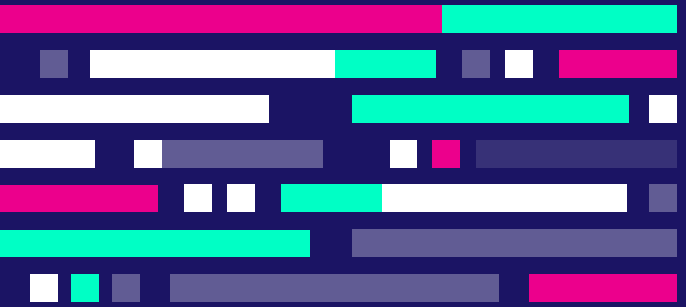
Survey of Boosting Methods

Caijun Qin

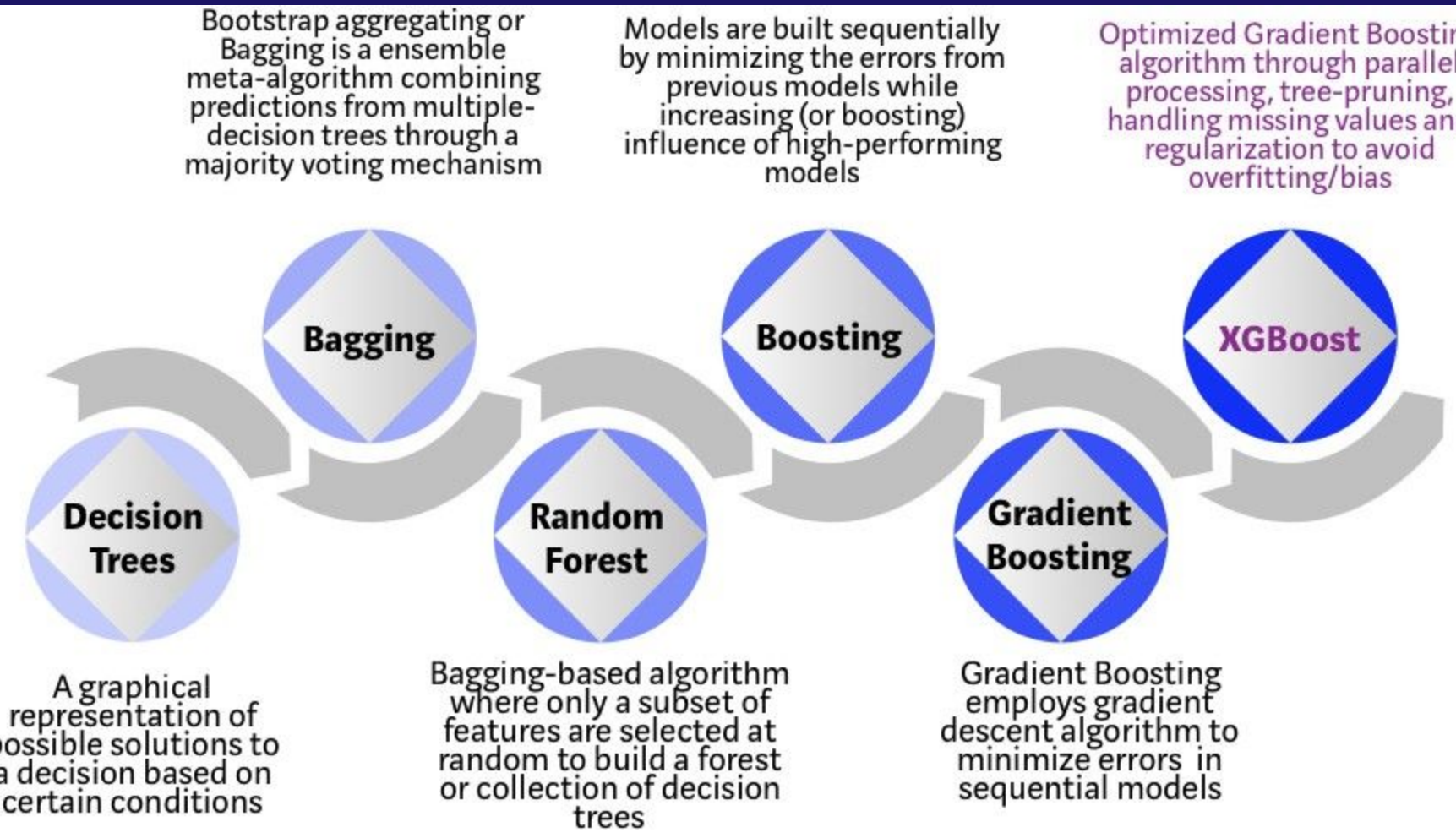


Definition

a family of algorithms that combines weak learners in synergy to a strong learner

Key Components

- Base estimator, or weak learner, is the type of regression or classifier algorithm used as repeatedly instantiated models that will collectively vote on a final estimate for a set of feature values
- Base estimator is usually a decision tree e.g. CART or a stump, but theoretically can be any type of machine learning model
- Note that most ensemble learning methods prefer trees and sometimes are only compatible with trees
- Stopping criteria, including maximum number of base estimators, tolerance for additional increase in accuracy, tree depth, etc.


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

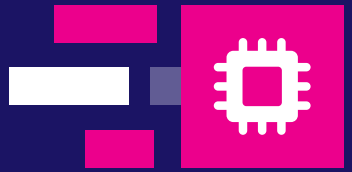
Overview of Methods

AdaBoost

Adapts base learners to observations with different weights

Gradient Boost

Use gradient descent to design additive model

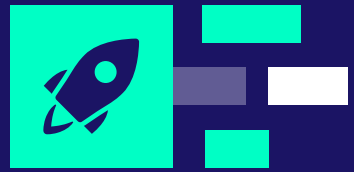


DeepBoost

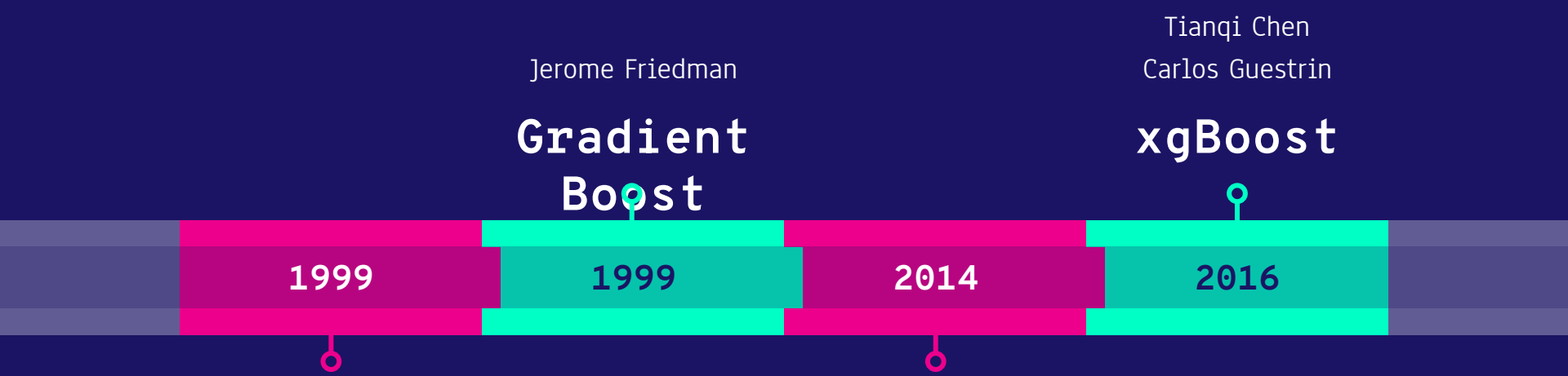
Neptune is the farthest planet from the Sun

xgBoost

“Extreme” version of gradient boost



Boost Algorithms Timeline



AdaBoost

Description

- Short for “Adaptive Boosting”
- Uses stumps as the base estimator
- A stump is just a tree with a depth of 1 i.e. split along a single feature
- Assigns weights to each observation relative to how hard it is to classify each observation
- Weights become updated each iteration by scaling by some factor
- Each iteration potentially sees a new input dataset from resampling with replacement of previous input data
- Observation weight determines probability of being chosen again
- Why? AdaBoost continues to fit on hard-to-classify observations longer
- Each weak learner has an associated importance weight α, which is the “amount of say” from that weak learner

Weight	Size	Class	Predictions
65	4.09	orange	orange
66	4.68	orange	orange
72	5.85	apple	apple
70	4.83	orange	orange
70	4.22	orange	orange
71	5.26	apple	apple
69	4.61	orange	orange
73	5.03	apple	apple

Algorithm 2 : AdaBoost.M1

1 **Input :** Dataset $D = \{(a_1, c_1), (a_2, c_2), \ldots, (a_N, c_N)\}$,
Base learner L and
Number of learning iteration T

2 Initialize equal weight to all training samples $w_i = 1/N$, $i = 1, 2, \ldots, N$

3 For $t = 1$ to T :

(a) Train a base learner h_t from D using D_t to training sample using w_i.

$$h_t = L(D, D_t)$$

(b) Compute error of h_t as

$$err_t = \frac{\sum_{i=1}^{N} w_i I(h_t(a_i) \neq c_i)}{\sum_{i=1}^{N} w_i}$$

(c) Compute the weight of h_t as

$$\alpha_t = \log\left(\frac{1 - err_t}{err_t}\right)$$

(d) Set $w_i \leftarrow w_i . \exp[\alpha_t I(h_t(a_i) \neq c_i)]$

4 **Output :** $H(a) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(a)$

Algorithm Simplified

1. Gather dataset and set stopping criteria e.g. number of iterations (T)
2. Initially set weights to 1 / N
3. Per iteration for iterations t=1, 2, ..., T:
   a. Train new base learner h_t
   b. Evaluate weighted error rate of h_t on dataset
   c. Use previously computed error rate to compute the “amount of say”, or α for h_t
   d. Update weight of each observation based on correct/incorrect estimate and α for h_t
4. Predict output for new set of feature values by majority vote of T base learners collectively together

What Happens with Weights?

Initial

All weights are initially the same, which is 1 / N.

Scale

Per iteration, each weight is scaled by an exponential factor. Correct prediction scales by fraction < 1, but incorrect prediction scales by number > 1. This is assuming that the base estimator has error rate < 0.5. Otherwise, scaling direction flips for bad base estimator.

Normalize

Divide scaled weights by their sum to normalize the weights. This means the final updated weights sum to 1.

Breakdown of α (alpha)

- α judges how accurate a base estimator handles predictions. Large magnitude indicates either a really good job or really bad job at distinguishing between different classes.
- (+) value indicates good estimator (low error rate), while (-) value indicates bad estimator (high error rate)
- α = 0 when error rate is exactly 0.5

(b) Compute error of h_t as

$$err_t = \frac{\sum_{i=1}^{N} w_i I(h_t(a_i) \neq c_i)}{\sum_{i=1}^{N} w_i}$$

(c) Compute the weight of h_t as

$$\alpha_t = \log\left(\frac{1 - err_t}{err_t}\right)$$

Variants of AdaBoost

LogitBoost - Essentially the same process as AdaBoost except loss function.

$$\sum_i \log\left(1 + e^{-y_i f(x_i)}\right)$$

Real AdaBoost or SAMME.R
GentleBoost
Penalized AdaBoost
More...

Example R Code

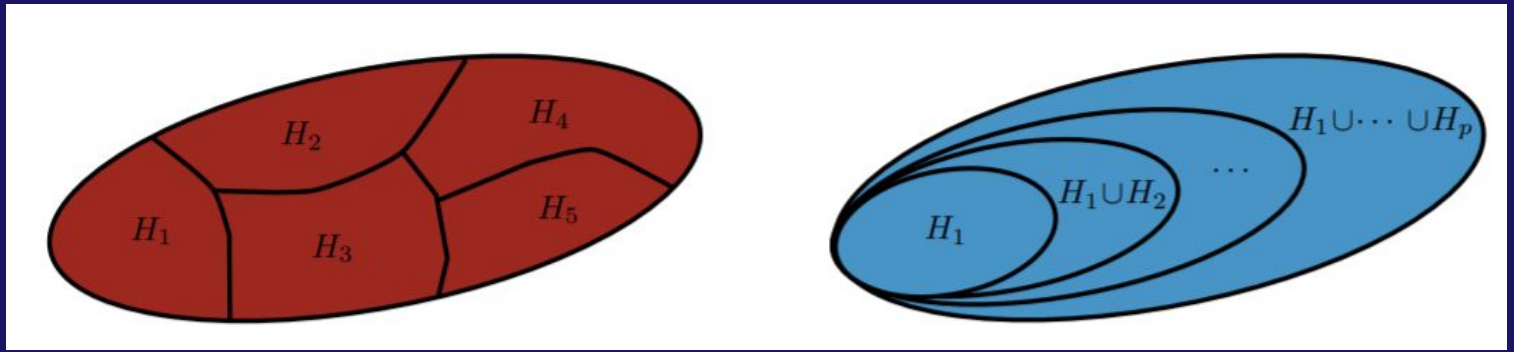
```
adaboost.model ←
fastAdaboost::adaboost(formula = y ~
., data = MASS::bacteria, nIter =
1000)
summary(object = adaboost.model)
adaboost.model$weights
```

Vector of weights (α) for 1000 trees)

DeepBoost

Description

- Developed by researchers at Google Research and Courant Institute
- Uses coordinate descent applied to convex objective
- Orders individual weak learners into families in order of complexity
- Complexity could be tree depth, for example



Formulas

$$\mathcal{F} = \mathrm{conv}\left(\bigcup_{k=1}^{p} H_k\right)$$

$$\mathrm{conv}(H) = \left\{ \sum_{t=1}^{T} \alpha_t h_t : \alpha_t \geq 0; \sum_{t=1}^{T} \alpha_t \leq 1; \forall t, h_t \in H \right\}$$

Maximum direction: definition based on the error

$$\epsilon_{t,j} = \frac{1}{2}\left[1 - \mathop{\mathrm{E}}_{i \sim \mathcal{D}_t}[y_i h_j(x_i)]\right],$$

where D_t is the distribution over sample at iteration t.

Example R Code

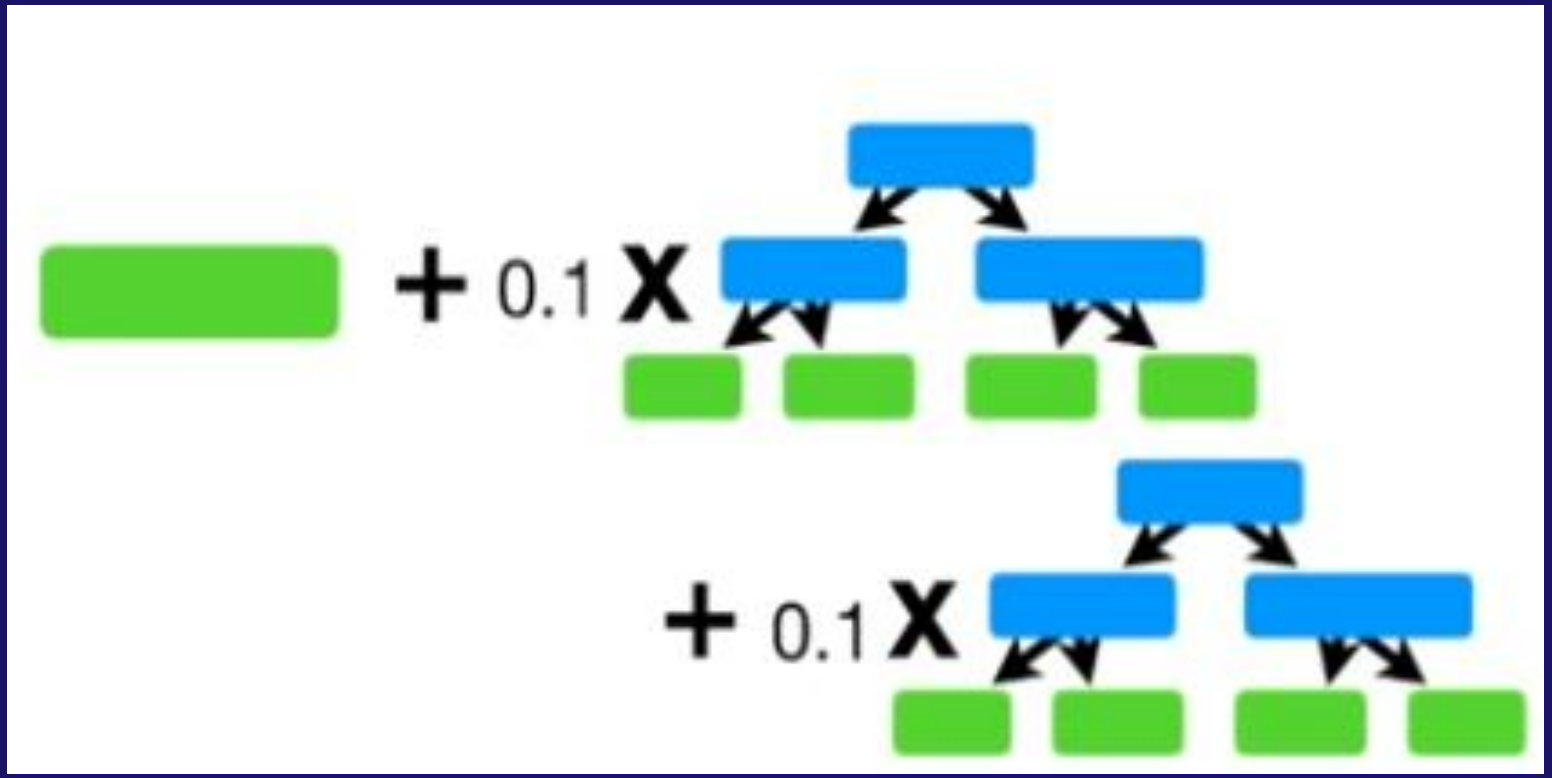
```
deepboost.model <-
deepboost::deepboost(formula = y ~
., data = MASS::bacteria, num_iter =
1000)
summary(object = deepboost.model)
deepboost.model
```

```
[1] "Model error: 0.136363640427589"
[1] "Average tree size: 14.7142858505249"
[1] "Number of trees: 7"
```

Gradient Boost

Description

- Additive model created by sequentially appending new weak learners with output values determined by gradient descent
- Nudges collective estimate of set of features closer and closer to true value by adding together pseudo-residuals
- Each new weak learner is based on pseudo-residuals from previous ensemble of existing weak learners
- Adjustable learning rate η that is multiplied to the predicted output by each weak learner after the first one


+ 0.1 X
+ 0.1 X

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 $\hat{f}(x) = f_M(x)$.

Algorithm Simplified

1. The first weak learner to start with is a constant value, specifically the mean of all true values. Just using f_0 will predict the same value regardless of input features.
2. For the m^{th} weak learner in m = 1, 2, ..., M:

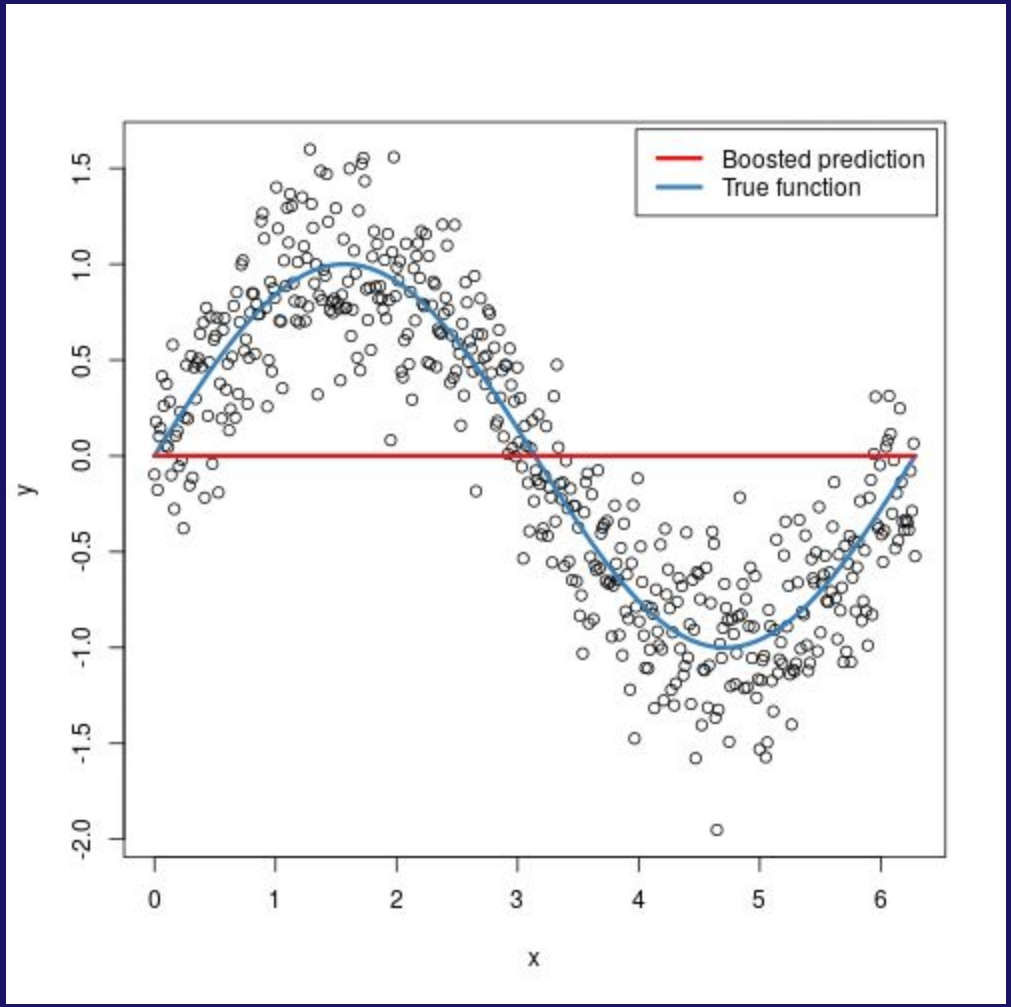
a. Compute partial derivative of chosen loss function for observation *i*; MSE would simply yield *y - y-hat*, where predicted *y-hat* is the outcome of using only the first *m - 1* weak learners for boosting

b. Let's call the previously computed *N* values pseudo-residuals. Fit a new tree (m^{th} one, in fact) based on the pseudo-residuals.

c. Convert each terminal node to a single value **γ**, which maximizes the overall "nudge" that minimizes residual gap for the whole node.

d. Append weak learner to the sequence of weak learners.

3. Predict on new data using complete set of *m* weak learners.


Boosted prediction
True function
y
x
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
0
1
2
3
4
5
6

Regression vs. Classification

Regression:

- ❏ Predicted outputs are usually within the same scope and numerical type (integer, decimal) as true outputs
- ❏ Loss function appropriate for regression, e.g. *MSE*
- ❏ Pseudo-residuals for *MSE* are simply what you expect for linear regression

$$e = y - \hat{y}$$

Classification:

- ❏ Predicted outputs are log odds, convertible to probabilities
- ❏ Loss function typically *log loss*
- ❏ Pseudo-residuals computed below

$$\frac{\sum Residual}{\sum[PreviousProb * (1 - PreviousProb)]}$$

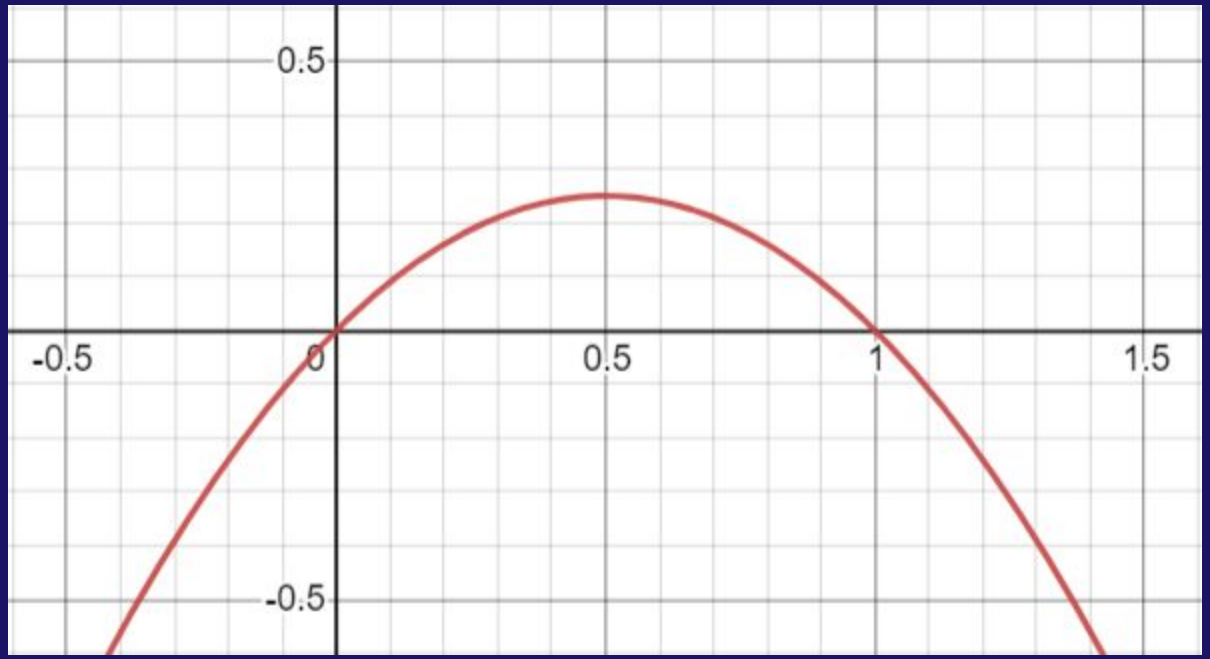
Loss Functions

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

$$LogLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} * \ log(p_{ij})$$

$$\mathrm{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})\right]$$

y=x(1−x)



Example R Code

```
gbm::gbm(formula = y ~ .,
distribution = "gaussian", data =
MASS::bacteria,
    n.trees = 1000, n.cores =
parallel::detectCores() - 1)
```

A gradient boosted model with
gaussian loss function.
1000 iterations were performed.
There were 5 predictors of which 2
had non-zero influence.

xgBoost

Description

- Short for “Extreme Gradient Boosting”
- Uses decision trees as the base estimator
- Base estimators typically have 8 to 32 terminal nodes (leaves), or a height of 3 to 5 when viewed as a binary tree
- Advanced version of gradient boosting that pushes to the limit in many aspects (penalization, tuning, hardware resources, cross-platform, multiple programming languages)
- Can handle sparse and missing data
- Many new optimizations over vanilla gradient boosting, including *Approximate Greedy Algorithm*, *Parallel Learning*, *Weighted Quantile Sketch*, *Sparsity-Aware Split Finding*

Important Formulas

Left: Regression, Right: Classification

$$Similarity\ Score = \frac{(\Sigma Residual)^2}{\#\ of\ Residual + \lambda}$$

$$Output\ Value = \frac{\Sigma Residual}{\#\ of\ Residual + \lambda}$$

$$Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$$

All formulas are derived by solving optimization problem for XgBoost

$$Gain = Left_{Similarity} + Right_{Similarity} - Root_{Similarity}$$

$$\text{Similarity Score} = \frac{(\sum Residual_i)^2}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$

$$\text{Output Value} = \frac{(\sum Residual_i)}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)] + \lambda}$$

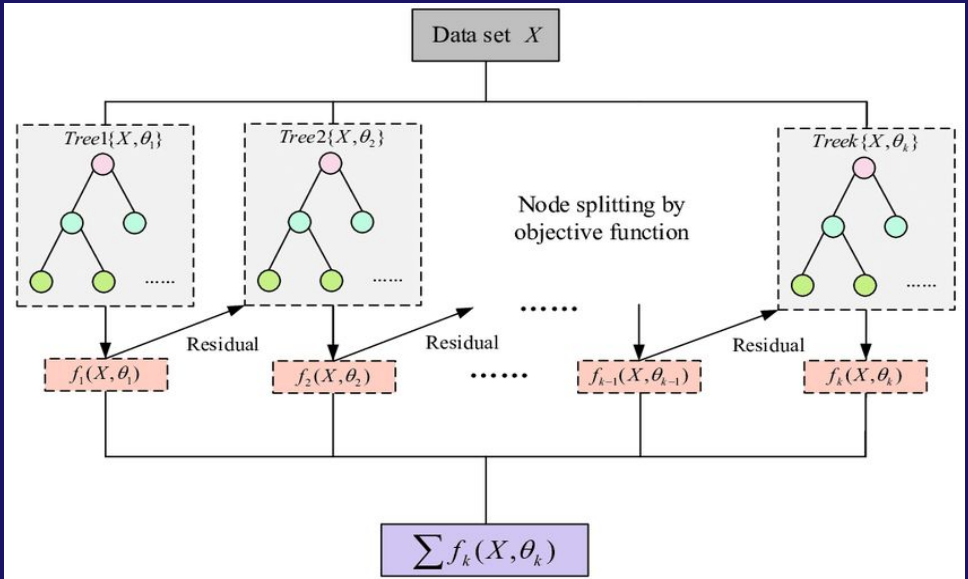
$Residual_i \rightarrow \hat{Y}$

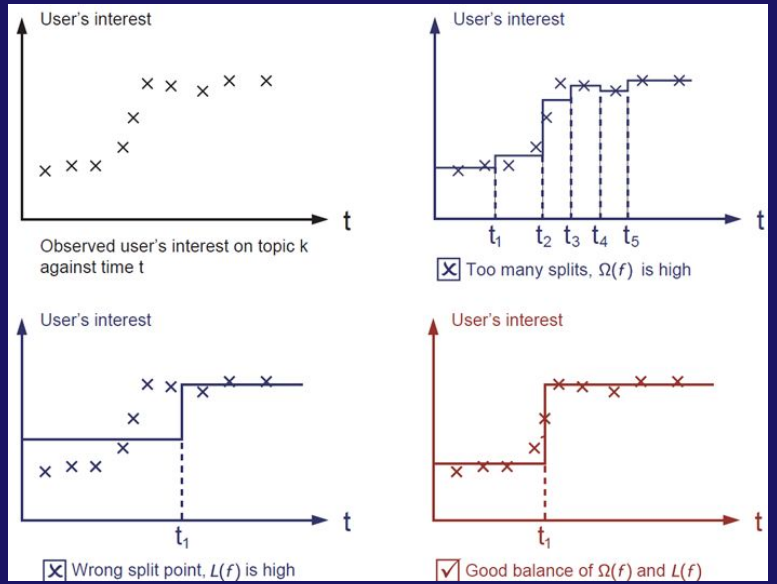
$Reg\_lamda \rightarrow \lambda = 0$

$\text{Previous Probability}_i \rightarrow \text{Previous } F_i(x)$

Similarity and Information Gain

- The similarity score for a node describes how much observations inside the node cluster on one side of the zero-mark
- Data with a highly positive or negative mean could be re-centered
- For a given feature P, let the values p_1, p_2, ..., p_k denote the feature values for the *k* observations falling into that node
- The node can be further split along a threshold occurs at maximization of information (similarity) gain by its L and R children


Data set X
$Tree1\{X,\theta_1\}$
$Tree2\{X,\theta_2\}$
$Treek\{X,\theta_k\}$
Node splitting by objective function
Residual
Residual
Residual
$f_1(X,\theta_1)$
$f_2(X,\theta_2)$
$f_{k-1}(X,\theta_{k-1})$
$f_k(X,\theta_k)$
$\sum f_k(X,\theta_k)$


User's interest
t
Observed user's interest on topic k against time t
User's interest
t_1 t_2 t_3 t_4 t_5
t
Too many splits, $\Omega(f)$ is high
User's interest
t_1
t
Wrong split point, $L(f)$ is high
User's interest
t_1
t
Good balance of $\Omega(f)$ and $L(f)$

Algorithm 1: XGboost algorithm

Data: Dataset and hyperparameters

Initialize $f_0(x)$;

for $k = 1, 2, \ldots, M$ **do**

Calculate $g_k = \frac{\partial L(y,f)}{\partial f}$;

Calculate $h_k = \frac{\partial^2 L(y,f)}{\partial f^2}$;

Determine the structure by choosing splits with maximized gain

$\mathbf{A} = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$;

Determine the leaf weights $w^* = -\frac{G}{H}$;

Determine the base learner $\hat{b}(x) = \sum_{j=1}^{T} wI$;

Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$;

end

Result: $f(x) = \sum_{k=0}^{M} f_k(x)$

Overfitting Prevention

- λ is a penalty constant in the denominator of formulas for *similarity score* and *output values* (estimate from weak learner if followed down to terminal node). Adding a constant value decreases similarity score to lessen the effect of a very few observations contributing to the score and overfitting the weak learner to themselves.
- Pruning can be quantified by a γ constant. Whenever *gain* for a terminal split is < γ, that split is undone, meaning the two child terminal nodes are fused back into the parent node.
- On the other hand, *cover* sets the minimum number of splits each weak learner should have. The formula is the denominator of similarity score without λ.

XGBoost Tree Methods

What is this?

Faster although approximate ways of splitting in weak learners.

Example R Code

```
xgboost.model ←
xgboost::xgboost(data =
bacteria.matrix[-0], label =
bacteria.matrix[0], nrounds = 1000)
xgboost.model
```

References

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d
https://medium.com/analytics-vidhya/gradient-boost-decomposition-pytorch-optimization-sklearn-decision-tree-regressor-41a3d0cb9bb7
https://blog.paperspace.com/gradient-boosting-for-classification/
https://laptrinhx.com/how-does-xgboost-work-863579087/
https://cs.nyu.edu/~mohri/talks/Princeton2015.pdf
https://analyticsindiamag.com/introduction-to-boosting-implementing-adaboost-in-python/

THANKS !

Any Questions?