Gradient Boosting

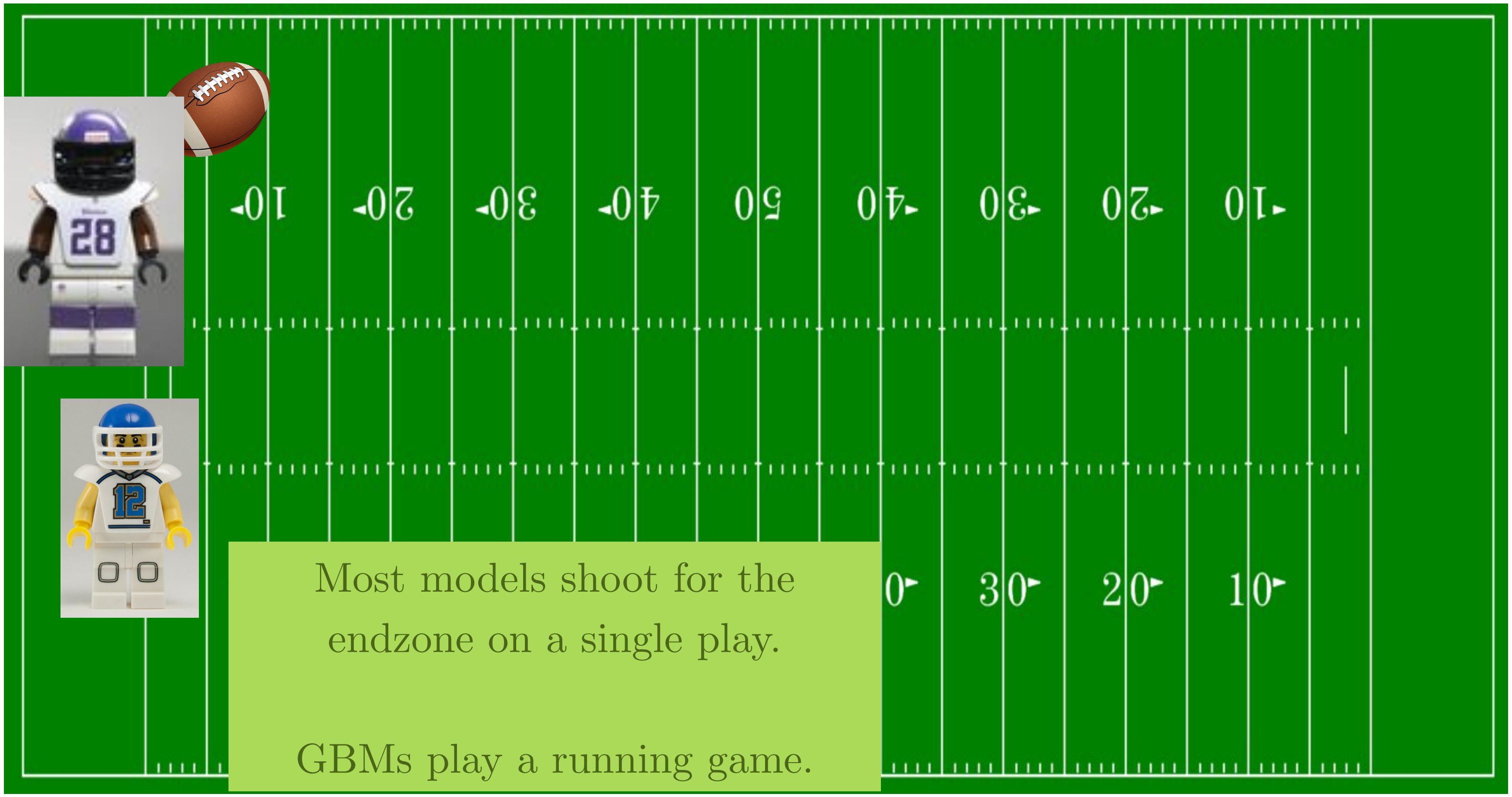
Stochastic Gradient Boosting, XGBoost, LiteGBM, CatBoost

Gradient Boosting Machines

• • •

(Jerome H. Friedman 1999-2001)

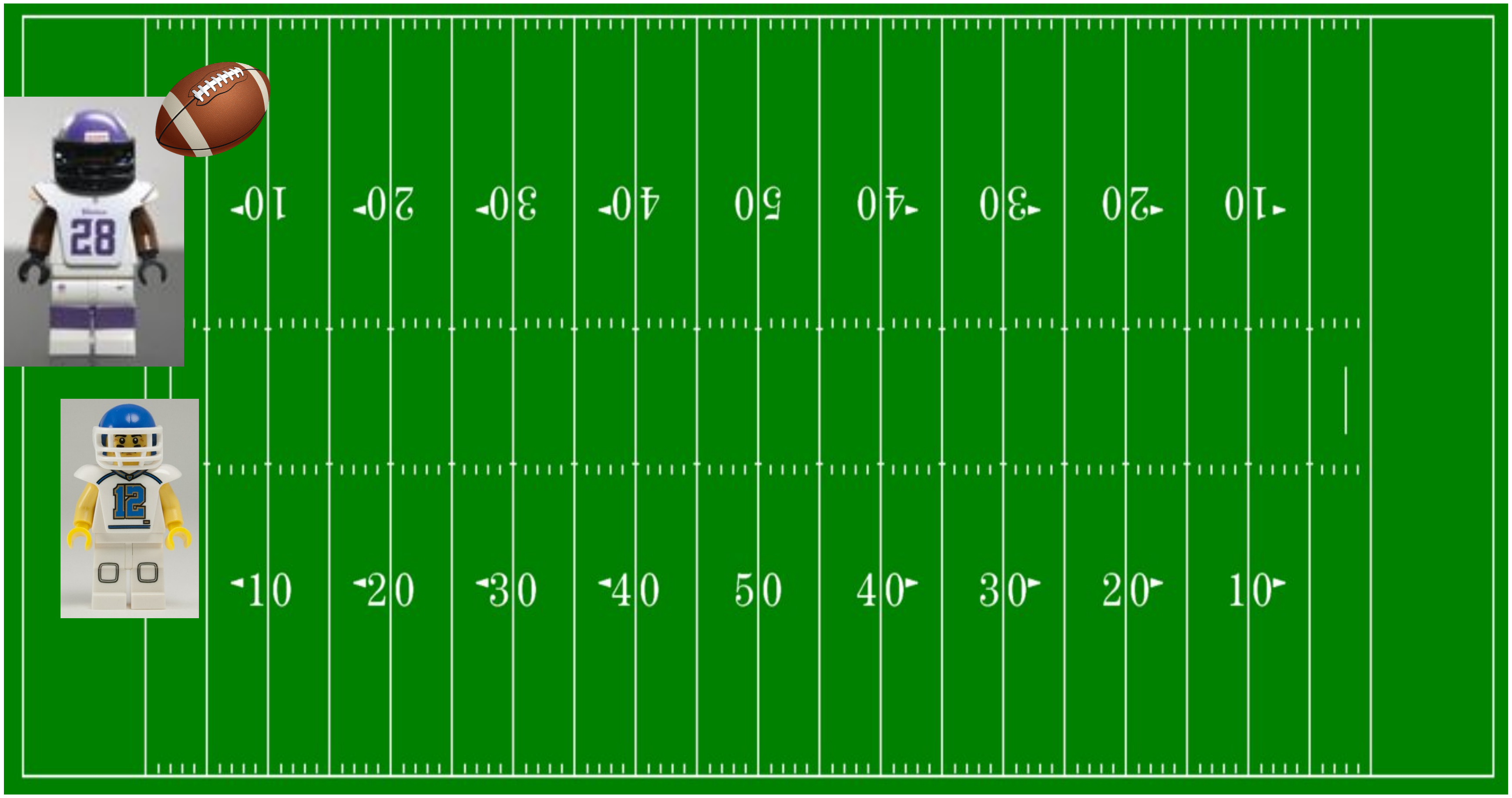
Football Strategy



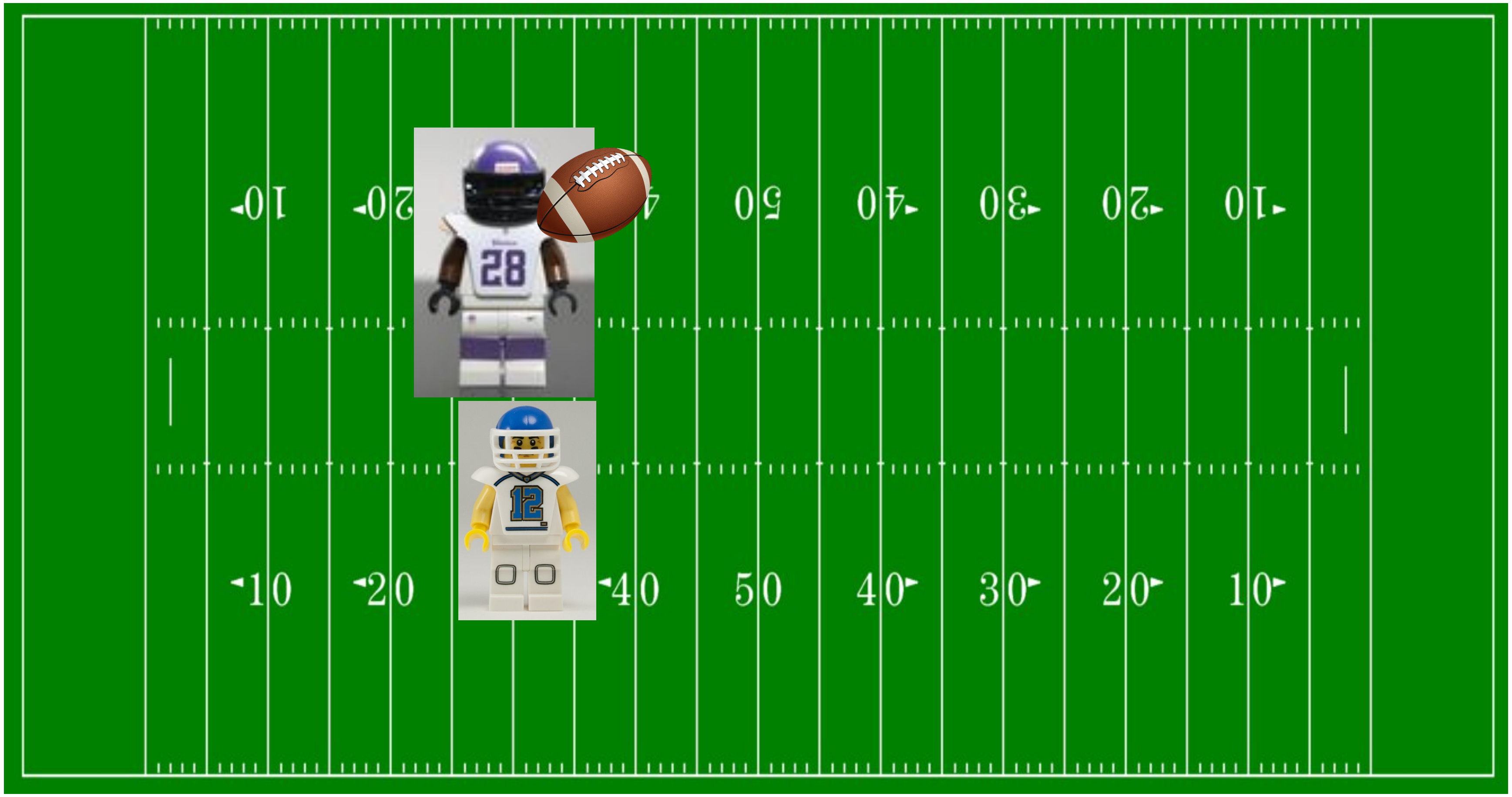
Most models shoot for the endzone on a single play.

GBMs play a running game.

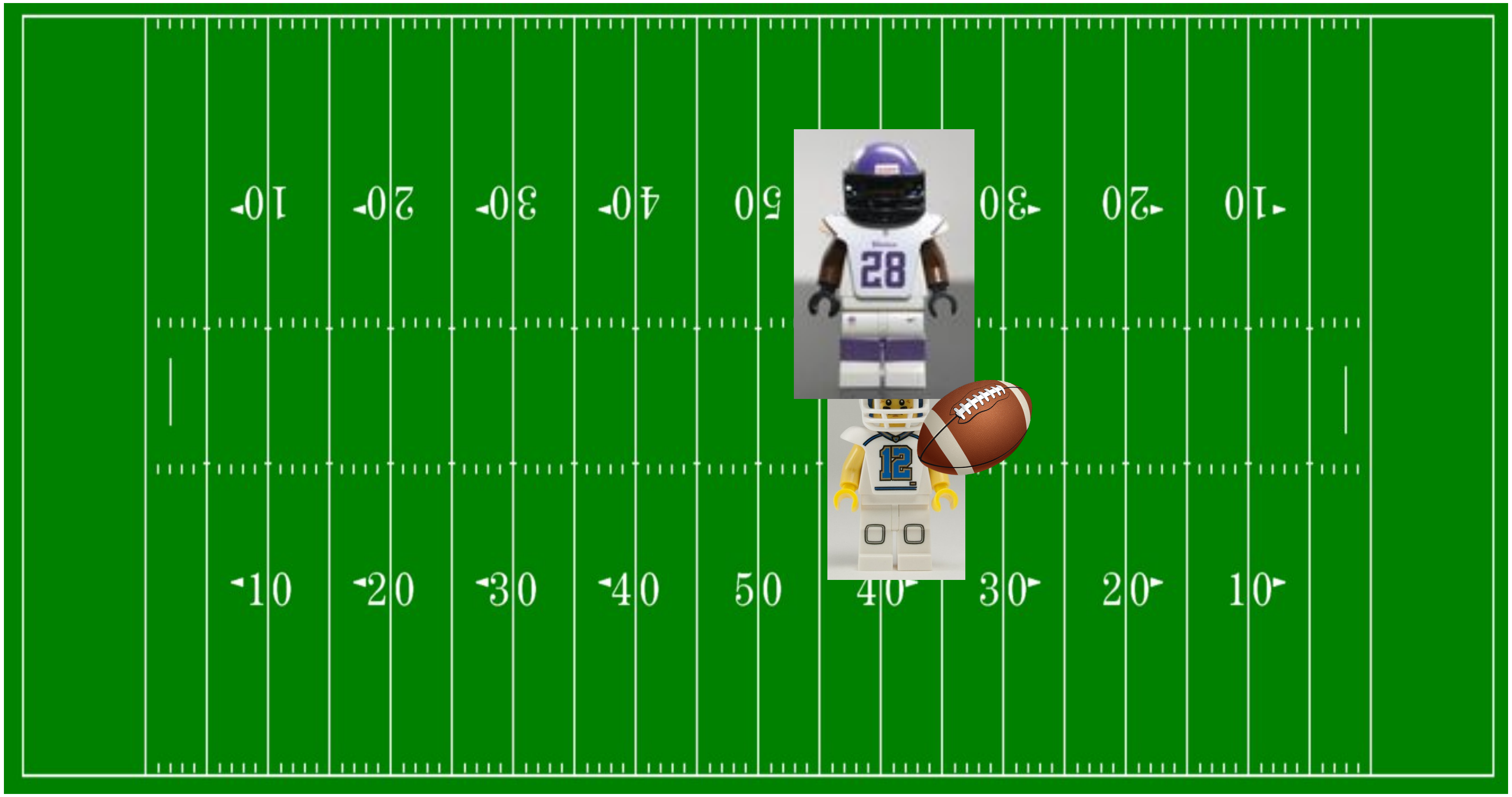
Football Strategy



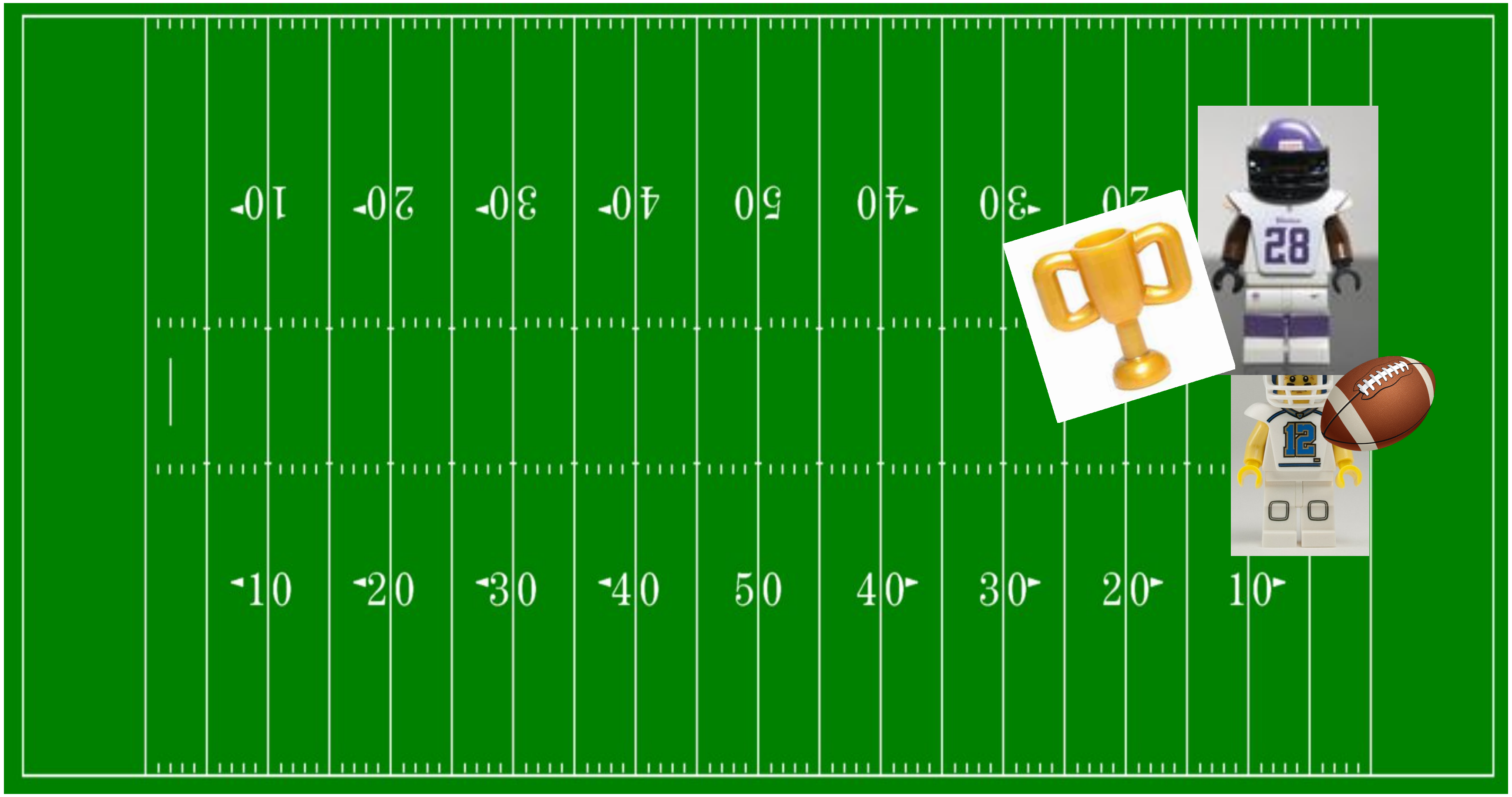
Football Strategy



Football Strategy



Football Strategy



Gradient Boosting Overview

Build a **simple** model, $f_1(x)$, trying to predict a target y

Don't even try to get very close (again, simple model). Error on $f_1(x)$ is expected to be large. (i.e. try a running play).

$$y = f_1(x) + \epsilon_1$$

error (remaining distance to endzone)

actual value (initial yards to endzone)

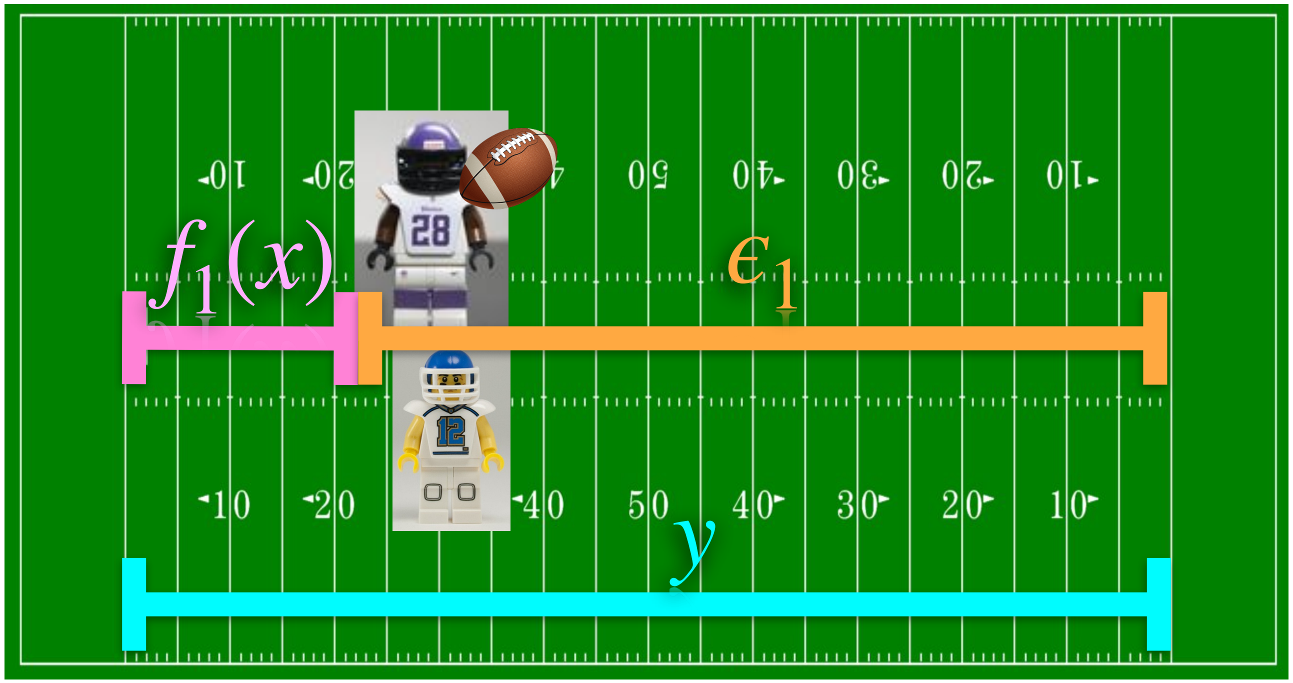
modeled value (distance of run)

Gradient Boosting Overview

Build a **simple** model, $f_1(x)$, trying to predict a target y

(i.e. try a running play)

$$y = f_1(x) + \epsilon_1$$



Gradient Boosting Overview

Now, let's try to ***predict that error*** with another simple model, $f_2(x)$. (Another running play)

$$\epsilon_1 = f_2(x) + \epsilon_2$$

Error from the first model (line of scrimmage to endzone)

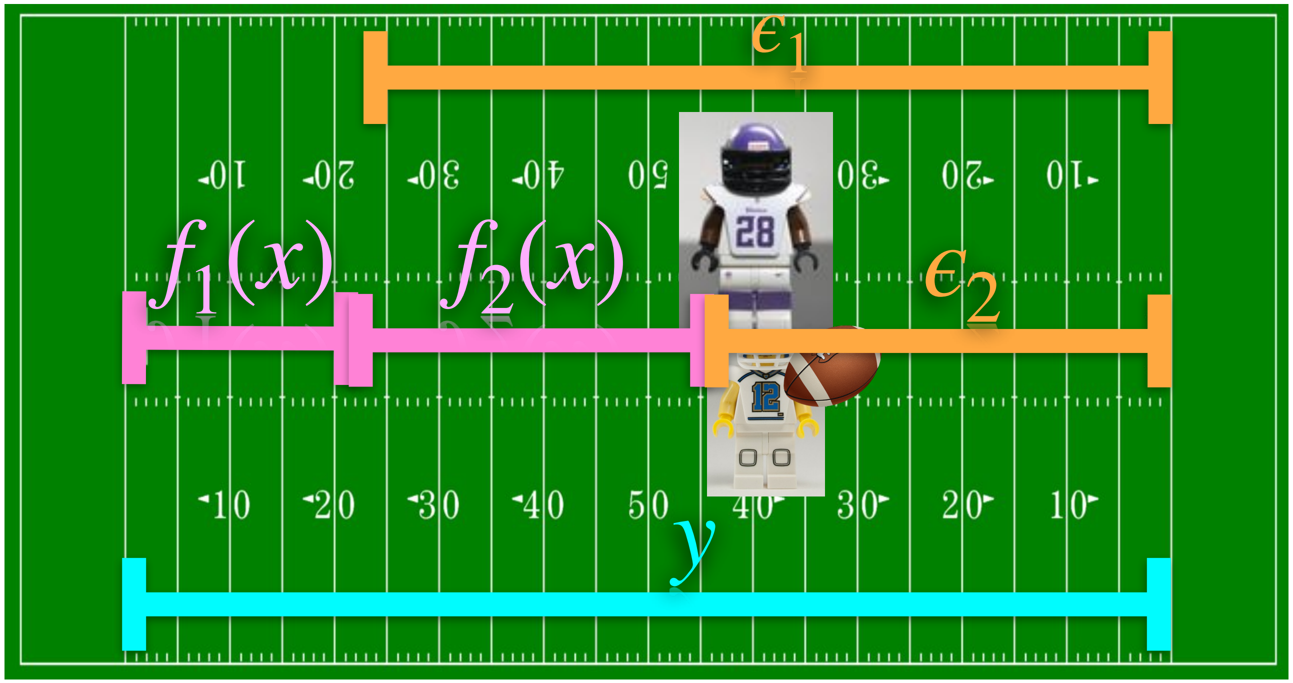
predicting the residual, ϵ_1 (yardage from second run)

error (remaining distance to endzone)

Gradient Boosting Overview

Now, let's try to predict that error with another simple model, $f_2(x)$. (Another running play)

$$\epsilon_1 = f_2(x) + \epsilon_2$$



Gradient Boosting Overview

Continue to add model after model, each one predicting the residuals from the previous round.

$$y = f_1(x) + f_2(x) + \ldots + f_k(x) + \epsilon$$

original modeled value

Predicts the residual, ϵ_1

predicting the residual, $\epsilon_{\mathrm{k-1}}$

presumably very small error

Gradient Boosting Summary

- At each round, we create a model to predict the residual from the previous round.
- If we're just going to continue to model error until it vanishes, what's the obvious problem we should be aware of?

Gradient Boosting and Overfitting

Gradient Boosting uses (at least) two forms of **regularization** to prevent overfitting:

1. A learning rate to effectively lessen the step-size taken at each step. Often called eta, $0 < \eta < 1$
   - $y = f_1(x) + \eta\, f_2(x) + \eta\, f_3(x) + \ldots + \eta\, f_k(x) + \epsilon_k$
   - Smaller values of eta => Less prone to overfitting
   - eta = 1 => no regularization
2. The number of trees/classifiers $f_i(x)$ used in the prediction
   - Larger number of trees => More prone to overfitting
   - Choose a number of trees by observing out-of-sample error

3. Other regularization parameters (λ, γ, L_2) have been introduced to most packages with the aim of reducing tendency to overfit.

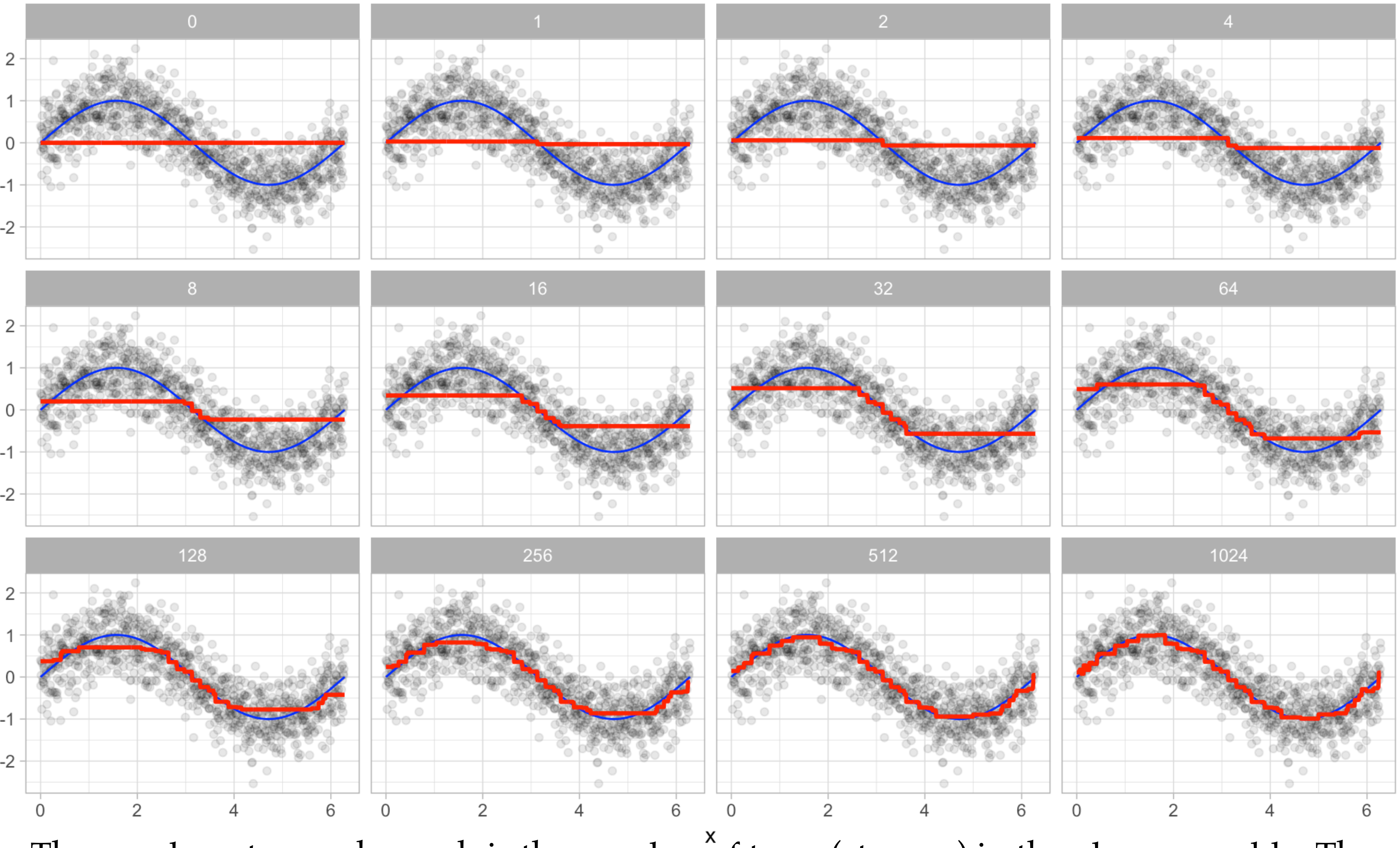
Gradient Boosted Trees

Gradient boosting yields a **additive ensemble model**

- There is no voting or averaging of individual models.
- The predictions from each model are summed together for final prediction.

The key to gradient boosting is **using "weak learners"**

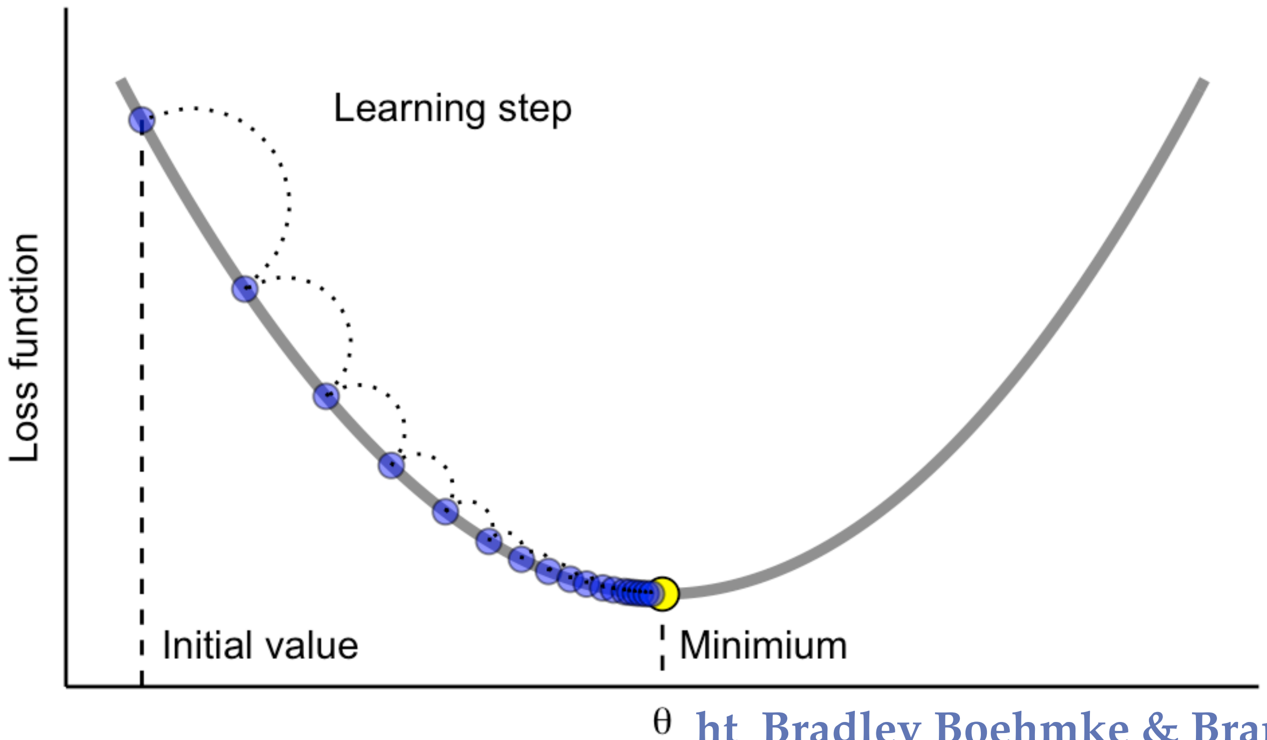
- Typically simple, shallow decision/regression trees
- Alone, make poor predictions but ensembled in this additive fashion provide superior results



The number atop each graph is the number of trees (stumps) in the gbm ensemble. The blue line is the true relationship y=sin(x)+ϵ. As the number of trees grows, the model approaches the true relationship. **ht Bradley Boehmke & Brandon Greenwell**

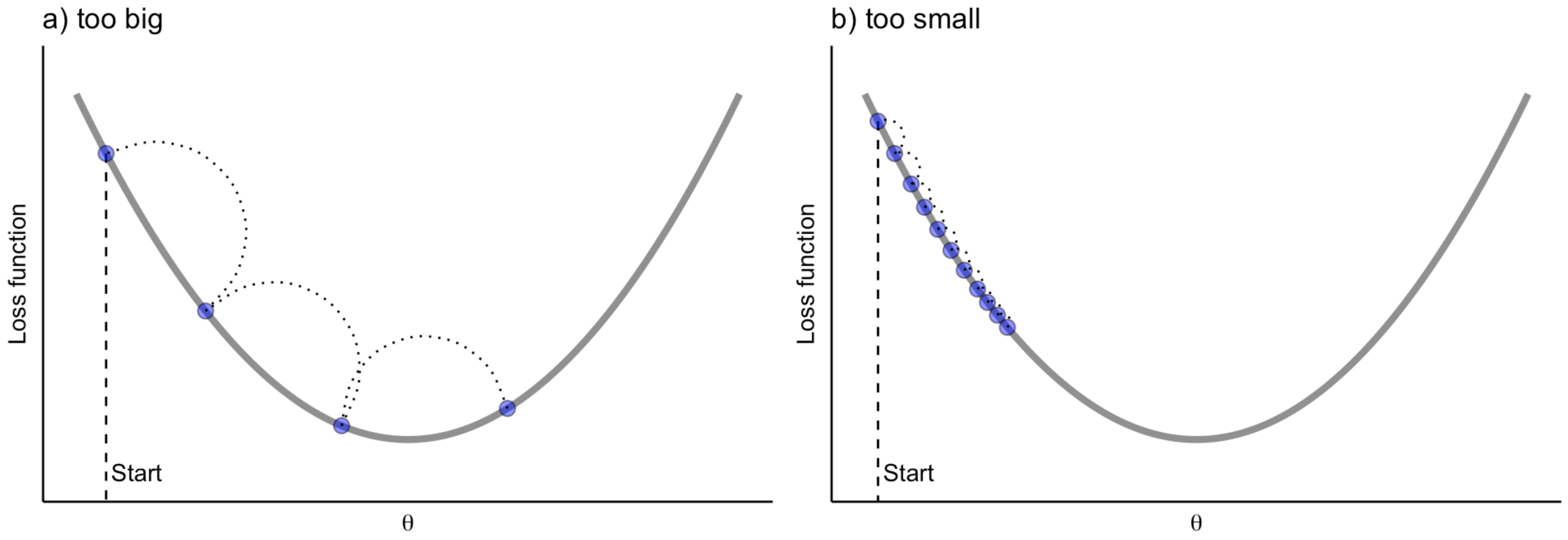
Gradient Descent

- Gradient Descent (Cauchy 1847) is a method that iteratively update parameters in order to minimize a loss (error) function by moving in the direction of steepest descent.
- Gradient Descent involves a learning rate (step-size)



ht Bradley Boehmke & Brandon Greenwell

η - The Learning Rate



ht Bradley Boehmke & Brandon Greenwell

Stochastic Gradient Descent

- Not all loss functions are convex (bowl-shaped)
- Local minima, plateaus on loss functions make gradient descent difficult.

- Stochastic gradient descent attempts to solve this problem by randomly sampling a fraction of the training observations for each tree in the ensemble.
- Makes the algorithm faster and more reliable, but may not always find the global minimum.

Gradient Boosting Summary

Advantages

- Exceptional model – one of most accurate available, generally superior to Random Forests when properly tuned and trained
- Can provide information on variable importance for the purposes of variable selection

Disadvantages

- Model **lacks interpretability** in the classical sense aside from variable importance
- The trees must be trained sequentially so **computationally this method is slow**er than Random Forest
- (At least one) extra tuning parameter over Random Forests, the regularization or shrinkage parameter, eta.
- Can be **hard to optimize tuning parameters** (time/complexity)
- Unlike random forests, GBM accuracy is **much more sensitive to hyperparameters**
  (small changes in settings => large changes in model accuracy)

Training a GBM

There is no secret recipe, grid search is typically infeasible so tuning parameters one at a time is common practice.

One suggested approach is as follows:

1. Start with a relatively high learning rate. Generally the default value of 0.1 works, range of 0.05–0.2 is often good
2. Determine the optimal number of trees for this learning rate
3. Fix tree-specific hyper parameters (depth/column sample/etc) and tune learning rate and assess speed vs. performance
4. Tune tree-specific parameters for decided learning rate
5. Once tree-specific parameters have been found, lower the learning rate to see if improvements result.

Recommended Implementations

GBM in SASViya

LiteGBM (Ke et al. 2017)

XGBoost

CatBoost

LiteGBM is generally faster than XGBoost with similar performance. CatBoost, LiteGBM and XGBoost differ in their treatment of categorical input variables, the way splits are searched, and whether they use standard or oblivious decision trees

Extreme Gradient Boosting (XGBoost)

"Extreme gradient boosting (XGBoost) is an optimized distributed gradient boosting library that is designed to be efficient, flexible, and portable across multiple languages (Chen and Guestrin 2016)."

Provides a few advantages over GBM:

1. Regularization: additional regularization parameters Gamma, L1, and L2 penalties
2. Early Stopping: settings to stop model assessment when additional trees offer no improvement
3. Parallel Processing: procedures to support GPU and Spark compatibility allowing for distributed processing. Doesn't fix problem that trees must be trained sequentially.
4. Loss Functions: flexibility to define custom objective functions and choose from a variety of existing loss functions
5. Different Base Learners: allows generalized linear models as well as tree-based ensembles.
6. Multiple Languages: XGBoost implementations for R, Python, Julia, Scala, Java, C++

Variable Importance in XGBoost

XGBoost provides 3 built-in measures of variable importance:

1. **Gain**: equivalent to metric in Random Forests, most common measurement of importance in overall model.
2. **Coverage**: measures the relative number of observations influenced by this feature
3. **Frequency**: percentage of splits in the whole ensemble that use this feature.

LightGBM

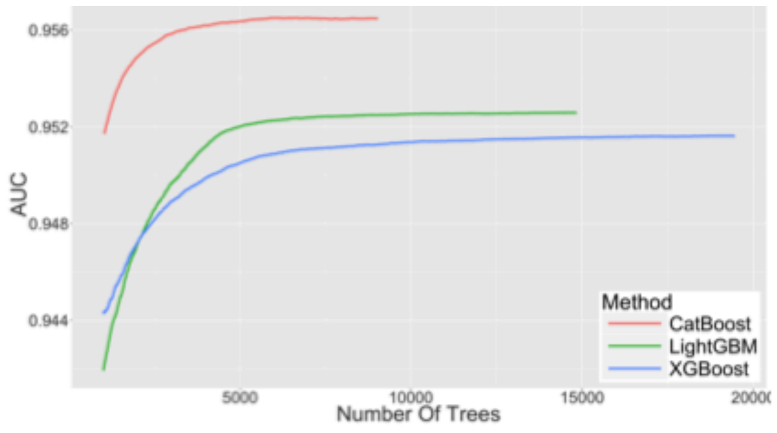
LightGBM employs novel **Gradient-based One-Side Sampling (GOSS)**

- Points with large gradients (read: large residuals) are more important for finding the optimal split point.
- GOSS uses all points with large gradients, and randomly samples points with small gradients.
- Can lead to a drastic reduction in the number of points used, hence the speed up.

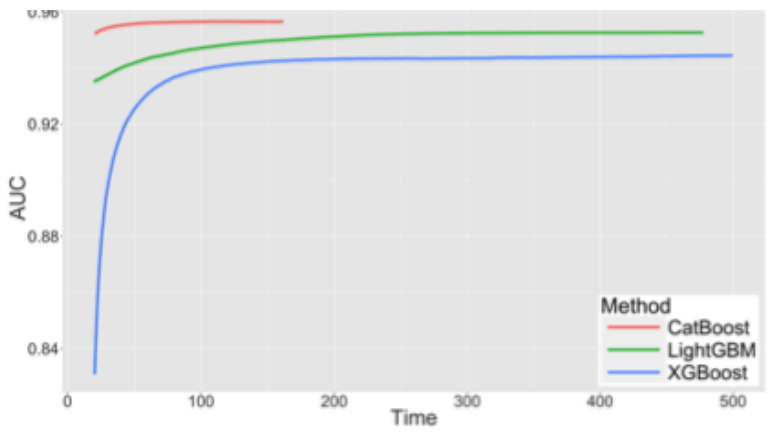
CatBoost

Selling point: Unique treatment of categorical inputs

- Clever mechanism for target-level encoding and feature combination
- “Oblivious trees” as base predictors: same splitting criterion used across an entire level of the tree.
- Fast training on GPU
- Original paper showed improvement in computation time AND accuracy over XGBoost and LiteGBM.



(a) AUC vs Number of trees



(b) AUC vs Time

For Self Study

• • •

An Introduction to “Old-fashioned” Boosting (Adaboost)

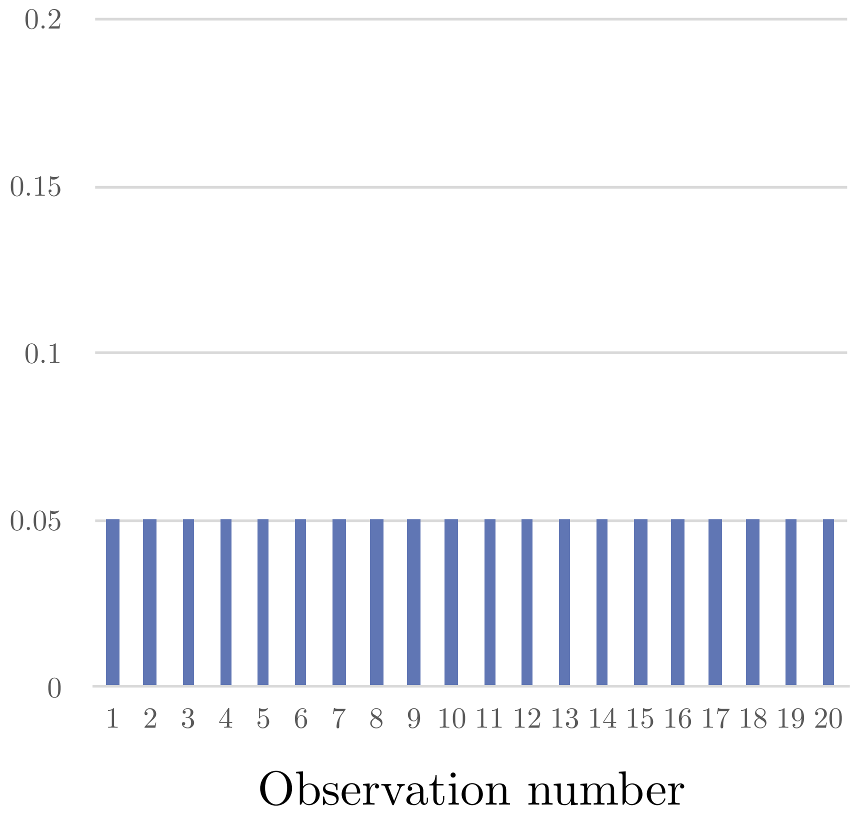
The original notion of *boosting* a model looked quite different from the modern approach outlined in first half of slidedeck.

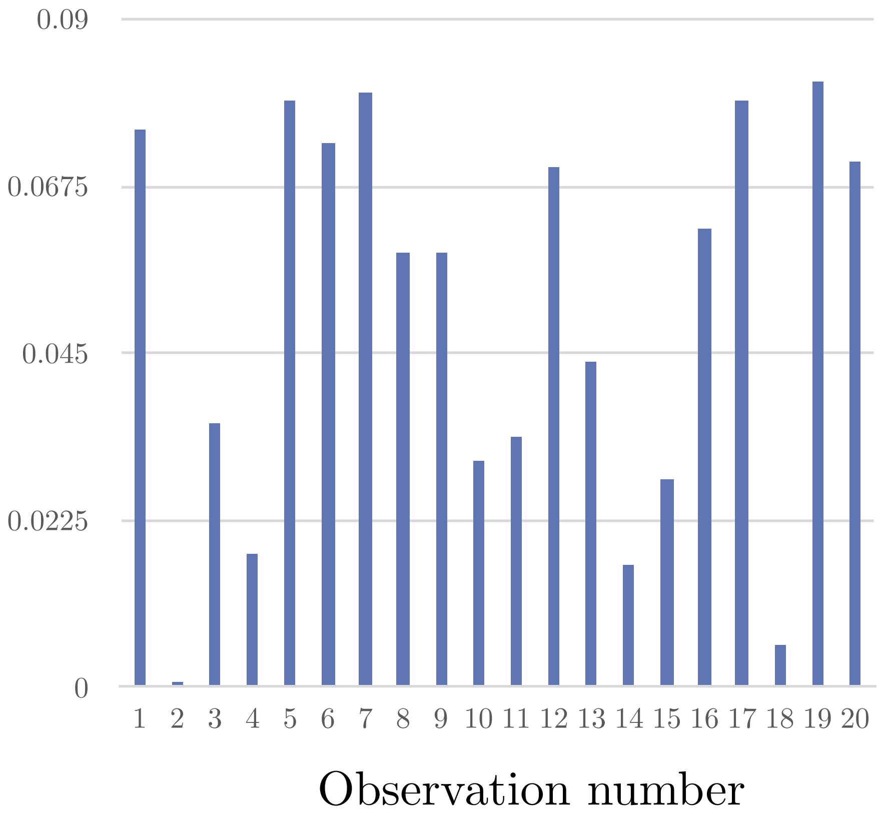
Boosting Overview

- Like bagging, going to draw a sample of the observations from our data with replacement
- Unlike bagging, the observations not sampled randomly
- Boosting assigns a weight to each training observation and uses that weight as a sampling distribution
  - Higher weight observations more likely to be chosen.
- May adaptively change that weight in each round
- **The weight is higher for examples that are harder to classify**

Bagging vs. Boosting

Probability of an observation being chosen for the sample at each round

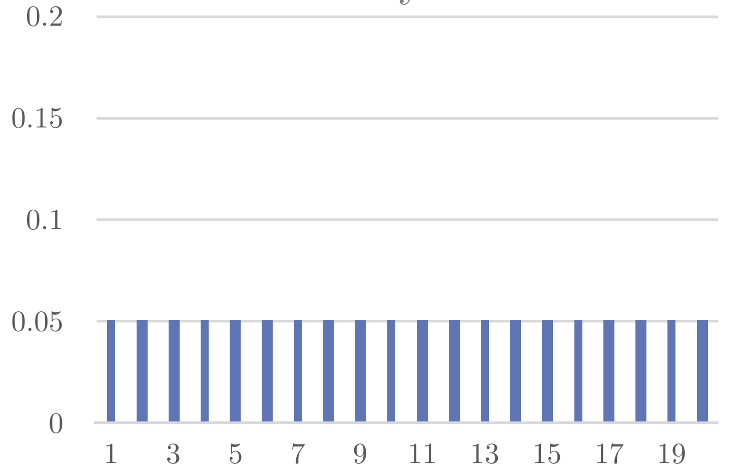




Bagging vs. Boosting

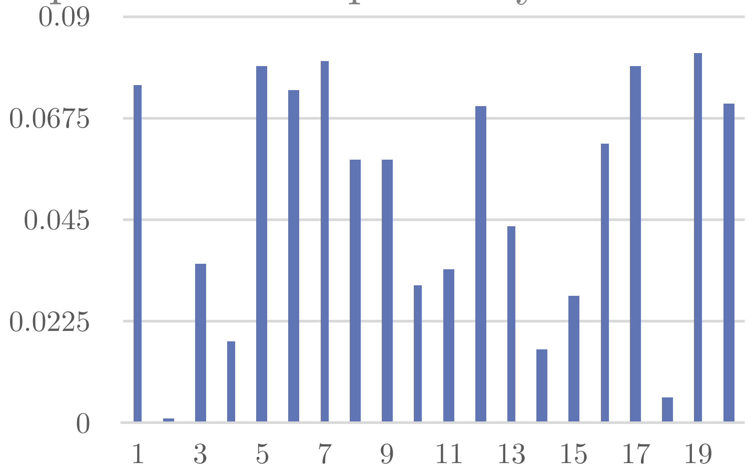
Only trying to create variability in the models by using training set variation.

Ensemble models built simultaneously, no time to evaluate accuracy.



Points with higher sampling probability were harder to predict accurately.

Want a chance to improve predictions sequentially



Boosting Example

input variable

- Same dataset used to illustrate bagging

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	1	1	1	-1	-1	-1	-1	1	1	1

target

- Start with equal weights for each observation
- Update weights each round based on the classification errors

Boosting Example

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Boosting: Weighted Ensemble

- Unlike Bagging, Boosted Ensembles usually weight the votes of each classifier by a function of their accuracy.
- If a classifier gets the higher weight observations wrong, it has a higher error rate.
- More accurate classifiers get higher weight in the prediction.

Boosting: Classifier weights

Errors made: First 3 observations

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Errors made: Middle 4 observations

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Errors made: Last 3 observations

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Boosting: Classifier weights

Errors made: First 3 observations

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Errors made: Middle 4 observations

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Errors made: Last 3 observations

x	0.2	0.2	0.4	0.4	0.4	0.4	[illegible]	[illegible]	[illegible]	[illegible]
y	1	1	-1	-1	-1	-1	[illegible]	[illegible]	[illegible]	[illegible]

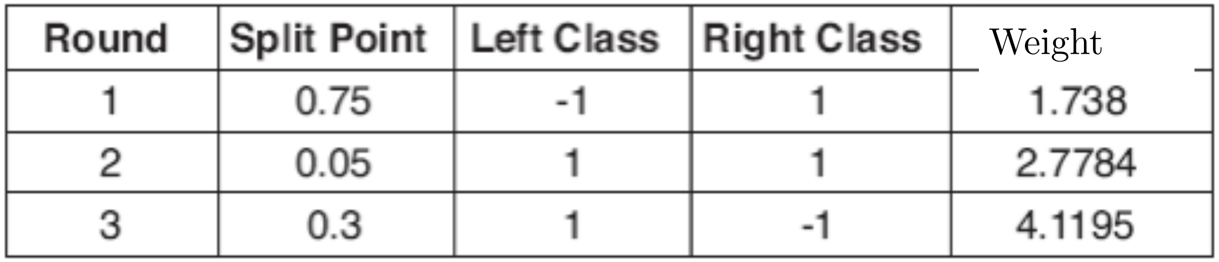
Lowest weighted error.
Highest weighted model.

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

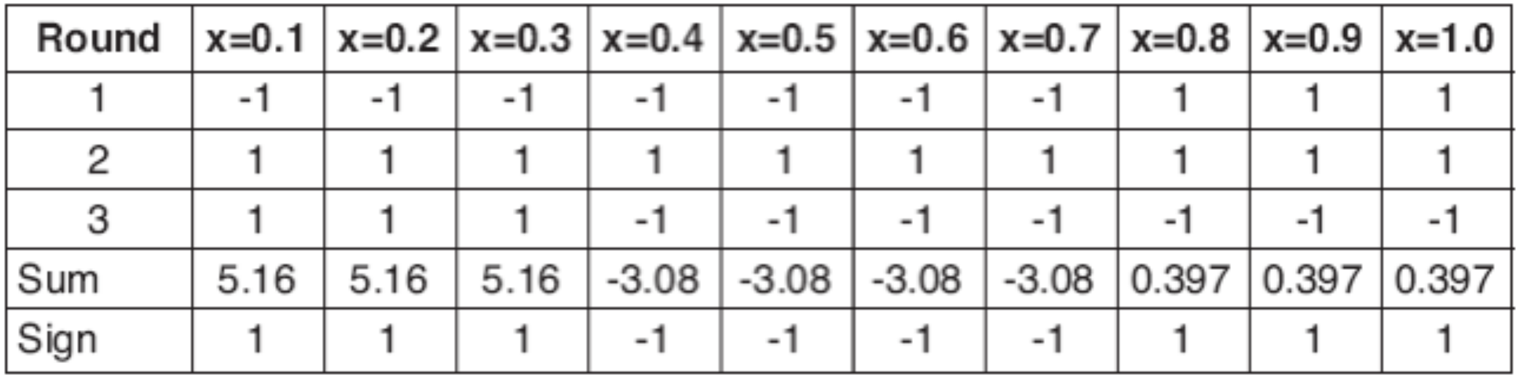
(b) Weights of training records

Boosting: Weighted Ensemble



Round	Split Point	Left Class	Right Class	Weight
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Classifier Decision Rules and Classifier Weights



Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Individual Classifier Predictions and Weighted Ensemble Predictions

Boosting: Weighted Ensemble

Round	Split Point	Left Class	Right Class	Weight
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Classifier Decision Rules and Classifier Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	[illegible]	[illegible]	[illegible]	[illegible]	1	1	1	1
3	1	1	[illegible]	[illegible]	[illegible]	[illegible]	-1	-1	-1	-1
Sum	5.16	5.16	[illegible]	[illegible]	[illegible]	[illegible]	.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

$$5.16 = -1.738 + 2.7784 + 4.1195$$

Individual Classifier Predictions and Weighted Ensemble Predictions

AdaBoost Details: The Classifier Weights

- Let w_j be the weight of observation j entering into present round.
- Let $m_j = 1$ if observation j is misclassified, 0 otherwise
- The error of the classifier this round is

$$\epsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j m_j$$

- The voting weight for the classifier this round is then

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$$

AdaBoost Details: Updating observation Weights

To update the observation weights from the current round (round i) to the next round (round $i+1$):

$w_j^{(i+1)} = w_j^i e^{-\alpha_j}$ if observation j was correctly classified

$w_j^{(i+1)} = w_j^i e^{\alpha_j}$ if observation j was misclassified

The new weights are then normalized to sum to 1 so they form a probability distribution.