Published in Towards Data Science

Simon Thiesen Follow
Feb 18, 2021 · 6 min read · Listen

CatBoost regression in 6 minutes

A brief hands-on introduction to CatBoost regression analysis in Python

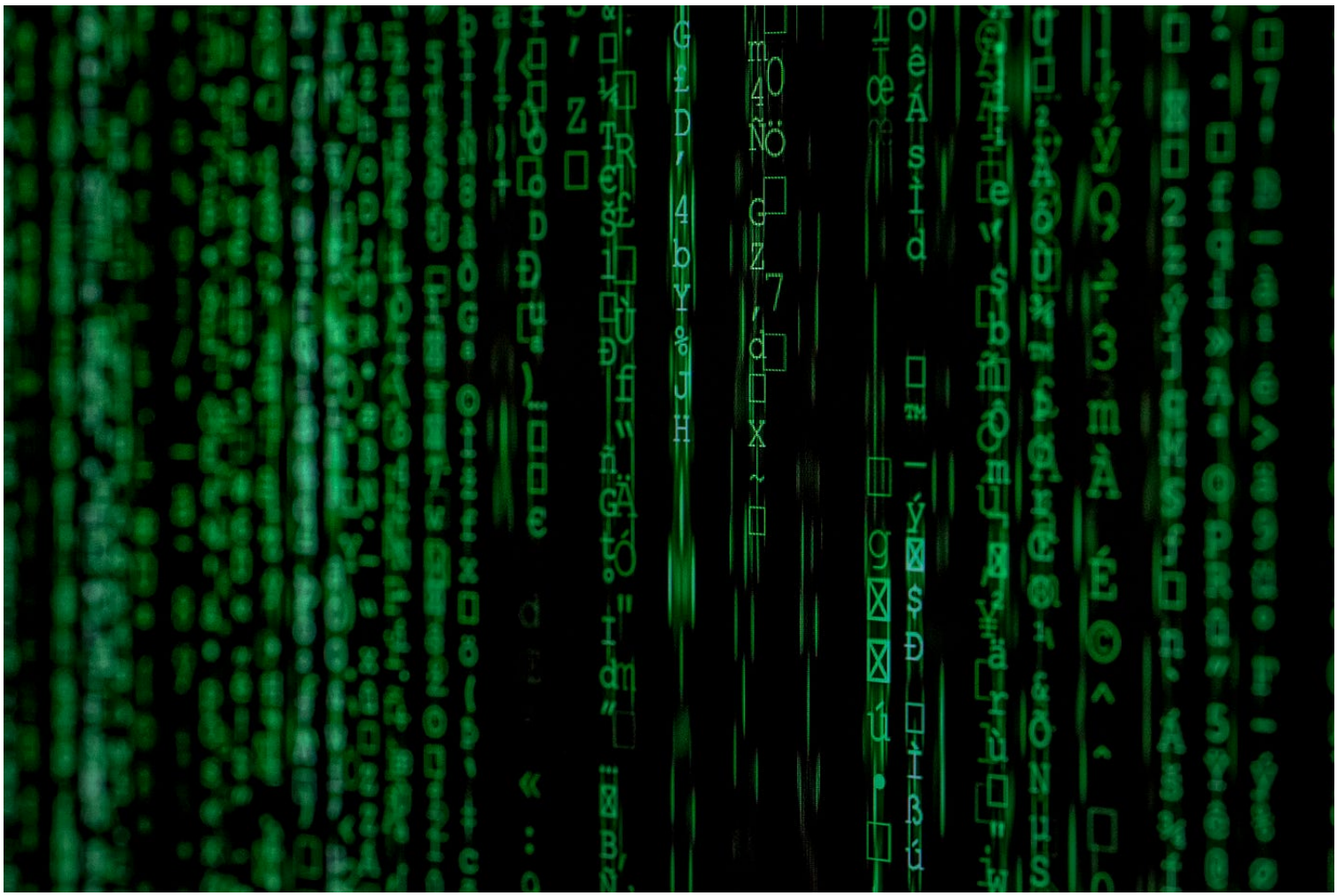


Photo by Markus Spiske on Unsplash

> *This article aims to provide a hands-on tutorial using the CatBoost Regressor on the Boston Housing dataset from the Sci-Kit Learn library.*

Table of Contents

2. Application

3. Final notes

Introduction

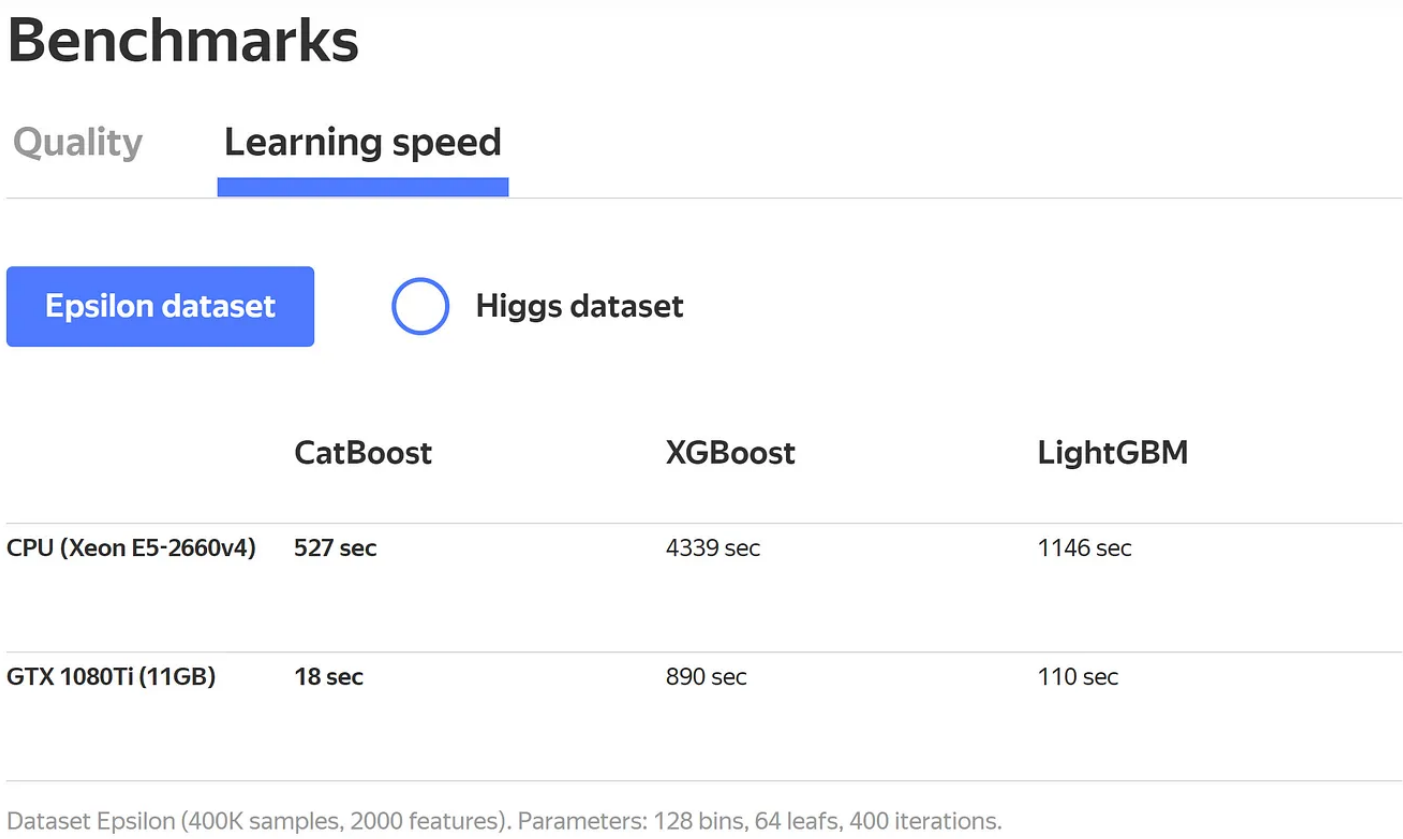
CatBoost is a relatively new open-source machine learning algorithm, developed in 2017 by a company named Yandex. Yandex is a Russian counterpart to Google, working within search and information services [1].

One of CatBoost's core edges is its ability to integrate a variety of different data types, such as images, audio, or text features into one framework. But CatBoost also offers an idiosyncratic way of handling categorical data, requiring a minimum of categorical feature transformation, opposed to the majority of other machine learning algorithms, that cannot handle non-numeric values. From a feature engineering perspective, the transformation from a non-numeric state to numeric values can be a very non-trivial and tedious task, and CatBoost makes this step obsolete.

CatBoost builds upon the theory of decision trees and gradient boosting. The main idea of boosting is to sequentially combine many weak models (a model performing slightly better than random chance) and thus through greedy search create a strong competitive predictive model. Because gradient boosting fits the decision trees sequentially, the fitted trees will learn from the mistakes of former trees and hence reduce the errors. This process of adding a new function to existing ones is continued until the selected loss function is no longer minimized.

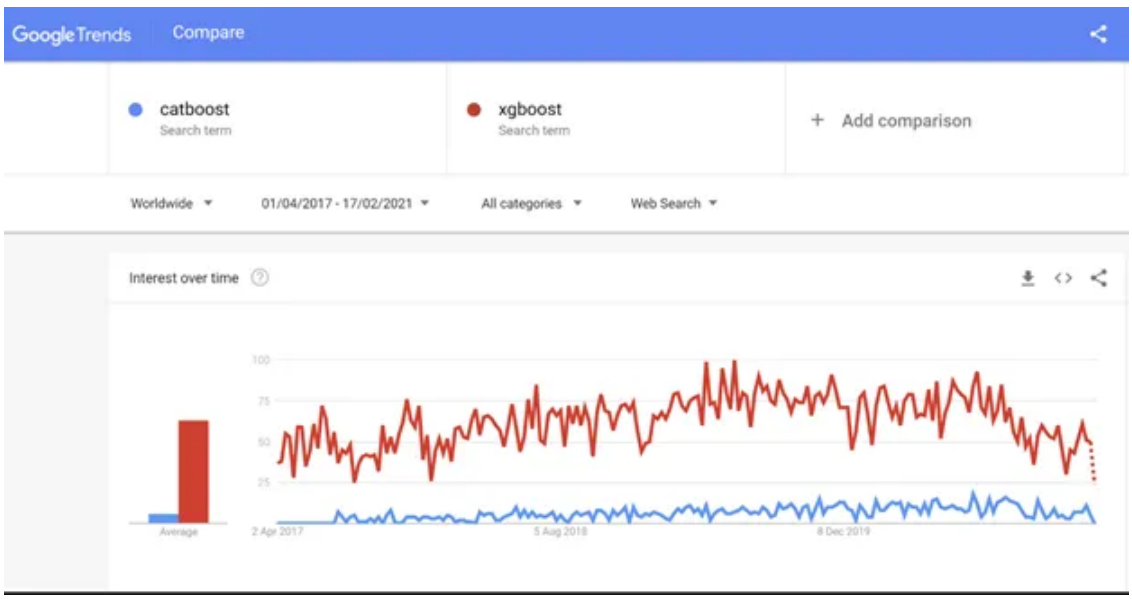
In the growing procedure of the decision trees, CatBoost does not follow similar gradient boosting models. Instead, CatBoost grows oblivious trees, which means that the trees are grown by imposing the rule that all nodes at the same level, test the same predictor with the same condition, and hence an index of a leaf can be calculated with bitwise operations. The oblivious tree procedure allows for a simple fitting scheme and efficiency on CPUs, while the tree structure operates as a regularization to find an optimal solution and avoid overfitting.

> *Compared computational efficiency:*



Learning speed, Yandex [2]

> *According to Google trends, CatBoost still remains relatively unknown in terms of search popularity compared to the much more popular XGBoost algorithm.*



Google Trends (2021) [3]

CatBoost still remains fairly unknown, but the algorithm offers immense flexibility with its approach to handling heterogeneous, sparse, and categorical data while still supporting fast training time and already optimized hyperparameters.

Application

The objective of this tutorial is to provide a hands-on experience to CatBoost regression in Python. In this simple exercise, we will use the Boston Housing dataset to predict Boston house prices. But the applied logic on this data is also applicable to more complex datasets.

So let's get started.

First, we need to import the required libraries along with the dataset:

```
import catboost as cb
import numpy as np
import pandas as pd
import seaborn as sns
import shap
import load_boston
from matplotlib import pyplot as pltfrom sklearn.datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.inspection import permutation_importance

boston=load_boston()

boston = pd.DataFrame(boston.data, columns=boston.feature_names)
```

Data exploration

It is always considered good practice to check for any Na values in your dataset, as it can confuse or at worst, hurt the performance of the algorithm.

```
boston.isnull().sum()
```

However, this dataset does not contain any Na's.

The data exploration and feature engineering phase are some of the most crucial (and time-consuming) phases when making data science projects. But in this context, the main emphasis is on introducing the CatBoost algorithm. Hence, if you want to dive deeper into the descriptive analysis, please visit EDA & Boston House Cost Prediction [4].

Training

Next, we need to split our data into 80% training and 20% test set.

The target variable is 'MEDV' — Median value of owner-occupied homes in $1000's.

```
X, y = load_boston(return_X_y=True)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state=5)
```

In order to train and optimize our model, we need to utilize CatBoost library integrated tool for combining features and target variables into a train and test dataset. This pooling allows you to pinpoint target variables, predictors, and the list of categorical features, while the pool constructor will combine those inputs and pass them to the model.

```
train_dataset = cb.Pool(X_train, y_train)
test_dataset = cb.Pool(X_test, y_test)
```

Next, we will introduce our model.

```
model = cb.CatBoostRegressor(loss_function='RMSE')
```

We will use the RMSE measure as our loss function because it is a regression task.

In situations where the algorithms are tailored to specific tasks, it might benefit from parameter tuning. The CatBoost library offers a flexible interface for inherent grid search techniques, and if you already know the Sci-Kit Grid Search function, you will also be familiar with this procedure.

In this tutorial, only the most common parameters will be included. These parameters include a number of iterations, learning rate, L2 leaf regularization, and tree depth. If you want to discover more hyperparameter tuning possibilities, check out the CatBoost documentation here.

```
grid = {'iterations': [100, 150, 200],
```

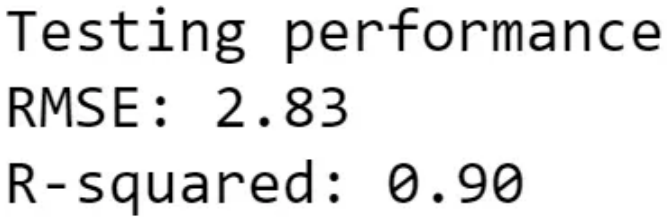

Performance evaluation

We have now performed the training of our model, and we can finally proceed to the evaluation of the test data.

Let's see how the model performs.

```
pred = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred)))
r2 = r2_score(y_test, pred)
```

```
print(“Testing performance”)
print(‘RMSE: {:.2f}’.format(rmse))
print(‘R2: {:.2f}’.format(r2))
```



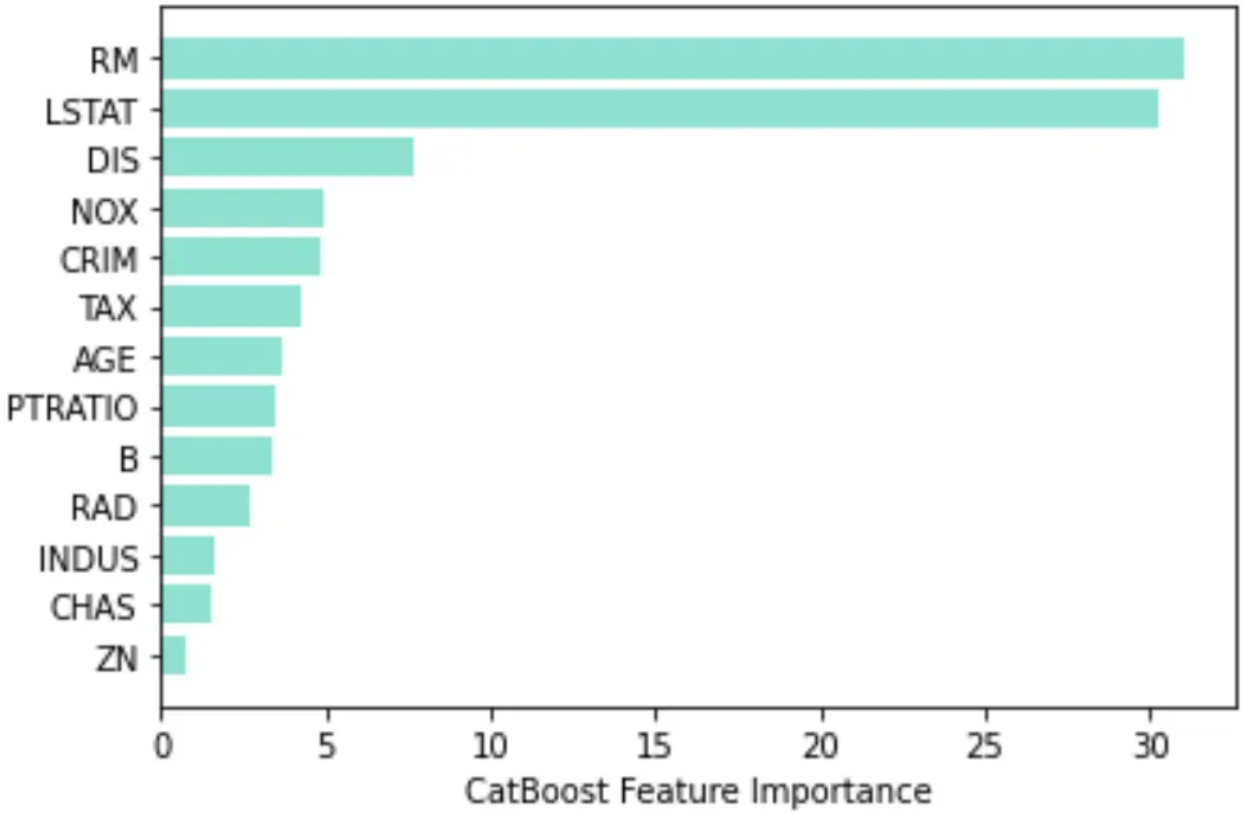
Test performance

As depicted above we achieve an R-squared of 90% on our test set, which is quite good, considering the minimal feature engineering.

Inference-wise, CatBoost also offers the possibility to extract Variable Importance Plots. Hence, a Variable Importance Plot could reveal underlying data structures that might not be visible to the human eye.

In this example, we are sorting the array in ascending order and making a horizontal bar plot of the features with the ***least*** important features at the bottom and ***most*** important features at the top of the plot.

```
sorted_feature_importance = model.feature_importances_.argsort()
plt.barh(boston.feature_names[sorte                    tance],
        model.feature_importances_[                    mportance],
        color='turquoise')
plt.xlabel("CatBoost Feature Importance")
```



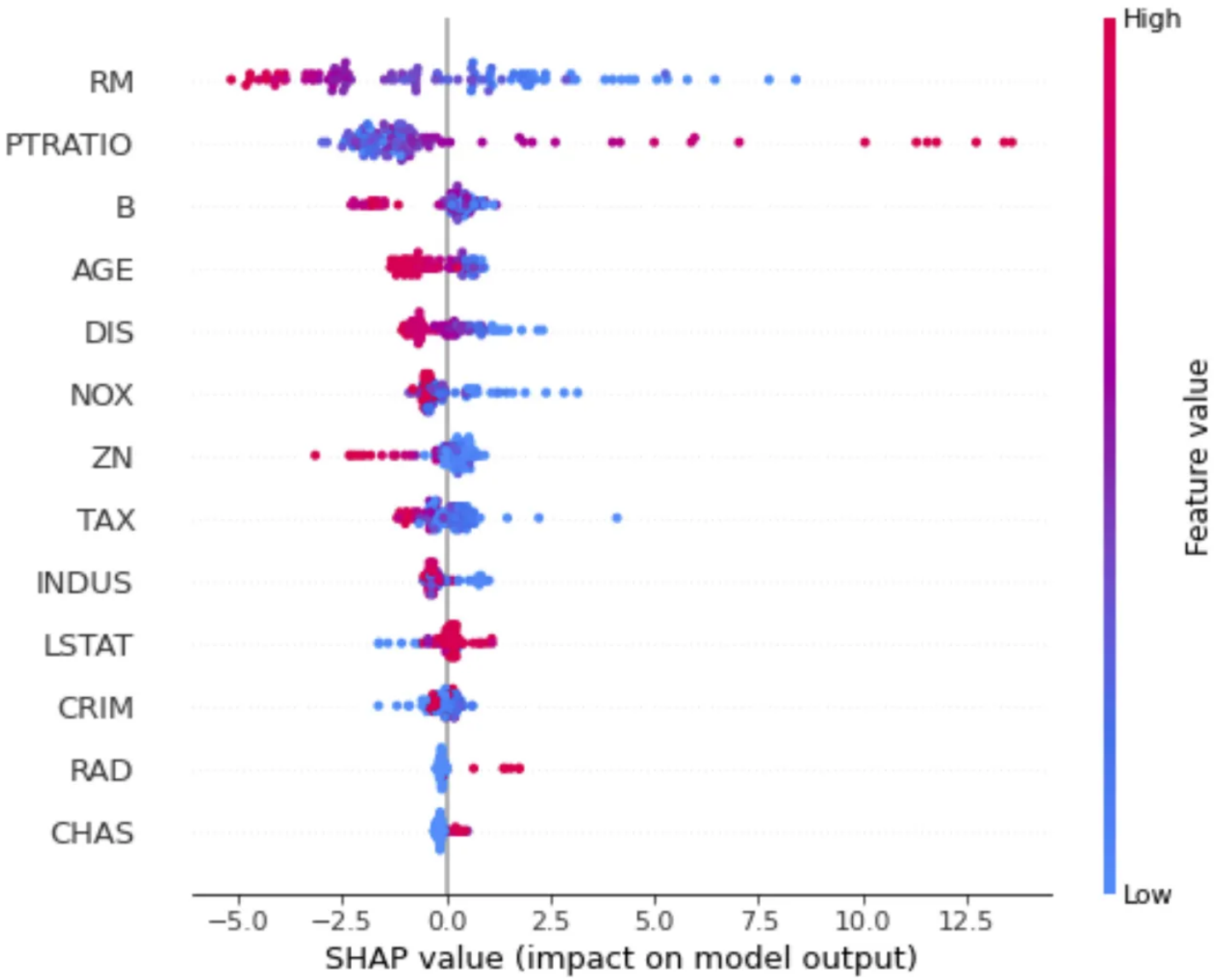
Variable Importance Plot

According to the illustration, these features listed above holds valuable information to predicting Boston house prices. The most influential variables are the average number of rooms per dwelling (RM) and the percentage of the lower status of the population (LSTAT).

SHapley Additive exPlanations (SHAP) plots are also a convenient tool to explain the output of our machine learning model, by assigning an importance value to each feature for a given prediction. SHAP values allow for interpreting what features driving the prediction of our target variable.

```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

shap.summary_plot(shap_values, X_test, feature_names =
boston.feature_names[sorted_feature_importance])
```



SHAP Plot

In the SHAP plot, the features are ranked based on their average absolute SHAP and the colors represent the feature value (red high, blue low). The higher the SHAP value, the larger the predictor's attribution. In other words, the SHAP values represent a predictor's responsibility for a change in the model output, i.e. prediction of Boston house prices. This reveals for example that larger RM are associated with increasing house prices while a higher LSTAT is linked with decreasing house prices, which also intuitively makes sense.

If you want to know more about SHAP plots and CatBoost, you will find the documentation here.

Final notes

So, in this tutorial, we have successfully built a CatBoost Regressor using Python, which is capable of predicting 90% of the variability in Boston house prices with an average error of 2,830$. Additionally, we have looked at Variable Importance Plots and the features associated with Boston house price predictions. If you want to learn more, I recommend you try out other datasets as well and delve further into the many approaches to customizing and evaluating your model.

Thanks for Reading!

Sources

[1] Yandex, Company description, (2020), https://yandex.com/company/

[2] Catboost, CatBoost overview (2017), https://catboost.ai/

[3] Google Trends (2021), https://trends.google.com/trends/explore?date=2017-04-01%202021-02-18&q=CatBoost,XGBoost

[4] A. Bajaj, EDA & Boston House Cost Prediction (2019), https://medium.com/@akashbajaj0149/eda-boston-house-cost-prediction-5fc1bd662673

Catboost Gradient Boosting Hands On Tutorials Python Machine Learning

---