Random Forests

Last time: WP(x) = Win Probability if you have possession at game-state x = (yardline, game seconds remaining, pointspread, timeouts, receive 2nd half kickoff, ——)

We introduced Decision Trees as a machine learning model to capture interacting relationships between variables.

Problem: decision trees are unstable and prone to overfitting.

Unstable: if you slightly perturb the training dataset, you can get a very different model

Overfitting: memorizing the random noise in the training dataset rather than the "true" underlying trend (signal)

→ show an example

What to do about this? Ask Leo Breiman,

> 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. [redacted]

- small changes in the training set or in the construction of a decision tree can lead to larger changes in the aggregated predictor

---

Bagging = Bootstrap – Aggregating

Training set T consists of N instance $n=1,\dots,N$ (e.g. N football plays)

put equal probabilities $p(n)=\frac{1}{N}$ on each instance.

Sample with Replacement (Bootstrap) N times from the training set T, forming a "new" Resampled training set $T^{(B)}$.

Then fit a decision tree $\hat{f}^{(B)}$ on $T^{(B)}$.

Lets do this, say, 1000 times, so we have 1000 bootstrapped decision trees $\{\hat{f}^{(1)},\dots,\hat{f}^{(1000)}\}$.

Aggregate $\hat{f}=\frac{1}{1000}\sum_{B=1}^{1000}\hat{f}^{(B)}$.

Why might bagging be a good idea?

1. First, pretend you have 1000 independently drawn datasets of size N, $\{T^{(1)}, \ldots, T^{(1000)}\}$.

Fit 1000 decision trees $\{\hat{f}^{(1)}, \ldots, \hat{f}^{(1000)}\}$ where $\hat{f}^{(B)}$ fit on $T^{(B)}$,

$$\hat{f} = \frac{1}{1000} \sum_{B=1}^{1000} \hat{f}^{(B)}$$ and

$$\mathbb{E}\hat{f} = \frac{1}{1000} \sum_{B=1}^{1000} \mathbb{E}\hat{f}^{(B)}$$ by linearity of expectation

$$= \frac{1}{1000} \sum_{B=1}^{1000} \mathbb{E}\hat{f}^{(1)}$$ because $\mathbb{E}\hat{f}^{(B)} = \mathbb{E}\hat{f}^{(1)}$ for all B

$$= \mathbb{E}\hat{f}^{(1)}.$$

The aggregated predictor has the same expected value as 1 individual decision tree.

$$\mathrm{VAR}(\hat{f}) = \mathrm{VAR}\left(\frac{1}{1000}\sum_{B=1}^{1000}\hat{f}^{(B)}\right)$$

$$= \frac{1}{1000^2}\sum_{B=1}^{1000}\mathrm{VAR}\left(\hat{f}^{(B)}\right) \quad \text{by independence}$$

$$= \frac{1}{1000^2}\sum_{B=1}^{1000}\sigma^2$$

$$= \frac{\sigma^2}{1000}$$

$\mu = \mathbb{E}\hat{f}^{(B)}$, $\sigma^2 = \mathrm{VAR}(\hat{f}^{(B)})$, then

$\mathbb{E}\hat{f} = \mu$ $\quad \mathrm{VAR}(\hat{f}) = \frac{\sigma^2}{1000}$.

Aggregating independent decision trees has same expected value but lower variance.

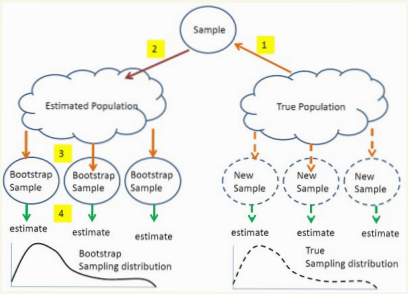
Assumed 1000 independently drawn training sets from some "true" function.

* In reality we have only one independently drawn dataset of size N, not 1000.

Bootstrapping: created 1000 different training datasets by Resampling N rows with replacement.

Idea behind Bagging one bootstrapped dataset is a somewhat close approximation to one independently drawn dataset from the "true" generating process.

(Recall the bootstrap lecture).



Sampling from the "true" underlying data generating process is similar to sampling from the observed training set because the observed training set itself is sampled from the "true" underlying data generating process.

Random Forest

Input: observed training data $T = (X, y)$

Hyperparameters: $n, m, A = 1000$

for $B = 1, \ldots, A$:

1. Bootstrap (row subsampling): Sample n of N rows of T with replacement
2. Column subsampling: Sample m of the columns (features) of X

These 2 things yield a new training set $T^{(B)}$ with n rows and m columns.

3. fit a decision tree $\hat{f}^{(B)}$ on $T^{(B)}$.

Aggregate: $\hat{f} = \frac{1}{1000} \sum_{B=1}^{1000} \hat{f}^{(B)}$.

* Random forest is much better than a single decision tree because it reduces variance.

Bias-Variance Tradeoff

out-of sample prediction error = Bias + Variance + Irreducible error

↓

lower for R.F. than dec. tree

Bias: $\mathbb{E}(\hat{f} - f)^2$

Same for 1 dec. tree and 1 Random Forest

Variance: R.F. has lower variance than 1 decision tree

Irreducible error: Same no matter what model you use

* one decision tree $\hat{f}(T)$ is more unstable and sensitive to the Random idiosyncracies of the traing set than 1000 aggregated decision trees $\hat{f}(T) = \frac{1}{1000} \sum_{B=1}^{1000} \hat{f}^{(B)}(T)$.

Boosting & XGBoost

- decision trees → overfit
- random forests → reduces overfitting by reducing variance

We can do better.

- boosting → also reduces bias

Boosting – train multiple models (e.g. decision trees) sequentially; train each successive model to optimize some loss function to improve the accuracy of the overall system.

Tree Boosting in a nutshell

- dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$
- n training examples $n = |\mathcal{D}|$
- m features (columns) $x_i \in \mathbb{R}^m$ (game-state)
- outcome $y_i \in \mathbb{R}$ (win/loss indicator)

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

f_k decision tree

To learn the set of functions (trees) $\{f_k\}$ used in this model, we minimize a Regularized objective function,

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

where $\Omega(f) = \gamma \cdot T + \frac{1}{2}\lambda \cdot \sum_j w_{jk}^2$

$y_i \in \{1, 0\}$ binary: $l(y_i, \hat{y}_i) =$ log loss

$= y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$

$y_i \in \mathbb{R}$ continuous: $l(y_i, \hat{y}_i) =$ squared error $= (y_i - \hat{y}_i)^2$

Regularization: $\Omega(f_k) =$ regularization term on the decision tree f_k

$T =$ # of leaves

$w_{jk} =$ weight value of tree k leaf j

$\longrightarrow$ making # leaves smaller and weights smaller

$\phi = \sum_k f_k = \underset{\phi}{\text{argmin}}\ \mathcal{L}(\phi)$

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

$\Omega(f_k) = \gamma \cdot T_k + \frac{1}{2}\lambda \cdot \sum_j w_{jk}^2$

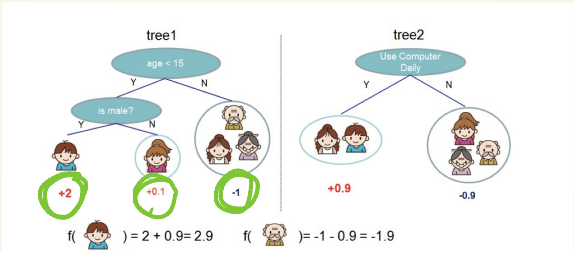


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

Ridge Regression $\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_j \beta_j^2$

$\sum_{i=1}^{n} (y_i - x_i^T \beta)^2$: squared error

$\lambda \sum_j \beta_j^2$: Regularized term

We wrote down a loss function $L(\phi) = L\left(\sum_k f_k\right)$ that we want to minimize in order to train our decision trees $\{f_k\}$. But how can we actually minimize this?

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

→ minimize in a sequential manner.

this is the essence of boosting!

At iteration t, $\hat{y}_i^{(t)}$ prediction of i^{th} datapoint.
To fit the decision tree f_t at the t^{th} iteration, lets use a modified version of L,

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

Assume we've already fit $t-1$ decision trees.
Then we have a prediction $\hat{y}_i^{(t-1)}$ from the aggregation of these $t-1$ decision trees.
Then the t^{th} aggregation looks like $\hat{y}_i^{(t-1)} + f_t(x_i)$.

Recall $\sum_{k=1}^{t} f_k = \left(\sum_{k=1}^{t-1} f_k\right) + f_t.$

Boosting = sequential optimization.

$$f_t = \underset{f_t}{\operatorname{argmin}}\ L^{(t)}(f_t)$$

How to actually minimize $L^{(t)}$?

↳ consider many decision trees f_t and simply select the one which minimizes $L^{(t)}$.

See Chen and Guestrin (2016) for more details.

Estimating in-game win probabilities in American Football

Ben Baldwin:

Estimate $WP_i(x) = WP(x \mid 1^{st} \text{ down and } 10)$

i = index of the i^{th} play

Game-state x_i = yardline, score differential, timeouts, game seconds remaining, pre-game point spread, and a few other variables

Win/loss outcome $y_i = 1$ if team with possession won the game else 0

Baldwin estimates WP_i using XGBoost.

He also estimates $\mathbb{P}_{FG}$, $\mathbb{E}$ next yardline after punting, $\mathbb{P}_{conversion}$ and knits these together to estimate $WP_{go}(x)$, $WP_{fg}(x)$, $WP_{punt}(x)$.

Read my paper "Analytics Have Some Humility: A Statistical View of Fourth Down Decision Making" (Appendix B) for more details.

Baldwin: Choose the decision which maximizes estimated value (win probability),

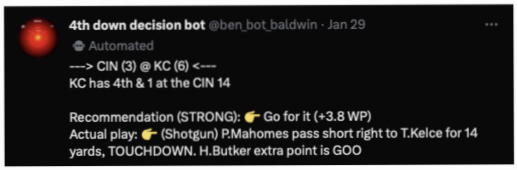
$$\underset{d \in \{Go, Fg, Punt\}}{\operatorname{argmax}} \; WP_d(x).$$

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)



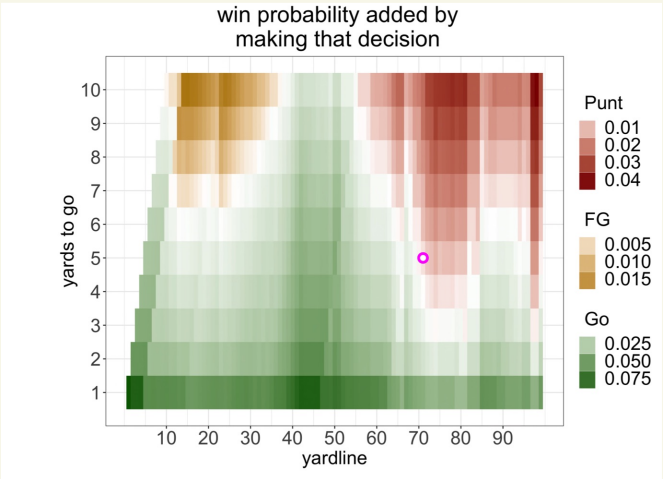
(b)

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

Gain = Value of Best − Value of 2nd best

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

See my Shiny App! Publicly available for download.