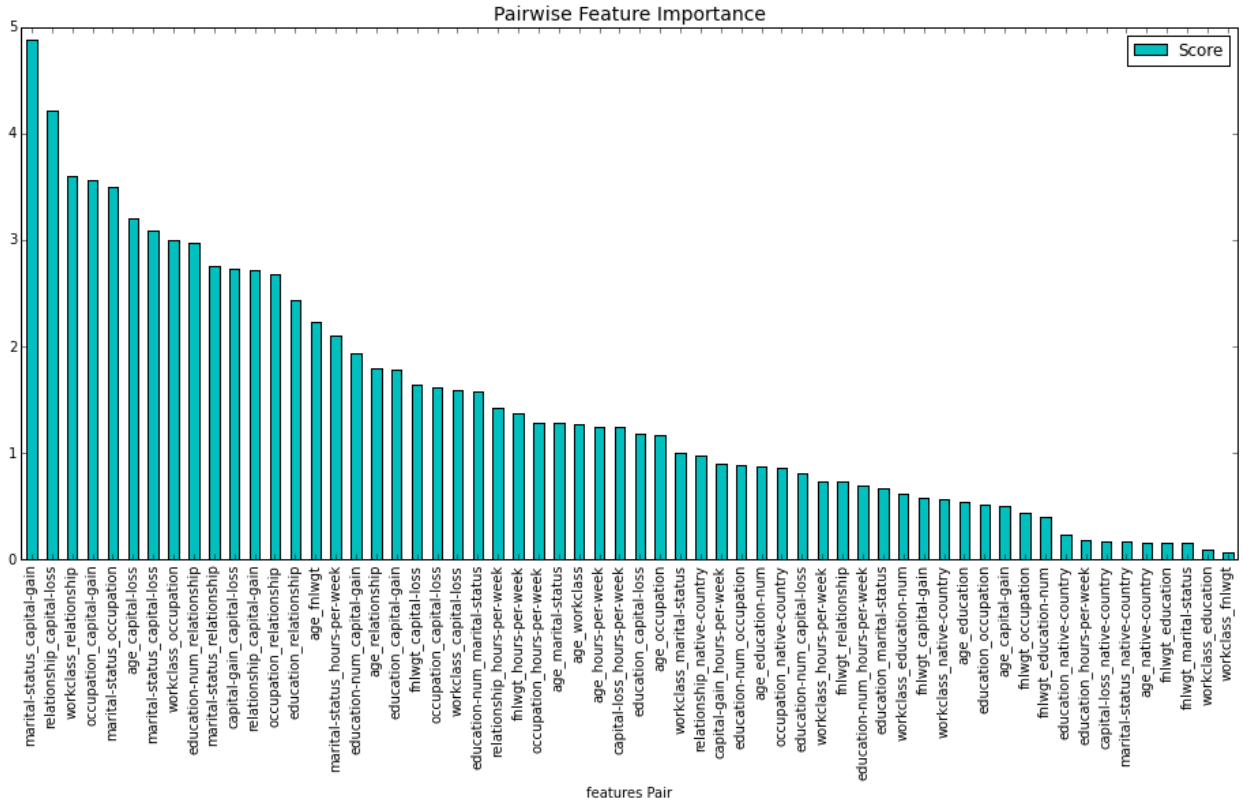
Method	Time
PredictionValuesChange	313 ms
LossFunctionChange	514 ms
Permutation	1.08 s
ShapValues	2.29 s

Interaction

With this parameter, you can find the strength of a pair of features (importance of two features together).

$$feature\_strength = \sum_{leafs} \left| \sum_{f1=f2} LeafValue - \sum_{f1!=f2} LeafValue \right|$$

In the output, you will get a list for each pair of features. The list will have 3 values, the first value is the index of the first feature in the pair, the second value is the index of the second feature in the pair and the third value is the feature importance score for that pair. For implementation details, please check the embedded notebook.



It is interesting to note that the top two features in the single feature importance don't necessarily make the strongest pair.

Notebook

[Dataset used in the notebook]

Object Importance

Why should you know it?

- Remove the most useless training objects from the training data
- Prioritize a batch of new objects for labeling based on which ones are expected to be the most “helpful”, akin to active learning

With this functionality, you can calculate the impact of each object on the optimized metric for test data. Positive values reflect that the optimized metric increases and negative values reflect that the optimized metric decreases. This method is an implementation of the approach described in this paper. Details of these algorithms are beyond the scope of this blog.

Catboost tutorial for Object Importance

catboost/tutorials
CatBoost tutorials repository. Contribute to catboost/tutorials development by creating an account on GitHub.
github.com

There are three types of `update_method` available in `cb.get_object_importance` :

- SinglePoint: The fastest and least accurate method
- TopKLeaves: Specify the number of leaves. The higher the value, the more accurate and the slower the calculation
- AllPoints: The slowest and most accurate method

For example, the following value sets the method to TopKLeaves and limits the number of leaves to 3:

```
TopKLeaves:top=3
```

Model Analysis Plots

Catboost has recently launched this functionality in its latest update**.** With this feature, we will be able to visualize how the algorithm is splitting the data for each feature and look at feature-specific statistics. More specifically we will be able to see:

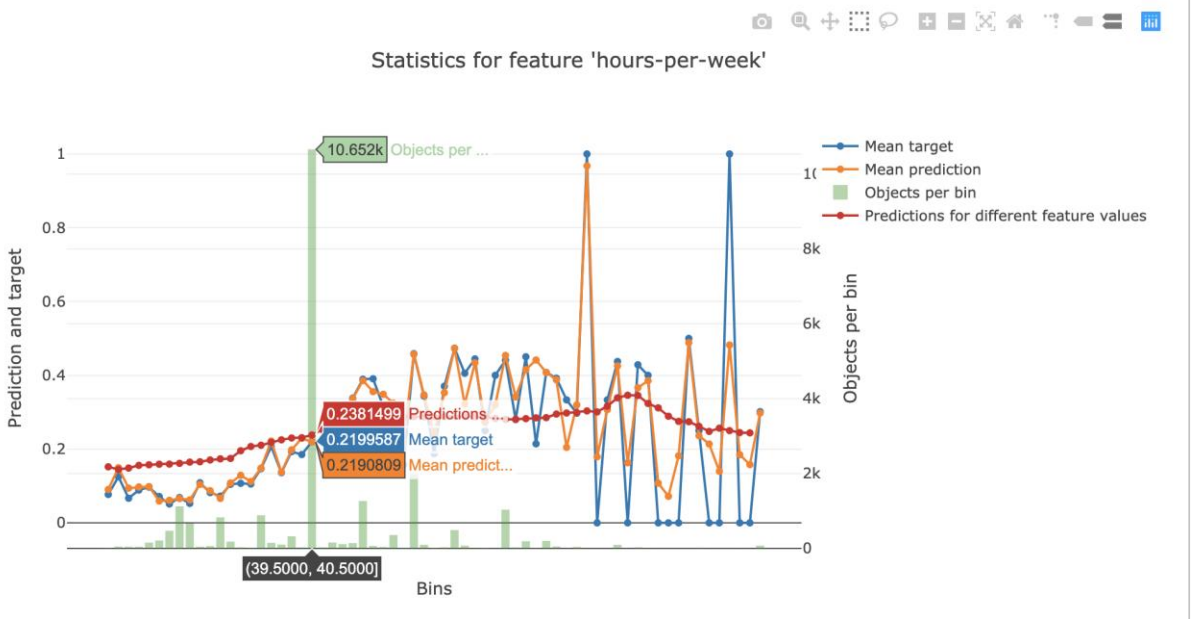
- Mean target value for each bin (bin is used for continuous feature) or category (currently only OHE features are supported)
- Mean prediction value for each bin/category
- Number of objects in each bin
- Predictions for different feature values: For every object, feature value is varied so that it falls into some bin. The model then predicts the target according to the new values of that feature and takes the mean of predictions in a bin (given by red points).

This plot will give us information like how uniform our splitting is (we don't want all the objects to go in one bin), whether our predictions are close to target (blue and orange line), the red line will tell us how sensitive our predictions are wrt a feature.

Numerical Feature Analysis

Numerical Feature

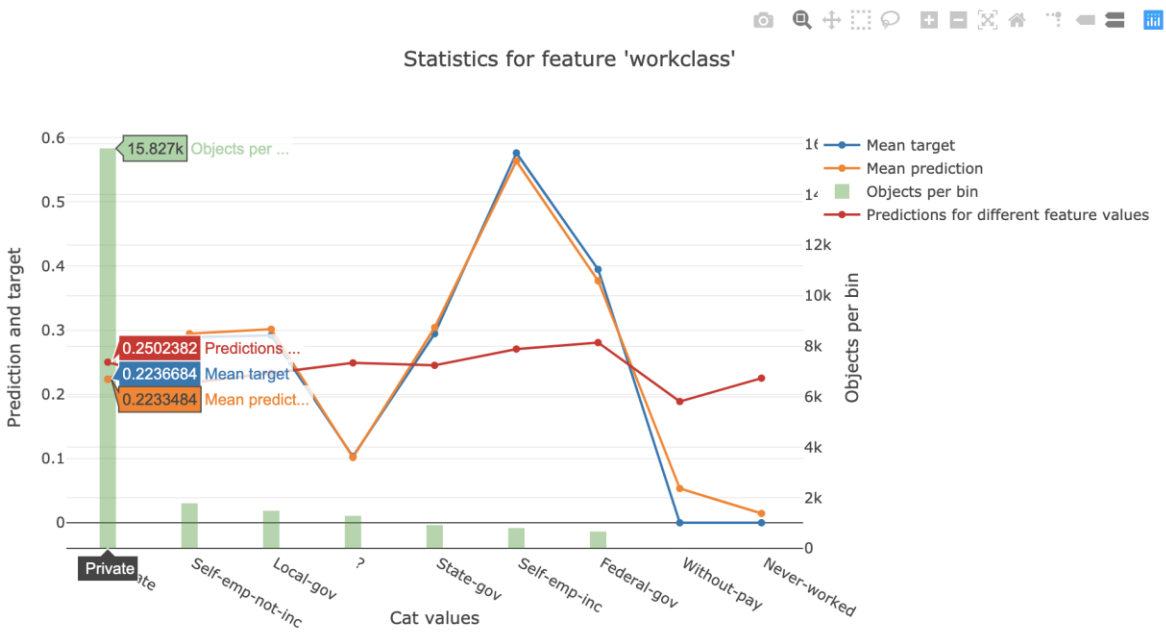
```
feature = 'hours-per-week'
res = model.get_feature_statistics(X_train, y_train, feature, plot=True)
```



One-Hot Encoded Feature Analysis

One-Hot Feature

```
feature = 'workclass'
model.get_feature_statistics(X_train, y_train, feature, plot=True)
```



Thank you for reading this. Hopefully, next time you will be able to make use of these tools to understand your models better.

After the positive reception from the machine learning community to my previous blog on CatBoost vs. Light GBM vs. XGBoost, the CatBoost team reached out to see if I was interested in covering more in-depth topics on the library. Thanks to the CatBoost team for helping answer my questions for this post!

About Me: I am currently working as a Data Scientist in Maps team at Uber. If you are interested in solving challenging problems at Uber, please reach out to me at LinkedIn. You can read my other blogs here.

References

- Catboost Documentation
- SHAP Values
- Catboost official website
- CatBoost Paper
- Notebook
- Paper on Finding Influential Training Samples
- Catboost Tutorials