







Bookmarks

- ▶ Start Here
- ▶ 1. The Big Picture
- ▶ 2. Data And Features
- ▶ 3. Exploring Data
- ▶ 4. Transforming Data
- ▶ 5. Data Modeling
- ▼ **6. Data Modeling II**
 - Lecture: SVC**
Quiz 
 - Lab: SVC**
Lab 
 - Lecture: Decision Trees**
Quiz 
 - Lab: Decision Trees**
Lab 

6. Data Modeling II > Lecture: Random Forest > Video

 Bookmark

How Do Random Forest Work?

MOD43



Lecture: Random Forest

Quiz



▶ 0:00 / 1:48

▶ 1.0x



Dive Deeper

Download video

Download transcript

.srt

Training

Random forests make use of two techniques when training, one occurs at the forest level, and the other at the individual tree level. First, like any supervised classifier, you'll pass in a training set of samples along with "truth" labels when you create an instance of the class. Instead of sharing the entire dataset with each decision tree, the forest performs an operation which is essentially a train / test split of the *training* data. Each decision tree in the forest randomly samples from the overall training data set. Through doing so, each tree exists in an independent subspace and the variation between trees is controlled. This technique is known as *tree bagging*, or bootstrap aggregating.

Tree bagging increases the accuracy of decision trees because while an individual decision tree might become hypersensitive to outliers and localized features, once all the results are averaged, the fringe results get blurred out. Therefore using random forests over a single tree decreases the variance of your classification results, without increasing the bias the way KNeighbor does when K is set too high. This technique only works if the individual trees are not correlated. If they were all trained on the same training set, they would only reinforce each other's decision. The bootstrapping or randomization of samples each tree is trained upon takes care of that.

Random forests also use one more trick. In addition to the tree bagging of training samples at the forest level, each individual decision tree further 'feature bags' at each node-branch split. This is helpful because some datasets contain a feature that is very correlated to the target (the 'y'-label). By selecting a random sampling of features every split, if such a feature were to exist, it wouldn't show up on as many branches of the tree and there would be more diversity of the features examined.

Predicting

After fitting the forest, a prediction can be made for unseen samples by using the majority vote label assigned from each tree. SciKit-Learn gives you access to an array containing each of the trained trees in the forest, in case you'd like to inspect them individually. SciKit-Learn also supports doing regression with decision trees and random forests. In these cases, the output needs to be a continuous variable, and is calculated as the average result of each tree.

Error Testing

Since each tree within the forest is only trained using a subset of the overall training set, the forest ensemble has the ability to error test itself. It does this by scoring each tree's predictions against that tree's out-of-bag samples. A tree's out of bag samples are those forest training samples that were withheld from a specific tree during training. There's nothing unique about splitting your data between training and testing sets except that you have an independent set of unseen samples to validate the accuracy of your training. Part of the random forest algorithm is the creation of independent sets for the training of each tree, so an overall *out-of-bag error* metric can be calculated for the forest ensemble. This error value is defined as the mean prediction error for each training samples using only those trees that didn't have the sample in their bootstrap.

One of the advantages of using the out of bag error is it eliminates the need for you to split your data into a training / testing before feeding it into the forest model, since that's part of the forest algorithm. However using the out-of-bag error metric often underestimates the actual performance improvement, and the optimal number of training iterations. Due to this and a few other reason's we'll discuss in the next module, SciKit-Learn recommends you maintain separate training and testing sets.



© edX Inc. All rights reserved except where noted. EdX, Open edX and the edX and Open EdX logos are registered trademarks or trademarks of edX Inc.

