Module 8: Ensembles

Live Session | SYS 6018 | Spring 2021

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:
  - Regression: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where $y_i \in \mathbf{R}$
  - Classification: $D = \{(x_1, g_1), (x_2, g_2), \ldots, (x_n, g_n)\}$ where $g_i \in \mathcal{G}$
  - There are n observations
- 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

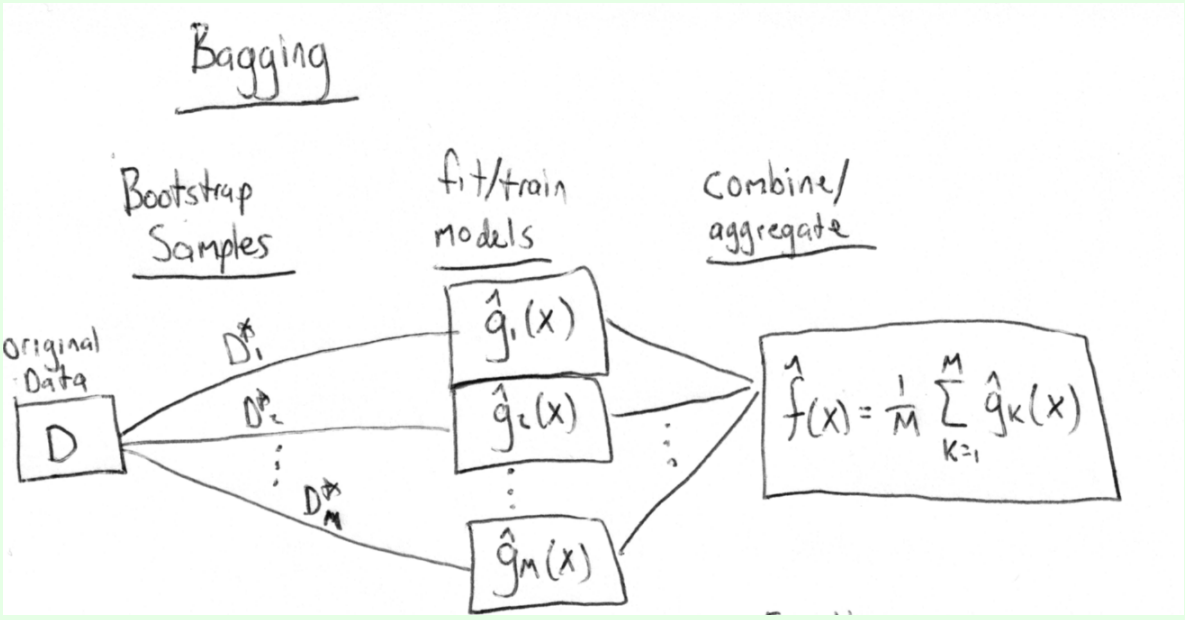
Bagging fits the *same base model* to *bootstrap samples* of the observed data and *averages* the predictions from each model.

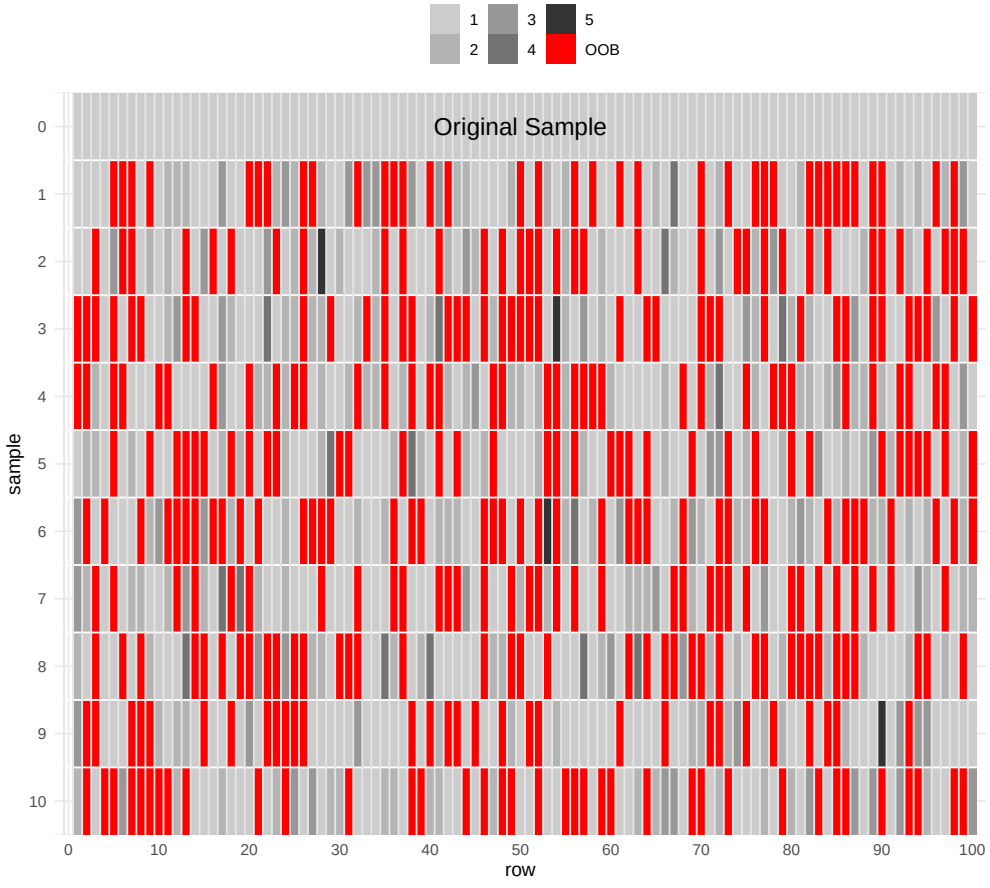
- The base model, $g(x)$, is usually a tree
- The predictions from the base models are averaged:

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

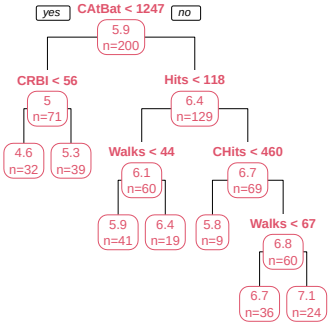
- M is the number of bootstrap samples (and thus base models)
- $\hat{g}(x \mid D_k^*)$ is the model fit to the k^{th} bootstrap sample

Bagging Illustration

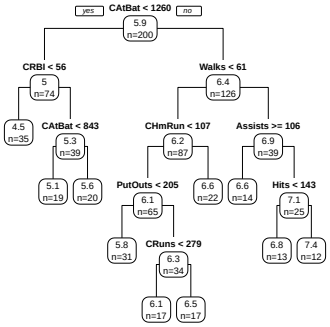




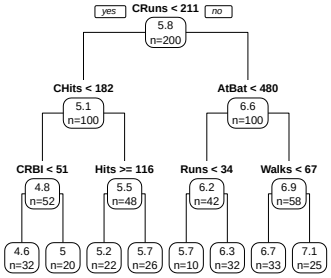
Original Tree



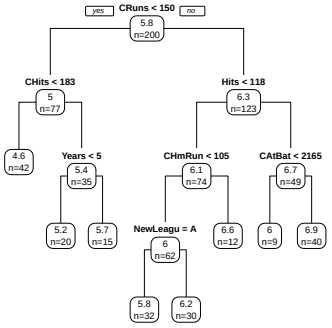
Bootstrap Tree: 1



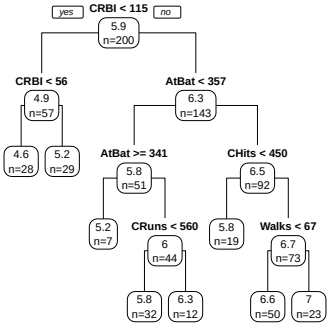
Bootstrap Tree: 2



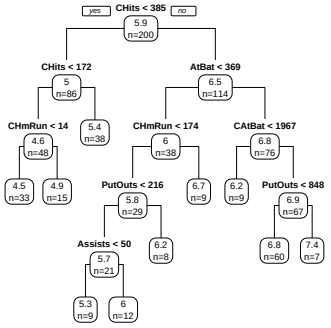
Bootstrap Tree: 3



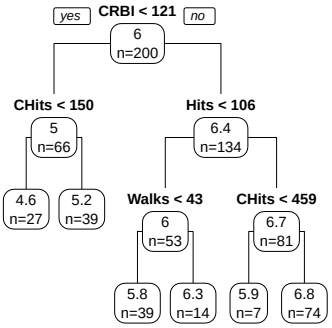
Bootstrap Tree: 4



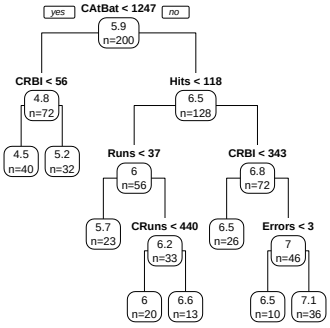
Bootstrap Tree: 5



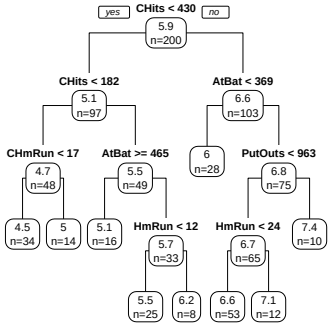
Bootstrap Tree: 6



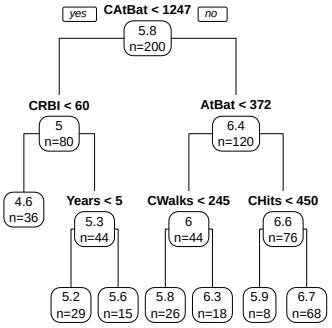
Bootstrap Tree: 7



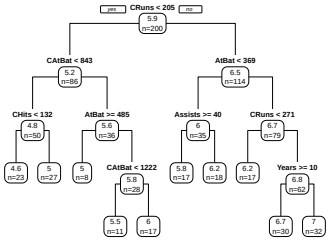
Bootstrap Tree: 8



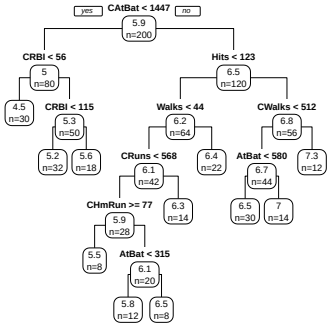
Bootstrap Tree: 9



Bootstrap Tree: 10



Bootstrap Tree: 11



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
- **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
- The special case of leave-one-out:

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

- **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 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

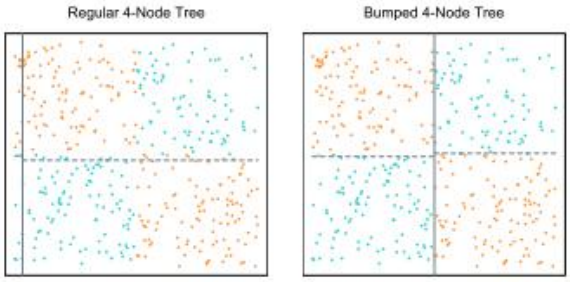


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two 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.*

2 Model Averaging and Stacking

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

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

- The estimated weight $\hat{w}_k$ determines how much the final aggregate model is influenced by model $\hat{g}_k(x)$

2.1 Model Selection

Model selection is the approach of choosing the *single best model*.

- In this setting, all $\hat{w}_k = 0$, except one is 1.
  - E.g., $\hat{w}_k \in \{0, 1\}$, $\sum_{k=1}^{M} \hat{w}_k = 1$

The best model is selected by: cross-validation (and repeated train/test), AIC/BIC, GCV, LOO-CV, OOB error, etc.

- We have done a type of *model selection* in choosing the optimal tuning parameters in ridge/lasso/best subsets.

Ensemble Motivation

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

2.2 Model Averaging

Let there be M candidate models.

- Assume one of the M models is correct (i.e., one model 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

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

2.2.1 BIC/AIC

AIC and BIC are popular approaches to model selection (along with cross-validation).

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

- $L_k(\hat{\theta}_k) = \max_{\theta \in \Theta_k} p(D \mid \theta, \mathcal{M} = k)$ 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 $-2 \log p(D \mid \mathcal{M} = k)$

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

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

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:

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

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

BIC Version

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

AIC Version

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

2.3 Stacking

A stacking model combines base models as a weighted sum

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

- 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 $\hat{w} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_M)$ 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*):

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - \sum_{k=1}^{M} w_k \hat{g}_k(x_i))^2$$

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*):

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - \sum_{k=1}^{M} w_k \hat{g}_k(x_i))^2$$

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

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

Algorithm: Stacking

1. Partition the data into V-folds $(D_1, D_2, \ldots, D_V)$
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 expect 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:

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

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

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

   - 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:

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

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

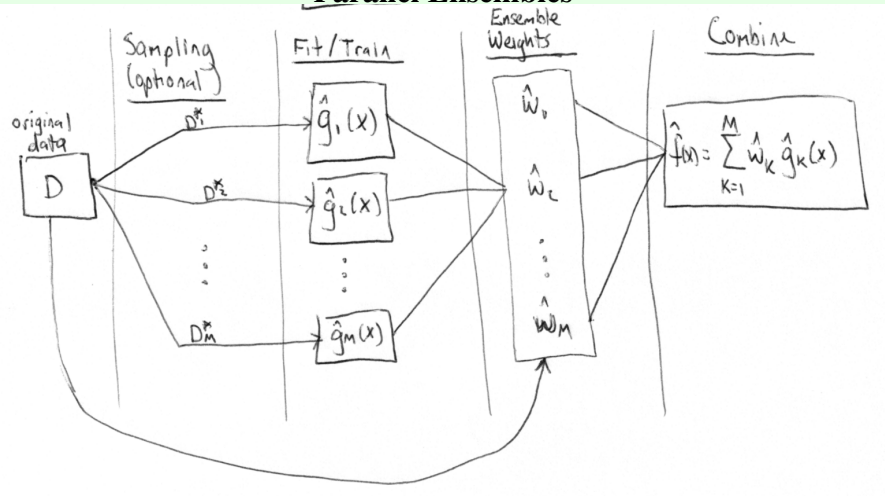
3 Ensemble Models

3.1 Boosting

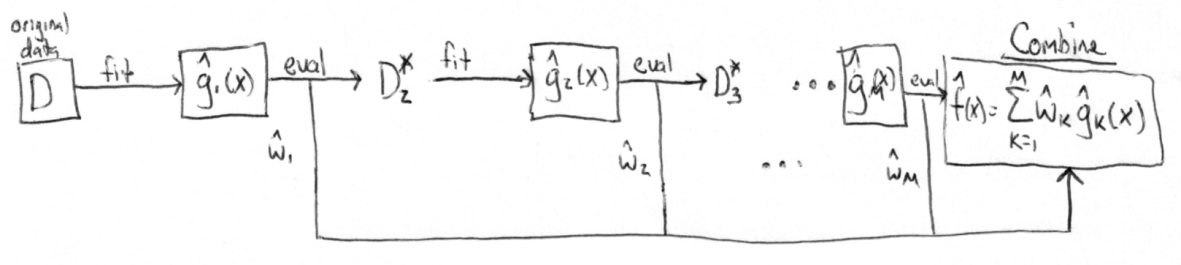
So far, we have focused on fitting the base models in *parallel*. In boosting, the base models are fit *sequentially*.

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
    - Slightly better prediction (e.g., for prediction contests) 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.
    - You can think of Naive Bayes and Linear/Logistic Regression as an ensemble that uses base models fit to a single feature
    - 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 responses 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
    - RF uses random subset of features for each split

3.2.2 Combining Models/Predictors

The base models can be combined in many different ways

1. Weighted sum/average
    - Model Averaging
    - Stacking
2. Use the median prediction
    - Bragging
3. Choose the best one
    - Model Selection
4. Parallel or Sequential?
    - Bagging is parallel
    - Boosting is sequential

4 Popular Boosting Implementations

4.1 GBM (Gradient Boosting Machine)

- R package `gbm`
- GBM Documentation

4.1.1 Model/Tree Tuning Parameters

- Tree depth (`interaction.depth`)
  - Grows trees to a depth specified by `interaction.depth` (unless there are not enough observations in the terminal nodes)
- Minimum number of observations allowed in the terminal nodes (`n.minobsinnode`)
- Sub-sampling (`bag.fraction`)
  - *Stochastic Gradient Boosting*
  - Sample (without replacement) at each iteration
- Loss Function (`distribution`)
  - The loss function is determined by the `distribution` argument
  - Use `distribution="gaussian"` for squared error
  - Other options are: `bernoulli` (for logistic regression), `poisson` (for Poisson regression), `pairwise` (for ranking/LambdaMart), `adaboost` (for the adaboost exponential loss), etc.

4.1.2 Boosting Tuning Parameters

- Number of iterations/trees (`n.trees`)
  - Use cross-validation (or out-of-bag) to find optimal value
  - Can use the helper function `gbm.perf()` to get the optimal value
- Shrinkage parameter (`shrinkage`)
  - Set small, but the smaller the `shrinkage`, the more iterations/trees need to be used
  - "Ranges from 0.001 to 0.100 usually work"
- Cross-validation (`cv.folds`)
  - `gbm` has a built in cross-validation
  - no way to manually set the folds

4.1.3 Computational Settings

- Number of Cores (`n.cores`)
  - Only used when cross-validation is implemented

4.2 xgboost (Extreme Gradient Boosting)

- R package `xgboost`
- xgboost Documentation
- xbgoost Paper

4.2.1 Model/Tree Tuning Parameters

- Different base leaners (`booster`)
  - `gbtree` is a tree
  - `gblinear` creates a (generalized) liner model (forward stagewise linear model)
- Tree building (`tree_method`)

  - To speed up the fitting, only consider making splits at certain quantiles of the input vector (rather than considering every unique value)
- Sub-sampling (`subsample`)
  - *Stochastic Gradient Boosting*
  - Sample (without replacement) at each iteration
- Feature sampling (`colsample_bytree`, `colsample_bylevel`, `colsample_bynode`)
  - Like used in Random Forest, the features/columns are subsampled
  - Can use a subsample of features for each tree, level, or node

Model Complexity Parameters

- Tree depth (`max_depth`)
  - Grows trees to a depth specified by `max_depth` (unless there are not enough observations in the terminal nodes)
  - Trees may not reach `max_depth` if the `gamma` or `min_child_weight` arguments are set.
- Minimum number of observations (or sum of weights) allowed in the terminal nodes (`min_child_weight`)
- Pruning (`gamma` or `min_split_loss`)
  - Minimum loss reduction required to make a further partition on a leaf node of the tree
  - The larger gamma is, the more conservative the algorithm will be
- ElasticNet type penalty (`lambda` and `alpha`)
  - `lambda` is an L_2 penalty
  - `alpha` is an L_1 penalty

- Recall that trees model the response as a *constant* in each region

$$\hat{f}_T(x) = \sum_{m=1}^{M} \hat{c}_m \, \mathbb{1}(x \in \hat{R}_m)$$

- Cost-complexity pruning found the optimal tree as the one that minimized the penalized loss objective function:

$$C_\gamma(T) = \sum_{m=1}^{|T|} \text{Loss}(T) + \gamma |T|$$

- XGBoost selects a tree at each iteration using the following penalized loss:

$$C_{\gamma,\lambda,\alpha}(T) = \sum_{m=1}^{|T|} \text{Loss}(T) + \gamma |T| + \frac{\lambda}{2} \sum_{m=1}^{|T|} \hat{c}_m^2 + \alpha \sum_{m=1}^{|T|} |\hat{c}_m|$$

- Loss Function (`objective`)
  - The loss function is determined by the `objective` argument
  - Use `reg:squarederror` for squared error
  - Other options are: `reg:logistic` or `binary:logistic` (for logistic regression), `count:poisson` (for Poisson regression), `rank:pairwise` (for ranking/LambdaMart), etc.

4.2.2 Boosting Tuning Parameters

- Shrinkage parameter (`eta` or `learning_rate`)
  - Set small, but the smaller the `eta`, the more iterations/trees need to be used
- Number of iterations/trees (`num_rounds`)
  - Use cross-validation (or out-of-bag) to find optimal value

- Cross-validation (`xgb.cv`)
  - `xgboost` has a built in cross-validation
  - It is possible to manually set the folds

4.2.3 Computational Settings

- Number of Threads (`nthread`)
- GPU Support (https://xgboost.readthedocs.io/en/latest/gpu/index.html)
  - Used for finding tree split points and evaluating/calculating the loss function

4.3 CatBoost

- R package: (https://github.com/catboost/catboost/tree/master/catboost/R-package)
- CatBoost Documentation

4.4 LightGBM

- R Package: https://github.com/microsoft/LightGBM/tree/master/R-package
- LightGBM Documentation