Random Forests and Gradient-Boosted Trees

2023-07-07

Bagging

Bootstrap aggregation (aka bagging) is a general approach for overcoming high variance

- Bootstrap: sample the training data with replacement – each bootstrap sample should have the same number of observations as the original sample

Bagging Algorithm

- Start with a specified number of trees B
- To generate a preidction for a new point:

Regression: take the average across the B trees

Classification: take the majority vote across B trees (could also use probabilities from the trees)

Improves prediction accuracy via **wisdom of the crowds** but at the expense of interpretability.

But what if these trees are quite similar to one another?

Random Forest Algorithm

Random Forests are **an extension of bagging**

For each tree b in 1,. . . ,B:

- construct bootstrap sample from the training data
- grow a deep, unpruned, complicated (aka overfit) **but with a twist**
- At each split: limit the variables considered to a random subset m (random subset without replacement) of the original p variables

Like bagging:

- Regression: take the average across the B trees
- Classification: take the majority vote across B trees (could also use probabilities from the trees)

split-variable randomization adds more randomness to make each tree more independent of each other

m is a tuning parameter

Example with MLB data

```
## # A tibble: 6 x 20
##       g    pa    hr     r   rbi    sb bb_percent k_percent   iso babip   avg
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>     <dbl> <dbl> <dbl> <dbl>
## 1    85   374    22    62    55     2        6.7      17.6 0.28  0.353 0.327
## 2    88   385    33    72    69     8       11.4      26   0.337 0.296 0.281
## 3    88   370    18    41    59     1        8.9      13   0.233 0.295 0.293
```

```
## 4    82   349    15    56    51     7       10.6      18.6 0.213 0.346 0.306
## 5    90   391    20    64    70     5       12        21.2 0.26  0.388 0.33
## 6    87   375    19    54    75    13       10.7       9.9 0.288 0.275 0.288
## # i 9 more variables: obp <dbl>, slg <dbl>, w_oba <dbl>, xw_oba <dbl>,
## #   w_rc <dbl>, bs_r <dbl>, off <dbl>, def <dbl>, war <dbl>
```

```
## Ranger result
##
## Call:
##  ranger(war ~ ., data = model_mlb_data, num.trees = 50, importance = "impurity")
##
## Type:                             Regression
## Number of trees:                  50
## Sample size:                      157
## Number of independent variables:  19
## Mtry:                             4
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.2359718
## R squared (OOB):                  0.8437078
```

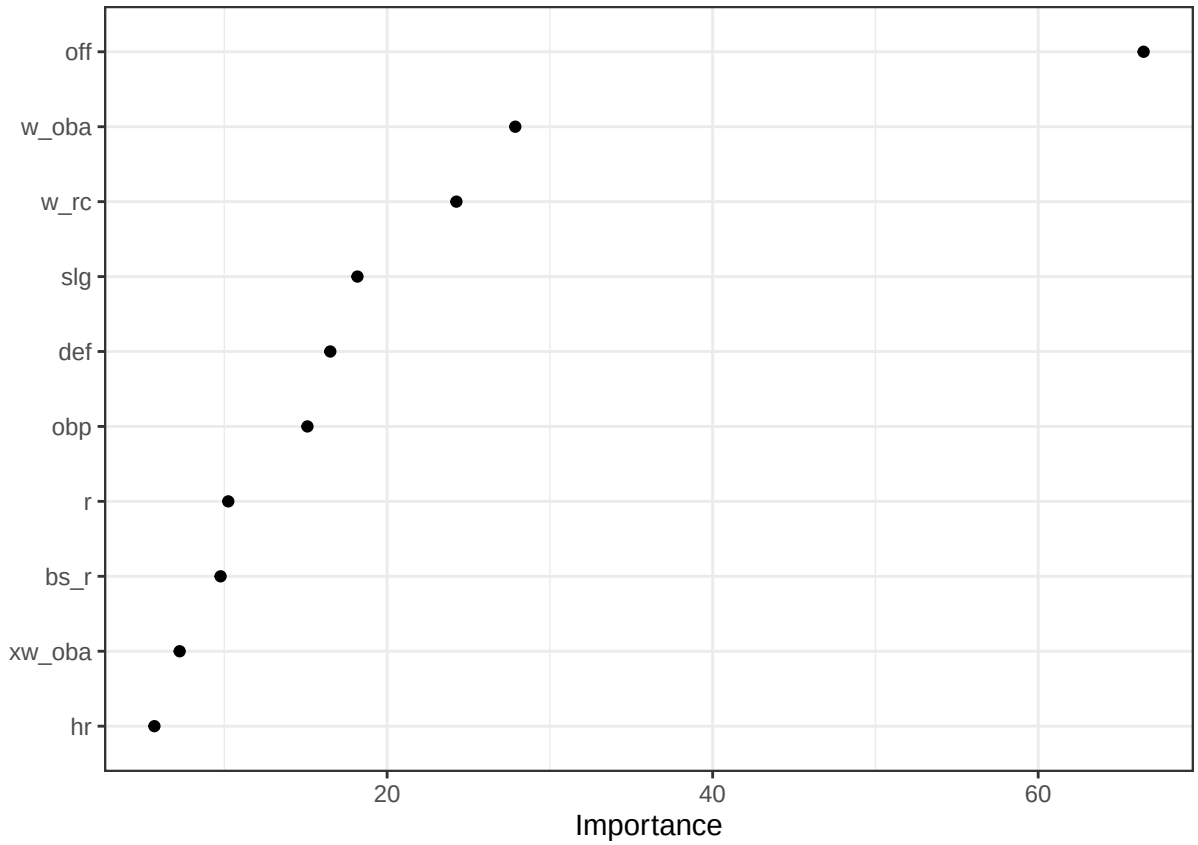
Out-of-Bag Estimate

Since the trees are constructed via bootstrapped data (samples with replacements) - each sample is likely to have duplicate observations / rows.

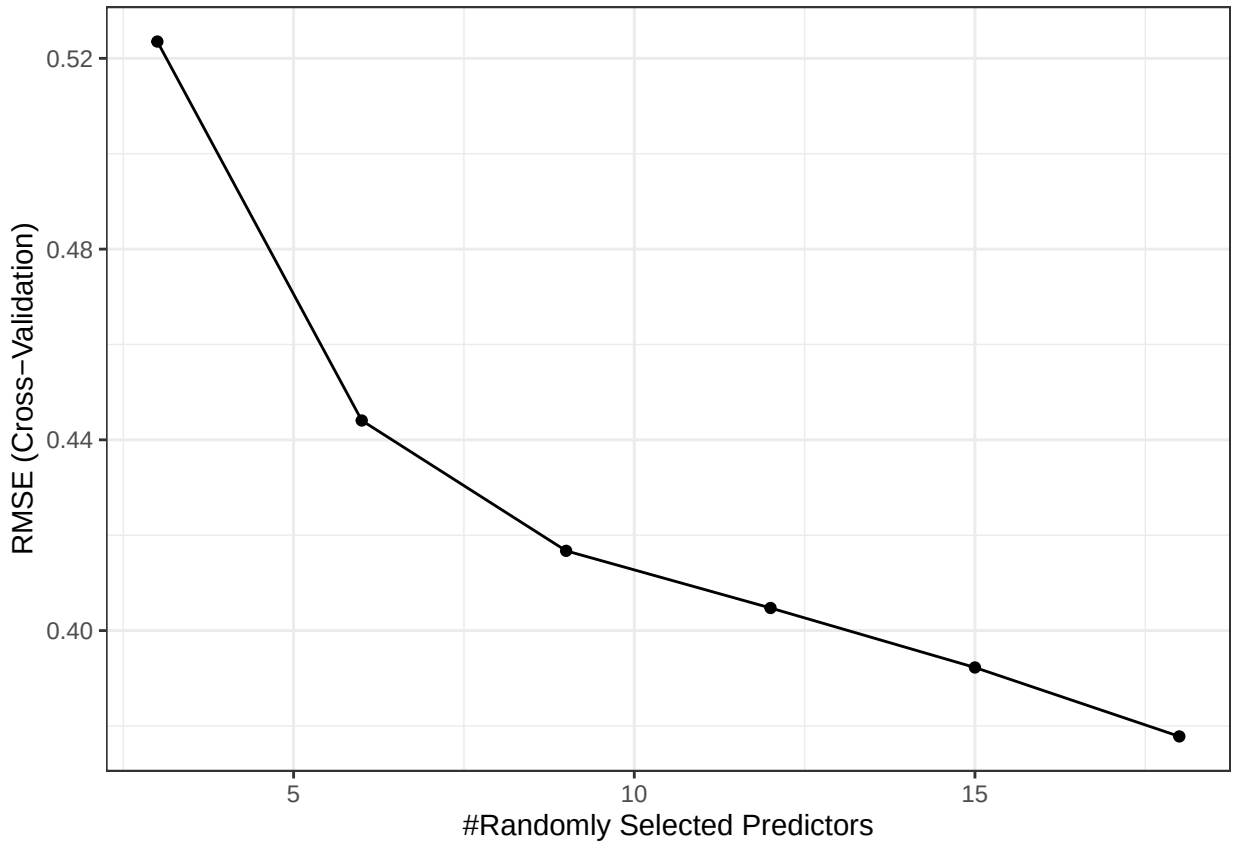
Out-of-bag (OOB): original observations not contained in a single bootstrap sample

- Can use the OOB samples to estimate predictive performance (OOB becomes better with larger datasets)
- On average about 63% of original data ends up in any particular bootstrap sample

Variance Importance



Tuning Random Forests



Boosting

Build ensemble models **sequentially**

- start with a weak learner, e.g. small decision tree with few splits
- each model in the sequence *slightly* improves upon the predictions of the previous models **by focusing on the observations with the largest errors / residuals** (i.e., up-weight the observations incorrectly predicted)

Boosted Trees Algorithm

regression setting

Write the prediction at step t of the search as $\hat{y}_i^{(t)}$, start with $\hat{y}_i^{(0)} = 0$

- Fit the first decision tree f_1 to the data: $\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$
- Fit the next tree f_2 to the residuals of the previous: $y_i - \hat{y}_i^{(1)}$
- Add this to the prediction: $\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + f_2(x_i) = f_1(x_i) + f_2(x_i)$
- Fit the next tree f_3 to the residuals of the previous: $y_i - \hat{y}_i^{(2)}$
- Add this to the prediction: $\hat{y}_i^{(3)} = \hat{y}_i^{(2)} + f_3(x_i) = f_1(x_i) + f_2(x_i) + f_3(x_i)$

Continue until some stopping criteria to reach final model as a **sum of trees**:

$$\hat{y}_i = f(x_i) = \sum_{b=1}^{B} f_b(x_i)$$

Gradient Boosted Trees

Regression boosting algorithm can be generalized to other loss functions (not just residuals) via gradient descent - leading to gradient boosted trees, aka gradient boosting machines (GBMs)

Update the model parameters in the direction of the loss function's descending gradient

SOME NOTES:

we must tune the learning rate (i.e., it is a hyperparameter)

Stochastic gradient descent can help with complex loss functions

- *Can take random samples of the data when updating - makes algorithm faster and adds randomness to get closer to global minimum (no guarantees!)*

Tuning GBMs

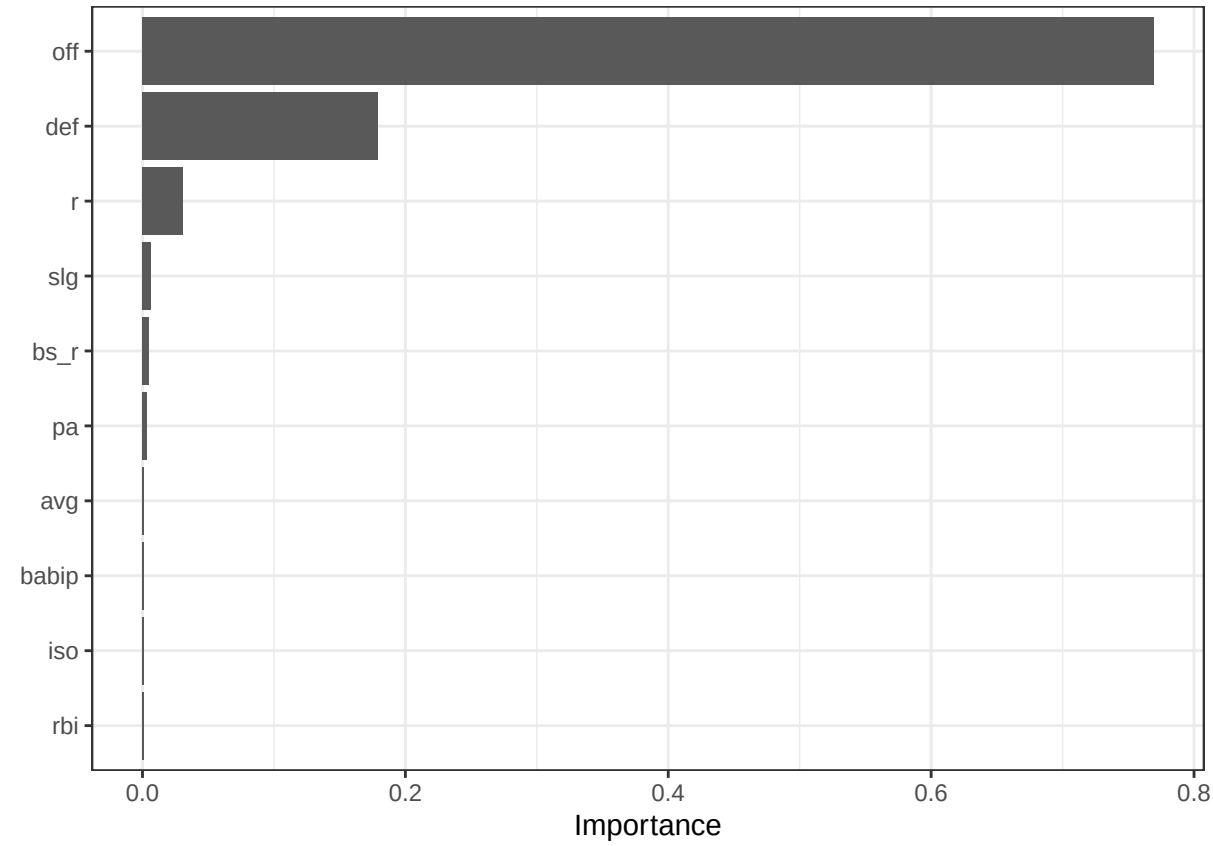
What we have to consider tuning (our hyperparameters):

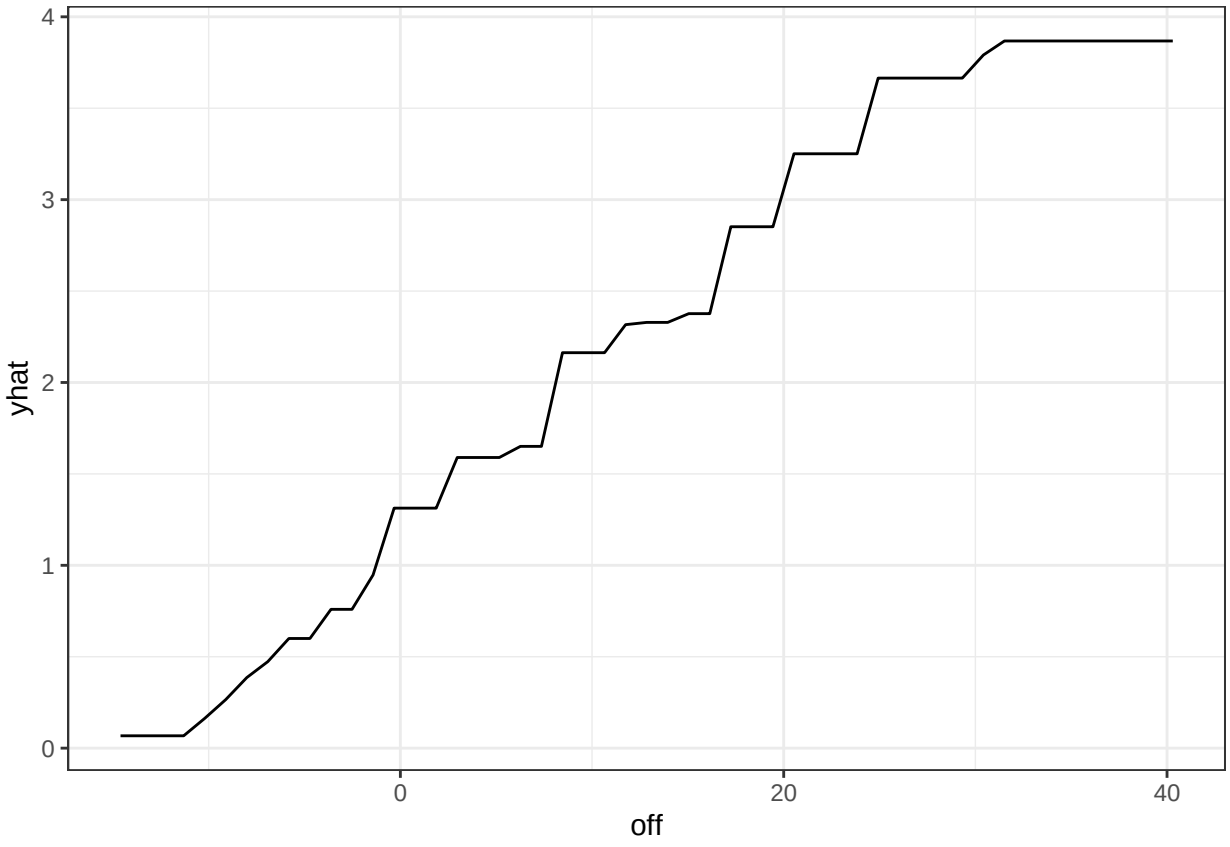
- number of trees B (nrounds)
- learning rate (eta), i.e. how much we update in each step
- these two really have to be tuned together
- complexity of the trees (depth, number of observations in nodes)
- XGBoost also provides more regularization (via gamma) and early stopping

More work to tune properly as compared to random forests... but GBMs have more flexibility in their usage for particular objective functions

Example

```
##     nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 130     200         1 0.3     0                1                1         1
```


off
def
r
slg
bs_r
pa
avg
babip
iso
rbi
0.0
0.2
0.4
0.6
0.8
Importance


4
3
2
1
0
yhat
0
20
40
off