

Lecture 6: Random Forests

Instructor: Justin J. Boutilier

===

In this lecture, we will introduce random forests. We will learn:

1. How random forests improve upon CART
2. How to use a random forests for explanation and prediction
3. How to train a random forest

*Corrections provided by Sarah Abdellatif

Random Forests

Random forests (RF) is a type of *ensemble learner*. The general idea is to train many CART models and combine their individual predictions by computing the average (for regression) or by computing the proportion of 1s / majority vote (for classification).

RF was first invented in 1995 by Dr. Tin Kam Ho – a researcher from Bell Labs (who is now at IBM Watson). It was extended by Prof. Leo Breiman in 2001 to the version we use today. Fun fact: Prof. Breiman was able to trade-mark the name *Random Forests* and that trademark is now owned by Minitab (a statistical software package).

Similar to CART, random forests can be used for both classification and regression problems. Although random forests were designed for prediction problems, they can also provide measures of feature importance that may be used for explanation problems (more on this later!).

Motivation

The motivation for developing random forests was to address the key limitation of CART models: the tendency to overfit the training data. As noted in Lecture 5, CART can severely overfit the training data leading to poor out-of-sample prediction accuracy (also called poor *generalizability*). Recall from Lecture 1 that overfitting is a result of high variance and intuitively, variance measures the sensitivity to a particular choice of dataset. In other words, CART is highly sensitive to the training data. To understand how random forest is able to reduce overfitting, we need to introduce the following concept.

Bootstrap aggregation

Bootstrap aggregation or *bagging* is a type of ensemble algorithm that helps to reduce model variance (and therefore overfitting). Along with CART and RF, bagging was invented by Prof. Breiman. All three (landmark) papers were solo-authored and have over 100,000 combined citations.

The bagging algorithm relies on sampling with replacement (also called bootstrapping). Given a training set of n observations denoted \mathcal{T} , we create M new training sets, $\mathcal{T}_1, \dots, \mathcal{T}_M$ by randomly sampling n observations from \mathcal{T} *with replacement*. Practically, this means that $\mathcal{T}_1, \dots, \mathcal{T}_M$ may have repeated observations. Note that each $\mathcal{T}_1, \dots, \mathcal{T}_M$ is the same size as the original training set \mathcal{T} . We can prove mathematically that for large n the proportion of unique observations in each $\mathcal{T}_1, \dots, \mathcal{T}_M$ is $1 - \frac{1}{e} \approx 63\%$. That means 37% of the observations are duplicates! We then train M CART models, one for each $\mathcal{T}_1, \dots, \mathcal{T}_M$. Each model provides a predicted target value and we either average (for regression) or compute the proportion of 1s / majority vote (for classification). More on this below.

Bagging improves model performance because it reduces the variance. Although each individual tree has high variance (and is overfit), the average prediction across all the trees does not (the proof leverages the Law of Large Numbers). The caveat here is that the individual trees should not be highly correlated with each other – two trees are correlated if they have the same splits. If we train each individual tree using the same training data \mathcal{T} , the resulting trees will be highly correlated (and many trees may be identical). Correlated trees are known to increase the bias of our predictions and bagging attempts to avoid this problem by training each tree on a different training set.

Extending bagging for random forests

Although each tree is trained using a (slightly) different training set, they may still be correlated. This is a major issue for applications with a small number of highly important features because each tree will split on those features and as a result, the trees will be correlated. To overcome this pitfall, we specialize the idea of bagging for random forests by only considering a random subset of features at each split (instead of all features). The suggested number of features to consider is a hyperparameter with the following default values:

- *Classification*: \sqrt{F}
- *Regression*: $\frac{1}{3}F$

where F is the total number of available features. This approach is sometimes called *feature bagging*. The combined randomness of the training set bagging and the feature bagging provides random forests with many uncorrelated trees, which ultimately reduces the model variance without increasing the bias.

Training a random forest

Combing all the above information, we train a random forest as follows:

1. Create M training sets, $\mathcal{T}_1, \dots, \mathcal{T}_M$ by randomly sampling n observations from \mathcal{T} with replacement
2. For each training set, train a CART model using a random subset of features (see Lecture 6 for details on training CART)

Out-of-bag error and hyperparameter tuning

Random forests have four key hyperparameters:

- *Number of trees*: larger (100s and 1000s) typically perform better. The more features you have, the more trees you want to train.
- *Number of features to consider at each split*: default of \sqrt{F} for classification and $\frac{1}{3}F$ for regression
- *Splitting criteria*: same as CART
- *Stopping criteria*: less important than CART due to bagging

Due to the bagging procedure used in random forests, we do not need an explicit validation set. Instead, we can use what is called *out-of-bag error*. Each sampled training set $\mathcal{T}_1, \dots, \mathcal{T}_M$ is called a *bag*. As noted above, each bag contains roughly 63% of the observations, leaving roughly 37% that are not used to train the corresponding tree. To obtain an out-of-bag error estimate, we run each observation \mathbf{x}_i through the random forest using only the trees that were *not* trained with the observation. We obtain a prediction and corresponding error for each observation, and we average over all observations. We can use out-of-bag error to tune our hyperparameters. However, we must remember to remove a test set before beginning the model training / bagging process so that we can evaluate our final (tuned) model.

Making predictions

Random forests are often very accurate in practice and have seen widespread success. One potential reason for this is that they typically require much less data than a neural network and provide more information about feature importance, allowing them to be applied to a broader range of problems.

To make a prediction in random forest, we first compute a prediction for each bag / CART. For regression problems, we denote the prediction $\hat{y}_{im} \in \mathbb{R}$ where i corresponds to the observation and m corresponds to the bag. For observation i , our final random forest regression prediction is:

$$\hat{y}_i = \frac{\sum_{m=1}^M \hat{y}_{im}}{M}$$

For classification problems, we denote the prediction as $\hat{y}_{im} \in \{0, 1\}$ where i corresponds to the observation and m corresponds to the bag. Note that \hat{y}_{im} has already been converted from a probability to a binary value by the CART model. Our final random forest classification prediction is:

$$\hat{p}_i = \frac{\sum_{m=1}^M \hat{y}_{im}}{M},$$

where \hat{p}_i is the probability that $\hat{y}_i = 1$. We can then chose to keep as is or round to a binary value.

Feature importance

As mentioned above, random forests provide some information about feature importance. There are many ways to compute feature importance and we will list two popular options:

- *Mean decrease in impurity / mean improvement in accuracy:* Each time we make a split, we improve the accuracy of our predictions (measured by our chosen metric). Each leaf node has a corresponding criteria value (e.g., Gini for classification, MSE for regression) and when we split that leaf node into two new child nodes, each of those new child nodes should improve the criteria value. For each feature, we measure the improvement every time the feature is split and average all the improvements (over all splits in all trees). We often compute a weighted average where the splits are weighted according to their depth (remember that higher depth means less important). We then scale all feature importance values to $[0, 1]$ to obtain a relative measure of feature importance.
- *Permutation importance:* Permutation importance is a general feature importance measure (and can be applied to any algorithm). We first obtain the performance metric value of our entire random forest model (e.g., AUC, MSE) using the out-of-bag validation or a validation set. To obtain the importance of a particular feature, we randomly permute the entries for that feature. In other words, we replace the feature value for each observation with a random number. We then recompute the model accuracy and the difference is used to represent feature importance. If there is no change in model accuracy, then the feature has little importance because changing it to a bunch of random numbers has no impact! Similar to above, we can scale all feature importance values to $[0, 1]$.

Extremely Randomized Trees

Extremely randomized trees or *extra trees* (ET) are a variation of the random forest algorithm. The ET algorithm has two key differences:

1. Each tree is trained using the same training set \mathcal{T} (i.e., no bagging)
2. Instead of finding the best split for each feature (by searching over all possible splits), we consider one random split for each feature. Note that this only applies to continuous features (binary features have only one possible split!). Once we determine a random split for each feature, we select the best feature (using our chosen metric).

Extra trees and random forests often perform similarly. The main differences are that extra trees are computationally faster, but may perform worse for high-dimensional datasets (i.e., a large number of features), especially if many of those features are worthless.