Ensembles

DS 6410 | Spring 2025

ensembles.pdf

Contents

1 Introduction to Ensemble Models

Ensemble models combine predictions from several individual models (individual models are also called *base learners*).

1.1 Notation

- Observed data:
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - n observations
  - Regression: $y_i \in \mathbf{R}$
  - Classification: $y_i \in \mathcal{G}$
- Base learners (i.e., individual models)
  - $\hat{g}_1(x), \hat{g}_2(x), \ldots, \hat{g}_M(x)$
  - There are M base models
- Ensemble Model
  - $\hat{f}(x) = \mathcal{F}(\hat{g}_1(x), \hat{g}_2(x), \ldots, \hat{g}_M(x))$
  - $\mathcal{F}$ is generic notation for methods of combining, aggregating, or using the information from all M models to make a prediction.
- Summary: Ensemble approaches differ in *which* base models are used how they are *combined*
- Benefits:
  - Collective Knowledge of Crowds / Mixture of Experts
  - Bagging: variance reducer
  - Boosting: bias reducer

1.2 Bagging

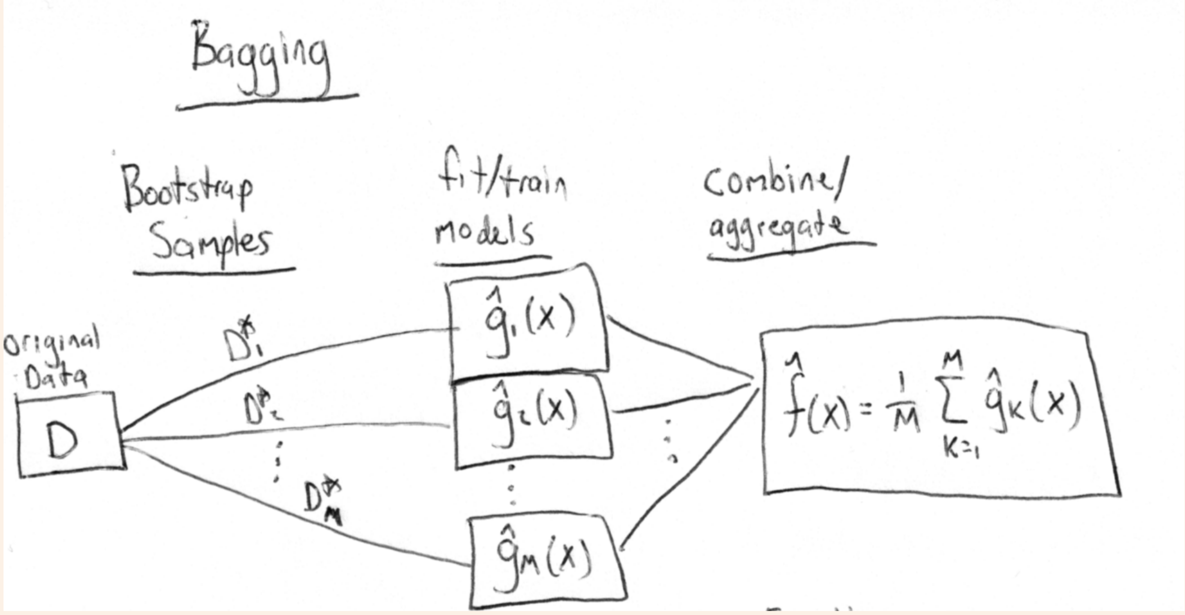
Bagging fits the *same base model* to *bootstrap samples* of the observed data and *averages* the predictions from each model.

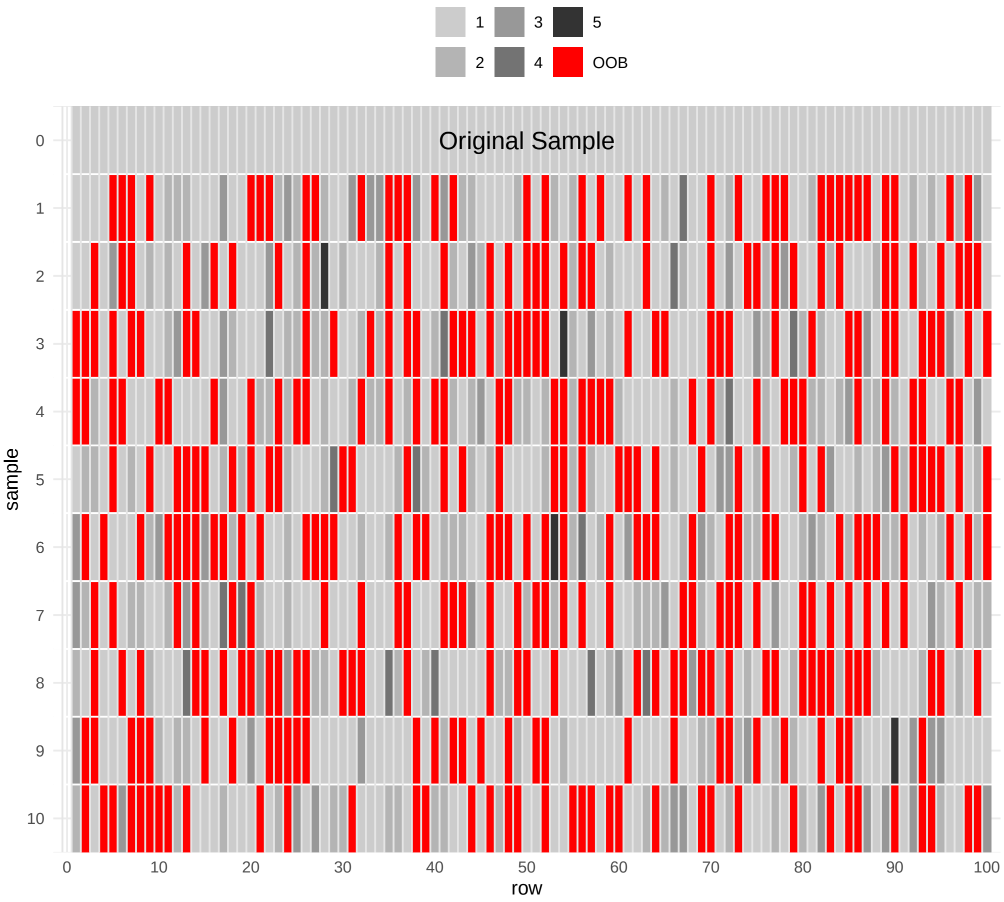
- The base model, $g(x)$, is usually a *deep* tree (or other high variance model)
- The predictions from the base models are averaged:

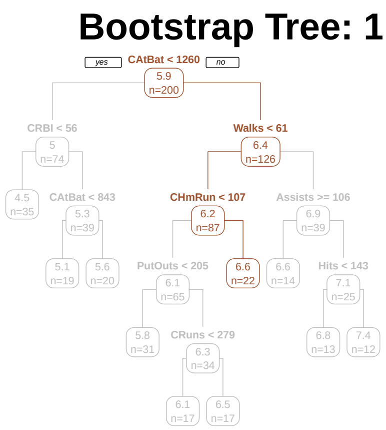
$$\hat{f}(x) = \frac{1}{M} \sum_{k=1}^{M} \hat{g}(x \mid D_k^*)$$

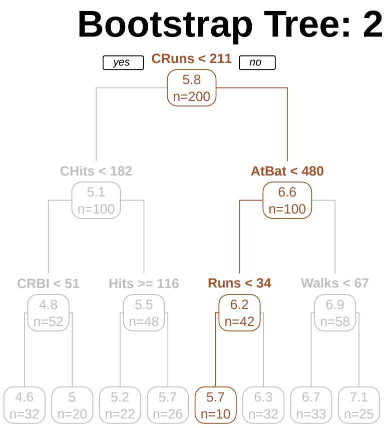
- M is the number of bootstrap samples (and thus base models)
- D_k^* is the k^{th} bootstrap sample
- $\hat{g}(x \mid D_k^*)$ is the predictions from the model fit to the k^{th} bootstrap sample

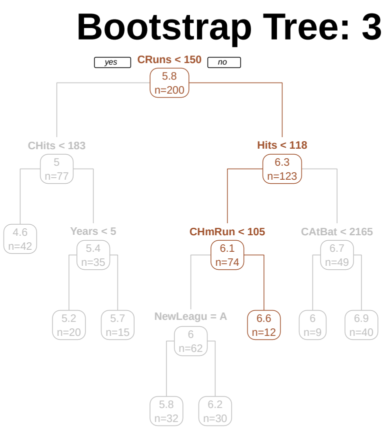
Bagging Illustration

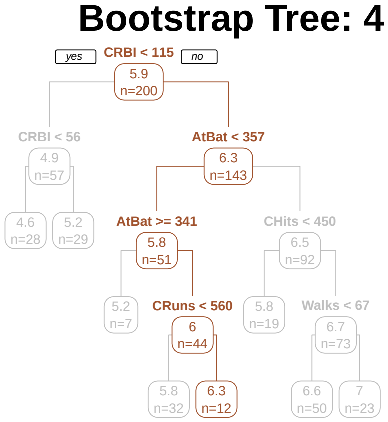


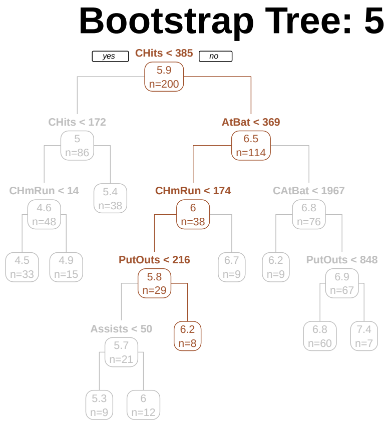

1
2
3
4
5
OOB
Original Sample
sample
0
1
2
3
4
5
6
7
8
9
10
row
0
10
20
30
40
50
60
70
80
90
100

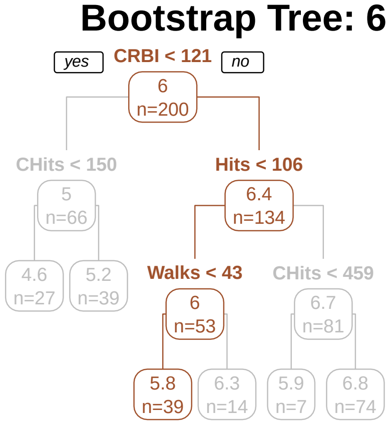

Bootstrap Tree: 1

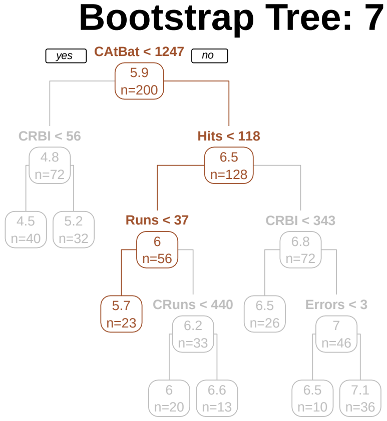

Bootstrap Tree: 2

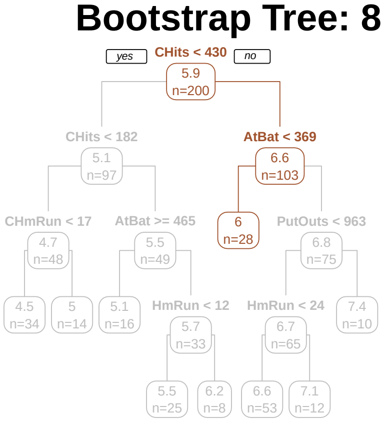

Bootstrap Tree: 3

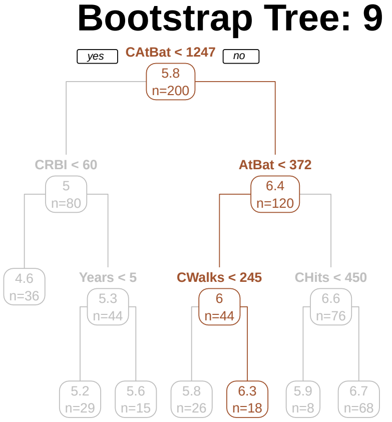

Bootstrap Tree: 4

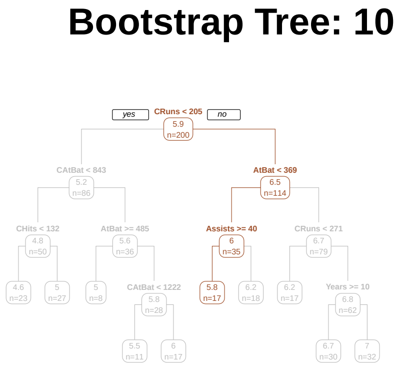

Bootstrap Tree: 5

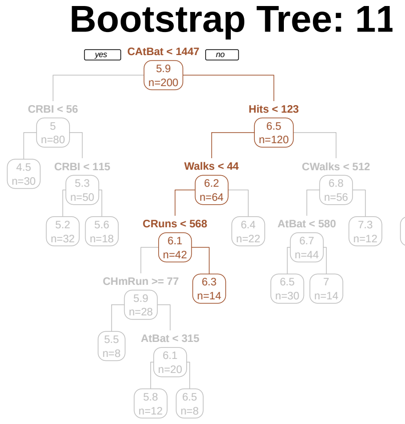

Bootstrap Tree: 6

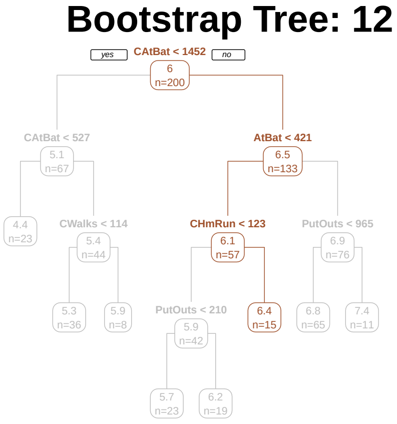

Bootstrap Tree: 7


Bootstrap Tree: 8


Bootstrap Tree: 9


Bootstrap Tree: 10


Bootstrap Tree: 11


Bootstrap Tree: 12

1.2.1 Bagging Variations

- **Random Forest** models fit trees to bootstrap data, but with the extra de-correlation step of only considering a subset of features for each split.
- **Sub-bagging**: D_k^* is a *sub-sample* (less than n) *without* replacement
- **Bragging**: use the *median* instead of the mean to combine predictions

$$\hat{f}(x) = \text{median}(\hat{g}(x \mid D_1^*), \hat{g}(x \mid D_2^*), \ldots, \hat{g}(x \mid D_M^*))$$

- **Bumping**: Like bagging, but *choose best model* instead of averaging.

$$\hat{f}(x) = \hat{g}(x \mid D_{\text{opt}}^*)$$

  - where $\text{opt} = \arg\min_k \sum_{i=1}^n L(y_i, \hat{g}(x_i \mid D_k^*))$
  - Include the original data D in the comparison
  - Thus, only a single (potentially bagged) dataset is being used for the final model

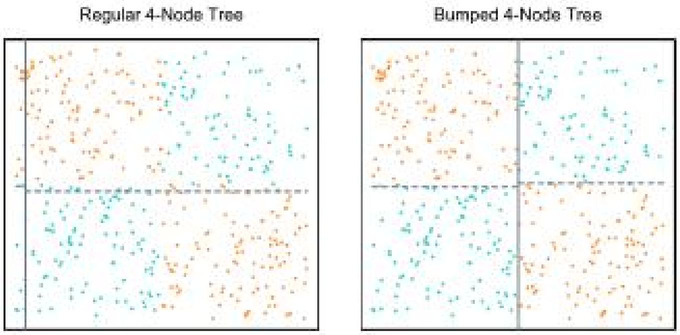


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm 20 times.*

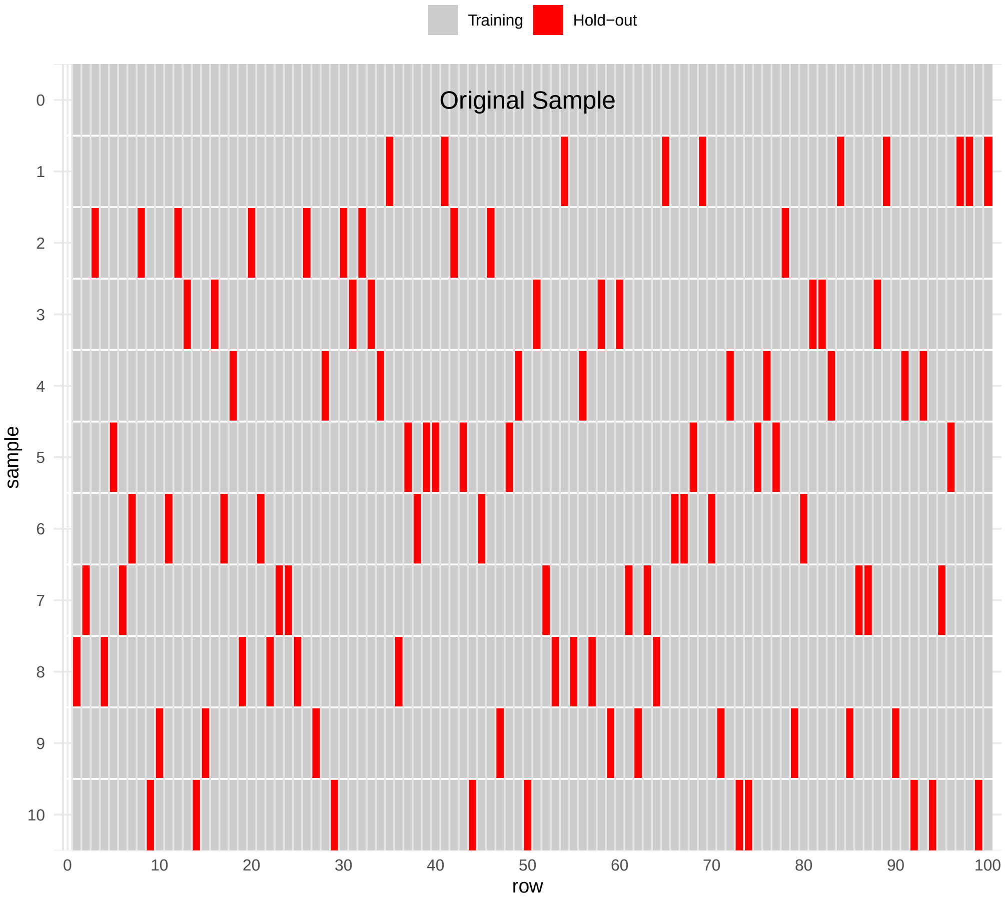
- **Cross-Validation Committee**: instead of using bootstrap samples, use cross-validation to make the different training sets

$$\hat{f}(x) = \frac{1}{M} \sum_{k=1}^{M} \hat{g}(x \mid D \setminus D_k)$$

- $D \setminus D_k$ are all the observations not included in the k^{th} fold.
  - In this notation, there are M folds
  - Special case of sub-bagging!
- The special case of leave-one-out:

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \hat{g}^{-i}(x)$$

- Same tuning parameters used for all base models



10-fold cross-validation sampling

Cross-Validation Committee Illustration

2 Model Averaging and Stacking

The basic idea of model averaging and (linear) stacking is simple to represent:

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x)$$

- The estimated weight $\hat{a}_k$ determines how much the final aggregate model is influenced by model $\hat{g}_k(x)$
- Bagging uses equal weights $a_k = 1/M$ (i.e., nothing is estimated) and the same base models (e.g., trees).

2.1 Model and Tuning Parameter Selection

Model selection is the approach of choosing the *single best model*. This also applies to choosing the best tuning parameter(s).

- In this setting, $\hat{a} = [\hat{a}_1, \ldots, \hat{a}_M]$ is *one-hot*
  - All $\hat{a}_k = 0$, except one is 1.
  - $\hat{a}_k \in \{0, 1\}$, $\sum_{k=1}^{M} \hat{a}_k = 1$

The best model is selected by: resampling (e.g., cross-validation, OOB error), AIC/BIC, GCV, LOO-CV, etc.

- We have done a type of *model selection* in choosing the optimal tuning parameters in ridge/lasso/mtry/k, etc.

Ensemble Motivation

We may be able to obtain better predictions if we combine all the models instead of just picking the best.

2.2 Bayesian Model Averaging

Let there be M candidate models.

- Assume one of the M models is correct (i.e., one of the models generated the data)
  - Let $\mathcal{M}$ denote the true model that generated the data
- Let $\pi_k = \Pr(\mathcal{M} = k)$ is the prior probability that model k is the true model
- $p(D \mid \mathcal{M} = k) = \int_\Theta p(D \mid \theta_k, \mathcal{M} = k) f(\theta_k \mid \mathcal{M} = k)\, d\theta_k$

The posterior probability of model k is

$$\Pr(\mathcal{M} = k \mid D) = \frac{p(D \mid \mathcal{M} = k) \cdot \pi_k}{\sum_m p(D \mid \mathcal{M} = m) \cdot \pi_m}$$

The best prediction (under a squared error loss) is

$$\begin{aligned} f^*(x) &= \mathrm{E}[Y \mid X = x, D] \\ &= \sum_{k=1}^{M} \Pr(\mathcal{M} = k \mid D) \cdot \mathrm{E}[Y \mid X = x, D, \mathcal{M} = k] \\ &= \sum_{k=1}^{M} a_k \cdot g_k(x) \end{aligned}$$

- $g_k(x) = \mathrm{E}[Y \mid X = x, D, \mathcal{M} = k]$ is the best prediction from model k.
- $a_k = \Pr(\mathcal{M} = k \mid D)$ is the posterior probability that model k is the correct model

Likelihood

In words, $P(D \mid \mathcal{M} = k)$ is the probability of observing data D given that the true model is k. However, most probability models have unknown model parameters θ. For example the Gaussian distribution has model parameters μ (the mean) and σ (the standard deviation).
The likelihood is a function of the model parameters:

$$L(\theta_k) = P(D \mid \theta_k,\, \mathcal{M} = k)$$

1. **Bayesian**. Specify a *prior* distribution $f_k(\theta)$ for the model parameters of model k.

$$\begin{aligned} p(D \mid \mathcal{M} = k) &= E_{f_k}[L(\theta)] \\ &= \int_{\Theta} L(\theta) f_k(\theta)\, d\theta \\ &= \int_{\Theta_k} p(D \mid \theta, \mathcal{M} = k) f_k(\theta)\, d\theta \end{aligned}$$

2. **Frequentist (Maximum Likelihood)**. Set θ to the value that maximizes the likelihood.

$$\begin{aligned} p(D \mid \hat{\theta}, \mathcal{M} = k) &= \arg\max_{\theta} L(\theta) \\ &= \arg\max_{\theta} p(D \mid \theta, \mathcal{M} = k) \end{aligned}$$

2.2.1 BIC/AIC

Recall that we considered AIC and BIC for model selection (along with cross-validation).

$$\begin{aligned} \mathrm{AIC}(k) &= -2 \log L(\hat{\theta}_k) + 2d(k) \\ \mathrm{BIC}(k) &= -2 \log L(\hat{\theta}_k) + \log n \cdot d(k) \end{aligned}$$

- $L(\hat{\theta}_k) = \max_{\theta \in \Theta_k} p(D \mid \theta, \mathcal{M} = k)$ is the maximized likelihood for model k
- $d(k)$ is the effective degrees of freedom for model k (under MLE)

It turns out that under certain settings (a bit beyond the scope of this course) that BIC is a good estimate of $-2 \log p(D \mid \mathcal{M} = k)$. Therefore,

$$\begin{aligned} \log p(D \mid \mathcal{M} = k) &\approx \underbrace{\log p(D \mid \hat{\theta}_k, \mathcal{M} = k) - \log n \cdot d(k)/2}_{-\frac{1}{2}\mathrm{BIC}(k)} \\ p(D \mid \mathcal{M} = k) &\approx e^{-\frac{1}{2}\mathrm{BIC}(k)} \end{aligned}$$

If we fill this into the posterior and set equal priors ($\pi_k = 1/M$) we get

$$\Pr(\mathcal{M} = k \mid D) = \frac{e^{-\frac{1}{2}\mathrm{BIC}(k)}}{\sum_m e^{-\frac{1}{2}\mathrm{BIC}(m)}}$$

Thus for any model where AIC/BIC can be calculated (i.e., there is a likelihood and estimated degrees of freedom) we can use the following ensemble:

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x)$$

- where $\hat{g}_k(x)$ is the prediction from model k
- And the weights are:

BIC Version

$$\hat{a}_k = \frac{e^{-\frac{1}{2}\text{BIC}(k)}}{\sum_m e^{-\frac{1}{2}\text{BIC}(m)}}$$

AIC Version

$$\hat{a}_k = \frac{e^{-\frac{1}{2}\text{AIC}(k)}}{\sum_m e^{-\frac{1}{2}\text{AIC}(m)}}$$

2.3 Linear Stacking

A *linear* stacking model combines base models as a weighted sum

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x)$$

- Strictly speaking, this is a bit more general that model averaging as the weights aren't constrained to sum to 1 or even be non-negative. (Although it is essentially the same idea.)
- Stacking is popular in prediction contests as it is a great way to combine models from teammates
- Notice that each model $\hat{g}_k(x)$ *and* the weights $\hat{a} = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_M)$ must be estimated.

Your Turn #1

1. In the *best subsets* and *step-wise* approaches, model $\hat{g}_k(x)$ is the best linear model with k predictors. What are the optimal weights if selected according to least squares (*using the training data*):

$$\hat{a} = \arg\min_{a \in \mathbf{R}^M} \sum_{i=1}^{n} (y_i - \sum_{k=1}^{M} a_k \hat{g}_k(x_i))^2$$

2. In lasso/ridge regression, model $\hat{g}_k(x)$ is the model corresponding to λ_k. What are the optimal weights if selected according to least squares (*using the training data*):

$$\hat{a} = \arg\min_{a \in \mathbf{R}^M} \sum_{i=1}^{n} (y_i - \sum_{k=1}^{M} a_k \hat{g}_k(x_i))^2$$

3. What would be a better way to select $\hat{a}$?

2.3.1 Linear Stacking (Single Hold-Out)

The main idea behind linear stacking is to find the weights using out-of-sample predictions.

Algorithm: Single Hold-out Stacking

1. Partition the data into a training and testing set $D = [D_{\text{train}}, D_{\text{test}}]$
2. Fit each model with the data from the training set and make predictions for the data in test set
   - Let $\hat{g}_k(x_i \mid D_{\text{train}})$ denote the prediction for *test* observation i using the *training* data D_{train}
3. The optimal weights are selected as:

$$\hat{a} = \arg\min_a \sum_{i \in D_{\text{test}}} L\left(y_i, \sum_{k=1}^{M} a_k \, \hat{g}_k(x_i \mid \mathcal{D}_{\text{train}})\right)$$

4. (optional) The final prediction is made by fitting each model with *all* the data

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x \mid D)$$

   - Note: the hold-out/test data is only used to estimate the weights

Because the prediction $\hat{y}_i | x_i$ is made from models that aren't trained with (x_i, y_i), the stacking weights are fairly adjusted for different model complexities.

- E.g., a model that is too complex (overfits) will not make good estimates on the hold-out data and hence should receive a low weight.

Stacking Features

Another way to view stacking is that each model creates a set of *new features* (feature engineering):

$$Z_{ik} = \hat{g}_k(x_i \mid D_{\text{train}})$$

- where $x_i \in D_{\text{test}}$.

and uses a simple model (e.g., linear regression or logistic regression) to estimate the weights:

$$\hat{a} = \arg\min_a \sum_{i \in D_{\text{test}}} L\left(y_i, \sum_{k=1}^{M} a_k \, Z_{ik}\right)$$

- E.g., using linear regression, $\hat{a} = (Z^{\mathsf{T}} Z)^{-1} Z^{\mathsf{T}} Y$
- Note: we could also use constrained optimization to force the weights to be non-negative and sum to one (model averaging).
- Non-linear stacking: treat the Z_k as new features in a non-linear model.
- Add in the original feature X_{test} as interactions if certain regions features space are better predicted by certain models.
  - i.e., weights vary over feature space.
  - trees may be good ensemble models for this if there is sufficient hold-out data.

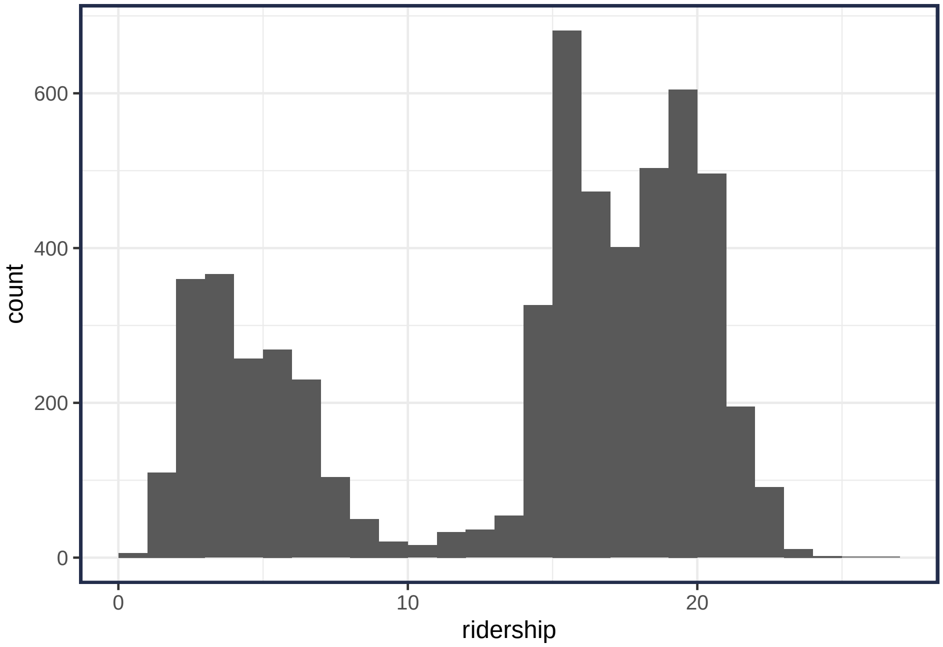
Hold-out Stacking Illustration

2.3.2 Linear Stacking Example

Implement a linear stacking model to predict ridership at a Chicago train station.

```r
library(modeldata)
library(tidymodels)
library(tidyverse)

data("Chicago", package = "modeldata") # load Chicago train ridership data
head(Chicago)
#> # A tibble: 6 x 50
#>   ridership Austin Quincy_Wells Belmont Archer_35th Oak_Park Western Clark_Lake
#>       <dbl>  <dbl>        <dbl>   <dbl>       <dbl>    <dbl>   <dbl>      <dbl>
#> 1     15.7   1.46         8.37     4.60       2.01     1.42     3.32      15.6
#> 2     15.8   1.50         8.35     4.72       2.09     1.43     3.34      15.7
#> 3     15.9   1.52         8.36     4.68       2.11     1.49     3.36      15.6
#> 4     15.9   1.49         7.85     4.77       2.17     1.44     3.36      15.7
#> 5     15.4   1.50         7.62     4.72       2.06     1.42     3.27      15.6
#> 6      2.42  0.693        0.911    2.27       0.624    0.426    1.11       2.41
#> # i 42 more variables: Clinton <dbl>, Merchandise_Mart <dbl>,
#> #   Irving_Park <dbl>, Washington_Wells <dbl>, Harlem <dbl>, Monroe <dbl>,
#> #   Polk <dbl>, Ashland <dbl>, Kedzie <dbl>, Addison <dbl>,
#> #   Jefferson_Park <dbl>, Montrose <dbl>, California <dbl>, temp_min <dbl>,
#> #   temp <dbl>, temp_max <dbl>, temp_change <dbl>, dew <dbl>, humidity <dbl>,
#> #   pressure <dbl>, pressure_change <dbl>, wind <dbl>, wind_max <dbl>,
#> #   gust <dbl>, gust_max <dbl>, percip <dbl>, percip_max <dbl>, ...
ggplot(Chicago, aes(ridership)) + geom_histogram(boundary = 0, binwidth = 1)
```



```
#: create train/test split
n_hold_out = 500
p_hold_out = n_hold_out / nrow(Chicago)
data_split = initial_split(Chicago, prop = 1 - p_hold_out)
```

```
#: model 1: RF
library(ranger)
set.seed(2023)
g1 = ranger(ridership ~ ., data = training(data_split))
Z1_rf = predict(g1, testing(data_split))$predictions
```

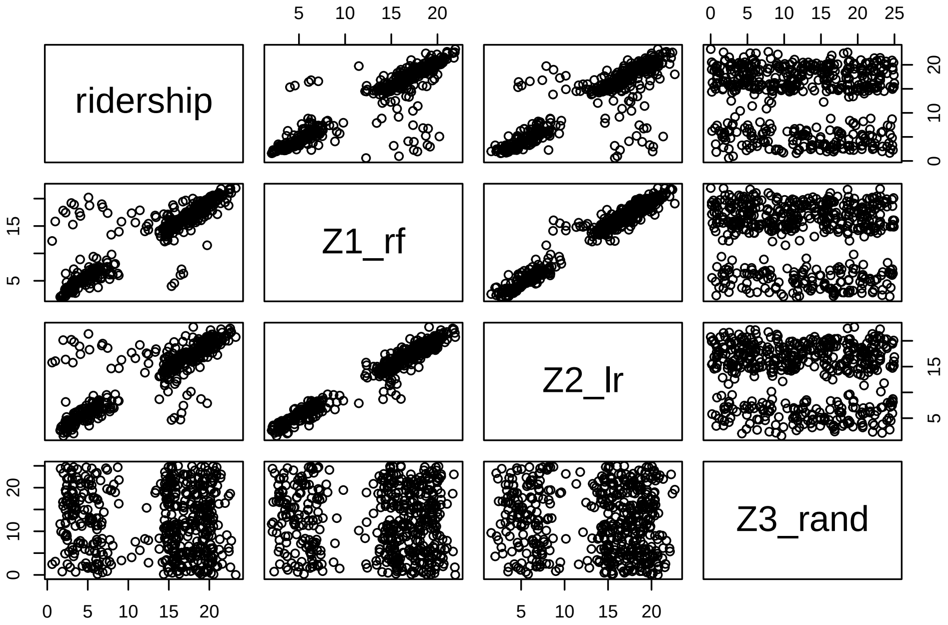
```
#: model 2: (penalized) linear regression. Use cv to select lambda.
library(glmnet)
X = makeX(  # one-hot encoding of categorical predictors
      train = training(data_split) %>% select(-ridership),
      test =  testing(data_split) %>% select(-ridership),
    )
set.seed(2023)
g2 = cv.glmnet(X$x, training(data_split)$ridership) # tune lambda with 10-fold cv
Z2_lr = predict(g2, X$xtest, s = "lambda.min")          # choose lambda.min
```

```
#: model 3: random model
set.seed(2023)
g3 <- function(data_test) runif(nrow(data_test), min = 0, max = 25)
Z3_rand = g3(testing(data_split))
```

```
#: linear regression stacking model
data_test = testing(data_split) %>% mutate(Z1_rf, Z2_lr, Z3_rand)
fit_stacking = lm(ridership ~ Z1_rf + Z2_lr + Z3_rand, data = data_test)
summary(fit_stacking)
#>
#> Call:
#> lm(formula = ridership ~ Z1_rf + Z2_lr + Z3_rand, data = data_test)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -15.793  -0.434   0.365   1.031  11.499
#>
```

```
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   0.1498     0.4109    0.36    0.716
#> Z1_rf         1.2249     0.1114   11.00   <2e-16 ***
#> Z2_lr        -0.2341     0.1119   -2.09    0.037 *
#> Z3_rand      -0.0111     0.0178   -0.62    0.533
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.88 on 496 degrees of freedom
#> Multiple R-squared:  0.806,  Adjusted R-squared:  0.805
#> F-statistic:  687 on 3 and 496 DF,  p-value: <2e-16
```

```
data_test %>% select(ridership, starts_with("Z")) %>% pairs()
```



```
#: final predictive base models
set.seed(2023)
g1_final = ranger(ridership ~ ., data = Chicago)
g2_final = cv.glmnet(do.call(rbind, X),
                     Chicago$ridership)
g3_final = g3

#: final predictive model (weighted sum of updated base models)
# Left as an exercise.
```

2.3.3 Linear Stacking (Cross-validated)

The main idea behind linear stacking is to find the weights using out-of-sample predictions.

Algorithm: Cross-Validated Stacking

1. Partition the data into V-folds $(D_1, D_2, \dots, D_V)$
2. Fit each model with the data from all folds except fold v and make predictions for the data in fold v
    - Repeat for all V folds

- Let $\hat{g}_k(x_i \mid D \setminus D_{v_i})$ denote the prediction for observation i using all the data except the data in the same fold as i (i.e., D_{v_i} is the data in the same fold as observation i)

3. The optimal weights are selected as:

$$\hat{a} = \arg\min_{a} \sum_{i=1}^{n} L\left(y_i, \sum_{k=1}^{M} a_k \, \hat{g}_k(x_i \mid D \setminus D_{v_i})\right)$$

4. (optional) The final prediction is made by fitting each model with *all* the data

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x \mid D)$$

- Note: cross-validation is only used to estimate the weights

Because the prediction $\hat{y}_i | x_i$ is made from models that aren't trained with (x_i, y_i), the stacking weights are fairly adjusted for different model complexities.

- E.g., a model that is too complex (overfits) will not make good estimates on the hold-out data and hence should receive a low weight

Stacking Features

Another way to view stacking is that each model creates a set of *new features* (feature engineering):

$$Z_{ik} = \hat{g}_k(x_i \mid D \setminus D_{v_i})$$

and uses a simple model (e.g., linear regression or logistic regression) to estimate the weights:

$$\hat{a} = \arg\min_{w} \sum_{i=1}^{n} L\left(y_i, \sum_{k=1}^{M} w_k \, Z_{ik}\right)$$

- E.g., using linear regression, $\hat{a} = (Z^{\mathsf{T}} Z)^{-1} Z^{\mathsf{T}} Y$
- Note: we could also use constrained optimization to force the weights to be non-negative and sum to one (model averaging)

Cross-Validation Stacking Illustration

2.4 Non-Linear Stacking

You are not limited to using a *linear* stacking model. But could potentially use any model (e.g., trees, knn) as the stacking model. Just pay attention to how much data is being used to estimate the stacking model parameters. If the stacking model is *too complex*, the high variance can offset good performance from the base models. This is the main reason why linear stacking models are most popular.

A *linear* stacking model combines base models as a weighted sum

$$\hat{f}(x) = \sum_{k=1}^{M} \hat{a}_k \hat{g}_k(x)$$

but a *non-linear* stacking model

$$\hat{f}(x) = \mathcal{F}(\hat{g}_1(x), \hat{g}_2(x), \ldots, \hat{g}_M(x), X_1, X_2, \ldots, X_p)$$

uses a non-linear combination of base model predictions, and optionally, the original predictor variables.

1. Split the data into training and hold-out/stacking sets (can use cross-validation too).
2. Fit M base models to *training data*.
3. Make predictions from each model on the *hold-out/stacking data*. Call these predictions $Z_1, Z_2, \ldots, Z_M$.
4. Fit and save a *non-linear* stacking model on the *hold-out/stacking data* using $Z_1, Z_2, \ldots, Z_M$ (and optionally $X_1, X_2, \ldots, X_p$) as the predictor variables.
5. Update the stacking predictor variables $Z_1, Z_2, \ldots, Z_M$ using all the available data.

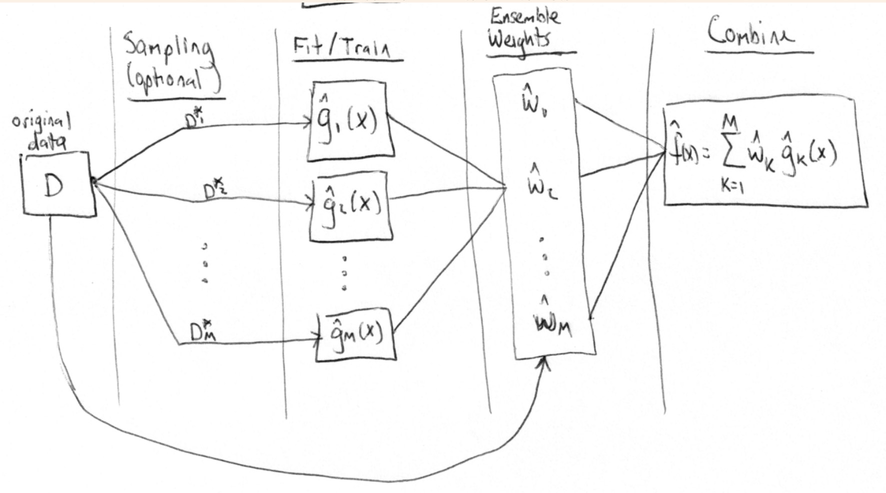
3 Ensemble Models

3.1 Boosting Preview

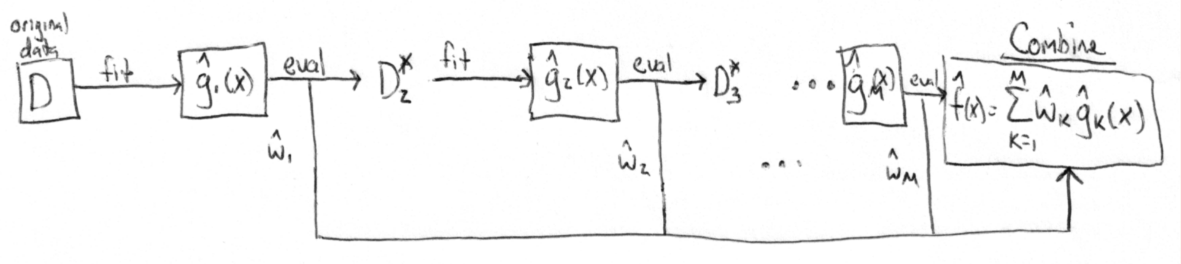
So far, we have focused on fitting the base models in *parallel*. In boosting, the base models are fit *sequentially*.

Sequential vs. Parallel ensembles

Parallel Ensembles



Boosting (Sequential Ensembles)



The general idea of *boosting* is to fit models sequentially, where each model depends on the combination of previous models.

There are two primary approaches:

1. Gradient Boosting: Sequentially fit models to the (pseudo) residuals, where the residuals are larger for observations that are poorly predicted.
2. AdaBoost: Sequentially fit to re-weighted data, where the weights are larger for observations that are poorly predicted.

Boosting is primarily a *bias* reducer

- The base models are often simple/weak (low variance, but high bias) models (like shallow trees)

3.2 Constructing Ensemble Models

Ensemble methods differ on (i) which base models are included and (ii) how the base models are combined to form a final prediction.

Here are a few thoughts on different ensemble configurations

- Think about how these impact the overall bias and variance (including model correlation) trade-off
- Some of these ideas were taken from: Dietterich T.G. (2000) Ensemble Methods in Machine Learning. In: Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science, vol 1857. Springer, Berlin, Heidelberg

3.2.1 Fitting Base Learners

1. Use same base learners (with different data/initialization) or different base learners.
   - Bagging and RF uses the same base learners, but fit with different (bootstraped) data
   - Better predictions may be achieved by using very different base learners (e.g., random forest, xgboost, GAM, ANN)

2. Use different data to train each model.
   - Bagging/RF uses bootstrap data to build different models
   - Boosting sequentially uses re-weighted or modified/residuals data

3. Use different sets of features to train each model.
   - RF randomly selects a sub-set of features for making each split
   - Has the potential to decrease correlation between base learners

4. Use different transformations of outcome variable to build models.
   - E.g., fit models to y and also $y' = \log y$ (and then backtransform)
   - E.g., one-vs-rest for classification
   - Gradient Boosting sequentially fits models to current residuals

5. Use randomness in model fitting.
   - use different initializations
   - set different random seeds
   - RF uses random subset of features for each split
   - Average multiple stochastic models (of same family/tuning) from different seeds

3.2.2 Combining Models/Predictors

The base models can be combined in many different ways

1. Weighted sum/average
   - Model Averaging
   - Stacking

2. Choose the best one
   - Model Selection
   - (all weights are zero except 1 weight is one)

3. Use the median prediction (Bragging)
4. Parallel vs. Sequential
   - Bagging is parallel
   - Boosting is sequential