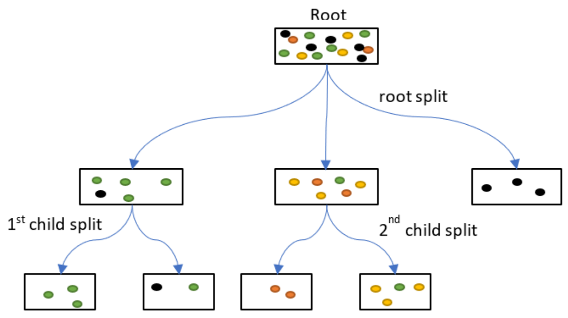
Decision Trees and Ensemble Learning Methods

Over the past few decades, ensemble learning methods have grown significantly and have been centered around aggregating decision trees. Classification and Regression Trees (CART) are a class of decision tree algorithms that are used for classification or regression problems. The CART model is represented by a binary decision tree in which each node represents a new decision. CART models are split at their nodes based on the Gini Impurity Test, which is given by $Gini = 1 - \sum_{i \in N} p_{i,k}^{2}$, where N is the set of all possible classes for the dependent variable, K is a category in an independent variable, and $p_{i,k}$ is the probability of the category K having class i. Once this equation is computed for every class in a given feature, the weighted sum of Gini scores for each class is computed, known as the Gini Index. The Gini index for each feature can be calculated, and the feature with the lowest Gini Index is selected as the feature that will dictate the first split of the tree. At each node, this process is repeated until a stopping point is reached. The stopping point is typically defined by two specified criteria: a certain percentage of data should be in the leaf nodes, typically 5%, or a max tree depth can be set by the user. The number of branches and nodes that a CART model will have depends on the noise in the features of the dataset and the number of features that the dataset has, and there is no set number.



Ensemble learning is a type of machine learning where multiple models, which are called weak learners, are combined to create a stronger model that can solve the same problem with more accuracy. Weak learners often have high variance or bias, and the goal of ensemble learning is to reduce bias or variance by combining weak learners into a strong learner. Bagging is one way that we can combine weak learners to build a strong learner.

Bagging stands for “bootstrap aggregating”, and it is a parallel ensemble method that trains each weak learner independently or concurrently. Bootstrapping is a statistical method that randomly draws samples of size *n* from an initial dataset without replacement; these samples are bootstrap samples that are meant to be representations of the true data distribution. In an ensemble learner, individual trees are trained for each bootstrap sample, such that if there are *T* samples, we can obtain a sequence of *T* outputs, $f_1(x), \ldots f_T(x)$. In a regression problem, given the bootstrap sample that each tree is now trained with, the models can be aggregated to produce a final output as $\overline{f(x)} = \frac{1}{T}\sum_{i=1}^{T} f_i(x)$, the average of the outputs for a given input for each tree. For classification problems, a majority vote, $\overline{f(x)} = sign(\sum_{i=1}^{T} sign(f_i(x)))$, is used by choosing the class that receives the majority of the votes as the output for the strong learner.

In a random forest model, multiple trees are combined to produce a forest of CART models to generate outputs with lower variance. CART models are trained on bootstrap samples, however, in a random forest model, samples are bootstrapped over both features and observations. This helps reduce correlation between trees because by sampling over features of the dataset, not all CART models look at the same information. The outputs from each tree are then averaged to produce a single output from the random forest model, our strong learner. Bagging is an ensemble learning method that looks to reduce variance, and this means that weak learners often have high variance individually and are combined to have a lower variance. With the random forest model, CART models that are deeper with more branches would be used as the base model because deeper trees typically have higher variances.

Boosting works in a similar fashion to bagging in that it looks to combine multiple weak learners into a strong learner, however, it differs in that boosting is not a parallel learning method, and so weak learners are not fitted independently. Instead, weak learners are trained sequentially, meaning that as weak learners are trained, new models are fitted by focusing on the observations that were poorly handled by the previous model. Thus, boosting is an adaptive process: as the next model is fitted, it adapts to the previous model and assigns more importance to the observations that the previous model struggled with. Boosting focuses on reducing bias, and so weak learners are often chosen to have high bias and low variance. Regarding CART models, shallow trees are used as base models because they have high bias with lower variance.

Specifically, boosting is typically implemented in one of two ways: adaptative boosting and gradient boosting. Adaptative boosting, or AdaBoost, uses a forest of tree stumps, which differ from regular trees in that they are usually cut to about one node and two leaves. In its first iteration, AdaBoost trains a tree stump for the dataset by weighting each observation equally and simply fitting a tree stump. In the next iteration, the tree stump is trained by assigning higher weights to the observations that were more difficult to classify in the first tree stump. Mathematically, AdaBoost works as follows. Consider the dataset $x_i \in R^n, y_i \in \{-1, 1\}$, where *x* represents the dataset with *n* attributes and *y* is a binary dependent variable. For the first iteration, all of the observations will have a weight $w = 1/N \in [0, 1]$, where *N* is the number of observations. Classification error is then computed for each sample as $\alpha = \frac{1}{2} ln(\frac{1-\varepsilon}{\varepsilon})$, where ε is the error of the tree stump for the observation. The weight of an observation for the next iteration will then become $w_i = w_{i-1} + e^{\pm\alpha}$. This process is then repeated for a given number of iterations. After a set number of iterations, the final forest should be trained to accurately classify observations.

Gradient boosting trains decision trees through boosting, but differs from AdaBoost in that Gradient boosting is based on a loss function, which computes the residuals of the model's outputs. As each new tree is constructed, the parameters of the tree that will minimize the loss function of the ensemble are selected through a gradient descent algorithm. Thus, gradient boosting continuously attempts to reduce the error of the ensemble model by minimizing the loss function. The process goes as follows: first a tree that only has one node is built, typically using the average value of the target variable. Next, the residuals of this tree node are computed; a larger tree, maybe with three or four leaves, is trained to predict the residuals of the original leaf. The residuals predicted by the new tree are scaled by a learning rate (which is fixed by the user), and then added back to the original prediction of the leaf to make new predictions. It is important to keep in mind that the learning rate is a number from 0 to 1, and a smaller learning rate improves accuracy but increases run time and efficiency when implemented; the default is 0.01.

Next, a new tree is constructed to predict the residuals of the previous tree, and these residuals should be smaller than the previous tree. Again, the residuals are multiplied by the learning rate and added back to the predictions of the previous tree to make new predictions. Each new tree is taking a step towards getting better predictions until the process is halted.

Mathematically, the gradient boosting algorithm works as such. A differentiable loss function is defined as $L(y_i, F(x)) = \frac{1}{2}(y_i - f(x))^2$, where y_i is the observed value and $F(x)$ is the predicted. The model is first initialized as $F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$, which produces a constant value γ that minimizes the loss function. This is the initial leaf that was described above, and it is usually the average of the target variable. To start building the first tree, first the residual for each prediction of the initial model, $r_{im} =- \left[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}\right] for\ i = 1,\ldots, n$, (this derivative is the "gradient" in gradient boost) where *i* is the observation number and *m* is the tree number, is computed. Next, a tree is fit to the residuals, r_{im} , and terminal regions $r_{j,m}$, where *j* is the number of the leaf in the new tree, are created. For each leaf in the tree, the output value is now calculated as $\gamma_{jm} = argmin_\gamma \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x) + \gamma)$. This function is funding the output for each leaf of the first regression tree that minimizes the loss for the leaf. The final predictions for the first tree are then made as $F_m(x) = F_{m-1}(x) + \eta(\gamma_{jm})$, where η is the learning rate. This is the first iteration of the gradient boosting model. The next iteration would have *m* = 2 and repeat this process again.

When simple gradient boosting is implemented, the Gradient Boosting Machine (GBM) algorithm is used. Extreme Gradient Boosting, or XGBoost, is an algorithm that is used to implement a more efficient method of gradient boosting. The main difference and advantage of XGBoost over GBM is that it utilizes regularized boosting, which prevents overfitting of any tree in the boosting process and allows for the model to be more generalized. This helps the model perform better during testing because it is not overfitted to the training set and it can be generalized to the test set, while GBMs tend to be overfitted to the training set. XGBoost also uses cross validation at each iteration, which allows for the model to evaluate its performance at each iteration to find the optimal stopping point, and parallel processing, which allows for XGBoost to be used with large datasets. In fact, one of the main points that makes XGBoost so useful is its efficiency.

References

Brownlee, Jason. “A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning.” *Machine Learning Mastery*, 14 Aug. 2020, https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/.

Dobilas, Saul. “CART: Classification and Regression Trees for Clean but Powerful Models.” *Medium*, Towards Data Science, 10 Mar. 2022, https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models-cc89e60b7a85.

Rocca, Joseph. “Ensemble Methods: Bagging, Boosting and Stacking.” *Medium*, Towards Data Science, 21 Mar. 2021, https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

Singh, Harshdeep. “Understanding Gradient Boosting Machines.” *Medium*, Towards Data Science, 4 Nov. 2018, https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

Victor Zhou. “A Simple Explanation of Gini Impurity.” *Victor Zhou*, Victor Zhou, 29 Mar. 2019, https://victorzhou.com/blog/gini-impurity/.

Yiu, Tony. “Understanding Random Forest.” *Medium*, Towards Data Science, 29 Sept. 2021, https://towardsdatascience.com/understanding-random-forest-58381e0602d2.