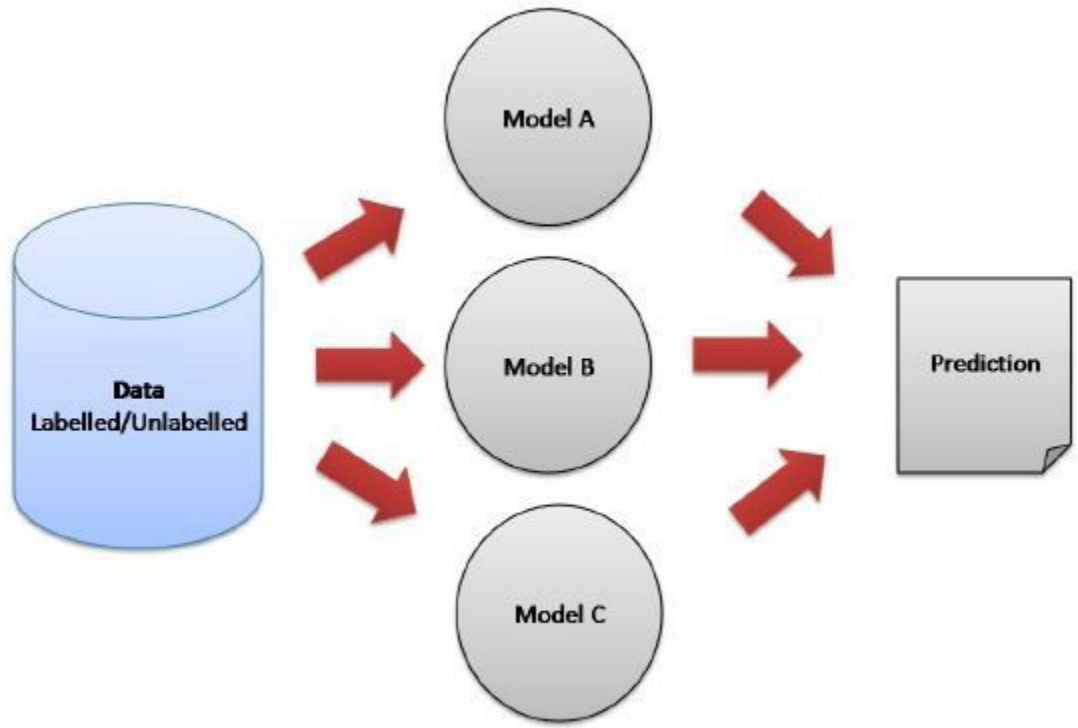
Ensemble Methods

Introduction

- Ensemble learning helps improve machine learning results by combining several models.
- Ensemble learning combines the predictions from multiple models to reduce the variance of predictions and reduce generalization error.
- This approach allows the production of better predictive performance compared to a single model.
- Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).


Model A
Data
Labelled/Unlabelled
Model B
Prediction
Model C

- Models can be different from each other for a variety of reasons, starting from the population they are built upon to the modeling used for building the model.

The differences can be due to :

1. Difference in Population.
2. Difference in Hypothesis.
3. Difference in Modeling Technique.
4. Diffence in Initial Seed.

Error in Ensemble Learning (Variance vs. Bias)

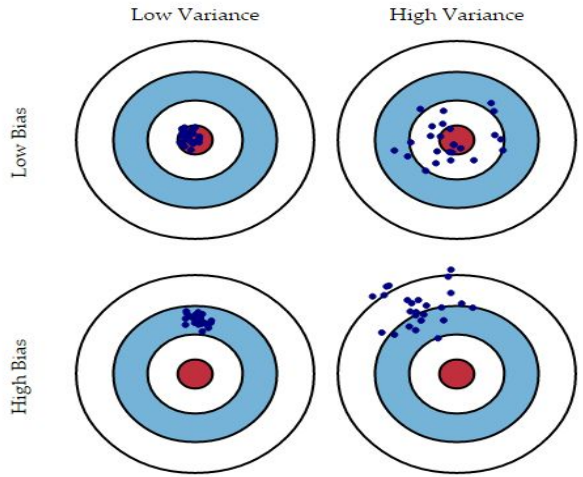
The error emerging from any model can be broken down into three components mathematically. Following are these component :

$$Err(x) = \left(E[\hat{f}\,(x)] - f(x)\right)^2 + E\left[\hat{f}\,(x) - E[\hat{f}\,(x)]\right]^2 + \sigma_e^2$$

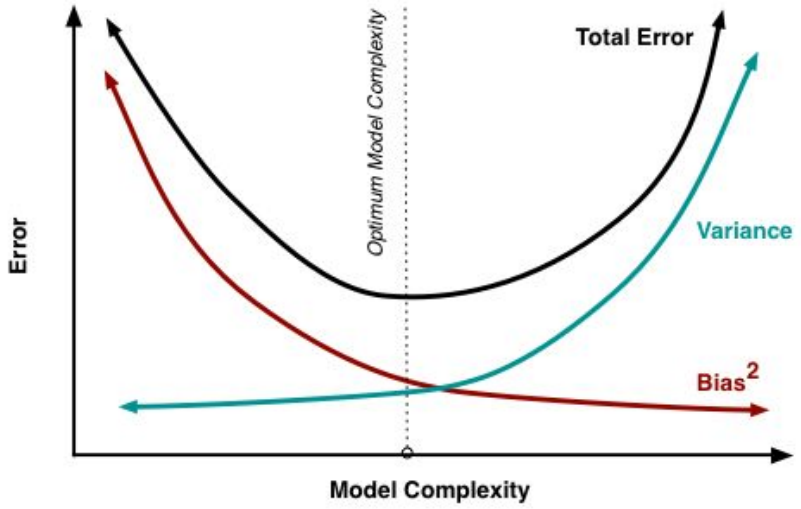
$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias error is useful to quantify how much on an average are the predicted values different from the actual value. A high bias error means we have a under-performing model which keeps on missing important trends.

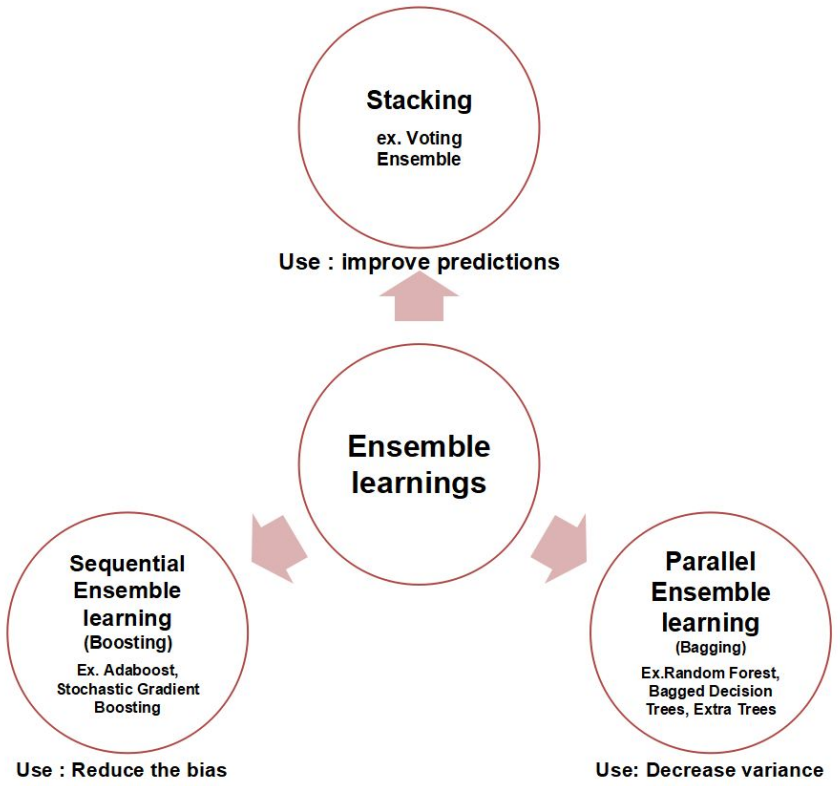
Variance on the other side quantifies how are the prediction made on same observation different from each other. A high variance model will over-fit on your training population and perform badly on any observation beyond training. Following diagram will give you more clarity (Assume that red spot is the real value and blue dots are predictions) :



- Normally, as you increase the complexity of your model, you will see a reduction in error due to lower bias in the model. However, this only happens till a particular point.
- As you continue to make your model more complex, you end up over-fitting your model and hence your model will start suffering from high variance.
-



Ensemble learning Types



Bootstrapping

- Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset.
- Bootstrap involves random sampling of small subset of data from the dataset. This subset can be replace. The selection of all the example in the dataset has equal probability. This method can help to better understand the mean and standard deviation from the dataset.
- Let’s assume we have a sample of ‘n’ values (x) and we’d like to get an estimate of the mean of the sample.

mean(x) = 1/n * sum(x)

We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure:

- Create many (e.g. m) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).
- Calculate the mean of each sub-sample.
- Calculate the average of all of our collected means and use that as our estimated mean for the data.
- For example, let's say we used 3 resamples and got the mean values 2.5, 3.3 and 4.7. Taking the average of these we could take the estimated mean of the data to be 3.5.

Having understood Bootstrapping we will use this knowledge to understand Bagging and Boosting.

Parallel Ensemble Learning(Bagging)

- Bagging, is a machine learning ensemble meta-algorithm intended to improve the strength and accuracy of machine learning algorithms used in classification and regression purpose. It additionally diminishes fluctuation of data(variance)and help to from over-fitting.
- Parallel ensemble methods where the base learners are generated in parallel
- Algorithms : Random Forest, Bagged Decision Trees, Extra Trees

Sequential Ensemble learning (Boosting)

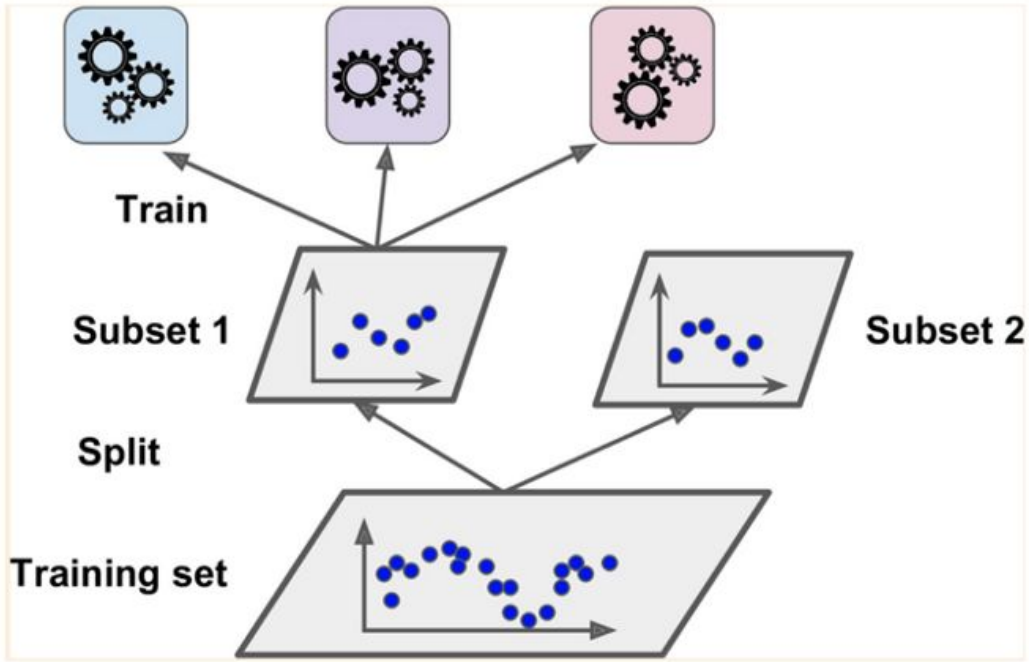
- Boosting, is a machine learning ensemble meta-algorithm for principally reducing bias, and furthermore variance in supervised learning, and a group of machine learning algorithms that convert weak learner to strong ones.
- Sequential ensemble methods where the base learners are generated sequentially.
- Example : Adaboost, Stochastic Gradient Boosting

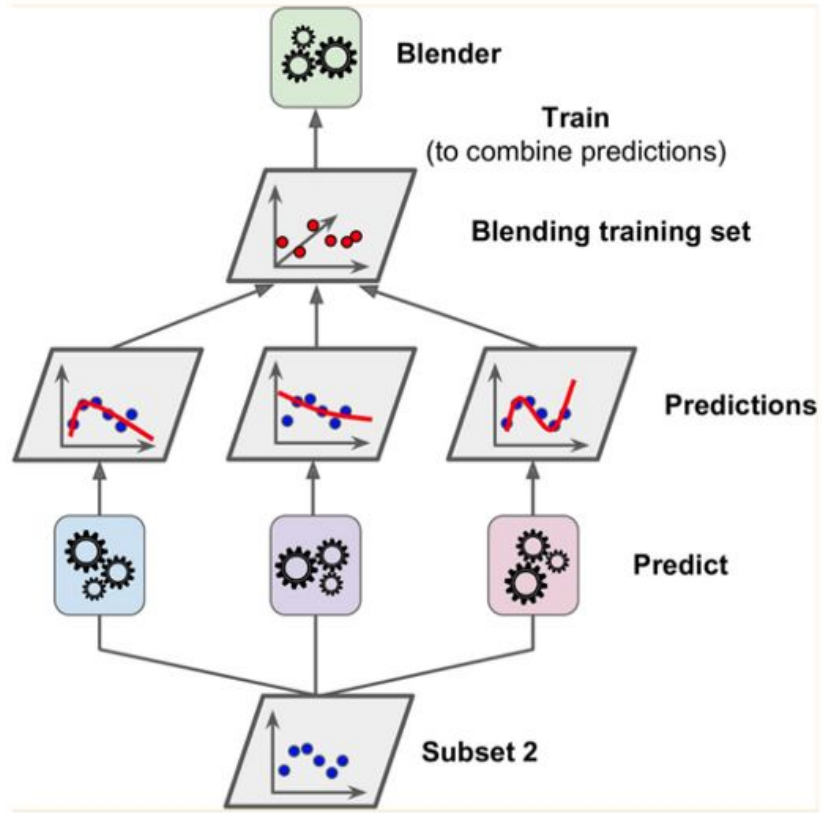
Stacking & Blending

Stacking is a way of combining multiple models, that introduces the concept of a meta learner. It is less widely used than bagging and boosting. Unlike bagging and boosting, stacking may be (and normally is) used to combine models of different types. The procedure is as follows:

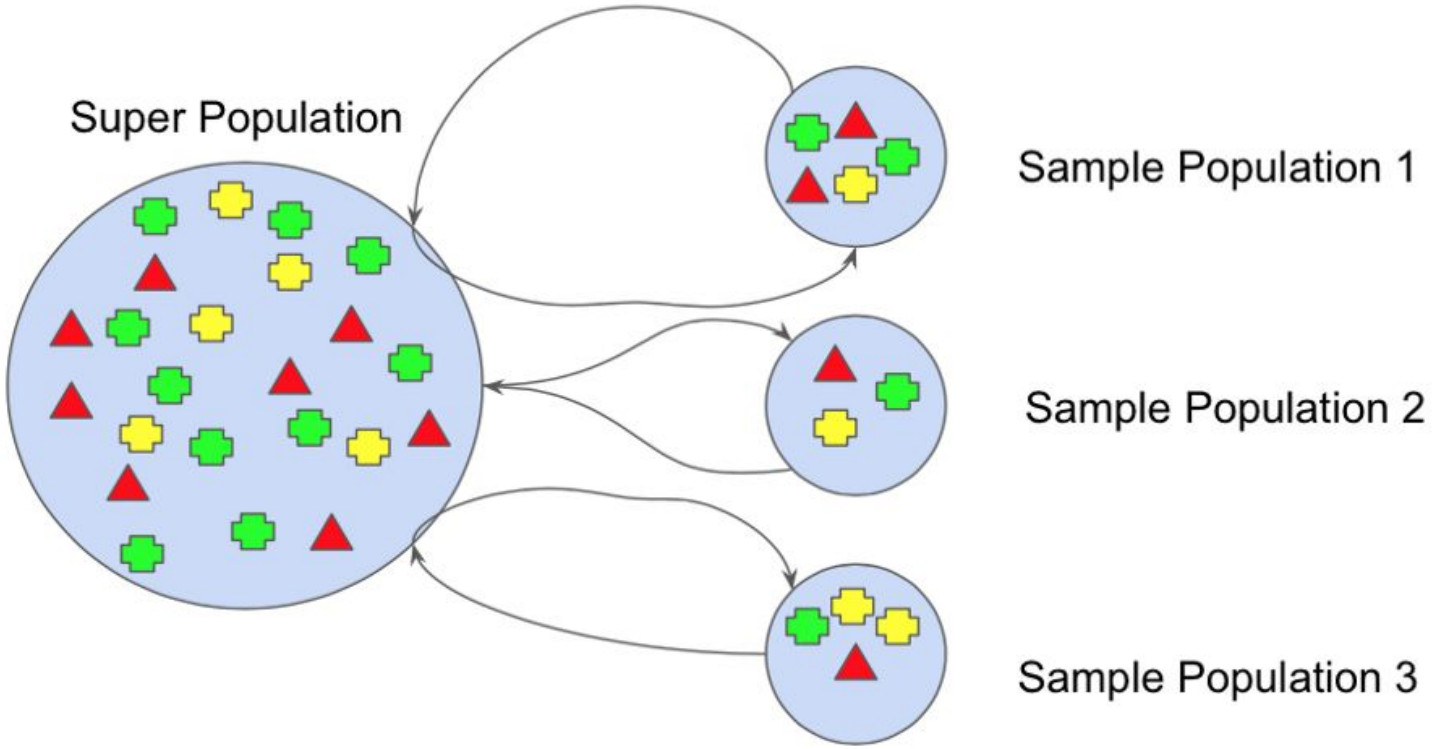
- Split the training set into two disjoint sets.
- Train several base learners on the first part.
- Test the base learners on the second part.
- Using the predictions from Test data sets as the inputs, and the correct responses as the outputs, train a higher level learner.
- Example : Voting Classifier

Blending is technique where we can do weighted averaging of final result.


Train
Subset 1
Subset 2
Split
Training set


Blender
Train
(to combine predictions)
Blending training set
Predictions
Predict
Subset 2

Visual Interpretation of Bootstrapping

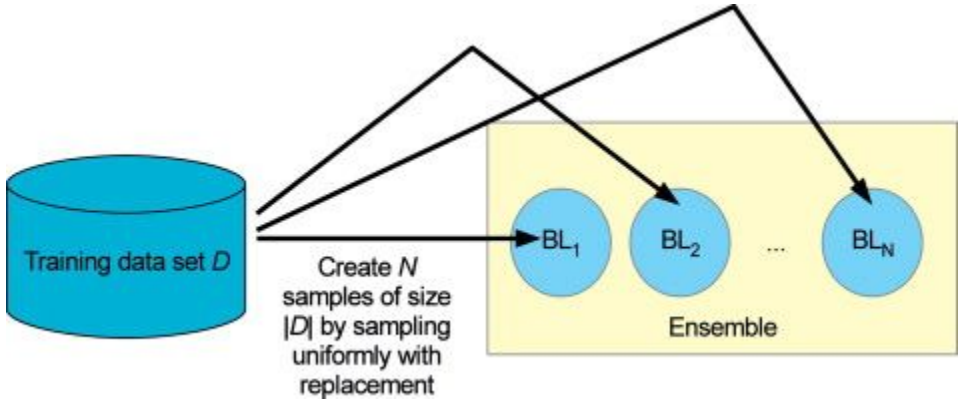


Bagging

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

- Suppose there are N observations and M features. A sample from observation is selected randomly with replacement(Bootstrapping).
- A subset of features are selected to create a model with sample of observations and subset of features.
- Feature from the subset is selected which gives the best split on the training data
- This is repeated to create many models and every model is trained in parallel
- Prediction is given based on the aggregation of predictions from all the models.

- The only parameters when bagging decision trees is the number of samples and hence the number of trees to include.
- This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement



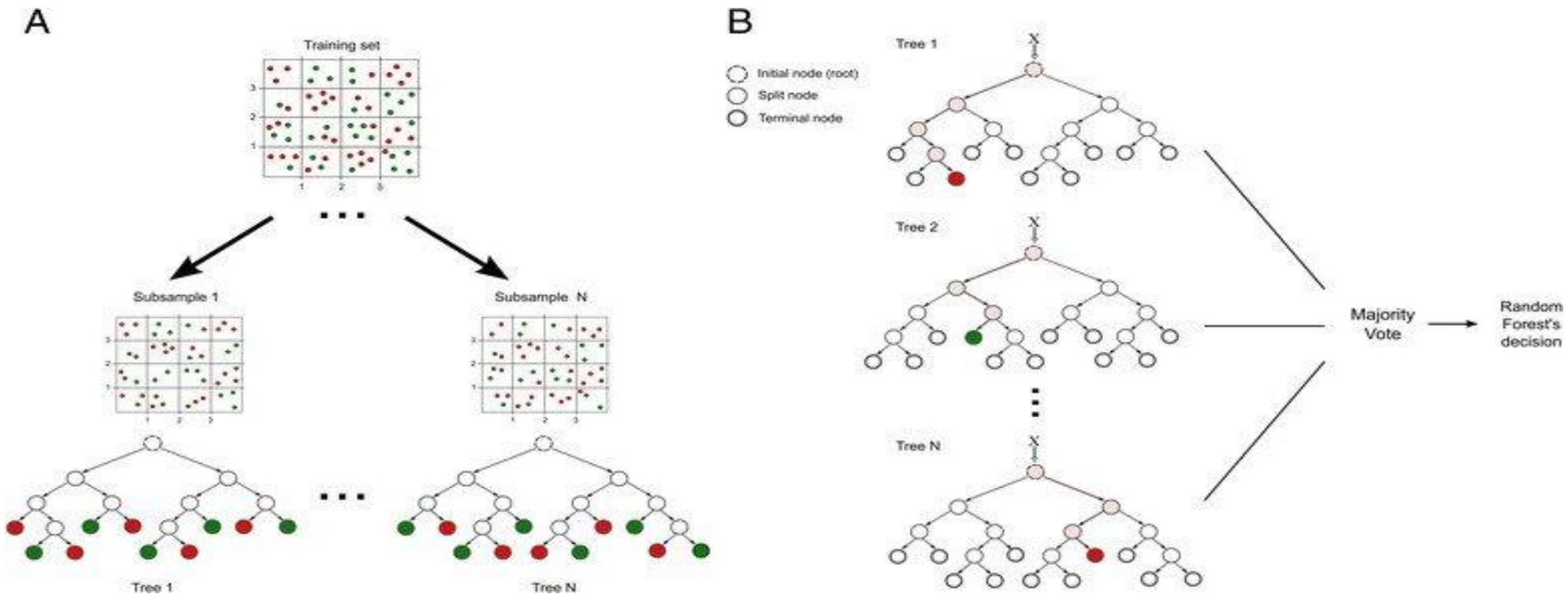
Problems with Bagging

- The problem with Bagging algorithm is it's using CART.
- CART uses Gini-Index, a greedy algorithm to find the best split.
- So we end up with trees that are structurally similar to each other. The trees are highly correlated among the predictions.
- Random Forest address this.

Random Forest Classifier

1. Take a random sample of size N with replacement from the data.
2. Take a random sample without replacement of the predictors.
3. Construct the first CART partition of the data.
4. Repeat Step 2 for each subsequent split until the tree is as large as desired. Do not prune.
5. Repeat Steps 1–4 a large number of times.

Example



- Each decision tree in the ensemble is built upon a random bootstrap sample of the original data, which contains positive (green labels) and negative (red labels) examples.
- Class prediction for new instances using a random forest model is based on a majority voting procedure among all individual trees.

- Bagging features and samples simultaneously: At each tree split, a random sample of m features is drawn, and only those m features are considered for splitting.
- Typically m = $\sqrt{}$ d or log2d, where d is the number of features
- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the “out-of-bag” error rate. random forests tries to improve on bagging by “de-correlating” the trees.
- Each tree has the same expectation.

Original Training Set

Col1	Col2	Col3	Col4	Col5	Col6
1	Sdf	200	A	1	.88
3	Fg	200	A	1	.67
2	Wdv	290	A	1	.36
4	Gh	345	B	0	.85
1	J	125	AB	0	.72
3	Xcv	543	B	0	.93
2	gbn	367	A	1	.18

Training Subsets via Bootstrapping

Col1	Col2	Col4	Col5	Col6
1	Sdf	A	1	.88
3	Fg	A	1	.67

Col1	Col3	Col4	Col5	Col6
1	200	A	1	.88
3	200	A	1	.67

Col2	Col3	Col4	Col5	Col6
Wdv	290	A	1	.36
Gh	345	B	0	.85

Col1	Col2	Col3	Col4	Col5
1	Sdf	200	A	1
2	Wdv	290	A	1

Col1	Col2	Col3	Col5	Col6
3	Fg	200	1	.67
2	Wdv	290	1	.36

Col1	Col2	Col3	Col4	Col6
1	Sdf	200	A	.88
3	Fg	200	A	.67

Col1	Col2	Col3	Col4	Col5
3	Fg	200	A	1
2	Wdv	290	A	1

Col2	Col3	Col4	Col5	Col6
Sdf	200	A	1	.88
Wdv	290	A	1	.36
J	125	AB	0	.72
Xcv	543	B	0	.93

Col2	Col3	Col4	Col5	Col6
Sdf	200	A	1	.88
Fg	200	A	1	.67
J	125	AB	0	.72
Xcv	543	B	0	.93

Hyperparameters

bootstrap : boolean, optional (default=True)

- Whether bootstrap samples are used when building trees.

min_samples_leaf : int, float, optional (default=1)

- *The minimum number of samples required to be at a leaf node:*
- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

n_estimators : integer, optional (default=10)

- The number of trees in the forest.

min_samples_split : int, float, optional (default=2)

- The minimum number of samples required to split an internal node:
- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

max_features : int, float, string or None, optional (default=”auto”)

- The number of features to consider when looking for the best split:
- If int, then consider max_features features at each split. -If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
- If “auto”, then max_features=sqrt(n_features).
- If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).
- If “log2”, then max_features=log2(n_features).
- If None, then max_features=n_features.

max_depth : integer or None, optional (default=None)

- The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

max_leaf_nodes : int or None, optional (default=None)

- Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

Advantages

- Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.
- Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
- You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

Disadvantages

- Random forests is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
- The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.

Finding important features

- Random forests also offers a good feature selection indicator.
- Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction. It automatically computes the relevance score of each feature in the training phase. Then it scales the relevance down so that the sum of all scores is 1.
- This score will help you choose the most important features and drop the least important ones for model building.
- Random forest uses gini importance or mean decrease in impurity (MDI) to calculate the importance of each feature.
- Gini importance is also known as the total decrease in node impurity. This is how much the model fit or accuracy decreases when you drop a variable. The larger the decrease, the more significant the variable is. Here, the mean decrease is a significant parameter for variable selection. The Gini index can describe the overall explanatory power of the variables.

Random Forests vs Decision Trees

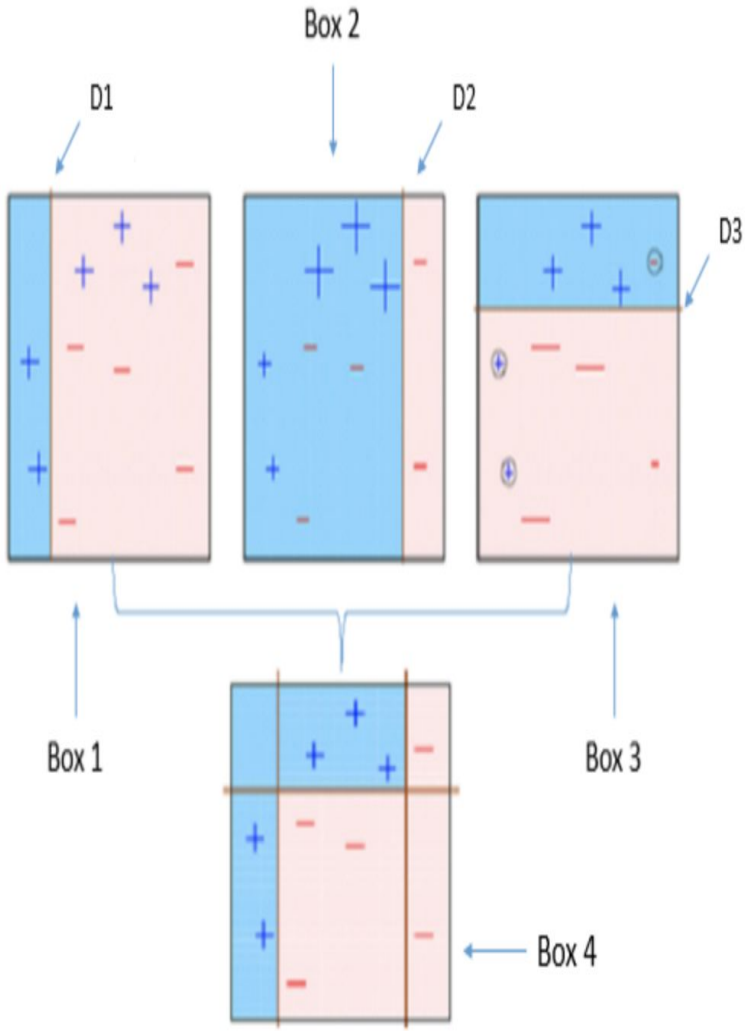
- Random forests is a set of multiple decision trees.
- Deep decision trees may suffer from overfitting, but random forests prevents overfitting by creating trees on random subsets.
- Decision trees are computationally faster.
- Random forests is difficult to interpret, while a decision tree is easily interpretable and can be converted to rules.

Boosting

2.Boosting

- Similar to bagging, boosting also creates an ensemble of classifiers by resampling the data, which are then combined by majority voting. However, in boosting, resampling is strategically geared to provide the most informative training data for each consecutive classifier.
- In essence, each iteration of boosting creates three weak classifiers: the first classifier C1 is trained with a random subset of the available training data.
- The training data subset for the second classifier C2 is chosen as the most informative subset, given C1 . Specifically, C2 is trained on a training data only half of which is correctly classified by C1 , and the other half is misclassified. The third classifier C3 is trained with instances on which C1 and C2 disagree.
- The three classifiers are combined through a three-way majority vote

- Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners.
- Unlike bagging that had each model run independently and then aggregate the outputs at the end without preference to any model. Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.



Box 1: You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or — (minus). The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as — (minus). In such case, we'll assign higher weights to these three + (plus) and apply another decision stump.

Box 2: Here, you can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors. This time with three -(minus). Again, we will assign higher weight to three — (minus) and apply another decision stump.

Box 3: Here, three — (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly. This time a horizontal line is generated to classify + (plus) and — (minus) based on higher weight of mis-classified observation.

Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

Which is the best, Bagging or Boosting?

- Bagging and Boosting decrease the variance of your single estimate as they combine several estimates from different models. So the result may be a model with higher stability.
- If the problem is that the single model gets a very low performance, Bagging will rarely get a better bias. However, Boosting could generate a combined model with lower errors as it optimises the advantages and reduces pitfalls of the single model.
- By contrast, if the difficulty of the single model is over-fitting, then Bagging is the best option. Boosting for its part doesn't help to avoid over-fitting; in fact, this technique is faced with this problem itself. For this reason, Bagging is effective more often than Boosting.