Ensembling Methods

We now cover methods by which we can aggregate the output of trained models. We will use Bias-Variance analysis as well as the example of decision trees to probe some of the trade-offs of each of these methods.

To understand why we can derive benefit from ensembling, let us first recall some basic probability theory. Say we have n independent, identically distributed (i.i.d.) random variables X_i for $0 \leq i < n$. Assume $\mathrm{Var}(X_i) = \sigma^2$ for all X_i. Then we have that the variance of the mean is:

$$\mathrm{Var}(\bar{X}) = \mathrm{Var}(\frac{1}{n}\sum_i X_i) = \frac{\sigma^2}{n}$$

Now, if we drop the independence assumption (so the variables are only i.d.), and instead say that the X_i's are correlated by a factor ρ, we can show that:

$$\begin{aligned}
\mathrm{Var}(\bar{X}) =& \mathrm{Var}(\frac{1}{n}\sum_i X_i) && (1)\\
=& \frac{1}{n^2}\sum_{i,j} \mathrm{Cov}(X_i, X_j) && (2)\\
=& \frac{n\sigma^2}{n^2} + \frac{n(n-1)\rho\sigma^2}{n^2} && (3)\\
=& \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2 && (4)
\end{aligned}$$

Where in Step 3 we use the definition of pearson correlation coefficient $\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_x \sigma_y}$ and that $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$.

Now, if we consider each random variable to be the error of a given model, we can see that both increasing the number of models used (causing the second term to vanish) as well as decreasing the correlation between models (causing the first term to vanish and returning us to the i.i.d. definition) leads to an overall decrease in variance of the error of the ensemble.

There are several ways by which we can generate de-correlated models, including:

- Using different algorithms
- Using different training sets
- Bagging Tries to approximate having different training sets. Ex: Random Forests (bagging for decision trees)
- Boosting Ex: Adaboost, xgboost (these are variances of boosting for decision trees)

While the first two are fairly straightforward, they involve large amounts of additional work. In the following sections, we will cover the latter two techniques, boosting and bagging, as well as their specific uses in the context of decision trees.

In statistics, bootstrap is handy when there is no analytical form or normal theory to help estimate the sampling distribution of the statistic of interest (say the ratio of variance and mean — remember that many traditional statistic methods like Hypothesis Testing use the statistic mean or variance, and rely on normality or the Central Limit for normality)

Using bootstrapping one get an “empirical bootstrap distribution” of the statistic of interest and one can derive the “bootstrap confidence interval” for the purposes of hypothesis testing.

1 Bagging

1.1 Boostrap

Bagging stands for ”Boostrap Aggregation” and is a **variance reduction** ensembling method. **Bootstrap** is a method from statistics traditionally used to measure uncertainty of some estimator (e.g. mean).

Say we have a true population P that we wish to compute an estimator for, as well a training set S sampled from P ($S \sim P$). While we can find an approximation by computing the estimator on S, we cannot know what the error is with respect to the true value. To do so we would need multiple independent training sets S_1, S_2, ... all sampled from P.

However, if we make the assumption that $S = P$, we can generate a new bootstrap set Z sampled with replacement from S ($Z \sim S$, $|Z| = |S|$). In fact we can generate many such samples $Z_1, Z_2, ..., Z_M$. We can then look at the variability of our estimate across these bootstrap sets to obtain a measure of error.

1.2 Aggregation

Now, returning to ensembling, we can take each Z_m and train a machine learning model G_m on each, and define a new **aggregate predictor**:

$$G(X) = \sum_m \frac{G_m(x)}{M}$$

This process is called **bagging**. Referring back to equation (4), we have that the variance of M correlated predictors is:

$$Var(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2 \leq \rho\sigma^2 + (1-\rho)\sigma^2 = \sigma^2$$

Bagging creates less correlated predictors than if they were all simply trained on S, thereby decreasing ρ. While the bias of each individual predictor increases due to each bootstrap set not having the full training set available, in practice it has been found that Text the decrease in variance outweighs the increase in bias. Also note that increasing the number of predictors M can't lead to additional overfitting, as ρ is insensitive to M and therefore overall variance can only decrease.

And sigma^2 is variance of any given one learning algorithm so bagging (boosting+aggregation) reduces variance of the model by averaging.

An additional advantage of bagging is called **out-of-bag estimation**. It can be shown that each bootstrapped sample only contains approximately $\frac{2}{3}$ of S, and thus we can use the other $\frac{1}{3}$ as an estimate of error, called out-of-bag error. In the limit, as $M \to \infty$, out-of-bag error gives an equivalent result to leave-one-out cross-validation.

Bagging is ideal to fight high variance, especially if the learning algorithm (random variable) is high variance and low bias (bagging will decrease variance and increase bias)

1.3 Bagging + Decision Trees

Recall that fully-grown decision trees are high variance, low bias models, and therefore the variance-reducing effects of bagging work well in conjunction with them. Bagging also allows for handling of missing features: if a feature is missing, exclude trees in the ensemble that use that feature in our of their splits. Though if certain features are particularly powerful predictors they may still be included in most if not all trees.

A downside to bagged trees is that we lose the interpretability inherent in the single decision tree. One method by which to re-gain some amount of insight is through a technique called **variable importance measure**. For each feature, find each split that uses it in the ensemble and average the decrease in loss across all such splits. Note that this is not the same as measuring how much performance would degrade if we did not have this feature, as other features might be correlated and could substitute.

A final but important aspect of bagged decision trees to cover is the method of **random forests**. If our dataset contained one very strong predictor, then our bagged trees would always use that feature in their splits and end up correlated. With random forests, we instead only allow a subset of features to be used at each split. By doing so, we achieve a decrease in correlation ρ which leads to a decrease in variance. Again, there is also an increase in bias due to the restriction of the feature space, but as with vanilla bagged decision trees this proves to not often be an issue. Finally, even powerful predictors will no longer be present in every tree (assuming sufficient number of trees and sufficient restriction of features at each split), allowing for more graceful handling of missing predictors.

1.4 Recap

To summarize, some of the primary benefits of bagging, in the context of decision trees, are:

+ Decrease in variance (even more so for random forests)
+ Better accuracy
+ Free validation set
+ Support for missing values

While some of the disadvantages include:

– Incrase in bias (even more so for random forests)

– Harder to interpret

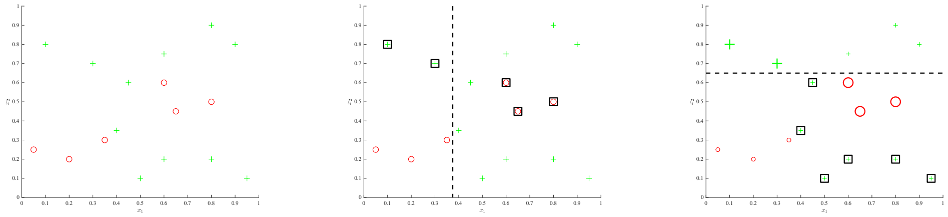
– Still not additive

– More expensive

2 Boosting

2.1 Intuition

Bagging is a variance-reducing technique, whereas boosting is used for **bias-reduction**. We therefore want high bias, low variance models, also known as **weak learners**. Continuing our exploration via the use of decision trees, we can make them into weak learners by allowing each tree to only make one decision before making a prediction; these are known as **decision stumps**.



We explore the intuition behind boosting via the example above. We start with a dataset on the left, and allow a single decision stump to be trained, as seen in the middle panel. The key idea is that we then track which examples the classifier got wrong, and increase their relative weight compared to the correctly classified examples. We then train a new decision stump which will be more incentivized to correctly classify these "hard negatives." We continue as such, incrementally re-weighting examples at each step, and at the end we output a combination of these weak learners as an ensemble classifier.

2.2 Adaboost

Having covered the intuition, let us look at one of the most popular boosting algorithms, **Adaboost**, reproduced below:

Algorithm 0: Adaboost

```
Input: Labeled training data $(x_1, y_1)$, $(x_2, y_2)$, ... $(x_N, y_N)$
Output: Ensemble classifer $f(x)$
$w_i \leftarrow \frac{1}{N}$ for $i = 1, 2..., N$
for $m = 0$ to $M$ do
    Fit weak classifier $G_m$ to training data weighted by $w_i$
    Compute weighted error $err_m = \frac{\sum_i w_i \mathbb{1}(y_i \neq G_m(x_i))}{\sum w_i}$
    Compute weight $\alpha_m = \log(\frac{1-err_m}{err_m})$
    $w_i \leftarrow w_i * \exp(\alpha_m \mathbb{1}(y_i \neq G_m(x_i)))$
end
$f(x) = \text{sign}(\sum_m \alpha_m G_m(x))$
```

https://mccormickml.com/2013/12/13/adaboost-tutorial/, based on original paper

The weightings for each example begin out even, with misclassified examples being further up-weighted at each step, in a cumulative fashion. The final aggregate classifier is a summation of all the weak learners, weighted by the negative log-odds of the weighted error.

We can also see that due to the final summation, this ensembling method allows for modeling of additive terms, increasing the overall modeling capability (and variance) of the final model. Each new weak learner is no longer independent of the previous models in the sequence, meaning that increasing M leads to an increase in the risk of overfitting.

The exact weightings used for Adaboost appear to be somewhat arbitrary at first glance, but can be shown to be well justified. We shall approach this in the next section through a more general framework of which Adaboost is a special case.

2.3 Forward Stagewise Additive Modeling

The **Forward Stagewise Additive Modeling** algorithm reproduced below is a framework for ensembling :

Algorithm 1: Forward Stagewise Additive Modeling

```
Input: Labeled training data $(x_1, y_1)$, $(x_2, y_2)$, ... $(x_N, y_N)$
Output: Ensemble classifer $f(x)$
Initialize $f_0(x) = 0$
for $m = 1$ to $M$ do
    Compute $(\beta_m, \gamma_m) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta G(x_i; \gamma))$
    Set $f_m(x) = f_{m-1}(x) + \beta_m G(x; y_i)$
end
$f(x) = f_m(x)$
```

Close inspection reveals that few assumptions are made about the learning problem at hand, the only major ones being the additive nature of the ensembling as well as the fixing of all previous weightings and parameters after a given step. We again have weak classifiers $G(x)$, though this time we explicitly parameterize them by their parameters γ. At each step we are trying to find the next weak learner's parameters and weighting so to best match the remaining error of the current ensemble.

As a concrete implementation of this algorithm, using a squared loss would be the same as fitting individual classifiers to the residual $y_i - f_{m-1}(x_i)$. Furthermore, it can be shown that Adaboost is a special case of this formulation, specifically for 2-class classification and exponential loss:

$$L(y, \hat{y}) = \exp(-y\hat{y})$$

For further details regarding the connection between Adaboost and Forward Stagewise Additive Modeling, the interested reader is referred to 10.4 Elements of Statistical Learning.

2.4 Gradient Boosting

In general, it is not always easy to write out a closed-form solution to the minimization problem presented in Forward Stagewise Additive Modeling. High-performing methods such as **xgboost** resolve this issue by turning to numerical optimization.

One of the most obvious things to do in this case would be to take the derivative of the loss and perform gradient descent. However, the complication is that we are restricted to taking steps in our model class – we can only add in parameterized weak learners $G(x, \gamma)$, not make arbitrary moves in the input space.

In **gradient boosting**, we instead compute the gradient at each training point with respect to the current predictor (typically a decision stump):

$$g_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}$$

We then train a new regression predictor to match this gradient and use it as the gradient step. In Forward Stagewise Additive Modeling, this works out to:

$$\gamma_i = \text{argmin}_\gamma \sum_{i=1}^{N} (g_i - G(x_i; \gamma))^2$$

2.5 Recap

To summarize, some of the primary benefits of boosting are:

+ Decrease in bias

+ Better accuracy

+ Additive modeling

While some of the disadvantages include:

– Increase in variance

– Prone to overfitting

For more on the theory behind boosting, John Duchi's excellent supplemental lecture notes are recommended.

Decision trees

RF is based on decision trees. In machine learning decision trees are a technique for creating predictive models. They are called decision **trees** because the prediction follows several branches of "if… then…" decision splits - similar to the branches of a tree. If we imagine that we start with a sample, which we want to predict a class for, we would start at the bottom of a tree and travel up the trunk until we come to the first split-off branch. This split can be thought of as a feature in machine learning, let's say it would be "age"; we would now make a decision about which branch to follow: "if our sample has an age bigger than 30, continue along the left branch, else continue along the right branch". This we would do until we come to the next branch and repeat the same decision process until there are no more branches before us. This endpoint is called a leaf and in decision trees would represent the final result: a predicted class or value.

At each branch, the feature thresholds that best split the (remaining) samples locally is found. The most common metrics for defining the "best split" are **gini impurity** and **information gain** for classification tasks and **variance reduction** for regression.

Single decision trees are very easy to visualize and understand because they follow a method of decision-making that is very similar to how we humans make decisions: with a chain of simple rules. However, they are not very robust, i.e. they don't generalize well to unseen samples. Here is where Random Forests come into play.

Ensemble learning

RF makes predictions by combining the results from many individual decision trees - so we cal them a **forest** of decision trees. Because RF combines multiple models, it falls under the category of ensemble learning. Other ensemble learning methods are gradient boosting and stacked ensembles.

Combining decision trees

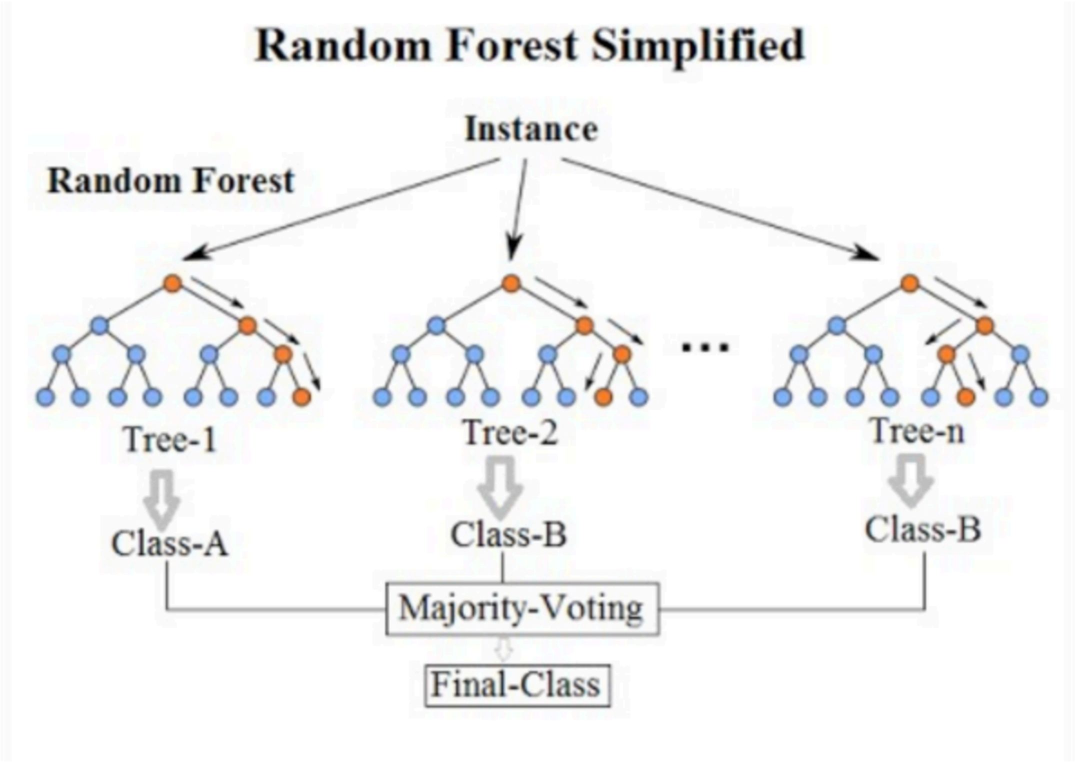
There are two main ways for combining the outputs of multiple decision trees into a random forest:

1. Bagging, which is also called Bootstrap aggregation (used in Random Forests)
2. Boosting (used in Gradient Boosting Machines)

Bagging works the following way: decision trees are trained on randomly sampled subsets of the data, while sampling is being done with replacement. Bagging is the default method used with Random Forests. A big advantage of bagging over individual trees is that it decrease the variance of the model. Individual trees are very prone to overfitting and are very sensitive to noise in the data. As long as our individual trees are not correlated, combining them with bagging will make them more robust without increasing the bias. The part about correlation is important, though! We remove (most of) the correlation by randomly sampling subsets of data and training the different decision trees on this subsets instead of on the entire dataset. In addition to randomly sampling instances from our data, RF also uses **feature bagging**. With feature bagging, at each split in the decision tree only a random subset of features is considered. This technique reduces correlation even more because it helps reduce the impact of very strong predictor variables (i.e. features that have a very strong influence on predicting the target or response variable).

bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

The final result of our model is calculated by averaging over all predictions from these sampled trees or by majority vote.


Random Forest Simplified
Instance
Random Forest
...
Tree-1
Tree-2
Tree-n
Class-A
Class-B
Class-B
Majority-Voting
Final-Class