Decision Trees & Random Forests

Jerry Peng

1 Decision Trees

A decision tree is a flowchart-like structure where each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. The figure 1 shows a simple example of decision tree.

Figure 1: Decision Tree Classifier

1.1 Building a Tree

1.1.1 Feature Selection

While there are multiple ways to select the best attribute at each node, two methods, information gain and Gini impurity, act as popular splitting criterion for decision tree models. They help to evaluate the quality of each test condition and how well it will be able to classify samples into a class.

Entropy (Information Gain) Used to measure the disorder or uncertainty in the data. The entropy of set S is calculated by using the formula:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where p_i is the proportion of the number of elements in class i to the number of elements in set S.

Gini Impurity Another measure used for decision making in trees, defined as:

$$G(S) = 1 - \sum_{i=1}^{n} p_i^2$$

Information Gain Calculated as the difference in entropy from before to after the set S is split on an attribute A. The higher the information gain, the better the attribute:

$$Gain(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

where S_v is the subset of S for which attribute A has value v

1.1.2 Splitting Nodes

The tree is constructed in a top-down recursive manner. At each step, the best split is chosen by selecting the feature with the highest information gain or lowest Gini impurity.

1.2 Pruning the Tree

Once a tree is built, it might have overly complex branches that do not generalize well from the training data. Pruning methods, like cost complexity pruning, are used to remove these branches and prevent overfitting.

1.2.1 Cost Complexity Pruning

Cost complexity pruning reduces the size of a decision tree by considering the trade-off between the tree's complexity and its fit to the training data.

Subtree Replacement: Begin by considering the smallest subtrees (those closest to the leaves) and decide whether or not to replace them with a leaf node. This decision is based on a complexity parameter α, which balances the trade-off between tree size and accuracy.

Complexity Parameter (α) Calculation The idea is to define a cost function:

$$C_\alpha(T) = C(T) + \alpha \times \text{number of leaves}$$

where $C(T)$ is the misclassification rate of the Tree T

Pruning Process The tree is pruned in such a way that the cost $C_\alpha(T)$ is minimized. For each internal node, the cost of pruning and not pruning is compared. If the cost of pruning is less than not pruning (i.e., replacing the subtree with a leaf node reduces the cost), the subtree is replaced.

1.3 Type of Decision Trees

1.3.1 Classification Trees

Classification trees are used for predicting discrete class labels. They are applicable when the output variable is categorical (like 'yes' or 'no', 'red' or 'blue').

Node Splitting Each node in the tree represents a question or a test on an attribute, which splits the data into subsets. The choice of which attribute to split on is based on measures like Gini impurity or information gain.

1.3.2 Regression Trees

Regression trees are used for predicting continuous quantities. They are applicable when the output variable is numerical (like height, price, or temperature).

- **Node Splitting**: Similar to classification trees, but instead of reducing Gini impurity or maximizing information gain, the decision criteria are based on minimizing the variance within each node. The measure used to decide the splitting point is the reduction in variance.

$$\text{Variance Reduction} = Var(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Var(S_v)$$

- **Prediction** In each leaf node, the prediction for regression trees is usually the mean or median of the values in that leaf.

2 Bagging & Boosting

2.1 Bagging

Definition Bagging is a method designed to improve the stability and accuracy of machine learning algorithms. It reduces variance and helps to avoid overfitting. Typically applied to decision tree methods, but it can be used with any type of method.

How it works?

1. **Bootstrap Sampling:** Multiple subsets of the original dataset are created using bootstrap sampling (random sampling with replacement).
2. **Model Building:** A separate model is trained on each of these subsets.
3. **Aggregation:** The individual models' predictions are then combined through averaging (for regression) or majority voting (for classification) to form a final prediction.

Reduction in variance Bagging reduces the variance of the prediction by aggregating the results of multiple models, each trained on slightly different data.

2.2 Boosting

Definition Boosting is an ensemble technique that aims to create a strong classifier from a number of weak classifiers. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until no further improvements can be made.

How it works?

1. **Sequential Model Training:** Unlike bagging, boosting involves incrementally building an ensemble by training each new model instance to emphasize the data points that previous models misclassified.
2. **Weight Updating:** In each iteration, the algorithm assigns higher weights to the incorrectly predicted observations, so that subsequent models focus more on difficult cases.
3. **Model Aggregation:** The final model prediction is typically a weighted sum of the individual predictors.

Reduction in bias Unlike bagging, boosting primarily works by reducing bias (and also variance to a lesser extent), building a highly accurate prediction rule by focusing on the most challenging parts of the training data.

3 Random Forest

A Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. It's primarily used for classification tasks but can also be applied to regression problems. The idea is to build multiple decision trees during training and output the mode of the classes (classification) or mean prediction (regression) of the individual trees for unseen data.

3.1 How does it work?

Bootstrap Sampling For each tree in the forest, a random sample of the data is selected using bootstrap sampling (sampling with replacement). This means each tree in the forest is built on a slightly different dataset.

Feature Randomization When building each tree, at each node, instead of searching for the most significant feature among all features, Random Forest randomly selects a subset of features and finds the best split among these features. This adds an additional layer of randomness to the model, which helps in reducing variance.

Tree Building Each tree in the forest is built to the largest extent possible (fully grown and unpruned). The randomness in feature selection and the bootstrap sampling leads to different trees.

Voting/Averaging For classification, each tree in the forest "votes" for a class, and the class receiving the most votes is the model's prediction. For regression, the average output of all trees is taken as the prediction.

3.2 Mathematical Perspective

Reduction in Variance The key mathematical idea behind Random Forest is that the ensemble of trees, each trained on different subsets of the same data, reduces the variance of the prediction without increasing the bias. This is because while each individual tree might have high variance, the average of many trees will have lower variance.

Law of Large Numbers The underlying principle is akin to the law of large numbers in statistics. As the number of independently trained trees in the forest increases, the collective decision made by the forest converges to a more stable and accurate prediction.

4 XGBoost

XGBoost, which stands for eXtreme Gradient Boosting, is an advanced implementation of gradient boosting algorithm. It has gained popularity in machine learning competitions for its efficiency, performance, and scalability.

XGBoost is based on the gradient boosting framework, which builds models sequentially, with each model attempting to correct the errors made by the previous ones. XGBoost has been optimized in terms of computational speed, use of memory, and overall performance. This is one of the reasons why it often outperforms other implementations of gradient boosting.

4.1 How does it work?

Sequential Model Training: Like other boosting methods, XGBoost trains models sequentially. Each new model incrementally reduces the loss function (a measure of how well the model is performing) of the entire system using gradient descent.

Gradient Descent and Boosting: The core principle is to build new models that predict the residuals or errors of prior models and then add them together to make the final prediction.

Regularization: XGBoost includes a regularization term in the loss function, which helps in reducing overfitting. This is a key difference from traditional gradient boosting.

Handling of Sparse Data: It can handle sparse data (missing values, zero values) and has built-in routines to handle such scenarios.

Tree Pruning: Unlike standard gradient boosting, which stops splitting a tree when it is no longer beneficial to do so, XGBoost actually grows the tree up to a certain depth and then prunes it back, which can result in more optimal trees.

System Optimization: XGBoost uses advanced computing techniques such as parallelization, cache awareness, and hardware optimization to improve efficiency.

4.2 Objective Function

XGBoost is an additive model composed of k base models. Assume that the tree model we want to train for the t-th iteration is $f_t(x)$, then there is:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where

- $\hat{y}_i^{(t)}$: the prediction result come from the model after t-th iteration based on the sample i
- $\hat{y}_i^{(t-1)}$: the prediction results from the tree $t-1$
- $f_t(x_i)$: the function/model for the tree t

The loss function for XGBoost with $\hat{y}_i$ and the corresponding true value y_i is defined as:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

where n is the number of the sample

Hence, the objective function for XGBoost is

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{i=1}^{t} \Omega(f_i)$$

where $\sum_{i=1}^{t} \Omega(f_i)$ is the sum of the complexity of t trees, and it is added to the formula as the regularization term

Since XGBoost is an algorithm in the boosting family, it follows forward step-by-step addition. Using the t-th model, the prediction function is

$$\hat{y}_i^t = \hat{y}_i^{(t-1)} + f_t(x_i)$$

and now we can revise the objective function to

$$\begin{aligned} Obj^t &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \sum_{i=1}^{t-1} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

By using Taylor's Theorem, the loss function can be rewritten to

$$l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) = l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)$$

where

- g_i is first-order derivative of the loss function

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$$

- h_i is second-order derivative of the loss function

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$$

By bringing the above second-order expansion into the objective function of XGBoost, the approximate value of the objective function can be obtained:

$$Obj^{(t)} \simeq \sum_{i=1}^{t}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

and by removing the constant values, we get the final objective function as

$$Obj^{(t)} \simeq \sum_{i=1}^{t}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Hence, we only need to find the first-order and second-order derivatives of the loss function at each step, and then optimize the objective function to get $f(x)$ at each step. Finally, we can get an overall model based on the additive model.

For more detailed information, please refer to the link https://zhuanlan.zhihu.com/p/83901304

References