Gradient Boosted Decision Trees-Explained

Soner Yıldırım Follow
Feb 17 · 6 min read ★

With detailed explanation of boosting and scikit-learn implementation

A decision tree builds upon iteratively asking questions to partition data. It is easier to conceptualize the partitioning data with a visual representation of a decision tree:

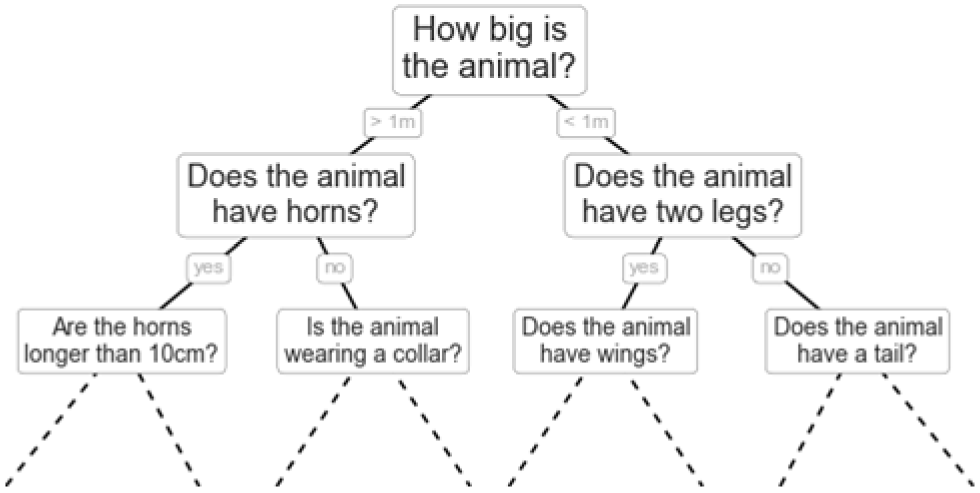

How big is the animal?
> 1m
< 1m
Does the animal have horns?
Does the animal have two legs?
yes
no
yes
no
Are the horns longer than 10cm?
Is the animal wearing a collar?
Does the animal have wings?
Does the animal have a tail?

Figure source

One decision tree is prone to overfitting. To reduce the risk of overfitting, models that combine many decision trees are preferred. These combined models also have better performance in terms of accuracy. **Random forests** use a method called **bagging** to combine many decision trees to create an ensemble. Bagging simply means combining in parallel. If you want to learn detailed information about decision trees and random forests, you can refer to the post below.

Decision Tree and Random Forest — Explained

Detailed explanation with theory and examples with code

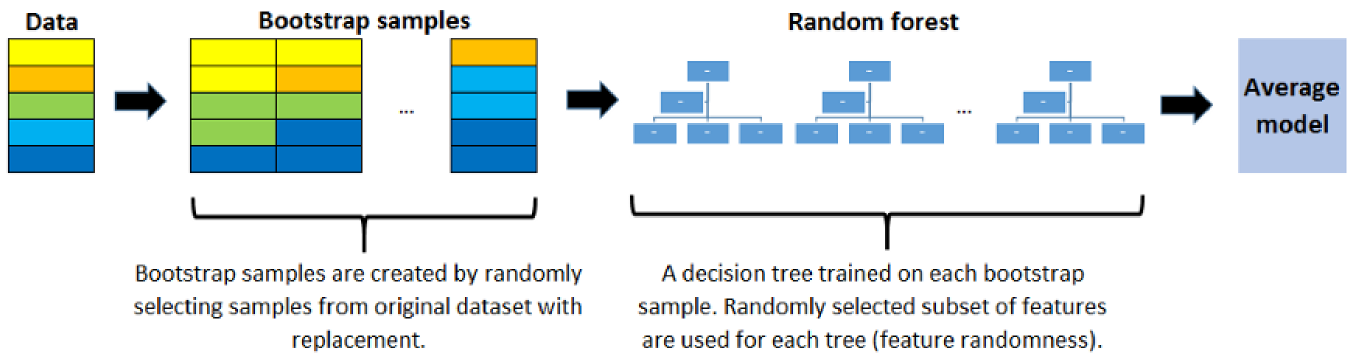
towardsdatascience.com

In this post, I will cover **gradient boosted decision trees** algorithm which uses **boosting** method to combine individual decision trees.

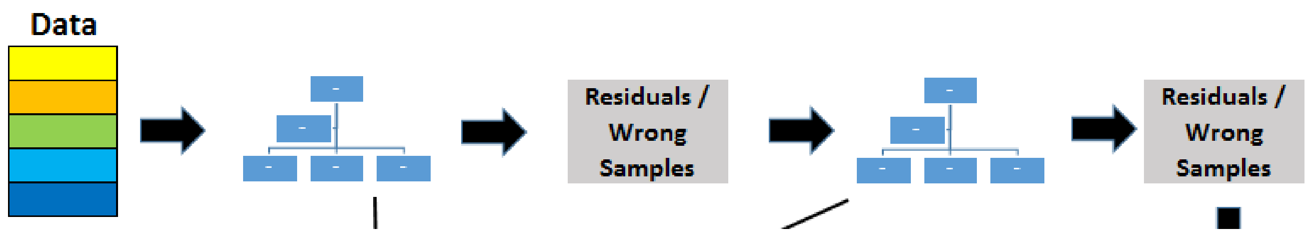
Boosting means combining a learning algorithm in series to achieve a strong learner from many sequentially connected weak learners. In case of gradient boosted decision trees algorithm, the weak learners are decision trees.

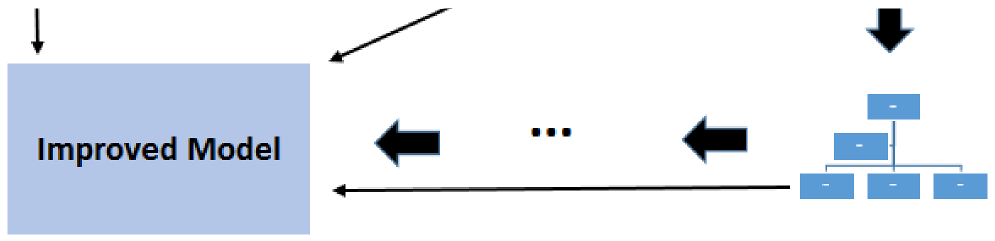
Each tree attempts to minimize the errors of previous tree. Trees in boosting are weak learners but adding many trees in series and each focusing on the errors from previous one make boosting a highly efficient and accurate model. Unlike bagging, boosting does not involve bootstrap sampling. Everytime a new tree is added, it fits on a modified version of initial dataset.

Since trees are added sequentially, boosting algorithms learn slowly. In statistical learning, models that learn slowly perform better.


Data
Bootstrap samples
Random forest
Average model
Bootstrap samples are created by randomly selecting samples from original dataset with replacement.
A decision tree trained on each bootstrap sample. Randomly selected subset of features are used for each tree (feature randomness).

Random Forests





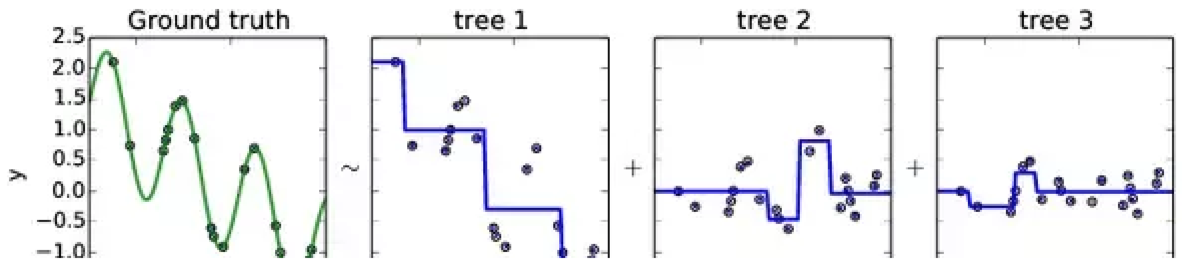
Gradient Boosted Decision Trees

Gradient boosting algorithm sequentially combines weak learners in way that each new learner fits to the residuals from the previous step so that the model improves. The final model aggregates the results from each step and a strong learner is achieved. **Gradient boosted decision trees** algorithm uses decision trees as week learners. A loss function is used to detect the residuals. For instance, mean squared error (MSE) can be used for a regression task and logarithmic loss (log loss) can be used for classification tasks. It is worth noting that existing trees in the model do not change when a new tree is added. The added decision tree fits the residuals from the current model. The steps are as follows:

- $f_1(x) \approx y$
- The residual is $y - f_1(x)$
- $f_2(x) \approx y - f_1(x)$
- The residual is $y - f_1(x) - f_2(x)$
- $f_3(x) \approx y - f_1(x) - f_2(x)$

Gradient boosted decision tree algorithm

The figure below visualize these steps:



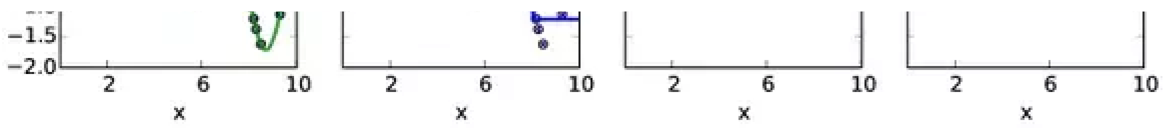
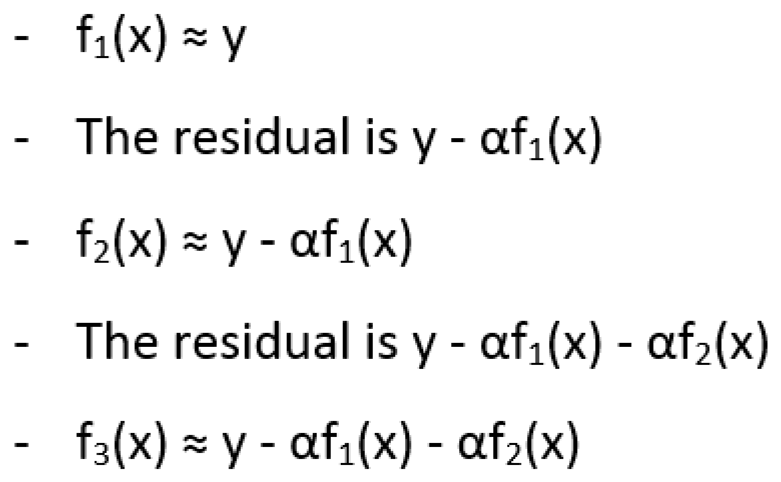


Figure source on Quora

Learning rate and n_estimators

Hyperparemetes are key parts of learning algorithms which effect the performance and accuracy of a model. **Learning rate** and **n_estimators** are two critical hyperparameters for gradient boosting decision trees. Learning rate, denoted as α, simply means how fast the model learns. Each tree added modifies the overall model. The magnitude of the modification is controlled by learning rate. The steps of gradient boosted decision tree algorithms with learning rate introduced:



- $f_1(x) \approx y$
- The residual is $y - \alpha f_1(x)$
- $f_2(x) \approx y - \alpha f_1(x)$
- The residual is $y - \alpha f_1(x) - \alpha f_2(x)$
- $f_3(x) \approx y - \alpha f_1(x) - \alpha f_2(x)$

Gradient boosted decision tree algorithm with learning rate (α)

The lower the learning rate, the slower the model learns. The advantage of slower learning rate is that the model becomes more robust and generalized. In statistical learning, models that learn slowly perform better. However, learning slowly comes at a cost. It takes more time to train the model which brings us to the other significant hyperparameter. **n_estimator** is the number of trees used in the model. If the learning rate is low, we need more trees to train the model. However, we need to be very careful at selecting the number of trees. It creates a high risk of overfitting to use too many trees.

Note on overfitting

One key difference between random forests and gradient boosting decision trees is the number of trees used in the model. Increasing the number of trees in random forests does not cause overfitting. After some point, the accuracy of the model does not increase by adding more trees but it is also not negatively effected by adding excessive trees. You still do not want to add unnecessary amount of trees due to computational reasons but there is no risk of overfitting associated with the number of trees in random forests.

However, the number of trees in gradient boosting decision trees is very critical in terms of overfitting. Adding too many trees will cause overfitting so it is important to stop adding trees at some point.

Pros and Cons

Pros:

- Highly efficient on both classification and regression tasks
- More accurate predictions compared to random forests.
- Can handle mixed type of features and no pre-processing is needed

Cons

- Requires careful tuning of hyperparameters
- May overfit if too many trees are used (n_estimators)
- Sensitive to outliers

Scikit-learn example

I will walk you through the steps of creating GradientBoostingClassifier() using one of the sample datasets available on scikit-learn.

Let’s start with importing dependencies:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
```

Load the dataset we just imported:

```
df = load_wine()
X = df.data
y = df.target
```

Divide the dataset into train and test sets using train_test_split module:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
                                                    random_state=42)
```

Then, we create a GradientDescentClassifier() object and fit train data:

```
clf_gbm = GradientBoostingClassifier()
```

```
clf_gbm.fit(X_train, y_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Then we predict the target values of test set using **predict** method and measure the accuracy of the model:

```
y_pred = clf_gbm.predict(X_test)
```

```
accuracy_score(y_test, y_pred)
```

```
0.9444444444444444
```

It is worth noting that data preparation, model creation and evaluation in real life projects are extremely complicated and time-consuming compared to this very simple example. I just wanted show you the steps of model creatinon. In real life cases, most of your time will be spent on data cleaning and preparation (assuming data collection is done by someone else). You will also need to spend a good amount of time on the accuracy of your model with hyperparameter tuning and re-evaluating many times.

A note on XGBOOST

XGBOOST (Extreme Gradient Boosting), founded by Tianqi Chen, is a superior implementation of Gradient Boosted Decision Trees. It is faster and has a better performance. XGBOOST is a very powerful algorithm and dominating machine learning competitions recently. I will write a detailed post about XGBOOST as well.

. . .

Thank you for reading. Please let me know if you have any feedback.

My articles on other Machine Learning Algorithms

- Naive Bayes Classifier — Explained
- Support Vector Machine — Explained
- Decision Trees and Random Forests — Explained

Machine Learning Data Science Gradient Boosting