WSABI Summer Research Lab **Summer 2025**

Lecture 18: Random Forests & Boosting

Instructor: Jonathan Pipping *Author: Ryan Brill*

18.1 Bagging

18.1.1 Motivation

In Lecture 16, we discussed **Decision Trees** as a machine learning model to capture interactions between variables. However, decision trees present a few issues. For one, they're quite sensitive to the data: if you slightly perturb the training dataset, you can get a very different fitted tree. This makes these models prone to **overfitting**, or memorizing the random noise in the training dataset rather than the "true" underlying relationship (or **signal**). This means decision trees tend to perform poorly on new data. What can we do about this? Consider the work of Leo Breiman in [LB]:

> *"Unstable methods can have their accuracy improved by perturbing and combining, that is, generate multiple versions of the predictor by perturbing the training set or construction method, then combine these multiple versions into a single predictor."*

This method of perturbing and combining multiple models is the motivation for **Bagging**.

18.1.2 The Method

Bagging, or Bootstrap Aggregating, is a method of combining multiple models to improve their performance. Consider a training set $\mathcal{T}$ consisting of N observations, $n = 1, \ldots, N$. The method is as follows:

1. Like in Lecture 9, sample N observations from $\mathcal{T}$ with replacement, forming a bootstrapped training set $\mathcal{T}^{(b)}$.
2. Fit a decision tree $\widehat{f}^{(b)}$ on $\mathcal{T}^{(b)}$.
3. Repeat this process B times, forming a collection of B decision trees $\left\{\widehat{f}^{(b)}\right\}_{b=1}^{B}$.
4. The final model is the average of the B decision trees: $\widehat{f} = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}^{(b)}$.

18.1.3 Why Bagging?

Why might bagging be a good idea? First, pretend we have 1000 independently-drawn datasets of size N from the same underlying distribution, $\{\mathcal{T}^{(i)}\}_{i=1}^{1000}$. Fit 1000 decision trees $\left\{\widehat{f}^{(i)}\right\}_{i=1}^{1000}$, where $\widehat{f}^{(i)}$ is the decision tree fitted on $\mathcal{T}^{(i)}$. Then, we average these 1000 decision trees to form a single model $\widehat{f}$:

$$\widehat{f} = \frac{1}{1000}\sum_{i=1}^{1000} \widehat{f}^{(i)}$$

Consider the expected value of $\widehat{f}$. By the linearity of expectation, we have:

$$\mathbb{E}\left[\widehat{f}\right] = \frac{1}{1000}\sum_{i=1}^{1000}\mathbb{E}\left[\widehat{f}^{(i)}\right]$$

Then since datasets are drawn independently from the same distribution, we have that

$$\mathbb{E}\left[\widehat{f}^{(i)}\right] = \mathbb{E}\left[\widehat{f}^{(1)}\right] \ \forall\ i$$

Thus, we have that

$$\begin{aligned}\mathbb{E}\left[\widehat{f}\right] &= \frac{1}{1000}\sum_{i=1}^{1000}\mathbb{E}\left[\widehat{f}^{(1)}\right]\\ &= \mathbb{E}\left[\widehat{f}^{(1)}\right]\end{aligned}$$

So the aggregated predictor $\widehat{f}$ has the same expected value as one individual decision tree. Now consider the variance of $\widehat{f}$:

$$\text{Var}\left(\widehat{f}\right) = \text{Var}\left(\frac{1}{1000}\sum_{i=1}^{1000}\widehat{f}^{(i)}\right)$$

By independence, we have that

$$\text{Var}\left(\widehat{f}\right) = \frac{1}{1000^2}\sum_{i=1}^{1000}\text{Var}\left(\widehat{f}^{(i)}\right)$$

And from the identical distribution assumption, we have that

$$\text{Var}\left(\widehat{f}^{(i)}\right) = \text{Var}\left(\widehat{f}^{(1)}\right) \ \forall\ i$$

So it follows that

$$\text{Var}\left(\widehat{f}\right) = \frac{1}{1000^2}\sum_{i=1}^{1000}\text{Var}\left(\widehat{f}^{(1)}\right)$$

Call the variance of a single decision tree $\text{Var}\left(\widehat{f}^{(1)}\right) = \sigma^2$. Then we have that

$$\begin{aligned}\text{Var}\left(\widehat{f}\right) &= \frac{1}{1000^2}\sum_{i=1}^{1000}\sigma^2\\ &= \frac{\sigma^2}{1000}\end{aligned}$$

What does this mean? Aggregating 1000 independent decision trees trained on 1000 i.i.d. datasets has the same expected value as a single decision tree, but with a variance reduced by a factor of 1000. This is a significant reduction! However, we only have access to one dataset drawn from the "true" generating process. How can we get around this? As we recall from Lecture 9, sampling with replacement from this dataset (bootstrapping) emulates the process of drawing 1000 i.i.d. datasets from the population, since our original data is itself a random sample from this population. We visualize this in Figure 18.1.

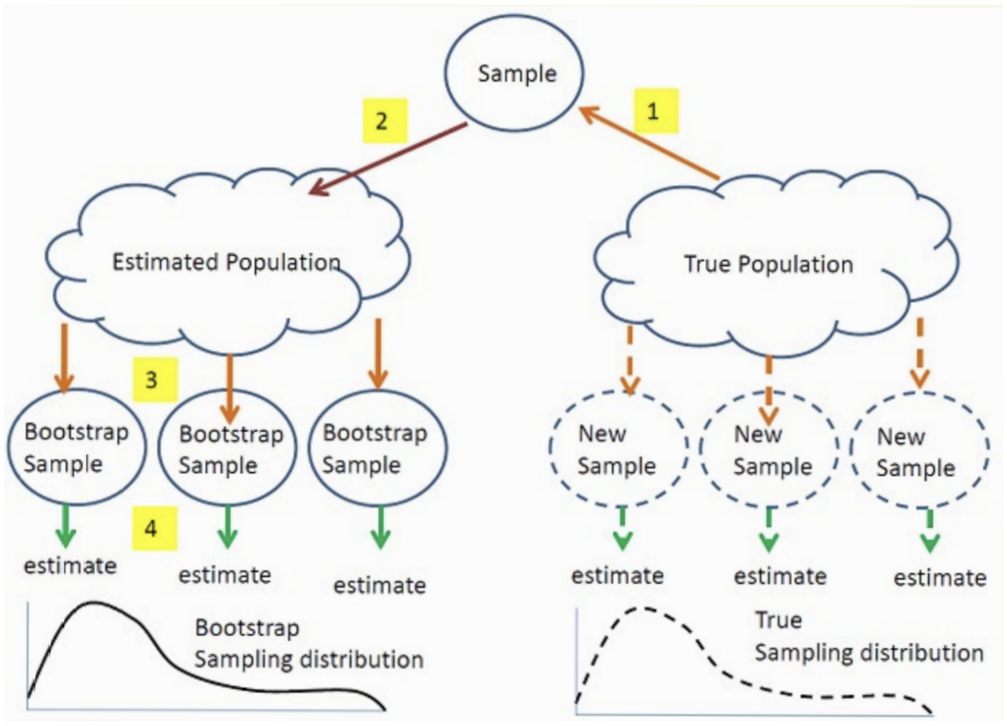


Figure 18.1: Bootstrapping approximates the process of drawing independent samples from the population.

18.1.4 Limitations

Note that bagging still has its limitations. For example, if we have a few dominant predictors in our dataset, bagging will create 1000 decision trees that are all very similar to each other, and the prediction errors will be **highly correlated**. These redundant trees limit the overall variance reduction that we can achieve, and increasing the number of trees doesn't help. This is where **Random Forests** come in.

18.2 Random Forests

18.2.1 Overview

In a random forest, we build a collection of decision trees, each of which is trained on a bootstrapped version of the original dataset. However, we also **randomly** select a subset of the predictors to split on, ensuring that the trees are not redundant! This is a form of **feature selection**, and it helps to reduce the variance of the overall model.

18.2.2 The Method

Consider once again observed training data of the form $\mathcal{T} = (\boldsymbol{X}, y)$, where $\boldsymbol{X} \in \mathbb{R}^{N \times p}$ and $y \in \mathbb{R}^N$. Then, for each $b \in \{1, \ldots, B\}$, we do the following:

1. Bootstrap: Sample n observations from N rows of $\mathcal{T}$ with replacement.
2. Sample Columns: Sample m predictors from p columns of $\boldsymbol{X}$ without replacement. This in combination with Step 1 yields a new training set $\mathcal{T}^{(b)} = \left(\boldsymbol{X}^{(b)}, y^{(b)}\right)$ with n observations and m predictors.
3. Fit a decision tree $\widehat{f}^{(b)}$ on $\mathcal{T}^{(b)}$.

Finally, we average the B decision trees to form a single model:

$$\widehat{f} = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}^{(b)}$$

Note that there are 3 hyperparameters to choose: B (the number of trees in the forest), n (the number of observations in each tree), and m (the number of predictors to sample for each tree). We can use cross-validation to choose these hyperparameters, and some common values to search over are listed here:

- $B \in \{100, 200, 500, 1000\}$
- $n \in \{N/2, N\}$
- $m \in \left\{\sqrt{p}, p/3, p/2\right\}$

18.2.3 Why Random Forests?

This model is called a **Random Forest** because it is a collection of bootstrapped decision trees, each trained on a **random subset** of the predictors. This aggregated model significantly reduces the variance in our predictions, and since the expected value of a random forest is the same as that of a single tree, their biases are the same. Then from our Bias-Variance decomposition in 18.1, we can expect the out-of-sample error for a random forest to be significantly lower than that of a single tree.

$$\text{MSE}\left(\widehat{f}\right) = \underbrace{\left(\mathbb{E}\left[\widehat{f} - f\right]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[\left(\widehat{f} - \mathbb{E}\left[\widehat{f}\right]\right)^2\right]}_{\text{Variance}} + \underbrace{\mathbb{E}\left[\epsilon^2\right]}_{\text{Irreducible Error}} \tag{18.1}$$

18.3 Boosting

18.3.1 Overview

To this point, we have discussed decision trees and random forests. We know that decision trees are extremely sensitive to the data, overfitting and performing very poorly on new data. We improved this significantly with random forests, which used an ensemble of decision trees to reduce variance and prevent overfitting. There's still a piece of our puzzle missing though: how can we reduce the bias in our model? **Boosting** offers a way to do this.

18.3.2 The Method

In Boosting, we train multiple decision trees sequentially, with each successive model trained to optimize some loss function which improves the fit of the overall system. Consider our same training set $\mathcal{T} = (\boldsymbol{X}, y)$, where $\boldsymbol{X} \in \mathbb{R}^{N \times p}$ and $y \in \mathbb{R}^N$. Then, we will train a collection of K decision trees and form a final model as an aggregation of these trees:

$$\widehat{y}_i = \varnothing(x_i) = \sum_{k=1}^{K} f_k(x_i)$$

To learn the set of functions (trees) $\{f_k\}_{k=1}^K$, used in this model, we minimize a Regularized objective function:

$$\mathcal{L}(\varnothing) = \sum_{i=1}^{N} \ell(y_i, \widehat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where ℓ is some loss function, and $\Omega(f_k)$ is a regularization term on the decision tree f_k, defined as

$$\Omega(f_k) = \gamma T_k + \frac{1}{2}\lambda \sum_{j=1}^{T_k} \omega_{jk}^2$$

where T_k is the number of leaves in tree k, ω_{jk} is the weight of leaf j in tree k, and γ and λ are regularization parameters. In a binary outcome setting ($y \in \{0, 1\}$), we use the log loss function, defined as

$$\begin{aligned} \ell(y_i, \widehat{y}_i) &= \text{log loss}(\widehat{y}_i, y_i) \\ &= y_i \log(\widehat{y}_i) + (1 - y_i)\log(1 - \widehat{y}_i) \end{aligned}$$

However, if we are in a continuous outcome setting ($y \in \mathbb{R}$), we use the squared error loss function, defined as

$$\ell(y_i, \widehat{y}_i) = (y_i - \widehat{y}_i)^2$$

Finally, we define our optimal $\widehat{\varnothing}$ as the sequence of functions that minimizes this objective function:

$$\widehat{\varnothing} = \sum_{k=1}^{K} \widehat{f}_k = \arg\min_{\widehat{\varnothing}} \mathcal{L}\left(\widehat{\varnothing}\right)$$

18.3.3 Training the Trees

Now that we have a loss function $\mathcal{L}(\varnothing) = \mathcal{L}\left(\sum_{k=1}^K f_k\right)$ to train our decision trees, how do we minimize it? We will do this **sequentially**, which is the essence of Boosting. At each iteration t, let $\widehat{y}_i^{(t)}$ be the prediction of the i^{th} observation. To fit the decision tree f_t at the t^{th} iteration, we use a modified version of $\mathcal{L}(\varnothing)$:

$$\mathcal{L}^{(t)}(f_t) = \sum_{i=1}^{N} \ell\left(y_i, \widehat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

where $\widehat{y}_i^{(t-1)} = \sum_{k=1}^{t-1} f_k(x_i)$ is the prediction of the i^{th} observation at the t^{th} iteration, using the previous $t-1$ trees. We can then fit the decision tree f_t by minimizing this modified objective function. Formally, since $\sum_{k=1}^t f_k = \left(\sum_{k=1}^{t-1} f_k\right) + f_t$, we have that

$$f_t = \arg\min_{f_t} \mathcal{L}^{(t)}(f_t)$$

We can then fit the decision tree f_t by minimizing this modified objective function, which is done by considering many possible trees and selecting the minimizer of $\mathcal{L}^{(t)}(f_t)$.

18.3.4 Example: NFL Win Probability

We will now implement boosting to estimate play-by-play NFL win probabilities, like Ben Baldwin did in [BB]. We will focus only on 1st down and 10's to simplify the problem. Our data (from the `nflfastR` package) contains the following information:

- `i`: the index of the i^{th} play
- `y_i`: the outcome of the game, which is 1 if the team in possession won and 0 otherwise
- `x_i`: the game state, which is a vector of the yard line, the score differential, the seconds remaining in the game, the number of timeouts remaining for each team, the pre-game point spread, and a few other variables

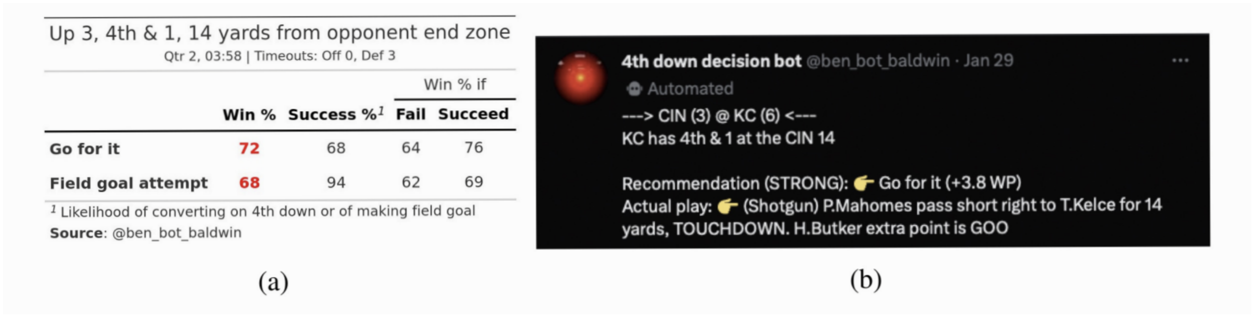
With this in mind, our goal is to estimate the following quantities:

$$\mathrm{WP}(x_i) = \mathrm{WP}(x_i \mid \text{1st down and 10})$$

In his paper, Ben Baldwin uses XGBoost to estimate $\mathbb{P}_{\mathrm{FG}}$, $\mathbb{E}[\text{next yard line after punting}]$, and $\mathbb{P}_{\text{conversion}}$, knitting these together to estimate $\mathrm{WP}_{\mathrm{Go}}(x_i)$, $\mathrm{WP}_{\mathrm{FG}}(x_i)$, and $\mathrm{WP}_{\mathrm{Punt}}(x_i)$. For more details, read Appendix B of [BYW]. From here, Baldwin recommends choosing the decision which maximizes estimated value (win probability). Mathematically, the optimal decision is expressed as

$$\underset{d \in \{\text{Go, FG, Punt}\}}{\arg\max} \quad \mathrm{WP}_d(x_i)$$

with the gain from the decision being the difference in win probability between the two best options. This model is public on Twitter, and an example of the model output is shown in Figure 18.2.



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

			Win % if	
	Win %	Success %[1]	Fail	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)

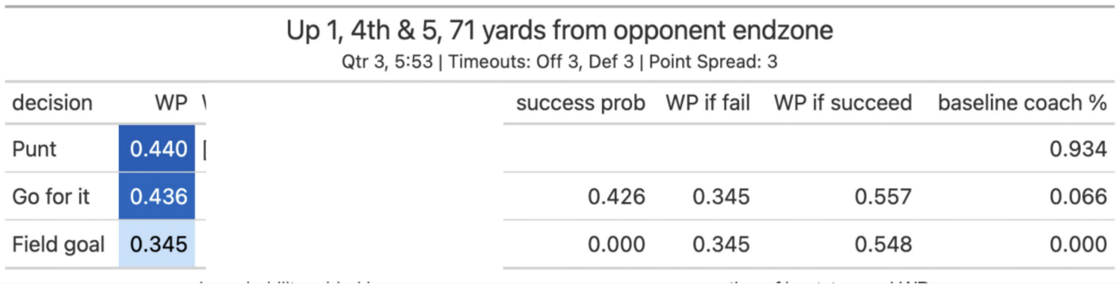
4th down decision bot @ben_bot_baldwin · Jan 29
Automated
---> CIN (3) @ KC (6) <---
KC has 4th & 1 at the CIN 14

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 18.2: Example of the output of Ben Baldwin's 4th down decision model.

Ryan implements a similar approach in [BYW], but includes a few more features to model the strength of each team's offense and defense. His code and Shiny app are available on GitHub, and examples of the output are shown in Figures 18.3 and 18.4.



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

Figure 18.3: Example output of Ryan's 4th down decision model.

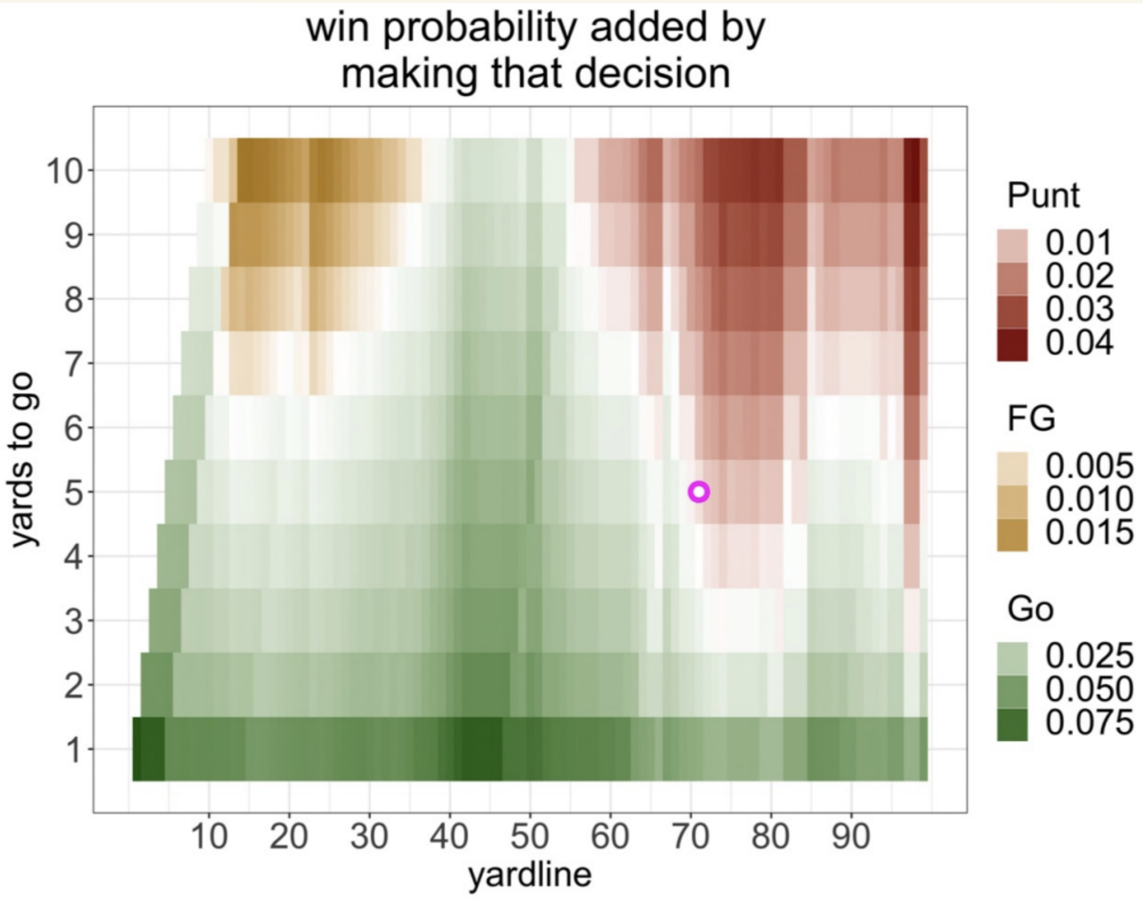


Figure 18.4: Estimated win probability gain from Ryan's 4th down decision model.

References

[BB] Baldwin, B., *NFL Win Probability from Scratch using XGBoost in R*, April 16, 2021.

[BL] Breiman, L., *Arcing Classifiers*, The Annals of Statistics, Vol. 26, No. 3, pp. 801-824, 1998.

[BYW] Brill, R. S., Yurko, R., & Wyner, A. J., *Analytics, Have Some Humility: A Statistical View of Fourth-Down Decision Making*, The American Statistician, 2025.