**An Intuitive Explanation of Random Forest and Extra Trees Classifiers**

[[](https://towardsdatascience.com/@frank.ceballos?source=post_page-----8507ac21d54b----------------------)](https://towardsdatascience.com/@frank.ceballos?source=post_page-----8507ac21d54b----------------------)

[Frank Ceballos](https://towardsdatascience.com/@frank.ceballos?source=post_page-----8507ac21d54b----------------------)

[Jul 14](https://towardsdatascience.com/an-intuitive-explanation-of-random-forest-and-extra-trees-classifiers-8507ac21d54b?source=post_page-----8507ac21d54b----------------------) · 7 min read

Photo by [Aperture Vintage](https://unsplash.com/@aperturevintage?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

**Purpose:** The purpose of this article is to provide the reader an intuitive understanding of [Random Forest](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) and [Extra Trees](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html) classifiers.

**Materials and methods:** We will use the Iris dataset which contains features describing three species of flowers. In total there are 150 instances, each containing four features and labeled with one species of flower. We will investigate and report on the accuracy of Decision Trees, Random Forest, and Extra Trees.

**Hardware**: We train and evaluate our models on an Apple workstation equipped with 8 GB 1600 MHz DDR3, with Inter(R)Core(TM) i7 with 2 CPUs @ 2.9 Ghz, and an Intel HD Graphics 4000 card. Let’s kick some tail.

**Note:** In the case you’re starting from scratch, I advise you follow this [article](https://towardsdatascience.com/i-want-to-be-the-very-best/installing-keras-tensorflow-using-anaconda-for-machine-learning-44ab28ff39cb) to install all the necessary libraries. Finally, it will be assumed that the reader is familiar with Python, Pandas, Scikit-learn, and decision trees. A detailed explanation about Scikit-learn decision trees can be found [here](https://towardsdatascience.com/scikit-learn-decision-trees-explained-803f3812290d).

**A Single Stump vs 1000 Stumps**

Suppose that we have a weak learner, a classifier whose accuracy is slightly better than a random decision, with a classification accuracy of 51 %. This could be a [decision stump](https://en.wikipedia.org/wiki/Decision_stump) , a decision tree classifier with its depth set to one. At first instance, it would appear that one shouldn’t bother which such a weak classifier; however, what if we consider putting together 1000 slightly different decision stumps (an ensemble), each with 51 % accuracy to make our final prediction? Intuitively, we can see that in average, 510 of these classifiers would correctly classify a test case and that 490 would misclassify it. If we collect the [hard votes](https://stats.stackexchange.com/questions/320156/hard-voting-versus-soft-voting-in-ensemble-based-methods) of each classifier, we could see that on average there would be about 20 more correct predictions; consequently, our ensemble would tend to have an accuracy higher than 51 %. Let’s see this in practice.

Here we will build a decision stump and compare its predictive performance to an ensemble of 1000 of them. The ensemble of decision trees is created using the Scikit-learn [BaggingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html). The decision stump and the ensemble will be trained on the Iris dataset which contains four features and three classes. The data is randomly split to create a training and test set.

Each decision stump will be built with the following criteria:

1. All the data available in the training set is used to build each stump.
2. To form the root node or any node, the best split is determined by searching in all the available features.
3. The maximum depth of the decision stump is one.

First we import all the libraries that we will use for this article.

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # 1. Importing Libraries # |
|  |

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # The usual stuff |
|  |

|  |
| --- |
| import pandas as pd |
|  |

|  |
| --- |
| import numpy as np |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Iris dataset |
|  |

|  |
| --- |
| from sklearn.datasets import load\_iris |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Data preprocessing and machine learning |
|  |

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split |
|  |

|  |
| --- |
| from sklearn.tree import DecisionTreeClassifier |
|  |

|  |
| --- |
| from sklearn.ensemble import BaggingClassifier |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # To measure performance |
|  |

from sklearn import metrics

Then, we load the data, split it, and train and compare a single stump vs an ensemble. The results are printed to the console.

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # 2. Stump vs Ensemble of 1000 Decision Stumps # |
|  |

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # Load data and store it into pandas DataFrame objects |
|  |

|  |
| --- |
| iris = load\_iris() |
|  |

|  |
| --- |
| X = pd.DataFrame(iris.data[:, :], columns = iris.feature\_names[:]) |
|  |

|  |
| --- |
| y = pd.DataFrame(iris.target, columns =["Species"]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Splitting Dataset |
|  |

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 20, random\_state = 100) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Defining the stump |
|  |

|  |
| --- |
| stump = DecisionTreeClassifier(max\_depth = 1) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Creating an ensemble |
|  |

|  |
| --- |
| ensemble = BaggingClassifier(base\_estimator = stump, n\_estimators = 1000, |
|  |

|  |
| --- |
| bootstrap = False) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Training classifiers |
|  |

|  |
| --- |
| stump.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
| ensemble.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Making predictions |
|  |

|  |
| --- |
| y\_pred\_stump = stump.predict(X\_test) |
|  |

|  |
| --- |
| y\_pred\_ensemble = ensemble.predict(X\_test) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Determine performance |
|  |

|  |
| --- |
| stump\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_stump) |
|  |

|  |
| --- |
| ensemble\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_ensemble) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Print message to user |
|  |

|  |
| --- |
| print(f"The accuracy of the stump is {stump\_accuracy\*100:.1f} %") |
|  |

print(f"The accuracy of the ensemble is {ensemble\_accuracy\*100:.1f} %")

The accuracy of the stump is 55.0 %  
The accuracy of the ensemble is 55.0 %

The results show that the ensemble of 1000 decision stumps obtained an accuracy of 55 %, showing that they are no better than a single decision stump. So what happened? Why are we not getting better results? Well we basically created 1000 decision stumps that were exactly the same. It’s like we asked a single person what their favorite food was 1000 times and, not surprisingly, obtained the same answer 1000 times.





**Random Forest Classifier**

In the previous section, we learned that having 1000 copies of the same decision stump in our ensemble is like having a single decision stump. Therefore, we will change the criteria for how we build each stump in order to introduce variation.

Each decision stump will be built with the following criteria:

1. A bootstrap will be created by randomly sampling the training set with replacement. The size of the bootstrap is set to equal the size of the training set.
2. To form the root node or any node, the best split is determined by searching in a subset of randomly selected features of size sqrt(number of features). In our case, each decision stump is allowed to inspect two out of the four features.
3. The maximum depth of the decision stump is one.

What we just described was the criteria to create a Random Forest. However, a Random Forest uses decision trees with a depth of one or greater. The term *random* stems from the fact that we randomly sample the training set, and since we have a collection of trees, it’s natural to call it a forest — hence Random Forest. To build the root node or any node in the tree, a random subset of features is selected. For each of these selected features, the algorithm searches for the optimal cutting point to determine the split for the given feature. The feature from the randomly selected subset that produces the purest split is then used to create the root node. The tree is grown to a depth of one, and the same process is repeated for all other nodes in the tree, until the desired depth of the tree is reached. Finally, it’s important to note that each tree is built separately using a different bootstrap, which introduces variation among the trees. Consequently, each tree makes different mistakes and when combined a strong classifier can be built. If you’re confused by all the jargon, [read this article](https://towardsdatascience.com/scikit-learn-decision-trees-explained-803f3812290d) that explains most of what I just described in a single paragraph.

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # 3. Stump vs Random Forest # |
|  |

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # Defining the stump |
|  |

|  |
| --- |
| stump = DecisionTreeClassifier(max\_depth = 1, splitter = "best", max\_features = "sqrt") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create Random Forest |
|  |

|  |
| --- |
| ensemble = BaggingClassifier(base\_estimator = stump, n\_estimators = 1000, |
|  |

|  |
| --- |
| bootstrap = True) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Training classifiers |
|  |

|  |
| --- |
| stump.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
| ensemble.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Making predictions |
|  |

|  |
| --- |
| y\_pred\_tree = stump.predict(X\_test) |
|  |

|  |
| --- |
| y\_pred\_ensemble = ensemble.predict(X\_test) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Determine performance |
|  |

|  |
| --- |
| stump\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_stump) |
|  |

|  |
| --- |
| ensemble\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_ensemble) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Print message to user |
|  |

|  |
| --- |
| print(f"The accuracy of the stump is {stump\_accuracy\*100:.1f} %") |
|  |

print(f"The accuracy of the Random Forest is {ensemble\_accuracy\*100:.1f} %")

Script 3 — Stump vs Random Forest. Notice how in line 5, we set splitter = “best” and in line 9 bootstrap = True. Your results may slightly vary since we did not fixed the seeds in the stump.

The accuracy of the stump is 55.0 %  
The accuracy of the Random Forest is 95.0 %

What?! So by simply introducing variation, we were able to obtain an accuracy of 95 %. In other words, decision stumps with low accuracies were used to build a forest. Variation was introduced among the stumps by building them on bootstraps — created by sampling the training set with replacement and allowing the stumps to only search subsets of randomly selected features to split the root node. Individually, each stump would obtain a low accurracy. However, when used in an ensemble, we showed that their accuracy skyrocketed! That, my friends, is what is commonly referred to as the wisdom of the crowd.

Just so that it sinks in, we can use weak classifiers as the base of an ensemble to obtain a high performing one.





**Extra Trees Classifier**

Similar to a Random Forest classifier we have the Extra Trees classifier — also known as Extremely Randomized Trees. To introduce more variation into the ensemble, we will change how we build trees.

Each decision stump will be built with the following criteria:

1. All the data available in the training set is used to built each stump.
2. To form the root node or any node, the best split is determined by searching in a subset of randomly selected features of size sqrt(number of features). The split of each selected feature is chosen at random.
3. The maximum depth of the decision stump is one.

Notice that in an Extra Trees classifier, the features and splits are selected at random; hence, “Extremely Randomized Tree”. Since splits are chosen at random for each feature in the Extra Trees Classifier, it’s less computationally expensive than a Random Forest.

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # 4. Stump vs Extra Trees # |
|  |

|  |
| --- |
| ############################################################################### |
|  |

|  |
| --- |
| # Defining the stump |
|  |

|  |
| --- |
| stump = DecisionTreeClassifier(max\_depth = 1, splitter = "random", max\_features = "sqrt") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create Extra Trees |
|  |

|  |
| --- |
| ensemble = BaggingClassifier(base\_estimator = stump, n\_estimators = 1000, |
|  |

|  |
| --- |
| bootstrap = False) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Training classifiers |
|  |

|  |
| --- |
| stump.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
| ensemble.fit(X\_train, np.ravel(y\_train)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Making predictions |
|  |

|  |
| --- |
| y\_pred\_tree = stump.predict(X\_test) |
|  |

|  |
| --- |
| y\_pred\_ensemble = ensemble.predict(X\_test) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Determine performance |
|  |

|  |
| --- |
| stump\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_stump) |
|  |

|  |
| --- |
| ensemble\_accuracy = metrics.accuracy\_score(y\_test, y\_pred\_ensemble) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Print message to user |
|  |

|  |
| --- |
| print(f"The accuracy of the stump is {stump\_accuracy\*100:.1f} %") |
|  |

print(f"The accuracy of the Extra Trees is {ensemble\_accuracy\*100:.1f} %")

Script 4— Stump vs Extra Trees. Notice how in line 5 splitter = “random” and the bootstrap is set to false in line 9. Your results may slightly vary since we did not fixed the seeds in the stump.

The accuracy of the stump is 55.0 %  
The accuracy of the Extra Trees is 95.0 %

**The Extra Trees classifier performed similarly to the Random Forest. However, there are performance differences that I would like to mention. Namely: Decision Trees show high variance, Random Forests show medium variance, and Extra Trees show low variance.**





**Closing Remarks**

If you reached the end of the article, by now you should understand the power of ensemble methods and know how Random Forest and and Extra Trees classifiers are built. I would like to mention that you should not use the Bagging classifier to build your Random Forest or Extra Trees classifier. More effective versions of these two classifiers are already built into Scikit-learn.

Ensemble methods are not limited to having weak learners as their base estimator. For example, you can determine the best three classifiers for a given task and use Scikit-learn [Voting Classifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html) to form an ensemble with these. You can optimize the weights of the ensemble and make predictions with them. Assuming that you have finely tuned your classifiers and all of them have similar performances, the Voting Classifier would gain an edge over any of them. To narrow your search of which classifiers to use, you can read this article that talks about [model design and selection using Scikit-learn](https://medium.com/i-want-to-be-the-very-best/model-design-and-selection-with-scikit-learn-18a29041d02a?source=post_stats_page---------------------------).