Data Science and Python

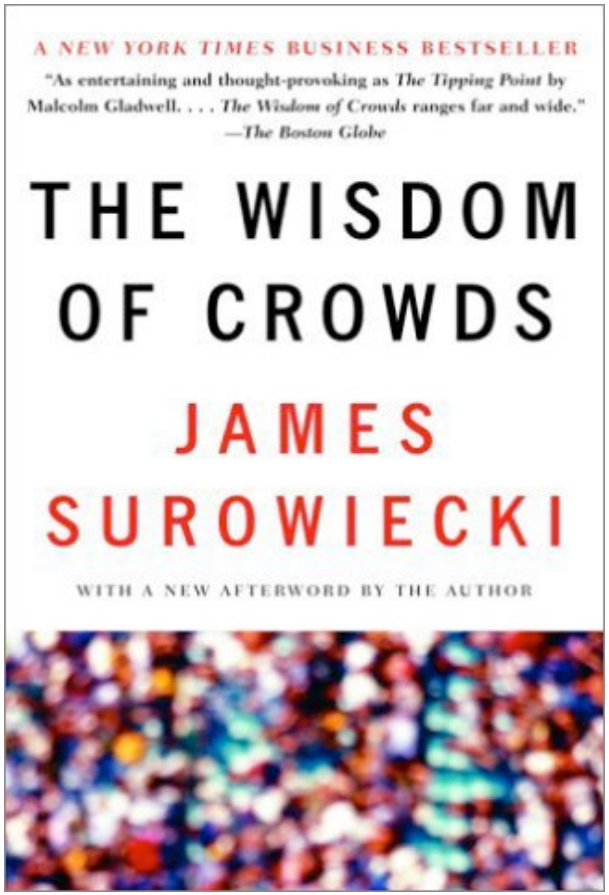
BRIAN D'ALESSANDRO

D'ATA SCIENTIST

ENSEMBLE METHODS

WISDOM OF CROWDS

This concept can be applied to Machine Learning. This is called Ensemble Learning



Conditions for This to Work

Diversity
Each person should have private information

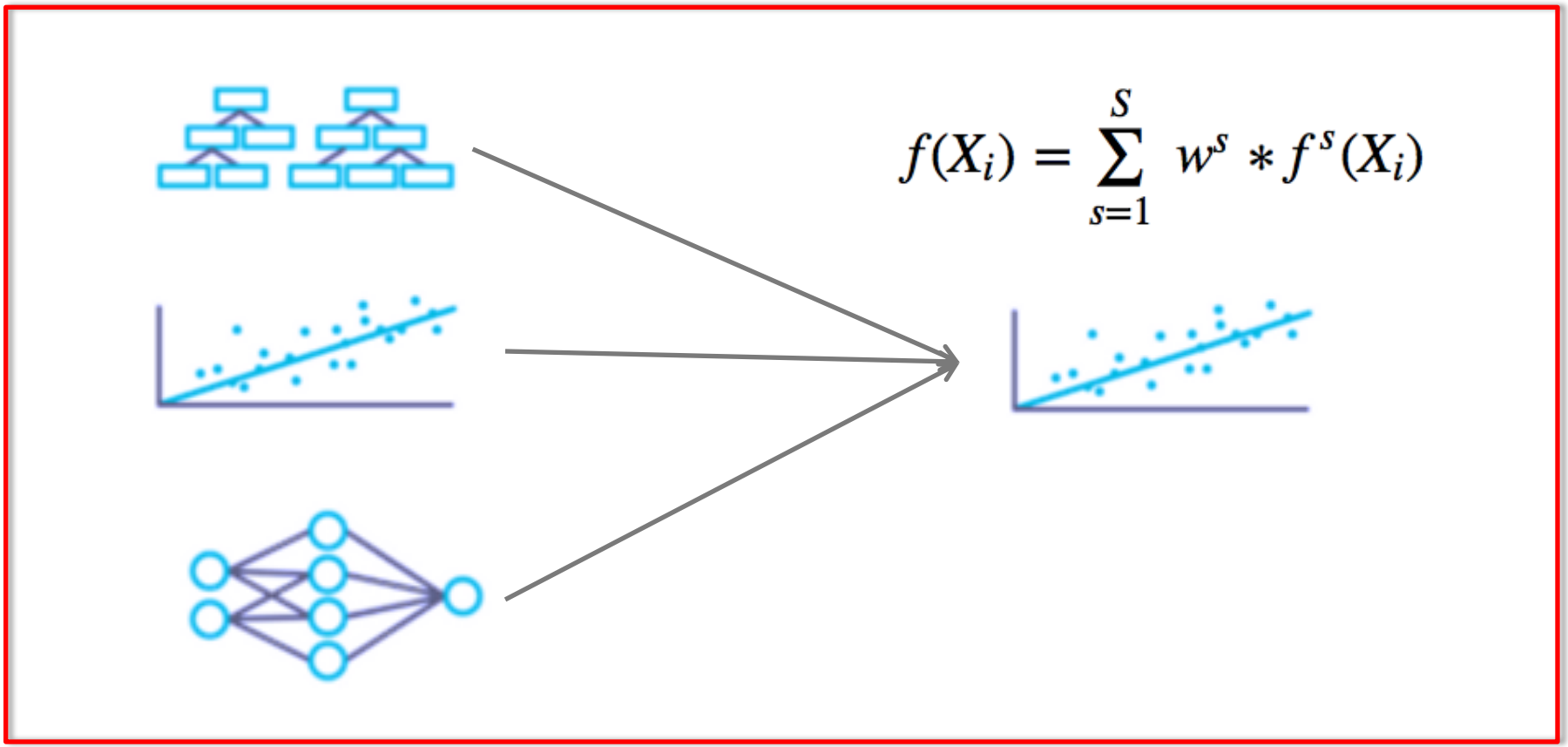
Independence
Peoples opinions aren't determined by opinions of others

Aggregation
Some mechanism exists for turning individual opinions into a collective decision.

A SIMPLE ENSEMBLE METHOD

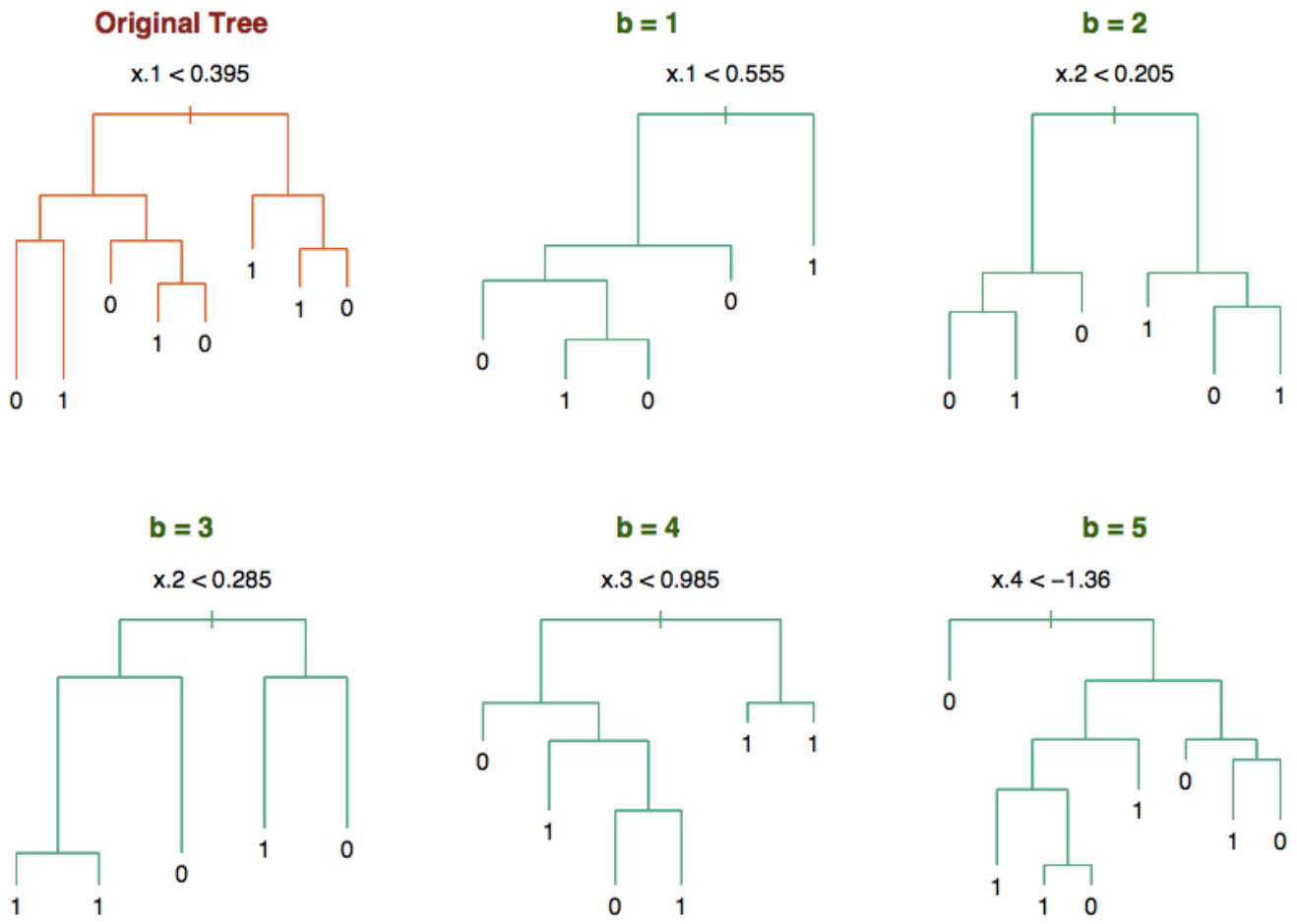
Stacking – Let $f^s(X_i)$ be some prediction on a sample using some arbitrary classifier s. Stacking is the process of taking a weighted combination of the predictions of a total of S different classifiers.

The weights can be a simple average, or learned via a secondary classification/regression process. Dstillery's MV model is a form of stacking ensemble.



WISDOM OF A CROWD OF TREES

Two common ensemble methods are Random Forests and Gradient Boosted Trees. Both learn a set of Decision Trees and make predictions based on some aggregation of the of individual tree predictions.



Source: ESL2

RANDOM FORESTS

BOOTSTRAP AGGREGATING

We can also use bootstrapping to improve the underlying predictions.

This is done via a procedure called Bootstrap Aggregating, or BAGGING.

The Bagging Procedure:

1. Create a bootstrap sample of the data
2. Fit the model on the bootstrap sample from step 1
3. Make prediction on train/test data using model from step 2
4. Repeat this N times, storing each prediction of step 3
5. Make a final prediction by averaging the bootstrapped predictions

BAGGING (FORMALIZED)

The Bagging Procedure:

We have data $D = [(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)]$ and we want to learn: $E[Y|X] = \hat{f}(X)$

Define a bootstrap sample D^b as N samples from D, sampled with replacement.

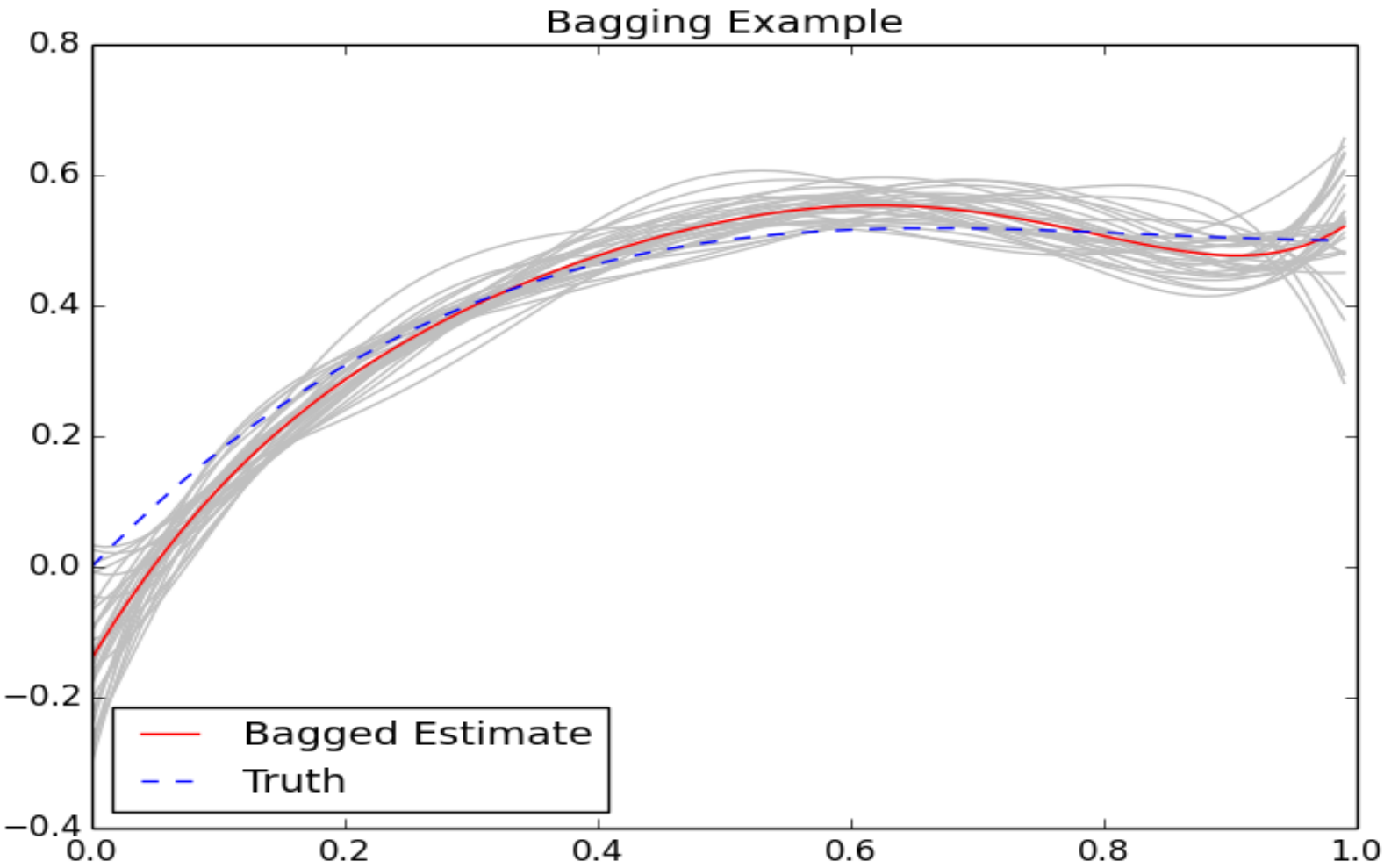
Let $E^b[Y|X] = \hat{f}^b(X)$ be the function learned from training set D^b.

Our bagged prediction is then the mean of all estimates of $f^b(X)$. I.e.,

$$\hat{f}_{bag}(X) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(X)$$

BAGGING

Random Forests use a technique called Bagging (Bootstrap Aggregating). The idea of Bagging is learn N models off of N bootstrap samples and average the N models together.



WHEN TO USE BAGGING

According to the seminal paper on bagging by Leo Breiman, Bagging:

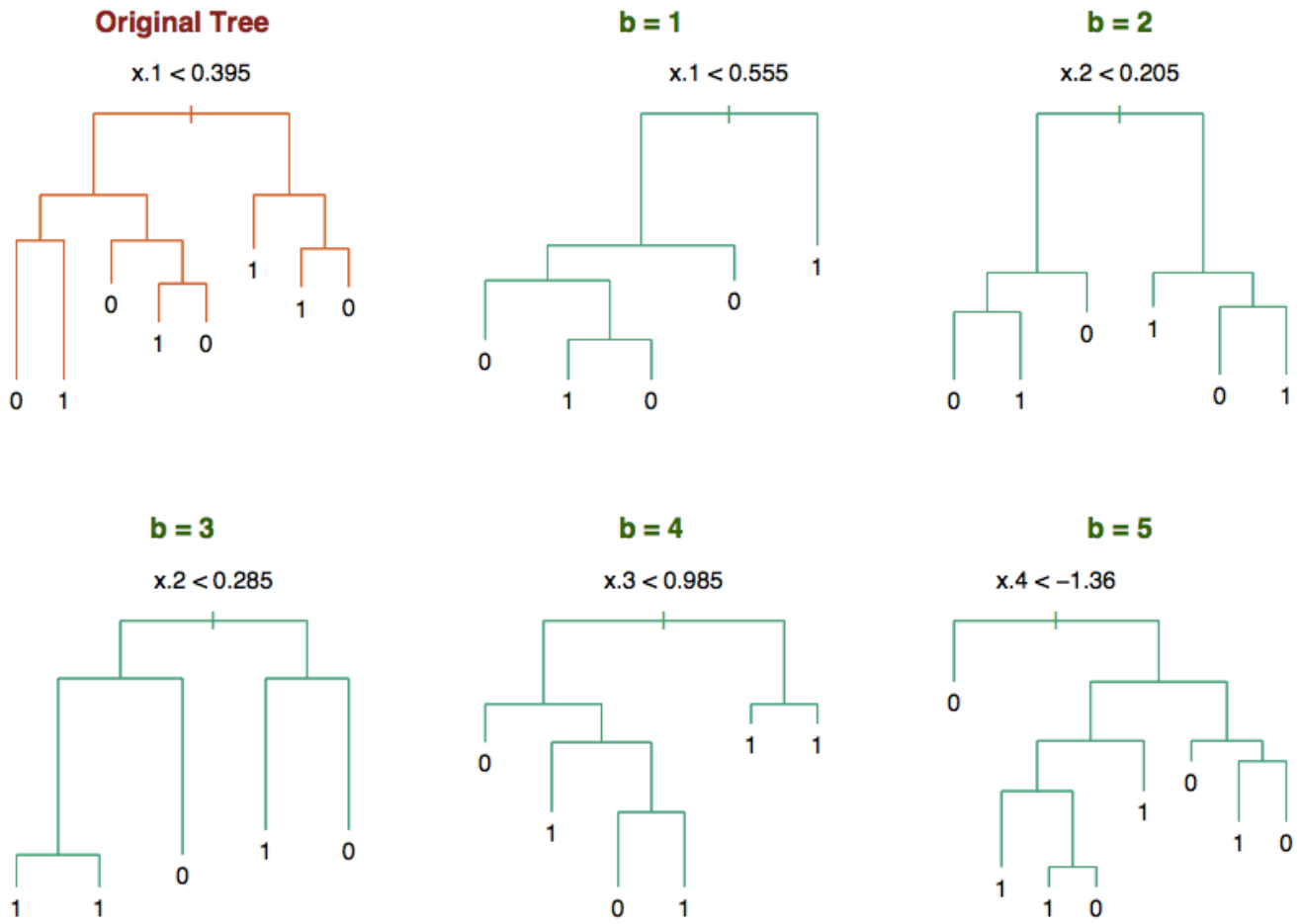
- ***"can push a good but unstable procedure a significant step towards optimality"***
- ***"can slightly degrade the performance of stable procedures"***

Q: What is an unstable procedure?

A: One in which small permutations of the training data result in dramatically different models

EXAMPLE UNSTABLE ALGORITHM

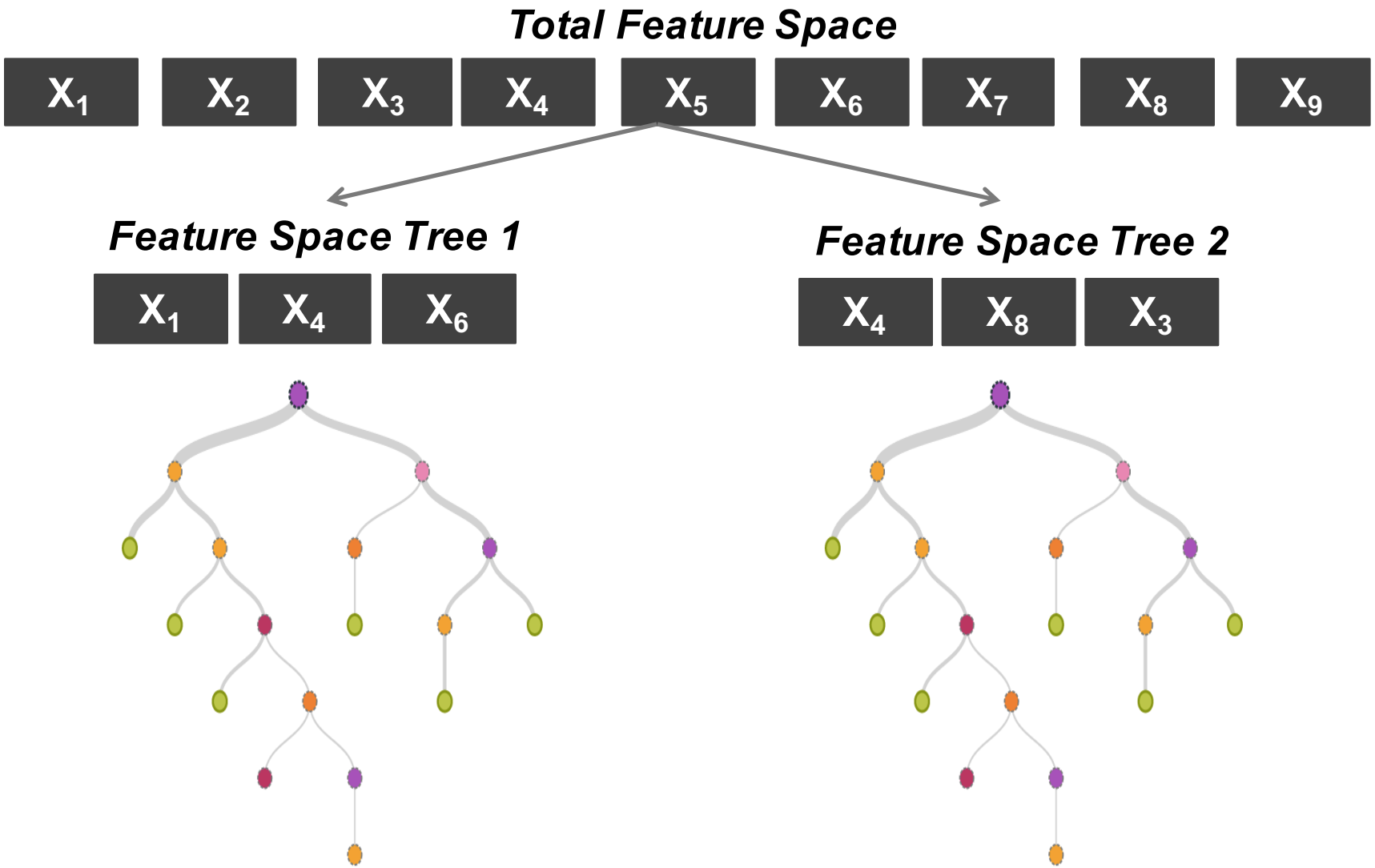
From ESL2: "…a decision tree on simulated data. Small variations of the data (generated by bootstrap sampling) produce wide variations in the learned tree."



Source: ESL2

DECORRELATING INDIVIDUAL TREES

For Bagging to work best, individual estimators should be as uncorrelated as possible. The RF algorithm accomplishes this by using a subset of randomly chosen features for each tree iteration.



THE RF ALGORITHM

The RF procedure is straightforward and easily parallelized.

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Source: ESL2

WHY IT WORKS

Bias

A decision tree is unstable, and has high variance, but can also have extremely low bias

- Can detect all manner of interaction effects
- Especially when allowed to grow very deep

The bias of the average of identically distributed trees is equal to the bias of the individual trees (in this case, very low).

Variance

If the variance of an individual tree is σ^2 and the pairwise correlation of any two trees is ρ, then the variance of the forest is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Randomly sampling features reduces the pairwise correlations ρ and reduces the 1st term above, while bootstrapping reduces the 2nd term above. Note though, that reducing the features decreases total variance but also increases the bias.

Source: ESL2

TUNING

RF's are quick to set up and might do fairly well straight out of the box. But nonetheless, tuning is always recommended.

Forest Level Parameters

- # trees (n_estimators) – increasing this decreases variance, but increases training time.
- # of features to sample (max_features) – the number of features sampled in each tree. Reducing this # increases the bias but decreases the RF variance.

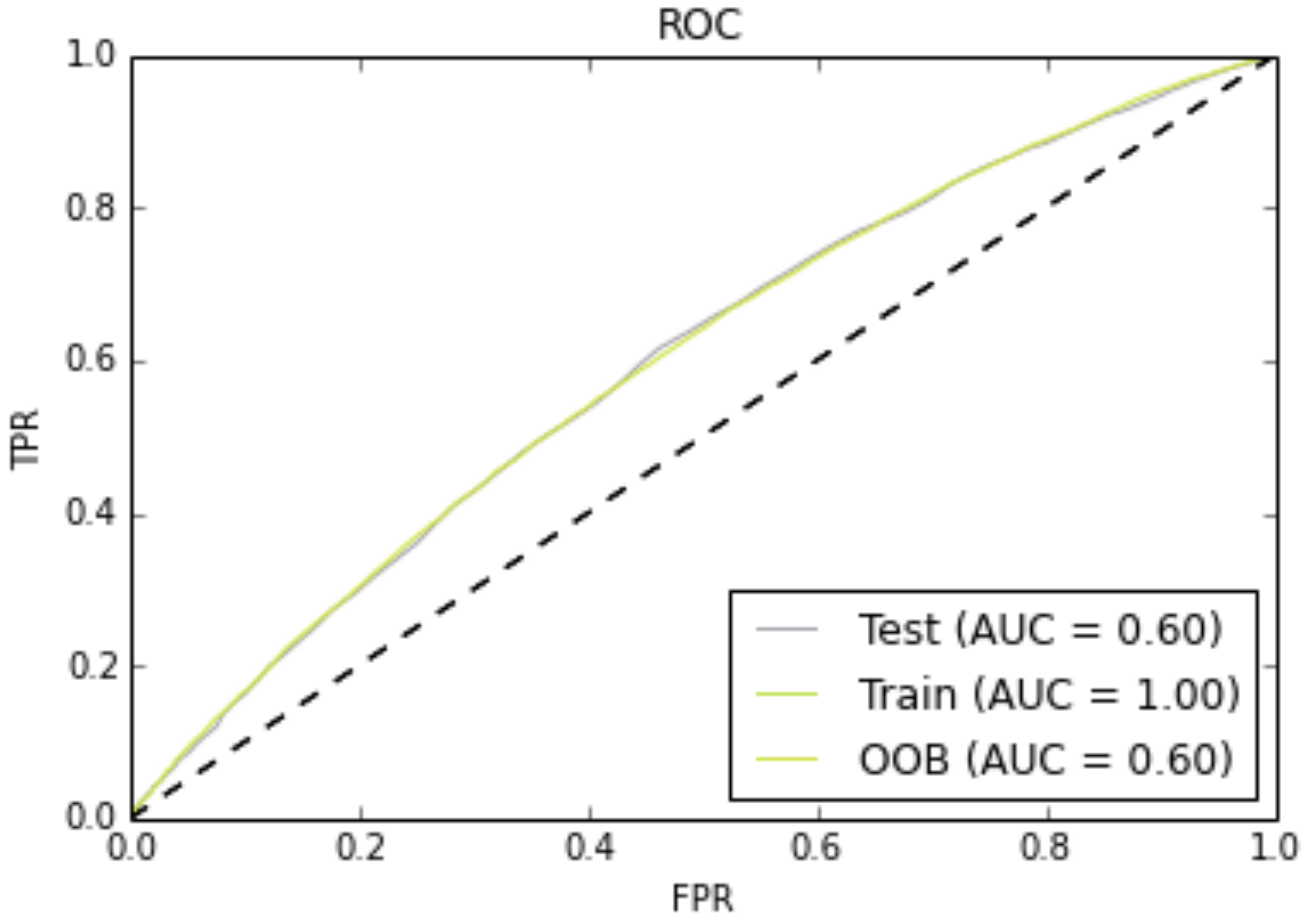
Tree Level Parameters

- # intermediate nodes (max_depth) – the size of the tree. Usually you don't want to limit this (i.e., set max_depth=None)
- Size of intermediate notes (min_sample_split) – the number of instances in an intermediate node, before splitting (usually good to set to 1).

We usually want to over-fit the individual trees, and as always, use some hold-out method to optimize Forest level parameters.

OUT-OF-BAG ERROR

Tuning via cross-validation can be very slow due to the number of trees that have to be built. Random Forests have a built in validation property called "out-of-bag" error estimation that can help us avoid x-validation. For each record, average $f(x_i, y_i)$ on all bootstrap iterations in which record i was not sampled. This out-of-bag prediction can then be used for out-of-sample error estimation.



Observations:

- Training AUC is 1!
- OOB AUC is almost = Holdout AUC

Results of random forest run on Churn data from HW 3.

GRADIENT BOOSTED TREES

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.

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

SO HOW DO WE SOLVE THIS?

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

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

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.

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

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.

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

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.

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

And then using the approximated gradient in the line search.

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

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

DIFFERENT 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	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Source: ESL2

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 in 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)

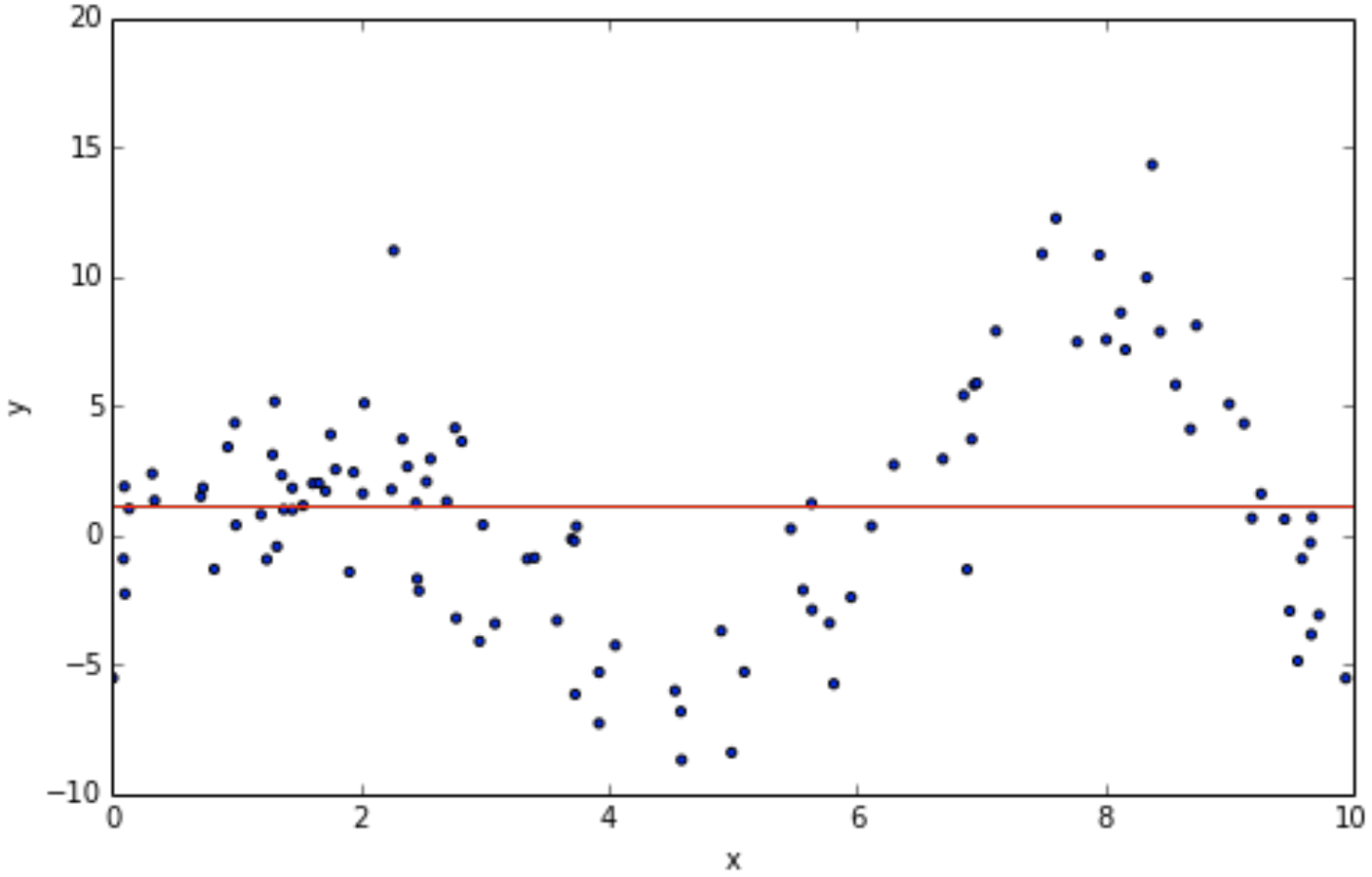
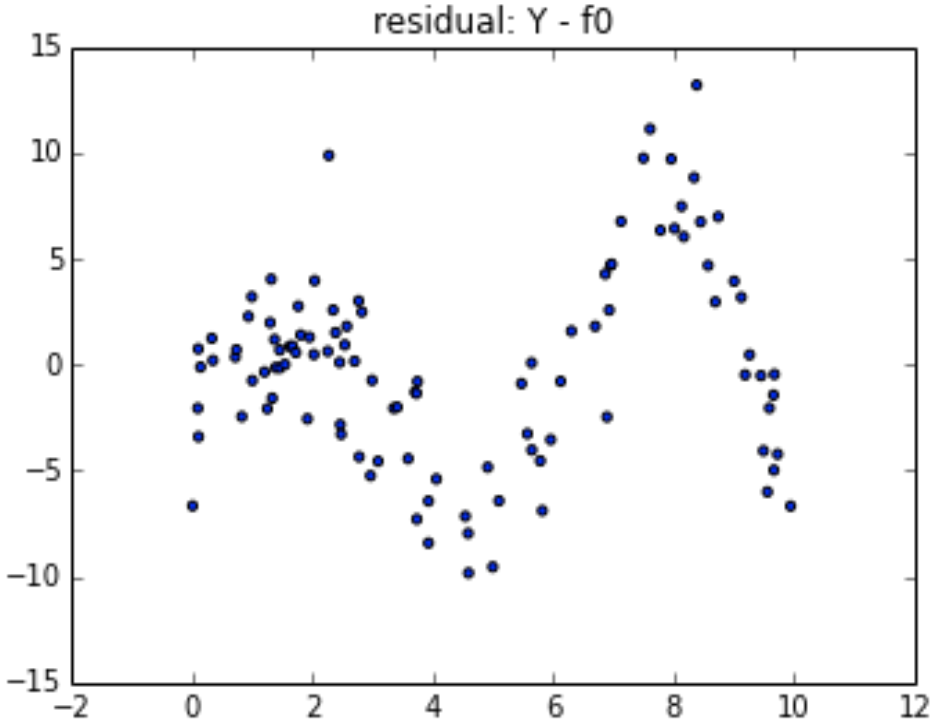


ILLUSTRATION (REGRESSION)

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 in 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 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



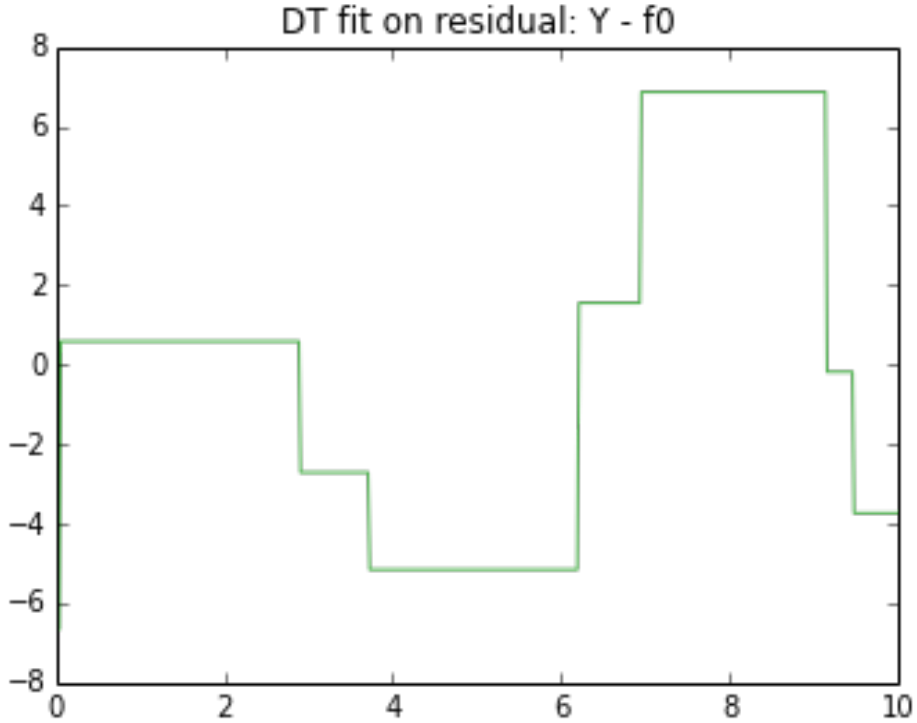
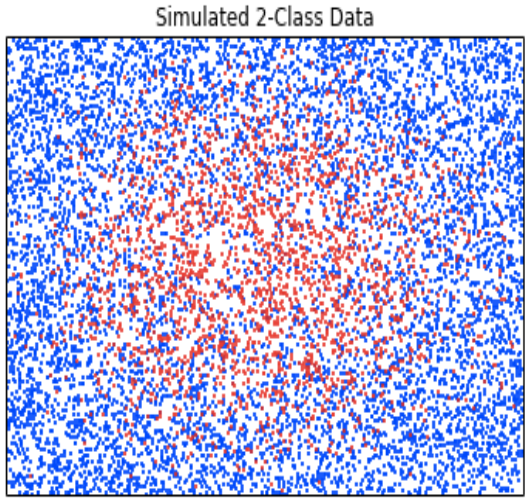


ILLUSTRATION (CLASSIFICATION)



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

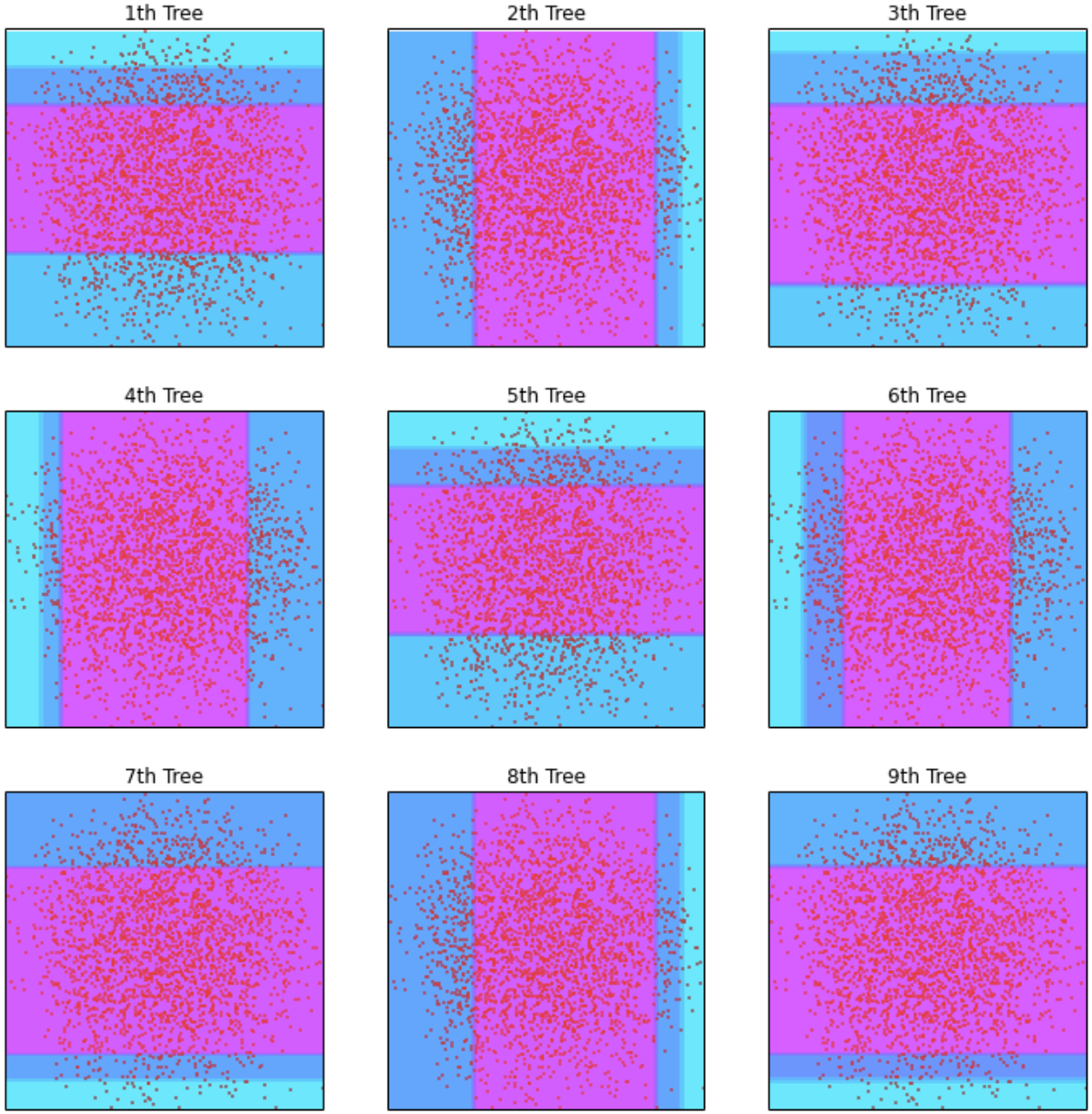
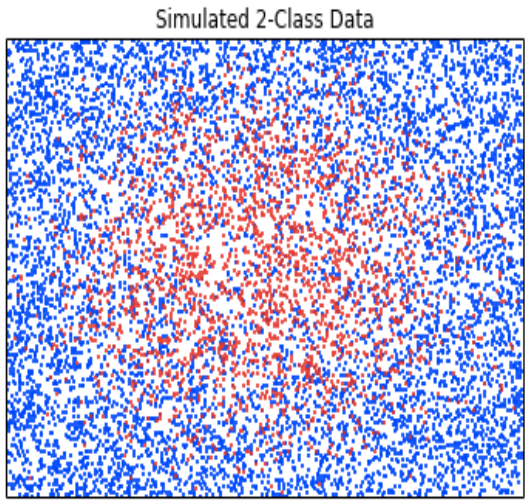
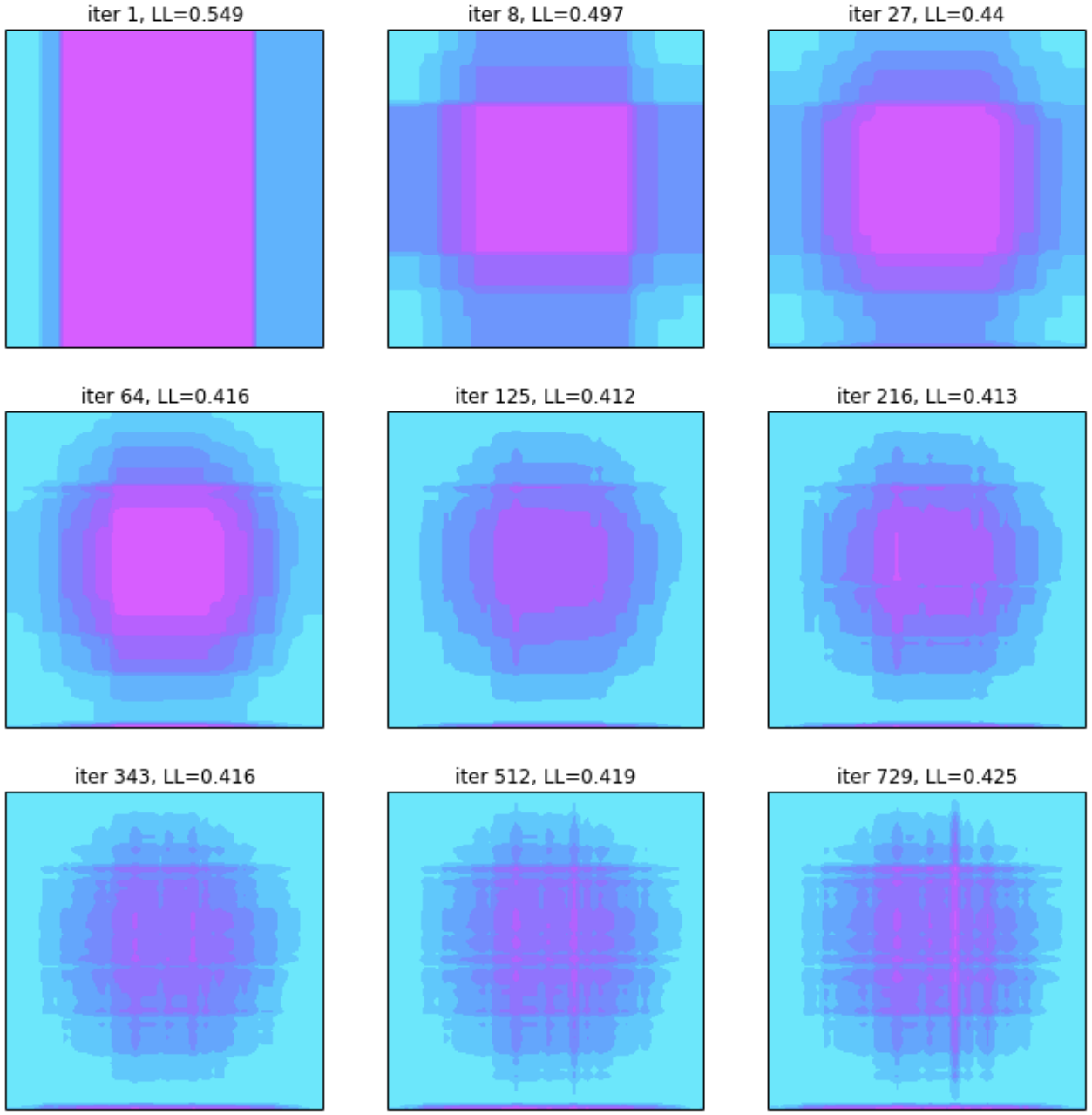


ILLUSTRATION (CLASSIFICATION)



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

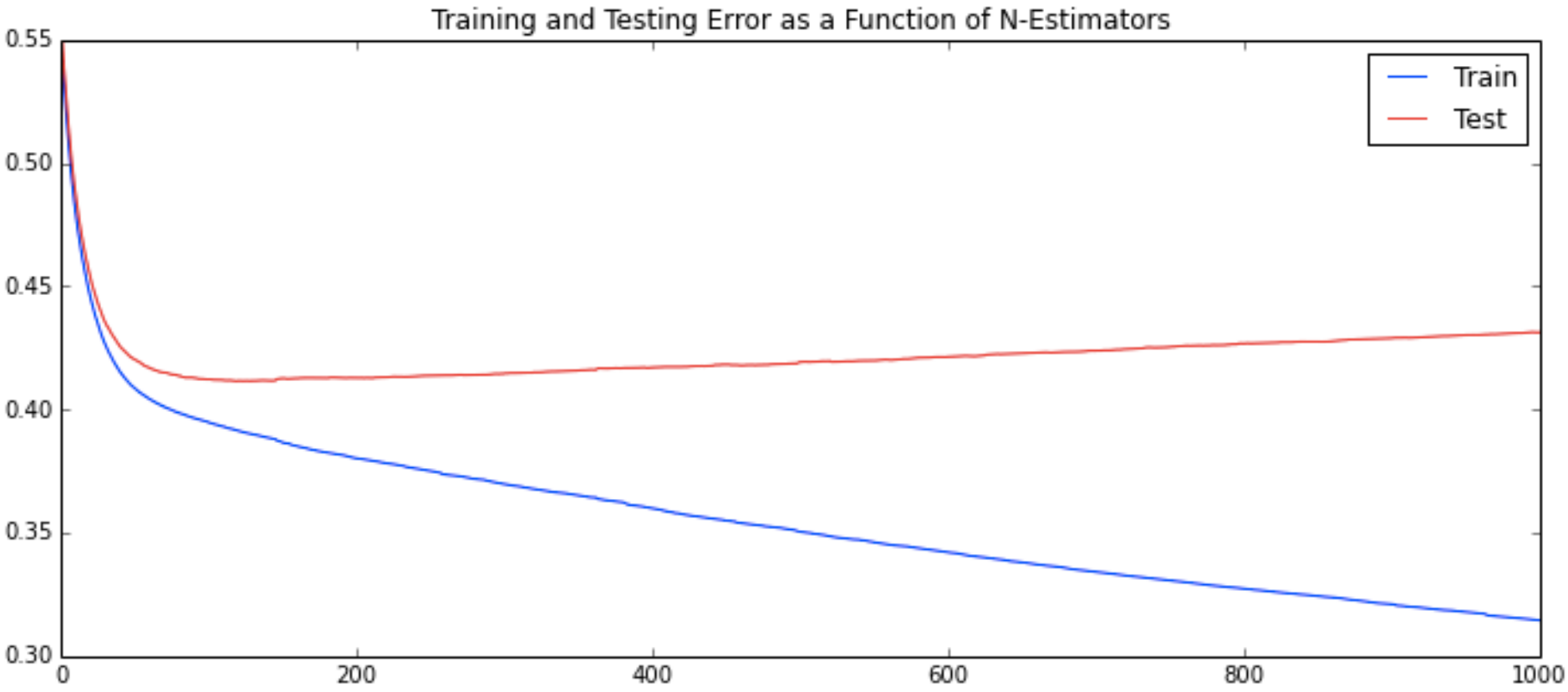


CONTROLLING COMPLEXITY

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.

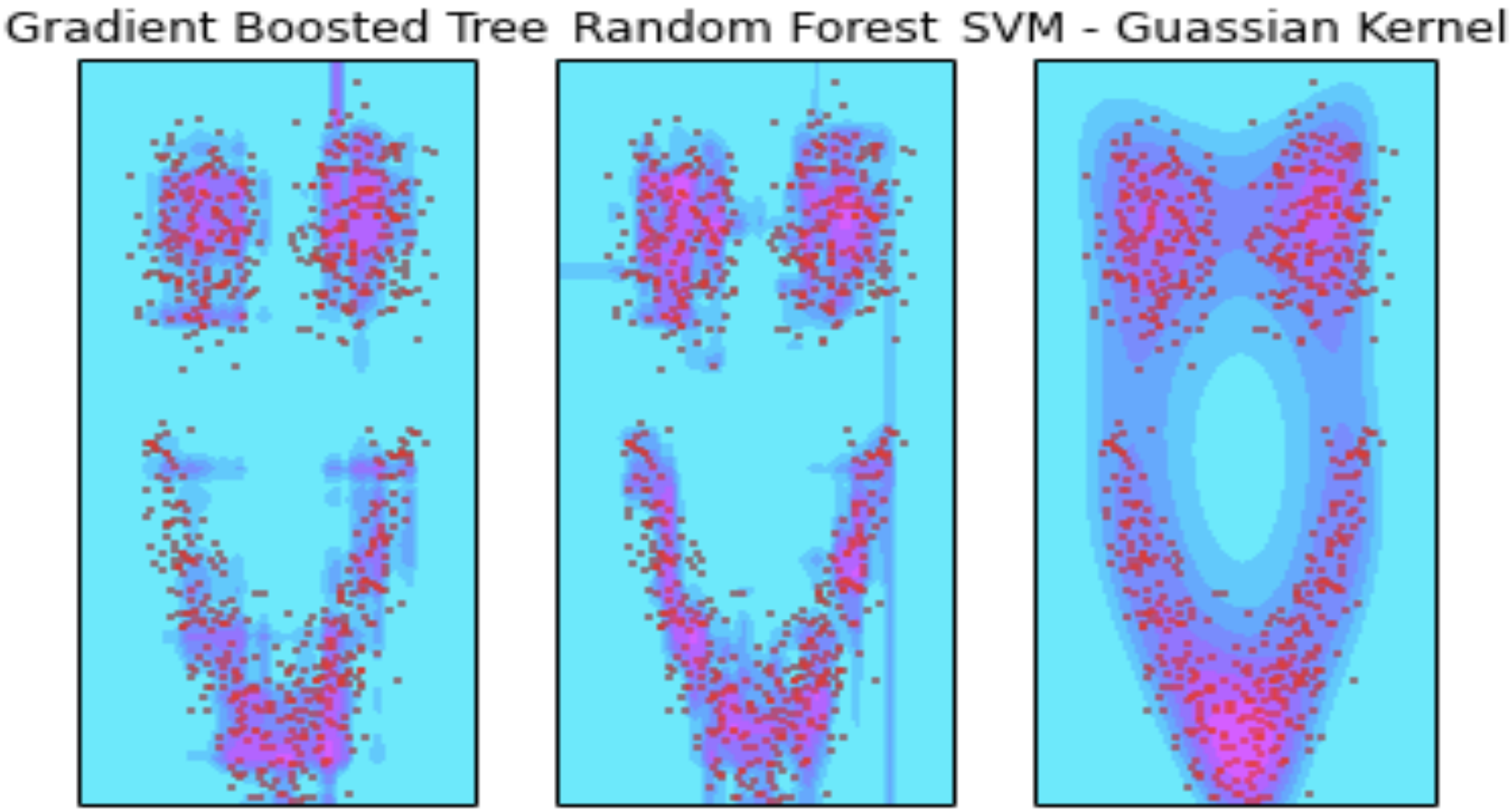
Additionally, we can add a regularization parameter to dampen the contribution of each tree.

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



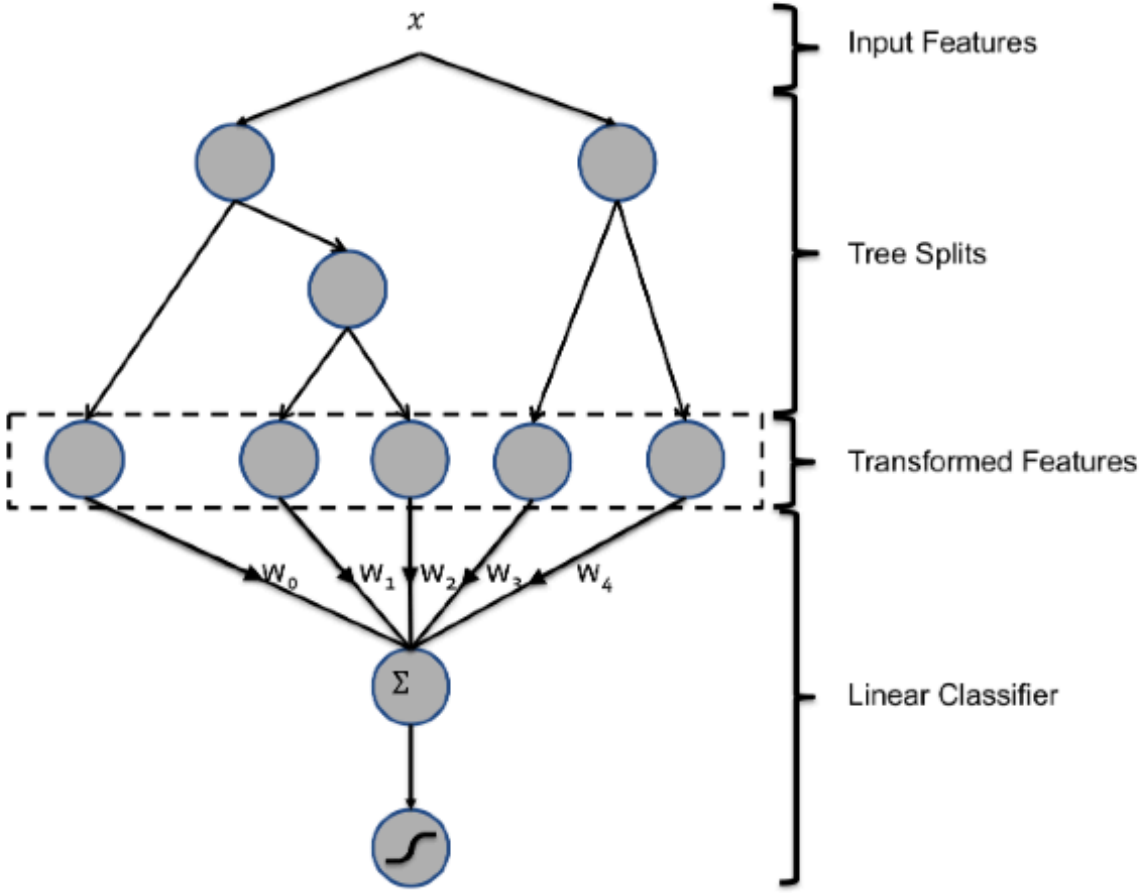
ADDING GBT TO YOUR TOOLKIT

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.



TREE BASED FEATURE ENGINEERING

Use the terminal nodes of the GBT trees as binary indicator features. Then build an SGD++ model on this enhanced feature set. Achieve the best of both worlds (scalability and performance).



Source: http://www.quinonero.net/Publications/predicting-clicks-facebook.pdf