Decision Tree

- **What it is:** The decision tree algorithm is a supervised machine learning algorithm that can be used for both regression and classification. It takes a flowchart-like structure in which each internal node represents a test on a feature, each leaf node represents a class label (classification) or a continuous value (regression) and branches represent conjunctions of features that lead to those labels/values.
- **How it works:** Decision tree involves carving up the predictor space into a number of simple regions (boxes) and then assigns a score/label for each box. The model building process decides where to draw the boxes and what the score/label in each box should be. More specifically, we first grow a large tree on the training data until the stopping criterion is reached (low enough total impurity or minimum number of data points in leaf). At each step of the tree-building (carving boxes into smaller boxes), we look for the best split at that particular step[1] by sliding the candidate cut point in each dimension until one of the resulting boxes has fewer than some minimum number of observations. The best split is a particular location in a particular dimension(variable) that minimizes the resulting total impurity[2] in the two boxes. For regression problem, impurity is usually measured by variance; for classification problems, impurity is commonly measured by Gini index and entropy or information gain. Second, cost complexity pruning is applied to the large tree to obtain a sequence of best subtrees[3]. And the final subtree is chosen based on the value of tuning parameter alpha.
- **Important tuning parameters**: Important user-chosen parameters are splitting criterion, max depth, min split size, min leaf size.
- **Pros and Cons of the model**: Generally, decision trees are fragile (if build slightly different) and unstable, prone to overfitting. Decision tree works well when the problem naturally divides by lines parallel to the different axes. And it doesn't well when the separation lines are at angles to the axis, or if the separation lines are substantially curved.

---

[1] rather than looking ahead and picking a split that will lead to a better tree in some future step (recursive binary splitting)

[2] sum of impurity of the box times number of records in that box

[3] the error function is prediction error + tuning parameter alpha*number of terminal nodes. Alpha is a nonnegative float. We tune the alpha with K-fold cross validation method, training and pruning the large trees on 1 to K-1th fold (getting a sequence of trees each responding to different value of alpha) and evaluating the MSE on the Kth fold. We pick the alpha that minimize the avg MSE on the validation set.

(Gradient) Boosted Tree

- **What it is:** A strong learner as a result of training a series of weak decision tree models. Boosting is a way of training a series of weak learners to result in a strong learner. Any weak learner can be used but trees are the most common.
- **How it works:** Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting regression approach instead learns slowly. Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals, rather than the outcome Y, as the response (trained to predict the residual error of the current model). We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm. By fitting small trees to the residuals, we slowly improve f in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals. In general, statistical learning approaches that learn slowly tend to perform well[4]. Adaptive boosting increases the weights on misclassified records so the next iteration can pay more attention to them
- **Important tuning parameters**: Important user-chosen parameters are loss function, number of trees B[5], max depth d[6],
  the shrinkage parameter λ[7].

[4] Note that in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown

[5] Boosting can overfit if number of trees is too large

[6] The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

[7] A small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance

Random Forest

- **What it is:** Random forest an ensemble of strong tree learners, each is a full model to predict the output. The final output is an average (regression) or vote (classification) across all the strong models.
- **How it works:** Build many independent decision trees that predicts the output, each has some randomness associated with it (either using only a randomly-chosen subset of variables for each tree OR using only a randomly-chosen subset of variables for each split iteration). Take the latter one as an example, each time a split in a tree is considered, a random sample of **m** predictors is chosen as split candidates from the full set of **p** predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $\mathbf{m \approx \sqrt{p}}$—that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors. In other words, in building a random forest, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.
- **Pros**: By adopting the ensemble method, random forests provide improvement related to robustness, stability and generalization[8]. Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors.

[8] Reasoning: random forest provides an improvement over bagged trees (which uses a random/bootstrapped subset of training records) by way of a small tweak that decorrelates the trees. Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of **bagged trees**, most or all of the trees will use this strong predictor (variable X) in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting. **Random forests overcome this problem** by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable. The main difference between bagging and random forests is the choice of predictor subset size **m**. For instance, if a random forest is built using $m = p$, then this amounts simply to bagging.

Neural Net

- **What it is:** Neural net is a supervised machine learning algorithm that maps inputs to an output with multiple adjustable parameters. A typical neural net consists of an input layer, some number of hidden layers and an output layer.
- **How it works:** All the independent variables (x's) form the input layer. The dependent variable y is the output layer. The hidden layer is a set of nodes or *neurons*. Each node in the hidden layer receives weighted signals from all the nodes in the previous layer and does a transform (e.g. sigmoid) on this linear combination of signals (weighted). The transform/activation function can be a logistic function (sigmoid[9]), or something else. The weights are trained by backpropagating[10] the error. That is, data is shown to the neural net, record by record. For each training record, every step for building neural network involves a guess, an error measurement (typical error/loss function is the square of the errors[11]) and a slight update in its weights (an incremental adjustment to the coefficients) as it slowly learns to pay attention to the most important features. The entire data set is passed through many times as the weights settle into a local optimum, each complete pass is called a training epoch. The essence of neural net is adjusting a model's weights in response to the error it produces, until you can't reduce the error any more[12].
- **Important tuning parameters:** # hidden layers, # nodes/layer, transform/activation function

9 Sigmoid activation function $= \frac{1}{1+\exp(-\sum a_i x_i)}$

10 At each layer, starting from the end, one propagates the error backwards to each node, and we calculate the gradient/slope of the error with respect to the node weights (how does the error vary as the weight is adjusted).

11 Typical loss function for NN: $E(y, \hat{y}) = |y - \hat{y}|^2$

12 https://pathmind.com/wiki/neural-network

Autoencoder

- **What it is:** The autoencoder is an unsupervised artificial neural network that is trained to efficiently compress and encode the input data and reconstruct the encoded data back a representation that is as close to the original input as possible. This method is usually used to detect anomalies in the input data.
- **How it works:** Autoencoder works in very similar way as a typical neural net, except that the output is the input data itself. By training an autoencoder on the entire data set. The model will learn to reproduce the data records as well as possible and will learn the nature of the input data. The records that aren't reproduced well are interpreted as unusual records, or anomalies. Therefore, a measure of the reproduction error (MSE)[13] is a measure of unusualness for that record and is thus a fraud score[14].

[13] Reproduction error: $s_i = \left(\sum_k |z_k'^i - z_k^i|^n \right)^{1/n}, \quad n \text{ anything}$

[14] https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726