Introduction to Data Science

GRADIENT BOOSTED DECISION TREES

BRIAN D'ALESSANDRO

GRADIENT BOOSTING TREES

We'll start by showing the basic functional form for classification:

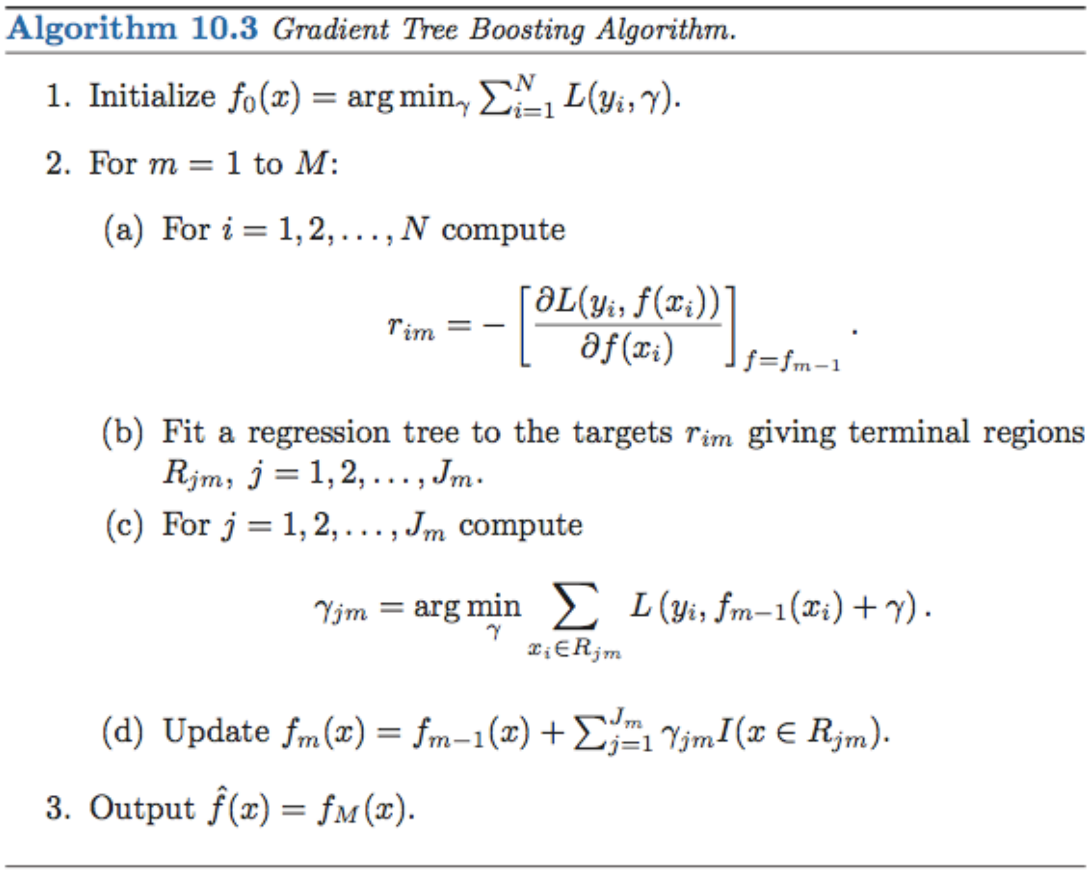
1. Start with a weighted sum of *m* individual decision trees

$$F^m(X) = \sum_{j=1}^{m} \gamma_j * T(X; \Theta_j)$$

2. Put the weighted sum into a sigmoid function to get a probability.

$$P(Y|X) = [1 + e^{-F^m(X)}]^{-1}$$

GENERAL ALGORITHM



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm}, j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Source: ESL2

Despite the fancy math used to get here, the algorithm is straightforward and intuitive.

We start with our best guess or model. We then compute the error at each observation, and fit a DT to that error.

The fitted DT becomes the next f(X) that gets added to the ensemble.

ILLUSTRATION (REGRESSION)

In Random Forests we see that each tree is learned on a bootstrap sample of the original data. GBT's take a completely different approach. GBT's rely on an iterative process, where each tree is fit on the residuals of the cumulative prediction from all prior trees. We'll demonstrate this on a regression problem first.

Step 0 – fit a null model to some data: f^0 = mean(Y)

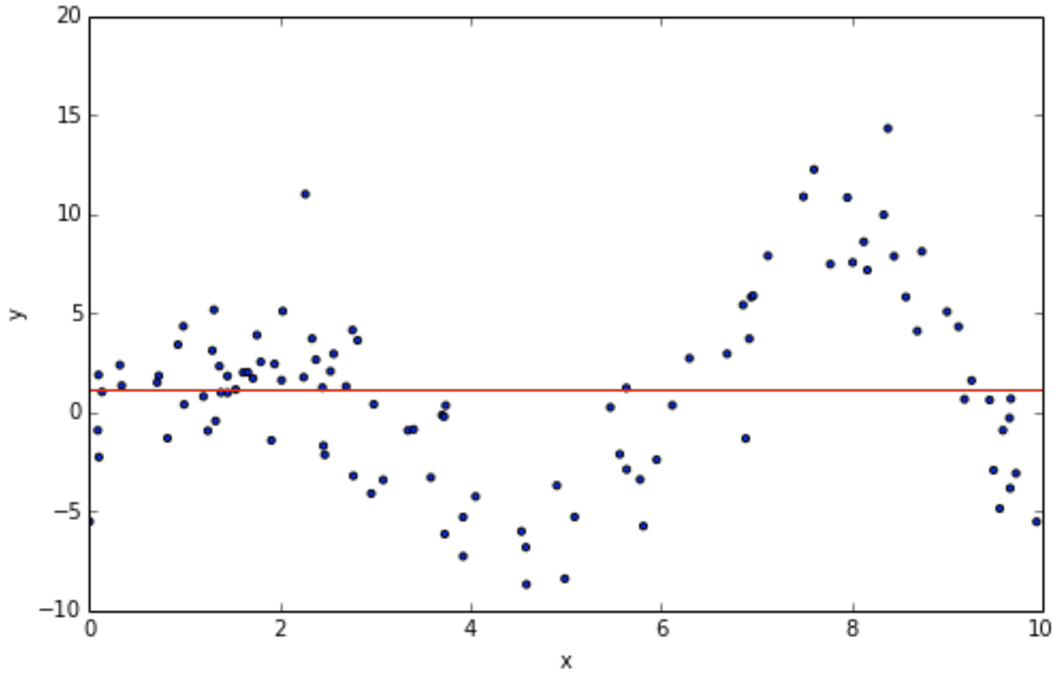
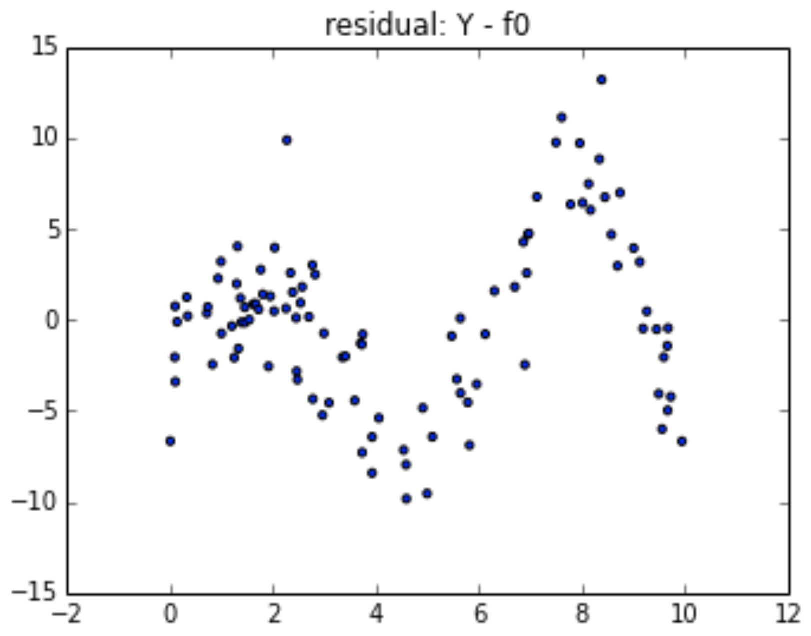
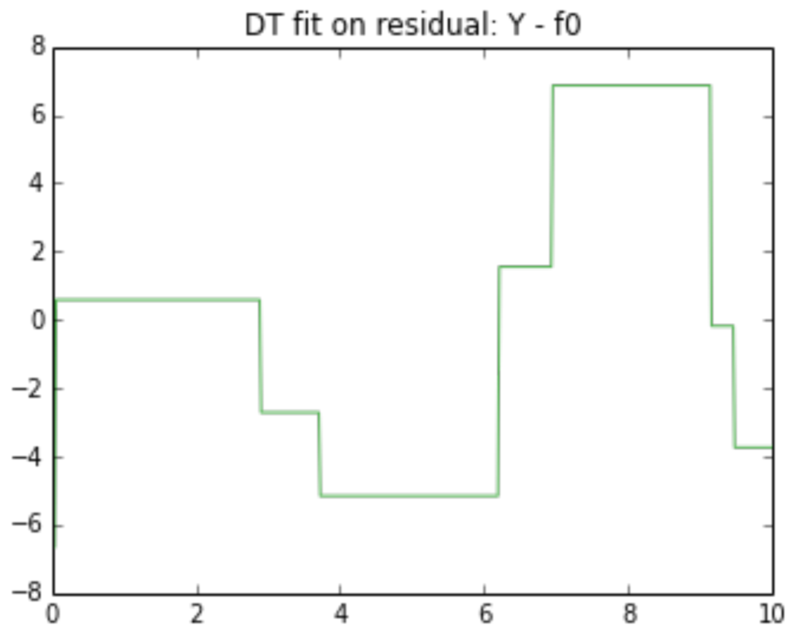


ILLUSTRATION (REGRESSION)

Step 1a – calculate residuals between Y and f^0
Step 1b – fit a new tree on $r^0 = (Y-f^0)$
Step 1c – apply line search to get weight of each tree
Step 1d – update function to get f^1





SO HOW DO WE SOLVE THIS?

First, we set it up as a greedy, stepwise optimization problem.

$$\boxed{F^m(X) = F^{m-1}(X) + \underset{f}{\operatorname{argmin}}\, \mathbb{L}(F^{m-1}(X) + f(X),\, Y)}$$

We assume $F^{m-1}(X)$ is fixed so we are greedily searching for the next additive function $f(X)$ that minimizes our loss function.

SO HOW DO WE SOLVE THIS?

First, we set it up as a greedy, stepwise optimization problem.

$$F^m(X) = F^{m-1}(X) + \underset{f}{\operatorname{argmin}} \mathbb{L}(F^{m-1}(X) + f(X), Y)$$

Which we solve approximately using a gradient descent approach.

$$\boxed{F^m(X) = F^{m-1}(X) - \gamma_m * \nabla_f \mathbb{L}(F^{m-1}(X) + f(X))}$$

We modify where we are at step *m-1*

In the negative direction of the error (or gradient) to move closer to the optimum

SO HOW DO WE SOLVE THIS?

First, we set it up as a greedy, stepwise optimization problem.

$$F^m(X) = F^{m-1}(X) + \underset{f}{\operatorname{argmin}} \, \mathbb{L}(F^{m-1}(X) + f(X), Y)$$

Which we solve approximately using a gradient descent approach.

$$F^m(X) = F^{m-1}(X) - \gamma_m * \nabla_f \, \mathbb{L}(F^{m-1}(X) + f(X))$$

We find gamma using an exact line search.

$$\boxed{\gamma^* = \underset{\gamma}{\operatorname{argmin}} \, \mathbb{L}(F^{m-1}(X) - \gamma \nabla_f \mathbb{L})}$$

This is a standard technique in optimization. First we find the optimal direction, and then we find the optimal step size

APPROXIMATING THE GRADIENT

In order to better generalize, we're better off approximating the gradient as a function of X. This can be done by fitting a tree to best approximate the gradient.

$$T(X;\Theta_m) = \underset{\Theta}{\operatorname{argmin}}(-\nabla_f \mathbb{L} - T(X;\Theta))^2$$

The gradient is defined only over the observed examples. To be able to generalize to unseen examples, we approximate it with a decision tree. The least squares approach enables us to learn a model that is close to the real-valued gradient.

And then using the approximated gradient in the line search.

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}}\ \mathbb{L}(F^{m-1}(X)\ -\ \gamma T(X;\Theta_m))$$

DIFFERENT ERROR/RESIDUAL FUNCTIONS

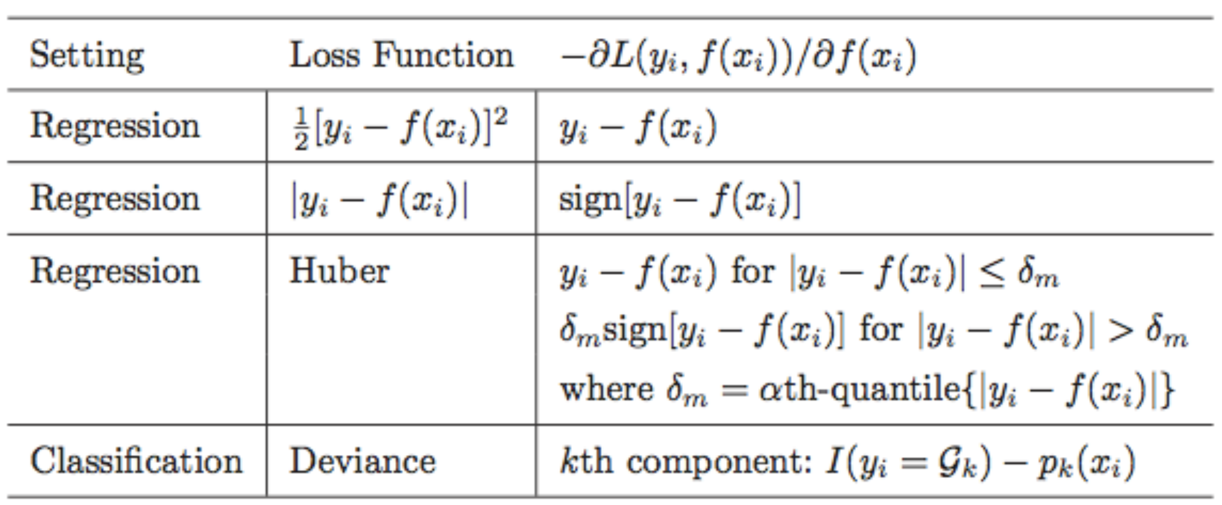
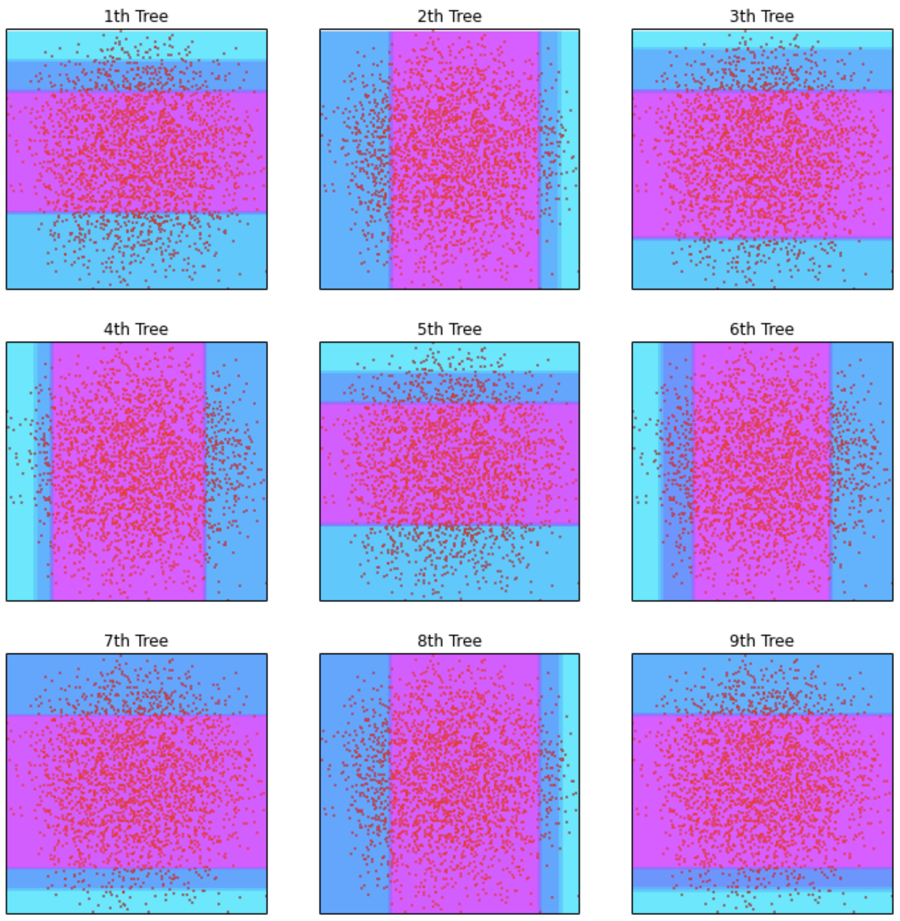


TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\|y_i - f(x_i)\|$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\|y_i - f(x_i)\| \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\|y_i - f(x_i)\| > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\|y_i - f(x_i)\|\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Source: ESL2

ILLUSTRATION (CLASSIFICATION)



In this example, we show the decision boundaries of the first 9 trees fit (separately) using the iterative GBT procedure.

We can see an interesting pattern, where each iteration learns a rectangular plane with alternating orientitation

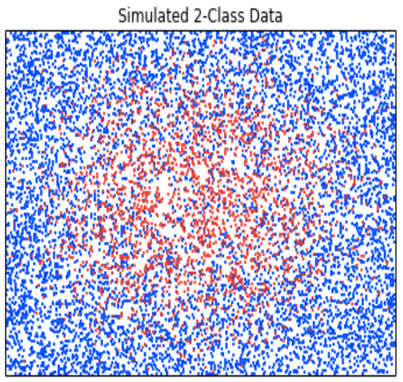
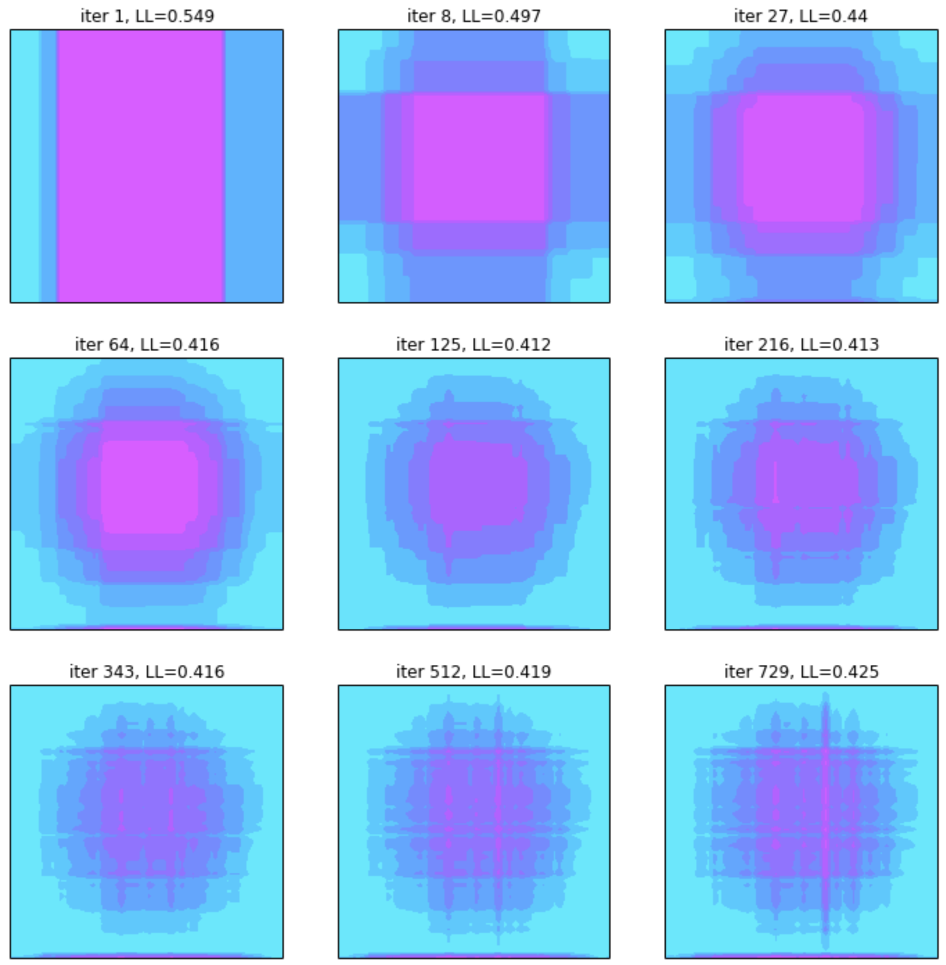
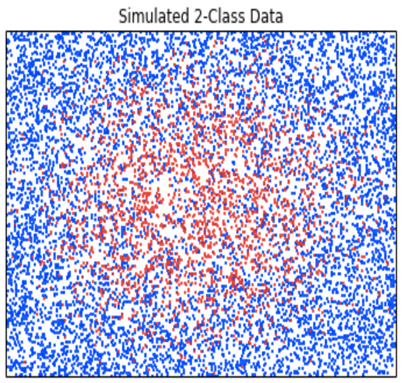


ILLUSTRATION (CLASSIFICATION)



This illustration shows the decision boundaries of the total classifier built off of N iterations of trees.

As the number of iterations increases, we see the classifier taking on the shape of the data. At the higher levels we also see the classification boundaries becoming more fragmented (and likely suggesting overfitting).

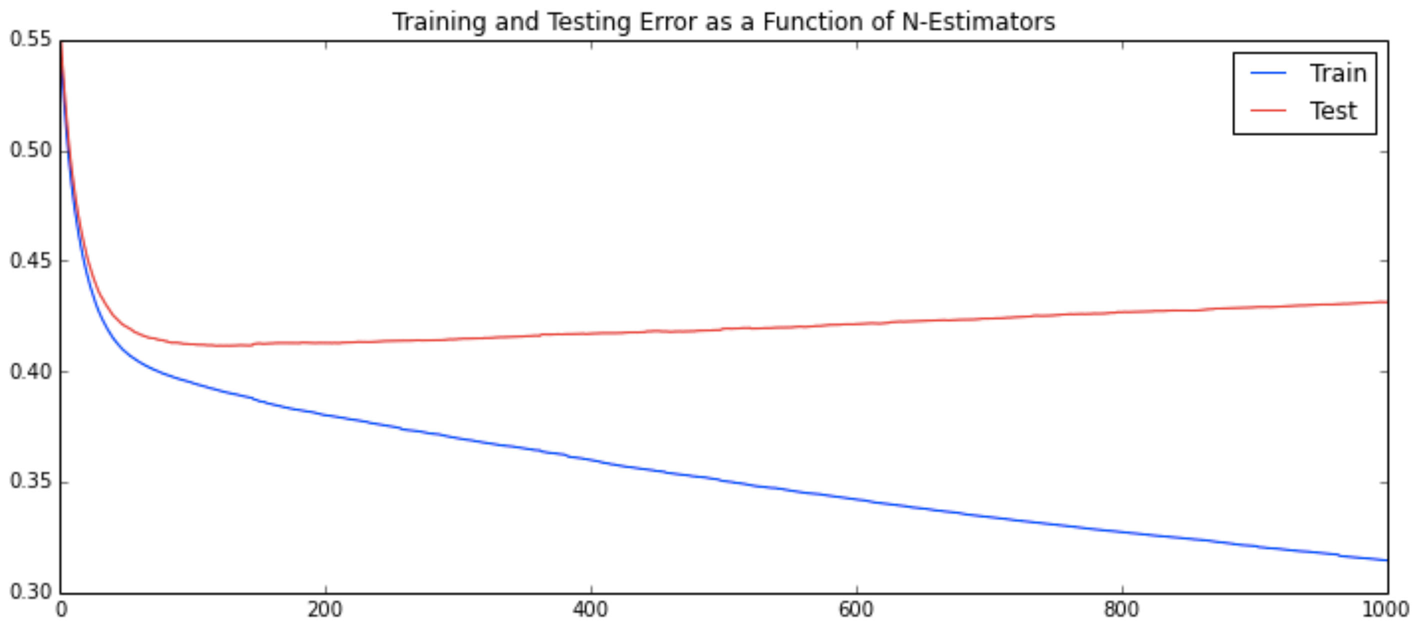


CONTROLLING COMPLEXITY

Like with Random Forests, our main level for controlling complexity is the number of trees. This can be found using cross-validation and grid search.

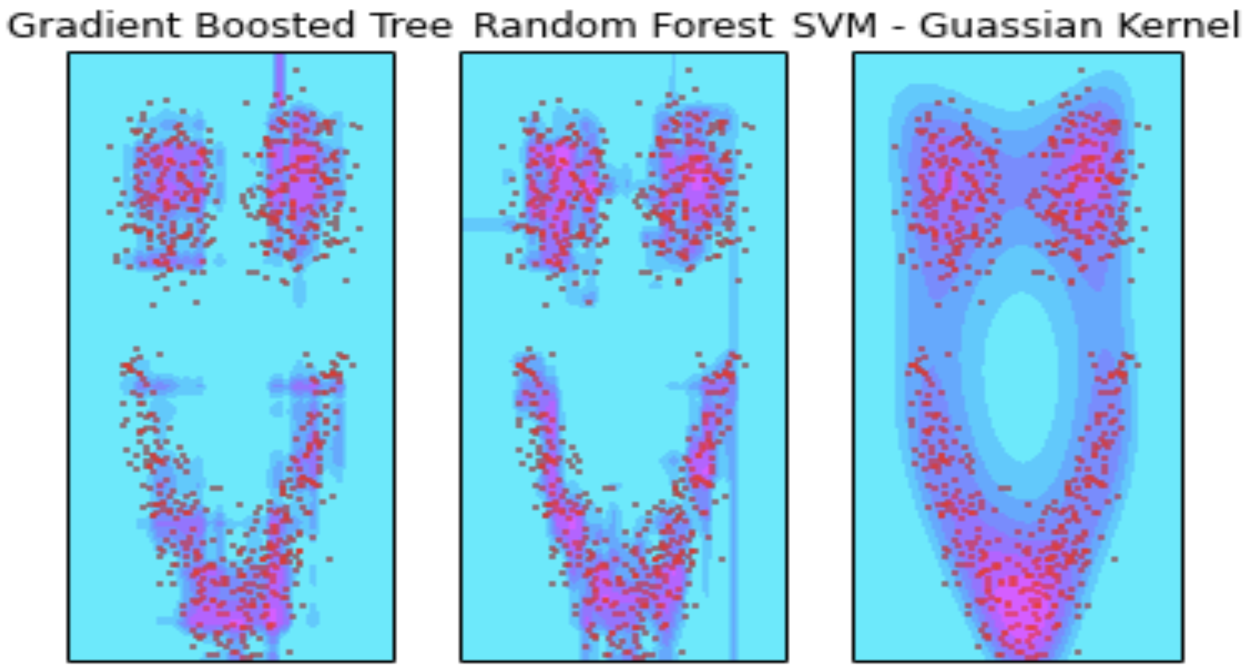
Additionally, we can add a regularization parameter to dampen the contribution of each tree.

$$F^m(X) = F^{m-1}(X) + \nu \gamma_m T(X; \Theta_m)$$



ADDING GBT TO YOUR TOOLKIT

GBT's are a great option for problems with non-linear decision boundaries. Their optimality is problem dependent, but are highly competitive with Random Forests and kernel based SVMs.



MODERN VARIANTS

As powerful as GBDTs are, newer variants of the algorithm have emerged and are generally considered the state of the art.

XGBoost

LightGBM

CatBoost

The exact differences are too detailed for the scope of this class. The main areas are in how they handle: ***Splitting, Categorical Variable Encoding, Feature Importance, Sampling, Missing Value Handling*** and ***Leaf Growth*****.**

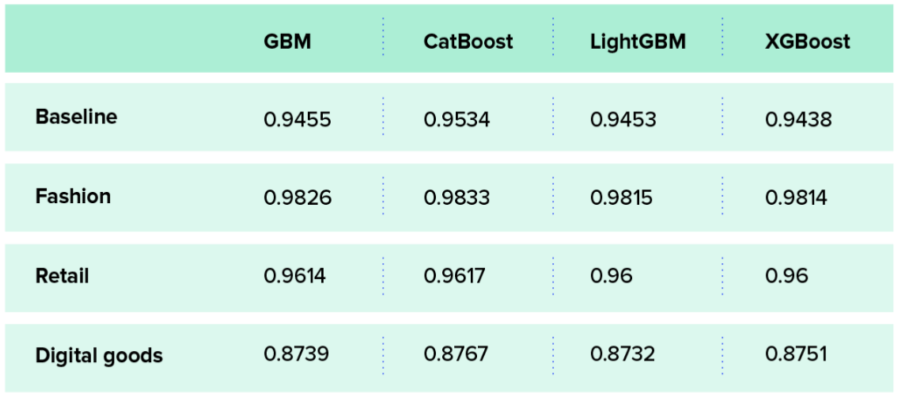
Which one to use is generally an empirical question.

MODERN VARIANTS

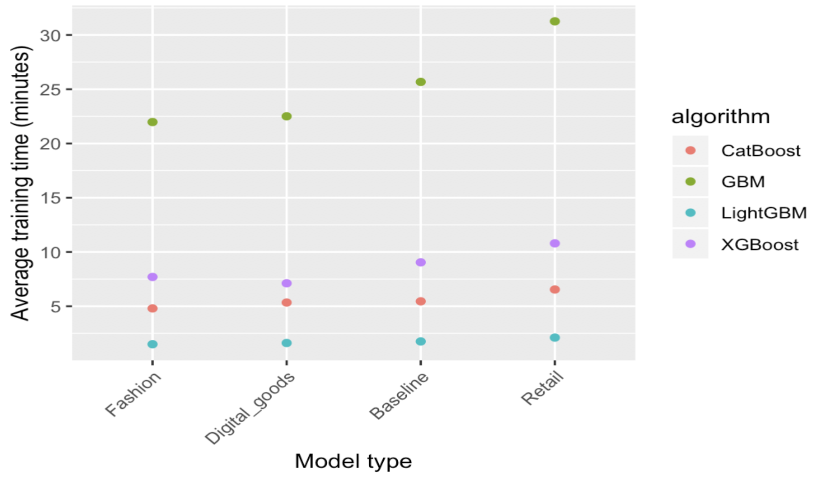
In this experiment the analyst tested the 4 GBM algorithms against 4 e-commerce fraud related data sets*, and compared the outcome using a weighted AUC metric.

Here we see that wAUC is fairly close for all algorithms in each data set. While they have different implementation details, the variants are fundamentally the same.

On the other hand, we see large variance in average training time. Speeding up training allows for more hyper-parameter tuning on larger data sets, which could ultimately lead to better performance.



	GBM	CatBoost	LightGBM	XGBoost
Baseline	0.9455	0.9534	0.9453	0.9438
Fashion	0.9826	0.9833	0.9815	0.9814
Retail	0.9614	0.9617	0.96	0.96
Digital goods	0.8739	0.8767	0.8732	0.8751



Source: https://medium.com/riskified-technology/xgboost-lightgbm-or-catboost-which-boosting-algorithm-should-i-use-e7fda7bb36bc