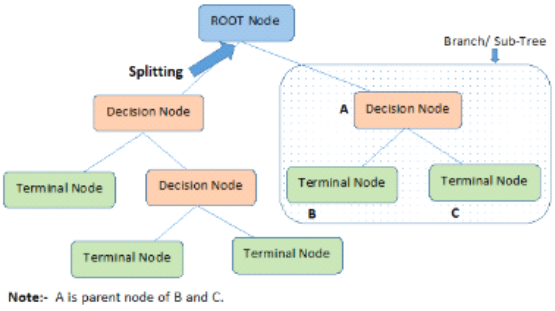
Trees

Saturday, October 20, 2018 2:09 PM

Decision Trees:
Decision tree builds classification or regression models in the form of a tree structure.

1. **Categorical Variable Decision Tree -** the value (class) obtained by terminal node in the training data is the mode of observations falling in that region
2. **Continuous Variable Decision Tree - I**n case of regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.

It breaks down a dataset into smaller and smaller subsets (homogenous subsets) while at the same time an associated decision tree is incrementally developed.



how does it identify the variable and the split - by using various algorithms
The algorithm selection is also based on type of target variables.

Let's look at the four most commonly used algorithms in decision tree:

Algorithms used for Classification:

1. Gini Index
2. Entropy-based (Information Gain)
3. Chi-square

Algorithm used for Regression:

4. Reduction in Variance

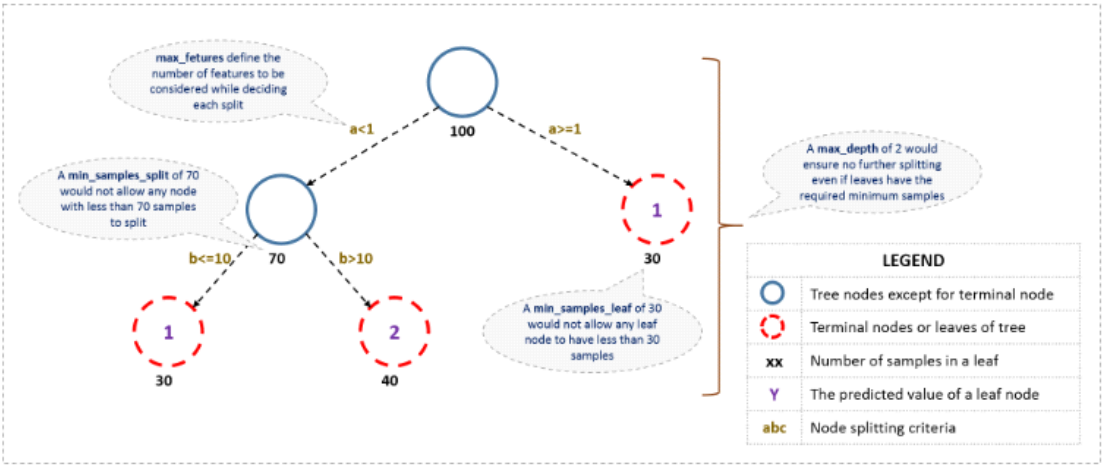
How to tune Decision Trees? / How do we improve the accuracy of DTs without causing over-fitting?

- Overfitting is one of the key challenges faced while modeling decision trees
    - Setting constraints on tree size (max_depth, min_samples_split, min_samples_leaf)
    - Tree pruning (no implementation available in Sklearn)

Why do we do pruning? - To offset the damage caused by the 'greedy approach' in building trees

What are the steps in pruning?

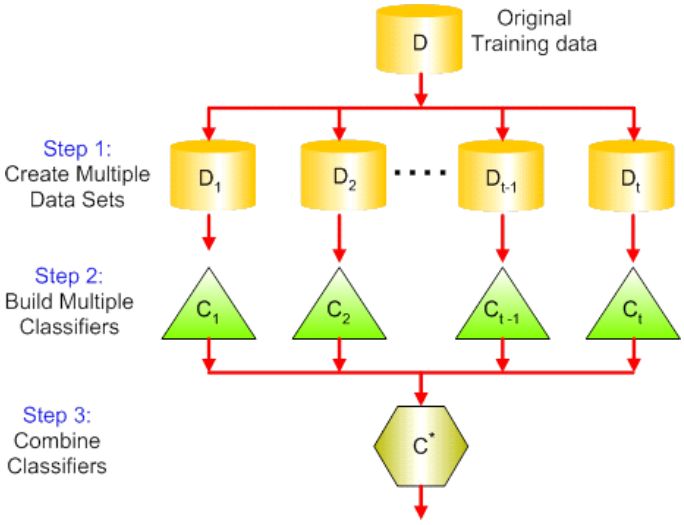
1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

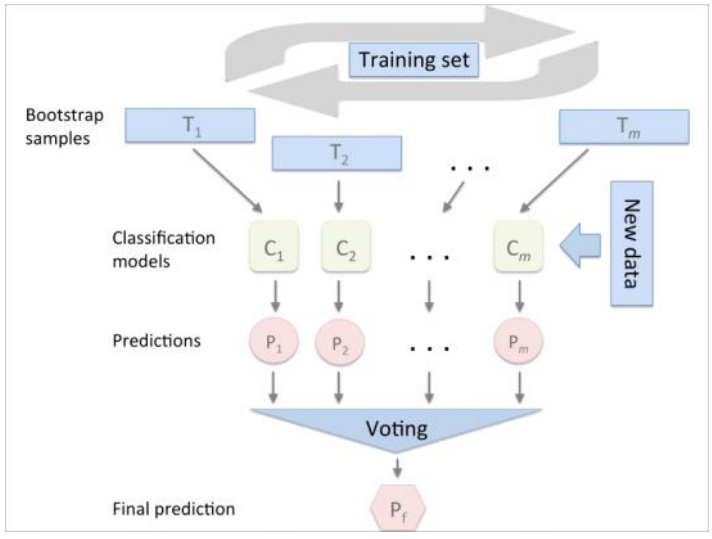


Random Forest = Decision Tree + Bagging:

Bagging = Bootstrap aggregating
Bootstrapping = In statistics, bootstrapping is any test or metric that relies on random sampling with replacement.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.





Algorithm Steps:

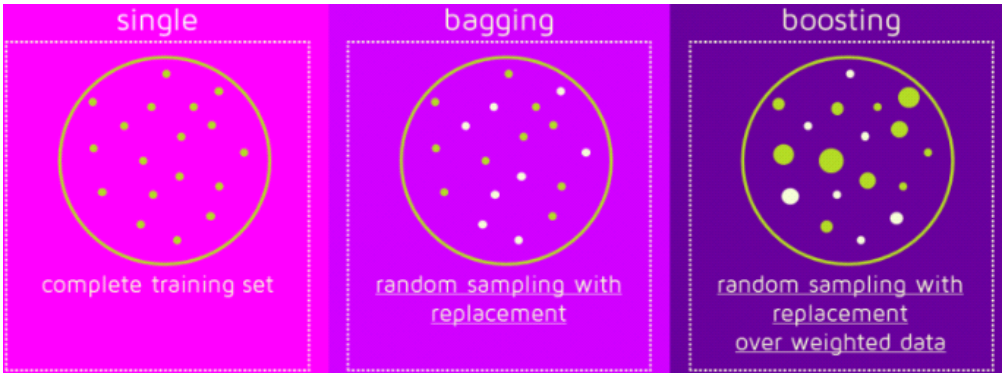
1. **Samples for the n_estimators**= Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. **max_features**=If there are M input variables, a number m<M is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
3. **Training_Stage: Build trees till till_criterion_is_met**: Each tree is grown to the largest extent possible and there is no pruning.
4. **Prediction Stage: Majority_votes/Average**: Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

One distinct advantage of Random Forest:
It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods

Boosting:
The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners.

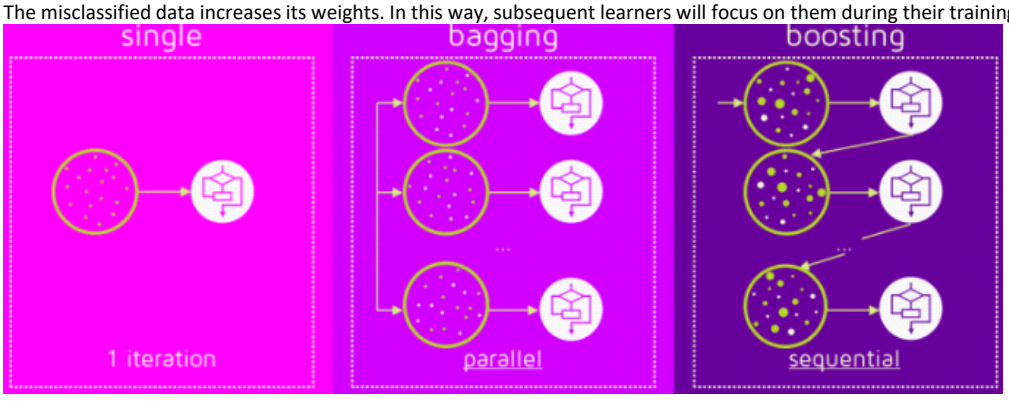
How boosting is different from bagging?

1. Sampling for the n_estimators



2. During Training: Sequential processing

The misclassified data increases its weights. In this way, subsequent learners will focus on them during their training.



3. In the Boosting training stage, the algorithm allocates weights to each resulting model. A learner with good a classification result on the training data will be assigned a higher weight than a poor one.

4. During Prediction phase:
   Boosting assigns a second set of weights, this time for the N classifiers, in order to take a weighted average of their estimates.

   Adaboost vs GBM:
   https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/24b929f0-6819-420e-943e-06fd4ac75813.xhtml

AdaBoost	GradientBoost
Both AdaBoost and Gradient Boost use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of the weak learners.	
In AdaBoost, shift is done by up-weighting observations that were misclassified before.	Gradient boost identifies difficult observations by large residuals computed in the previous iterations.
In AdaBoost "shortcomings" are identified by high-weight data points.	In Gradientboost "shortcomings" are identified by gradients.
Exponential loss of AdaBoost gives more weights for those samples fitted worse.	Gradient boost further dissect error components to bring in more explanation.
AdaBoost is considered as a special case of Gradient boost in terms of loss function, in which exponential losses.	Concepts of gradients are more general in nature.

^ elaborating on the last point, Gradient Boosting can take many objective functions:

From sklearn implementation of GBM:

class `sklearn.ensemble.` **`GradientBoostingClassifier`** (*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto', validation_fraction=0.1, n_iter_no_change=None, tol=0.0001*) [source]

Gradient Boosting for classification.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

Read more in the User Guide.

Parameters: **loss : *{'deviance', 'exponential'}, optional (default='deviance')***

loss function to be optimized. 'deviance' refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss 'exponential' gradient boosting recovers the AdaBoost algorithm.

Process Steps:
Adaboost - Steps 1 & 2 - sampling and training:

- adaptive boosting **changes the sample distribution** by modifying the weights attached to each of the instances. It **increases the weights of the wrongly predicted instances** and **decreases the ones of the correctly predicted instances**. The weak learner thus focuses more on the difficult instances.

Step 3 - Assigning weights to weak classifier:

- During prediction phase, each weak learner is given a weightage based on their predictive power (accuracy)

GBM - Steps 1 & 2 - sampling and training:

- gradient boosting doesn't modify the sample distribution. Instead of training on a newly sample distribution, the **weak learner trains on the remaining errors** (so-called **pseudo-residuals**) of the strong learner. It is another way to give more importance to the difficult instances. At each iteration, the pseudo-residuals are computed and a weak learner is fitted to these pseudo-residuals.

Step 3 - Assigning weights to weak classifier:

- During prediction phase, the contribution of the weak learner (so-called multiplier) to the strong one isn't computed according to his performance on the newly distribution sample but using **a gradient descent optimization proces**s

Xgboost over GBM:

- XGBoost is also known as 'regularized boosting'
- XGBoost allow users to define custom optimization objectives and a lot of other options too (like below)
- https://xgboost.readthedocs.io/en/latest/parameter.html