RANDOM FOREST

Advantages:-

* Assumption free
* They are often accurate,
* At the core of this algorithm is a Decision Tree so, Random Forests shares all its advantages but more powerful.
* It’s a data robust algorithm
* do not require feature scaling,
* categorical feature encoding, and
* need little parameter tunning.
* They can also be more interpretable than other complex models such as neural networks.
* Random forests are powerful not only in classification/regression but also for purposes such as outlier detection, clustering, and interpreting a data set (e.g., serving as a rule engine with inTrees).
* To address overfitting, and reduce the variance in Decision Trees; it utilized, for the first time, the statistical technique of Bootstrapping and combined the results of training multiple models into a single, more powerful learning model.
* The true potential of Random Forests comes from combining the results different Decision Trees.

Disadvantages:-

* There can be a bias when multi-level categorical features exist in a data set.
* importance scores can be misleading when features are redundant.
* You can’t see how the model makes decisions. with Random Forests, you trade-off interpretability for performance. the decision process gets much more complex and impossible to visualization.
* Correlated features will be given equal or similar importance, but overall reduced importance compared to the same tree built without correlated counterparts.
* Random Forests and decision trees, in general, give preference to features with high cardinality ( Trees are biased to these type of variables ).

What is a random forest

* Even though Decision Trees is simple and flexible, it is greedy algorithm. It focuses on optimizing for the node split at hand, rather than taking into account how that split impacts the entire tree. A greedy approach makes Decision Trees run faster, but makes it prone to overfitting.
* In Regression tasks overfitting is detected by high-variance, while in Classification tasks it’s by a high generalization error.
* Random Forests was developed specifically to address the problem of high-variance in Decision Trees.
* A forest of Bagged Decision Trees.
* A random forest consists of multiple random decision trees. Two types of randomnesses are built into the trees.
  + First, each tree is built on a random sample from the original data. **(Sampling of training data)**
  + Second, at each tree node, a subset of features are randomly selected to generate the best split. **(Sampling of features)**
  + this guarantees that the trees are de-correlated
* While Forest part of Random Forests refers to training multiple trees, the Random part is present at two different points in the algorithm.
  + There’s the randomness involved in the Bagging process. But then, you also pick a random subset of features to evaluate the node split. This is what guarantees that each tree is different and, therefore, ensures each model produces a slightly different result.
  + And while you could be thinking that randomly sampling features at each split introduces yet another hyperparameter you might need to tune, that’s not the case. The model takes care of it for you!
  + You can certainly tune this hyperparameter. However, there’s mathematical consensus about randomly picking a number of features that is equal to the square root of the total number of available features in the dataset.
* Each model is different
  + Like Decision Trees, the algorithm optimizes for the local split. But, instead of exploring all possible splits for each feature in the dataset, it randomly picks a subset of those features.
  + This randomness reduces the number of outcomes the algorithm needs to evaluate at each split and, it makes each trained tree is slightly different.
* **No holdout set required**
  + In Machine Learning you typically split the dataset into training and testing sets, in an effort to evaluate model performance with observations it has never seen before. This becomes a challenging problem when you have a small dataset or the cost, and effort, of collecting more data is high.
  + Out-of-bag Error for Random Forest evaluation
  + With Random Forests you can use the entire dataset to train and evaluate the model. The Bagging process takes care of it for you! Since you’re generating N smaller, random datasets picked with replacement, there’s always going to be a set of points that was not used to create the tree.

|  |
| --- |
| https://miro.medium.com/max/700/1*e34ym6401RB7SJQs6KfToA.png |
| https://miro.medium.com/max/700/1*4x8K9XRAZXRycFEXRtw5qA.png |
| The same process as above is applied to build multiple trees. The figure below illustrates the flow of applying a random forest with three trees to a testing data instance.  https://miro.medium.com/max/1050/1*EYFSWWF0FJJvTk4L44qFUQ.png |

Steps:

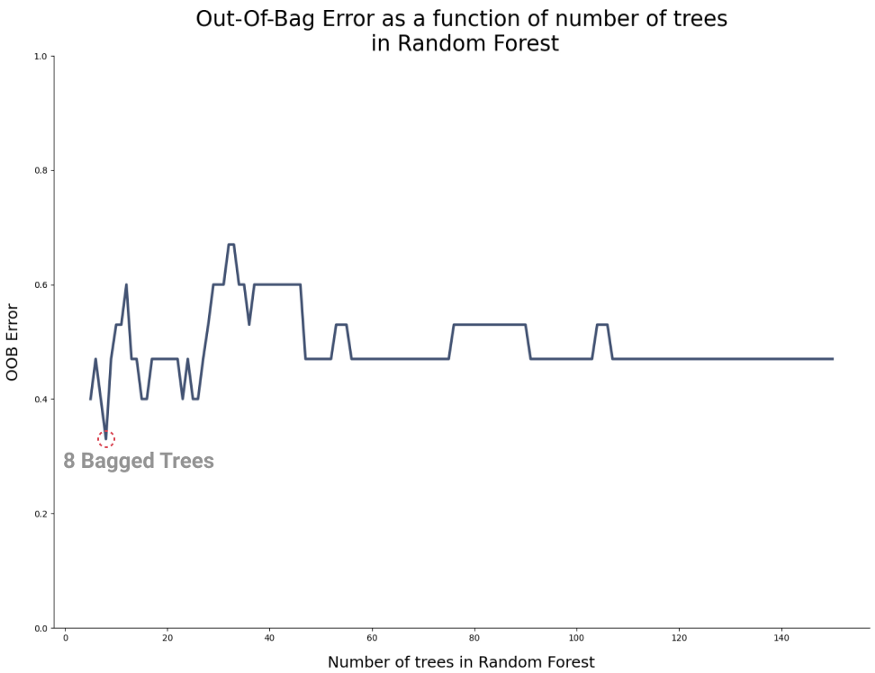
1. Take the original dataset and create N bagged samples of size n, with n smaller than the original dataset.
2. Train a Decision Tree with each of the N bagged datasets as input. But, when doing a node split, don’t explore all features in the dataset. Randomly select a smaller number, M features, from all the features in training set. Then pick the best split using impurity measures, like Gini Impurity or Entropy.
3. Aggregate the results of the individual decision trees into a single output.
4. Average the values for each observation, produced by each tree, if you’re working on a Regression task.
5. Do a majority vote across all trees, for each observation, if you’re working on a Classification task.

Reducing model variance with Random Forests

|  |
| --- |
| https://miro.medium.com/max/875/1*tc9muYY5AEDANN0G0Cu_NA.png |
| With DT: https://miro.medium.com/max/389/1*E8ADqor8csxvSPQ7r3EE0g.png |
| https://miro.medium.com/max/875/1*HiDDktOqf-mORomW8_SPCw.png |
| With RF: https://miro.medium.com/max/460/1*KIcGngSsqEDnpBPxa0IuxQ.png  By default the algorithm trains and combines the result of 100 bagged trees. This is one of the many hyperparameters you can tune in Random Forests.  You could see the generalization error decline significantly, from 67% to 33%. |

Find the Optimal Number of Trees

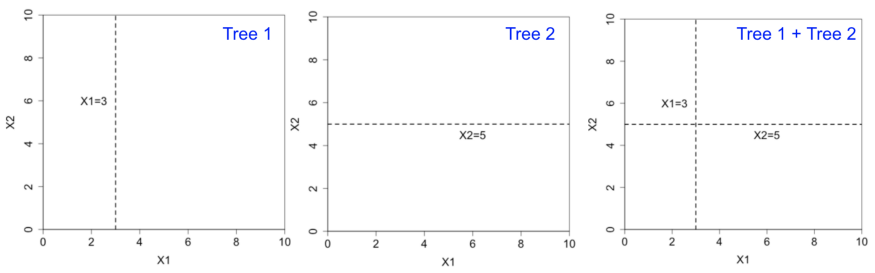
* Instead of tuning this hyperparameter by trail and error, you decided plot the Out-Of-Bag-Error(~Cross val error) as a function of the number of trees in the Random Forest.
* Training a large forest doesn’t improve the model.
* Looking at the plot, it’s easier to see the Out-Of-Bag Error getting worse past the sweet-spot of 8 trees. Then the model hits a performance plateau, when the forest gets bigger than 50 trees.



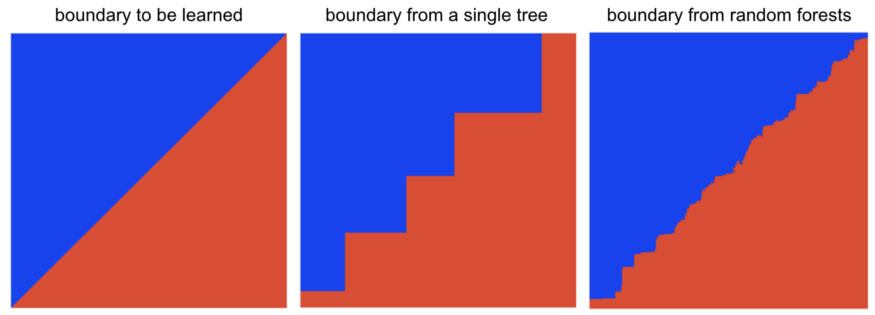
Why random forests outperform decision trees

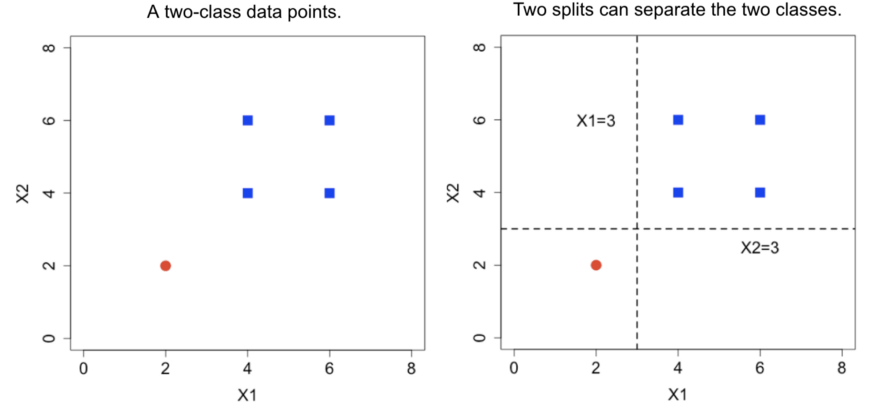
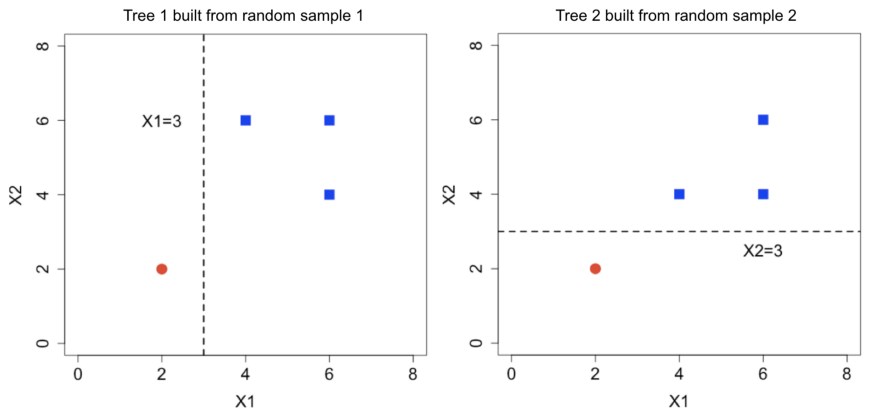
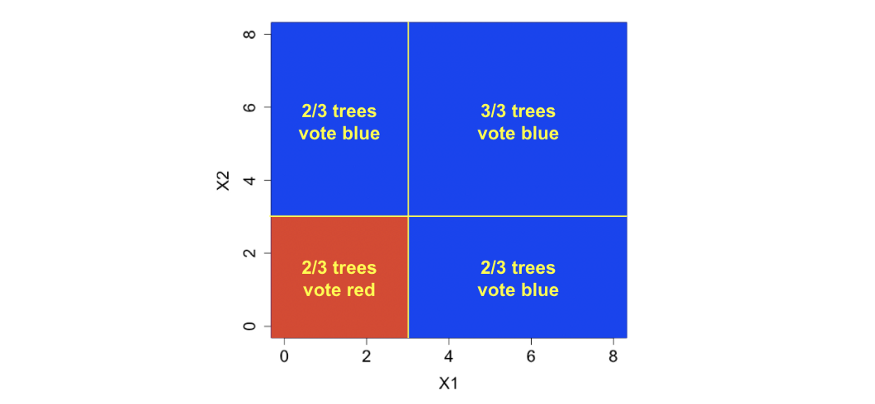
|  |
| --- |
| https://miro.medium.com/max/563/1*EFBVZvHEIoMdYHjvAZg8Zg.gif |
|  |

* **Higher Resolution In The Feature Space**
* ForestTrees are unpruned. While a single decision tree like CART is often pruned, a random forest tree is fully grown and unpruned, and so, naturally, the feature space is split into more and smaller regions.
* Trees are diverse. Each random forest tree is learned on a random sample, and at each node, a random set of features are considered for splitting. Both mechanisms create diversity among the trees.
* By combining the two trees, there are four regions that can be labeled differently.



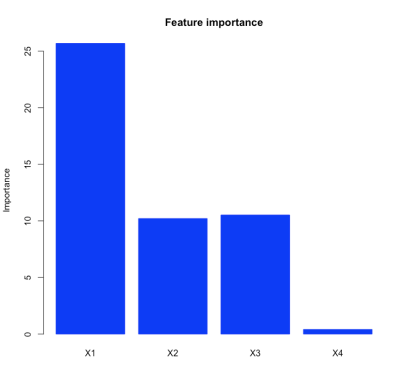
* Unpruned and diverse trees lead to a high resolution in the feature space. For continuous features, it means a smoother decision boundary.



* HANDLING OVERFITTING
* A single decision tree needs pruning to avoid overfitting. The following shows the decision boundary from an unpruned tree. The boundary is smoother but makes obvious mistakes (overfitting).
* The randomness and voting mechanisms in random forests elegantly solve the overfitting problem.
* 
* The two splits, however, result in very different decision boundaries. Decision trees often use the first variable to split, and so the ordering of the variables in the training data determines the decision boundary.
* Now consider random forests. For each random sample used for training a tree, the probability that the red point missing from the sample is 33%. So roughly 1 out of 3 trees is built with all blue data and always predict class blue. The other 2/3 of the trees have the red point in the training data. Since at each node a random subset of features is considered, we expect roughly 1/3 of the trees use x1, and the rest 1/3 uses x2.
* 
* By aggregating the three types of trees, the decision boundary shown below is now symmetric for x1 and x2. As long as there are enough trees, the boundary should be stable and does not depend on irrelevant information such as the ordering of variables.
* 
* Although random forests are accurate, they are considered as black-box models (hard to explain).

Feature Importance:-

* A feature’s importance score measures the contribution from the feature. It is based on the impurity reduction of the class due to the feature.



Feature selection using Random forest comes under the category of Embedded methods. Embedded methods combine the qualities of filter and wrapper methods. They are implemented by algorithms that have their own built-in feature selection methods. Some of the benefits of embedded methods are :

* They are highly accurate.
* They generalize better.
* They are interpretable

Random forests consist of 4 –12 hundred decision trees, each of them built over a random extraction of the observations from the dataset and a random extraction of the features. Not every tree sees all the features or all the observations, and this guarantees that the trees are de-correlated and therefore less prone to over-fitting. Each tree is also a sequence of yes-no questions based on a single or combination of features. At each node (this is at each question), the three divides the dataset into 2 buckets, each of them hosting observations that are more similar among themselves and different from the ones in the other bucket. Therefore, the importance of each feature is derived from how “pure” each of the buckets is.

* For classification, the measure of impurity is either the Gini impurity or the information gain/entropy.
* For regression the measure of impurity is variance.
* Therefore, when training a tree, it is possible to compute how much each feature decreases the impurity. The more a feature decreases the impurity, the more important the feature is. In random forests, the impurity decrease from each feature can be averaged across trees to determine the final importance of the variable.
* To give a better intuition, features that are selected at the top of the trees are in general more important than features that are selected at the end nodes of the trees, as generally the top splits lead to bigger information gains.

Python code on how to select features:-

* In all feature selection procedures, it is a good practice to select the features by examining only the training set. This is to avoid overfitting.
  + X\_train,y\_train,X\_test,y\_test=train\_test\_split(data,test\_size=0.3)
  + sel= SelectFromModel(RandomForestClassifier(n\_estimators = 100))
  + sel.fit(X\_train, y\_train)
* To see which features are important we can use get\_support method on the fitted model.
  + sel.get\_support()
  + It will return an array of boolean values. True for the features whose importance is greater than the mean importance and False for the rest.

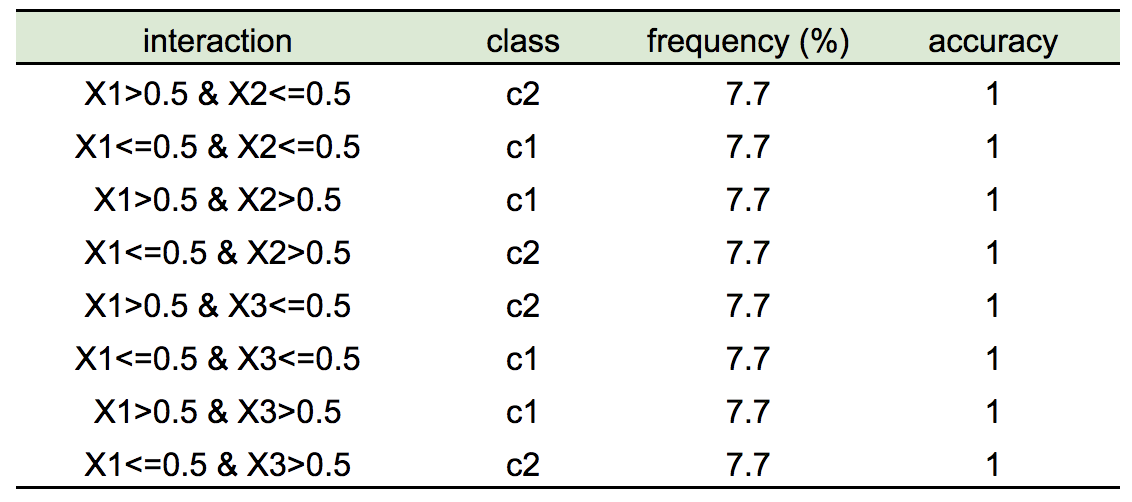
Partial dependency plot

* The importance score of a feature doesn’t tell how a feature and the class are correlated. The partial dependency plot can visualize the marginal effect of a feature on the class probability.

|  |  |
| --- | --- |
| When a feature is correlated to the class, the plot looks like the left figure below, indicating that X1 ≤ 0.5 and X1>0.5 are associated with different classes. | However, in our illustrative dataset, the partial dependence plot looks like the right figure — it does not indicate any relationship between X1 and the class, even though X1 has the largest importance score.  The reason is that X1 has to interact with X2 or X3 to be predictive of the class. X1 alone is not predictive. Therefore, partial dependence plots can be misleading for this case. |
| https://miro.medium.com/max/1680/1*shkbvFdyvn196odzFIgKLw.png | https://miro.medium.com/max/1680/1*my5rEC76WAf_YfDt5l36tQ.png |

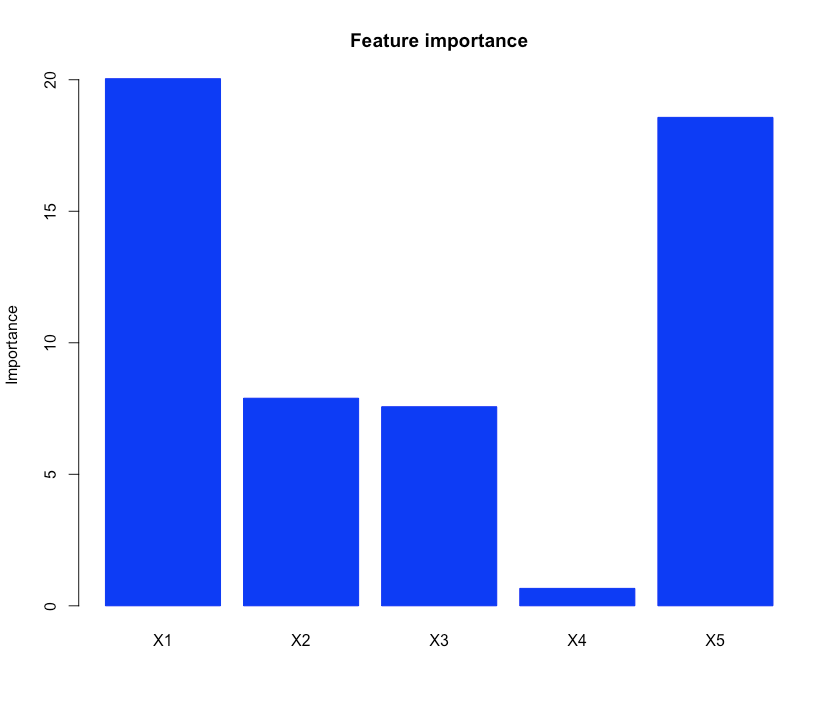
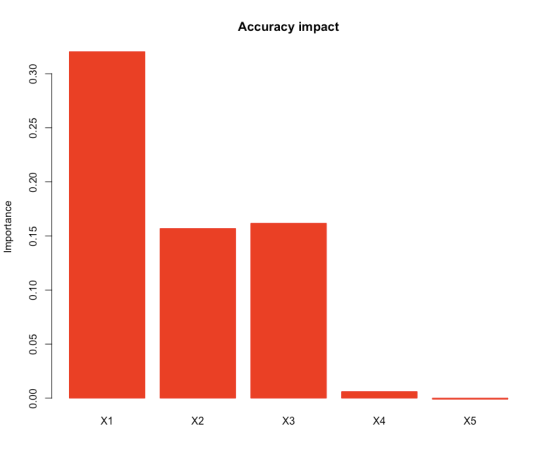
inTrees

* Neither importance scores nor partial dependency plots tell how multiple features interact with the class. The inTrees framework can be used to gain a clearer picture of what happens inside a random forest.
* For the illustrative dataset, the highly-predictive interactions and their associated classes can be extracted with inTrees and shown below. The frequency (0%-100%) measures the popularity of an interaction in the random forest, and accuracy (0–1) measures how accurate an interaction predicts the class.



Bias towards features with more categories

* For the illustrative dataset, let’s add a random feature X5 with 30 categories. Even the feature is irrelevant to the class, the importance score of X5 is larger than truly informative features X2 and X3, indicating an incorrect bias towards features with more categories.

…

* One solution is to perform feature selection. For example, in the randomForest R package, one can use features’ impact on accuracy (importance$MeanDecreaseAccuracy) to evaluate features. The accuracy impact plot below shows X5’s accuracy impact is quite small compared to the truly informative features, indicating the feature is confusing the model and should be removed before fitting a classifier.

Handling redundant features

* When features are similar to each other, the importance scores of these features can be misleading. In the illustrative dataset, X2 and X3 are identical and they “share” the importance scores (shown the left figure below). When there are more redundant features, the importance of each feature becomes even smaller.
* This may not hurt the accuracy performance but could be misleading in interpretation. One solution would be the regularized random forest (RRF). In the tree building process, RRF memorizes the features used in previous tree nodes, and prefer these features in splitting future tree nodes, therefore avoiding redundant features in the trees. The right figure below shows the importance scores from RRF.

|  |  |
| --- | --- |
| https://miro.medium.com/max/923/1*6NqVItJZg7DB9HibvJb89Q.png | https://miro.medium.com/max/923/1*0xF_rhmspSnXM7Kp2ngmDg.png |

Outlier detection with random forests

* Clustering with random forests can avoid the need of feature transformation (e.g., categorical features). In addition, some other random forest functions can also be used here, e.g., probability and interpretation. Here we demonstrate the method with a two-dimensional data set plotted in the left figure below.
* The idea is to generate a random data set that contrast with the original data. Here we randomly permute each feature. The two data sets are labeled with two classes (say, “normal” and “random”), respectively. The combined data set is shown in the right figure below.

|  |  |
| --- | --- |
| Original data  https://miro.medium.com/max/1476/1*nHNCFexPouUFV8i4rOi4CA.png | Generate a two-class data set. class 1: original data; class 2: the same size as the original data but with X1 and X2 randomly permuted.  https://miro.medium.com/max/1476/1*fsYeH97Z2pb_hfQHW8AloA.png |

* A random forest is built on the dataset. Then the classifier can be applied to test data instances. If the predicted class is “random”, then it is identified as an outlier. The outliers identified are illustrated in the left figure below.
* We can get insights into which features contribute to the outlier detection by looking at the importance score. For illustration, we add a random feature X3 that is irrelevant to the classes. The importance scores are shown in the right figure below. X1 and X2 are identified as important features, while X3 is less important.

|  |  |
| --- | --- |
| https://miro.medium.com/max/1346/1*u-FjSt6BCUN46z8UMKmJkA.png | https://miro.medium.com/max/923/1*aecONs-9EIokoy7cJRJYjA.png |

Clustering with random forests

* Similar to outlier detection, clustering with random forests saves efforts in feature preprocessing.
* The procedure is similar to outlier detection. First, create a synthetic dataset of the same size as the original data. Then label the original data and synthetic class with two different classes. A random forest is then built for the classification problem.
* From the built random forest, a similarity score between each pair of data instances is extracted. The similarity of two data instances is measured by the percentage of trees where the two data instances appear in the same leaf node.
* With the similarity scores, clustering algorithms such as hierarchical clustering can then be used for clustering. The figures below show the clustering results with the number of cluster pre-defined as 2 and 4 respectively.

|  |  |
| --- | --- |
| clustering with 2 clusters.  https://miro.medium.com/max/1346/1*y21hXrmsnRfNvIvfVbcEfA.png | clustering with 4 clusters  https://miro.medium.com/max/1346/1*Wrhen01tzaOKCvL8mN3Gjw.png |

Why make decision tress highly uncorrelated?

* **Feature bagging:** bootstrap aggregating or bagging is a method of selecting a random number of samples from the original set with replacement. In feature bagging the original feature set is randomly sampled and passed onto different trees. This is done to decrease the correlation among trees. A feature with unmatched great importance will cause every decision tree to choose it for the first and possible consequent splits, this will make all the trees behave similarly and ultimately more correlated which is undesirable. Our aim here is to make highly uncorrelated decision trees.
* We need highly uncorrelated decision trees because “average error of a bunch of perfectly random errors is zero” hence by decreasing the correlation and making each tree split as randomly as possibly (random in the sense of feature selection, we still aim to find the best split in the randomly selected set of columns), we get better predictions devoid of error.
* **Aggregation:** The core concept that makes random forests better than decision trees is aggregating uncorrelated trees. The idea is to create several crappy model trees (low depth) and average them out to create a better random forest. Mean of some random errors is zero hence we can expect generalized predictive results from our forest. In case of regression we can average out the prediction of each tree (mean) while in case of classification problems we can simply take the majority of the class voted by each tree (mode).

**CODING FOR RANDOM FOREST**

<https://towardsdatascience.com/random-forests-and-decision-trees-from-scratch-in-python-3e4fa5ae4249>

* **x:** independent variables of training set. To keep things minimal and simple I am not creating a separate fit method hence the base class constructor will accept the training set.
* **y:** the corresponding dependent variables necessary for supervised learning (Random forest is a supervised learning technique)
* **n\_trees :** number of uncorrelated trees we ensemble to create the random forest.
* **n\_features:** the number of features to sample and pass onto each tree, this is where feature bagging happens. It can either be sqrt, log2 or an integer. In case of sqrt, the number of features sampled to each tree is square root of total features and log base 2 of total features in case of log2.
* **sample\_size:** the number of rows randomly selected and passed onto each tree. This is usually equal to total number of rows but can be reduced to increase performance and decrease correlation of trees in some cases (bagging of trees is a completely separate machine learning technique)
* **depth:** depth of each decision tree. Higher depth means more number of splits which increases the over fitting tendency of each tree but since we are aggregating several uncorrelated trees, over fitting of individual trees hardly bothers the whole forest.
* **min\_leaf:** minimum number of rows required in a node to cause further split. Lower the min\_leaf, higher the depth of the tree.

|  |  |  |  |
| --- | --- | --- | --- |
| sklearn | ensemble | RandomForestClassifier  RandomForestRegressor | apply |
|  |  | decision\_path |
|  |  | feature\_importances\_ |
|  |  | fit |
|  |  | get\_params |
|  |  | mro |
|  |  | predict |
|  |  | predict\_proba |
|  |  | score |
|  |  | set\_params |
|  |  | 'AdaBoostClassifier', |  |
|  |  | 'AdaBoostRegressor', |  |
|  |  | 'BaggingClassifier', |  |
|  |  | 'BaggingRegressor', |  |
|  |  | 'BaseEnsemble', |  |
|  |  | 'ExtraTreesClassifier', |  |
|  |  | 'ExtraTreesRegressor', |  |
|  |  | 'GradientBoostingClassifier', |  |
|  |  | 'GradientBoostingRegressor', |  |
|  |  | 'IsolationForest', |  |
|  |  | 'RandomForestClassifier', |  |
|  |  | 'RandomForestRegressor', |  |
|  |  | 'RandomTreesEmbedding', |  |
|  |  | 'StackingClassifier', |  |
|  |  | 'StackingRegressor', |  |
|  |  | 'VotingClassifier', |  |
|  |  | 'VotingRegressor', |  |