XGBoost

Boosting train multiple models sequentially, then aggregate these models, to improve the accuracy of the overall system

* Boosting, like Bagging, Reduces Variance

* Boosting often works (slightly) better than Random Forests in practice because it also Reduces Bias (by upweighting the instances it gets wrong)

* XGBoost fits very quickly!

Tree Boosting in a Nutshell

1. Regularized learning objective

Dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

n examples $|\mathcal{D}| = n$, m features $x_i \in \mathbb{R}^m$,

outcome $y_i \in \mathbb{R}$

A tree ensemble model uses K additive functions (decision trees) to predict the output,

$$\boxed{\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)}$$

$f_k \in \mathcal{F} = \text{CART}$

$\mathcal{F} = \{f(x) = w_{q(x)}\} \; (q : \mathbb{R}^m \to T, \; w \in \mathbb{R}^T)$

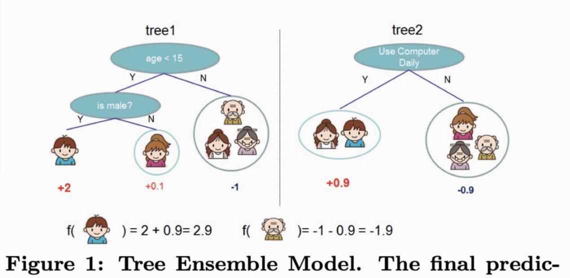


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

q = structure of each tree that maps an example x to a corresponding leaf index

w = weights of the tree

w_i = score on ith leaf

To learn the set of functions (trees) used in the model, we minimize a Regularized objective,

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2.$

Loss function l measures diff. b/t true y_i and pred. $\hat{y}_i$

for cts y_i: $l(y, \hat{y}_i) = (y_i - \hat{y}_i)^2$

for binary y_i: $l(y, \hat{y}_i) = y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$

Regularization term Ω penalizes each tree to mitigate overfitting

$T \longrightarrow$ fewer leaves

$\|w\|^2 \longrightarrow$ smaller weights

2. Gradient Tree Boosting

Train the model in an additive manner.

Let $\hat{y}_i^{(t)}$ be the prediction of the i^{th} training data instance at the t^{th} iteration.

To fit the decision tree f_t at the t^{th} iteration, modify our objective function,

$$\boxed{\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)}$$

Greedily add the f_t which most improves our model by minimizing $\mathcal{L}^{(t)}$ (Boosting!)

2^{nd} order Taylor approximation:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$, $h_i = \partial^2_{\hat{y}^{(t-1)}} \left(y_i, \hat{y}_i^{(t-1)}\right)$

are 1^{st} and 2^{nd} order gradient statistics on the loss function ℓ.

Remove constant terms (w.r.t. f_t) and simplify:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

Define $I_j = \{ i \mid q(x_i) = j \}$ = the instance set of leaf j in tree structure q

Expand Ω:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

For a fixed tree structure $q(x)$, i.e., given I_j compute the optimal weight w_j^* of leaf j by

$$\frac{\partial \tilde{\mathcal{L}}^{(t)}}{\partial w_j} = 0$$

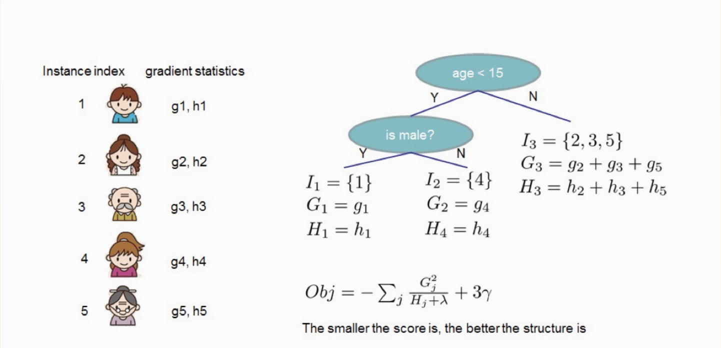
$$\Rightarrow w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

and then, with these weights, the optimal value of objective is

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\left(\sum_{i \in I_j} h_i + \lambda\right)} + \gamma T.$$

Scoring Function to evaluate the quality of a tree structure q

(like gini impurity for decision trees except works for a wider range of loss functions ℓ.)



Normally it is impossible to enumerate all possible tree structures q, evaluate each with $\mathcal{L}^{(t)}(q)$, and choose the best.

Instead we a Greedy algorithm: begin with a single leaf and iteratively add branches to the tree.

Assume I_L, I_R are the instance sets of the left and right node after a split. Let $I = I_L \cup I_R$.

Loss Reduction after split:

$$\mathcal{L}_{split} = \tilde{\mathcal{L}}^{(t)}(q_{\text{after split}}) - \tilde{\mathcal{L}}^{(t)}(q_{\text{before split}})$$

$$= \left[-\frac{1}{2} \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} + \gamma \right]$$

$$- \left[-\frac{1}{2} \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{1}{2} \frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \gamma \cdot 2 \right]$$

$$= \frac{1}{2} \left[\frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} + \frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma .$$

This formula $\mathcal{L}_{split}$ is used in practice to evaluate the split candidates!

* With small or moderate data ($n \le 5$ million Rows), which is true for NFL WP play-by-play data, use Exact Greedy split finding algorithm:

enumerate over all possible splits on all the features, and choose the best split according to $\mathcal{L}_{split}$

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split with max score

* additional ways to Reduce overfitting:
  - Shrinkage: Scales newly added weights w_j by a factor η after each step of tree boosting (learning rate) which Reduces the inflnence of each individual tree and leaves space for future trees to improve the model
  - column (feature) subsampling: also used in Random Forest

Settings

booster — e.g. gbtree to use Trees as weak learners f_k

objective l — e.g. RMSE, logloss

num Rounds — How many trees to fit sequentially

Monotone constraints — force trees to be monotonic

XGBoost Hyperparameters (to be tuned)

learning_rate alias:eta η	0.3	[0, inf)	Decreasing prevents overfitting.	Shrinks the tree weights in each round of boosting.
max_depth	6	[0, inf)	Decreasing prevents overfitting.	The depth of the tree. 0 is an option in a loss-guided growing policy.
gamma alias: min_split_loss γ	0	[0, inf)	Increasing prevents overfitting.	Low values, usually lower than 10, are standard.
min_child_weight	1	[0, inf)	Increasing prevents overfitting.	The minimum sum of weights required for a node to split.
subsample	1	(0, 1]	Decreasing prevents overfitting.	Limits the percentage of training rows for each boosting round.
colsample_bytree	1	(0, 1]	Decreasing prevents overfitting.	Limits the percentage of training columns for each boosting round.
max_delta_step	0	[0, inf)	Increasing prevents overfitting.	Only recommended for extremely imbalanced datasets.
lambda λ	1	[0, inf)	Increasing prevents overfitting.	L2 regularization of weights.

Recall: Task Estimate $V_1(x) = WP(x \mid 1^{st}$ down and $10)$. using Machine Learning.

Game-state X yardline, score differential, timeouts game seconds remaining, point spread, Receive 2^{nd} half kickoff

Model Setup i = index of i^{th} play in dataset of NFL plays

$y_i = 1$ if team with possession on play i wins, else 0

x_i = game-state vector of play i

$$\text{logit}\, \mathbb{P}(y_i = 1) = f(x_i) + \varepsilon_i$$

Obtain the best possible $\hat{f}(x_i)$ (best predictive performance (logloss))

⟶ use XGBoost to obtain $\hat{f}$ (Ben Baldwin)

* Make the 4^{th} down decision in $\{$Go, FG, Punt$\}$ which maximizes estimated win probability.

Example Plays

Up 3, 4th & 1, 14 yards from opponent end zone
Qtr 2, 03:58 | Timeouts: Off 0, Def 3

	Win %	Success %[1]	Win % if Fail	Win % if Succeed
Go for it	72	68	64	76
Field goal attempt	68	94	62	69

[1] Likelihood of converting on 4th down or of making field goal
Source: @ben_bot_baldwin

(a)

4th down decision bot @ben_bot_baldwin · Jan 29
Automated
---> CIN (3) @ KC (6) <---
KC has 4th & 1 at the CIN 14

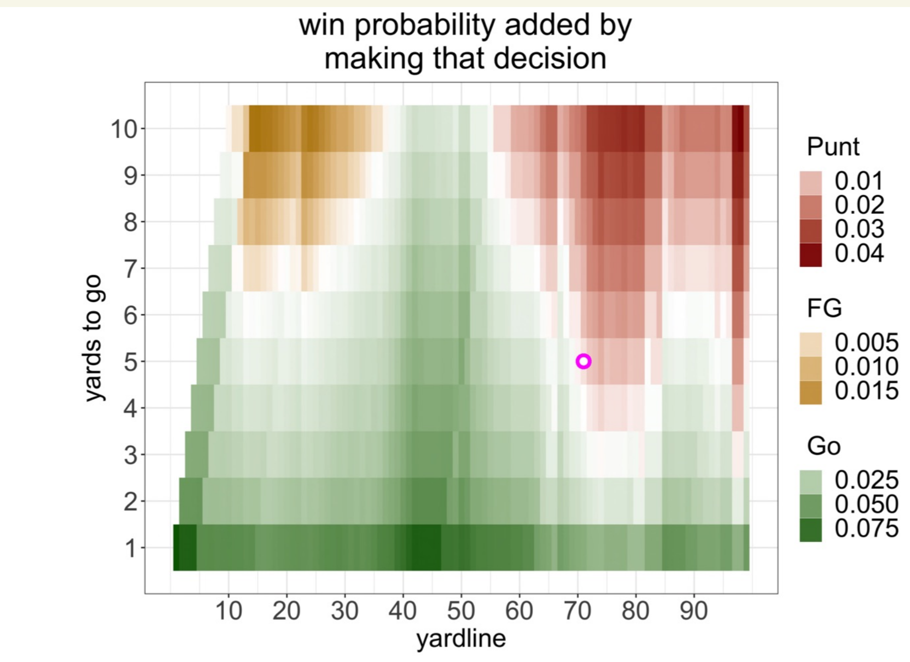
Recommendation (STRONG): 👉 Go for it (+3.8 WP)
Actual play: 👉 (Shotgun) P.Mahomes pass short right to T.Kelce for 14 yards, TOUCHDOWN. H.Butker extra point is GOO

(b)

Figure 6: Baldwin's decision making for example play 1.

Up 1, 4th & 5, 71 yards from opponent endzone
Qtr 3, 5:53 | Timeouts: Off 3, Def 3 | Point Spread: 3

decision	WP	success prob	WP if fail	WP if succeed	baseline coach %
Punt	0.440				0.934
Go for it	0.436	0.426	0.345	0.557	0.066
Field goal	0.345	0.000	0.345	0.548	0.000

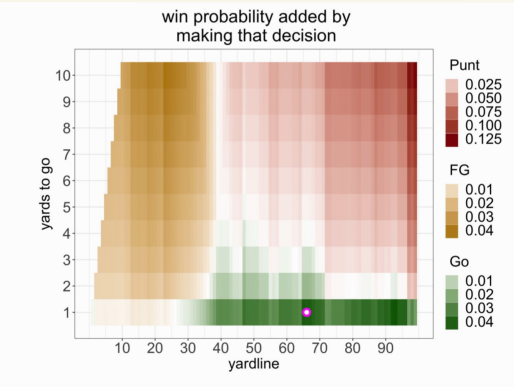


Commanders have the ball against the Colts in week 8 of 2022

Up 6, 4th & 1, 66 yards from opponent endzone

Qtr 4, 2:00 | Timeouts: Off 2, Def 0 | Point Spread: -6.5

decision	WP	WP gain CI	boot %	success prob	WP if fail	WP if succeed	baseline coach %
Go for it	0.927	[0.00295, 0.05236]	0.962	0.691	0.783	0.991	0.229
Punt	0.886		0.038				0.771
Field goal	0.783		0.000	0.000	0.783	0.980	0.000

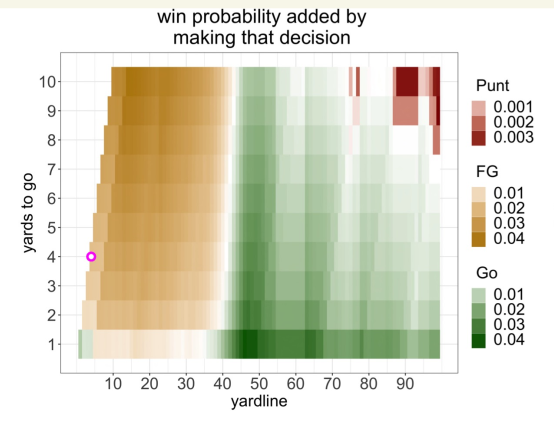


Raiders have ball vs. Rams in week 14 of 2022 (infamous Baker game)

Down 7, 4th & 4, 4 yards from opponent endzone

Qtr 1, 6:02 | Timeouts: Off 3, Def 3 | Point Spread: 8.5

decision	WP	WP gain CI	boot %	success prob	WP if fail	WP if succeed	baseline coach %
Field goal	0.183	[-0.03993, 0.04263]	0.385	0.987	0.11	0.184	0.849
Go for it	0.165		0.615	0.467	0.11	0.228	0.151
Punt	0.099		0.000				0.000



Bears have ball against Jets in week 12 of 2022