COMP 642 — Machine Learning February 07, 2023

Lecture 9

Lecturer: Anshumali Shrivastava Scribe By: Ziang Tang, Jiehui Li, Harshit Khanna

Disclaimer: *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

1 Decision Trees

Decision Trees are a crucial algorithm for machine learning in predictive modeling. This technique has been around for decades, with modern variations such as random forest being among the most effective methods available.

1.1 Classification and Regression Trees

Classification and Regression Trees or CART for short is a term introduced by Leo Breiman to refer to Decision Tree algorithms which can be used for both classification and regression problems in predictive modeling. Although it is traditionally known as "decision trees," some platforms, such as R, use the more modern term CART. The CART algorithm forms the basis for other advanced algorithms such as bagged decision trees, random forest, and boosted decision trees.

The representation of the CART model is a binary tree, which can be thought of as a simple binary tree from algorithms and data structures. The root node represents a single input variable and its split point, while the leaf nodes contain the output variable used for predictions. For instance, if a dataset has two inputs, height and weight, and the output is sex (male or female), a simple binary decision tree would look like this:

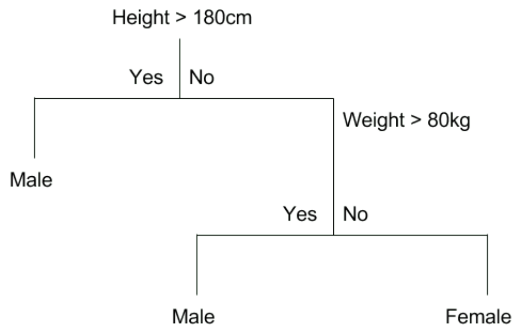


Figure 1: Example Decision Tree

With this binary tree representation of the CART model, making predictions is relatively straightforward. For a new input, the tree is traversed by starting at the root node and evaluating the specific input. A learned binary tree can be thought of as a partitioning of the input space, with each input variable representing a dimension. The decision tree splits this space into rectangles (when there are two input variables) or hyper-rectangles (for more inputs). New data filters through the tree and lands in one of these rectangles, and the output value for that rectangle is the prediction made by the model, giving a boxy decision boundary.

For example, given the input of [height = 190 cm, weight = 90 kg], we would traverse the above tree as follows:

Height > 180 cm: Yes

Weight > 80 kg: Yes

Therefore: Male

It is very important to note some properties of Decision Trees:

$$\begin{gathered} D_1 \bigcap D_2 = \phi \\ D_1 \bigcup D_2 = D \end{gathered} \tag{1}$$

where D = The parent node D1, D2 = child nodes

Decision Trees and CART are simple to understand and interpret, and can handle both numerical and categorical variables. They are also robust to outliers and can handle non-linear relationships between the input and target variables. However, they can be prone to overfitting, especially if the tree is allowed to grow too deep, so various techniques, such as pruning or ensemble methods like Random Forest, are used to improve their performance.

1.2 Strategy of Greedy Algorithm

Creating a CART model involves choosing input variables and the best split points on those variables to form a tree structure. This process is accomplished using a greedy algorithm that minimizes a cost function. The tree construction stops when a predetermined stopping criterion is reached, such as a minimum number of training examples in each leaf node.

Building a binary decision tree involves dividing the input space using a technique called recursive binary splitting, which involves trying and testing different split points using a cost function. The split with the lowest cost is selected for each input variable and all possible split points.

For regression predictive modeling problems, the cost function minimized in selecting split points is the sum of the squared errors between the actual output values and the predicted values for all training examples within a rectangle. This cost function is calculated as the $\sum(y - prediction)^2$, where y is the actual output for a training example and prediction is the predicted output for that example.

For classification the Gini index function and Entropy function can be used which provide an indication of how "pure" the leaf nodes are (how mixed the training data assigned to each node is). First let's see the Entropy function.

$$E = \sum_{i=1}^{K}(-f_i * \log f_i)$$

Where E is the entropy over all classes, f_i are the proportion (or frequency) of training instances with label i in the rectangle of interest. We can then calculate the information gain by

$$IG(parent, left, right) = E(parent) - [n_1E(left) + n_2E(right)]$$

Where $n_1E(left) + n_2E(right)$ is the weighted average of entropy for each node. In the greedy algorithm every time we choose the attribute to split on which can result in the least disorder (entropy) and the most information gain. To further illustrate this splitting criterion, let's see an example.

Consider an example where we are building a decision tree to predict whether a loan given to a person would result in a write-off or not. Our entire population consists of 30 instances. 16 belong to the write-off class and the other 14 belong to the non-write-off class. We have two features, namely "Balance" that can take on two values: "<50K" or ">50K", and "Residence" that can take on three values: "OWN", "RENT" or "OTHER". I'm going to show you how a decision tree algorithm would decide what attribute to split on first and what feature provides more information, or reduces more uncertainty about our target variable out of the two using the concepts of Entropy and Information Gain.

Feature1: Balance

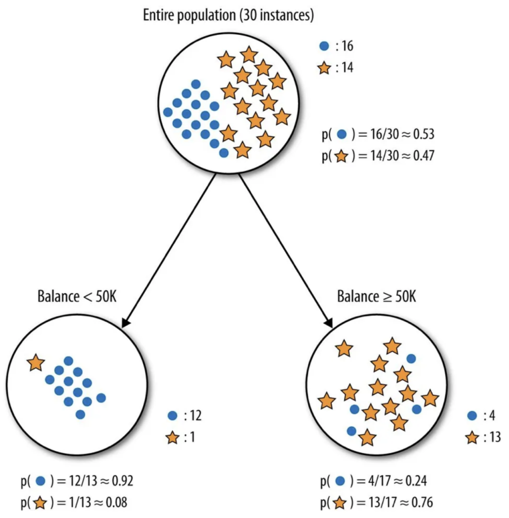


Figure 2: Splitting on Feature "Balance". Provost, Foster; Fawcett, Tom. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking

Let's calculate the entropy for the parent node and see how much uncertainty the tree can reduce by splitting on Balance.

$$E(Parent) = -\frac{16}{30}\log(\frac{16}{30}) - \frac{14}{30}\log(\frac{14}{30}) \approx 0.99$$

$$E(Balance < 50K) = -\frac{12}{13}\log(\frac{12}{13}) - \frac{1}{13}\log(\frac{1}{13}) \approx 0.39$$

$$E(Balance > 50K) = -\frac{4}{17}\log(\frac{4}{17}) - \frac{13}{17}\log(\frac{13}{17}) \approx 0.79$$

$$E(Balance) = \frac{13}{30} \times 0.39 + \frac{17}{30} \times 0.79 = 0.62$$

$$IG(Parent, Balance) = E(Parent) - E(Balance) = 0.99 - 0.62 = 0.37$$

Splitting on feature "Balance" leads to an information gain of 0.37 on our target variable. Let's do the same thing for feature, "Residence" to see how it compares.

Feature2: Residence

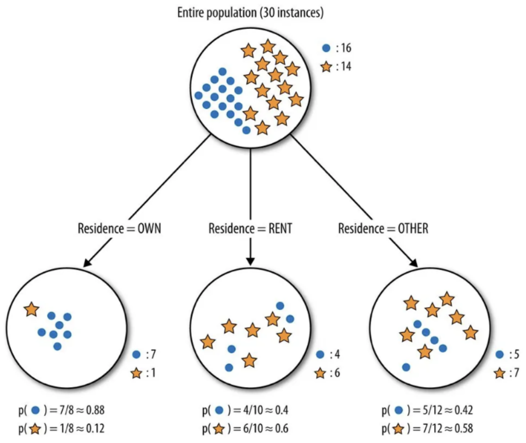


Figure 3: Splitting on Feature "Residence". Provost, Foster; Fawcett, Tom. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking

With the same steps as above we can see that the entropy of "Residence" $E(Residence) = 0.86$ and splitting on feature "Ralance" leads to an information gain of 0.13 on our target variable.

By itself the feature Balance provides more information about our target variable than Residence. It reduces more disorder in our target variable. A decision tree algorithm would use this result to make the first split on our data using Balance. From here on, the decision tree algorithm would use this process at every split to decide what feature it is going to split on next. In a real world scenario , with more than two features the first split is made on the most informative feature and then at every split the information gain for each additional feature needs to be recomputed because it would not be the same as the information gain from each feature by itself. The entropy and information gain would have to be calculated after one or more splits have already been made which would change the results. A decision tree would repeat this process as it grows deeper and deeper till either it reaches a pre-defined depth or no additional split can result in a higher information gain beyond a certain threshold which can also usually be specified as a hyper-parameter.

Another function to measure the "purity" is Gini Index function.

$$G = \sum_{k=1}^{K} (p_k * (1- p_k))$$

Where G is the Gini index over all classes, p_k are the proportion of training instances with class k in the rectangle of interest. A node that has all classes of the same type (perfect class purity) will have G=0, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a G=0.5.

1.3 Strategy of Growing and Pruning

The stopping criterion in decision tree construction plays a crucial role in determining the performance of the tree. To further improve performance, pruning can be done after the tree has been learned.

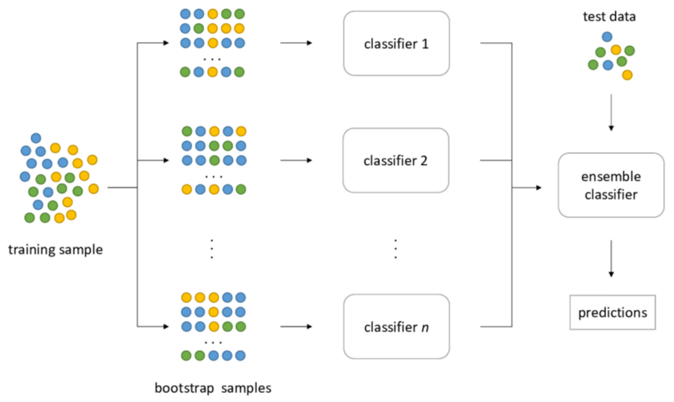


Figure 4: Bagging Overview. Oleksandr, Proskurin. Bagging in Financial Machine Learning: Sequential Bootstrapping.

The complexity of a decision tree is measured by the number of splits in the tree, and simpler trees are often preferred as they are easier to understand and less prone to overfitting the data.

The simplest and fastest pruning method involves evaluating the effect of removing each leaf node in the tree using a hold-out test set. The nodes are removed only if it leads to a decrease in the overall cost function on the entire test set. This process continues until no further improvements can be made.

More advanced pruning methods, such as cost complexity pruning, also known as weakest link pruning, can be used. In this method, a learning parameter (alpha) is used to weigh the decision to remove nodes based on the size of the subtree.

2 Bagging

2.1 Why Bagging: Idea of Ensemble Classifiers

Using a single decision tree could be problematic because decision tree can be highly sensitive to small changes in data. One solution of the problem is that instead of just using a single decision tree, we can train multiple decision trees to make it more robust. Each decision tree is a different way of classifying the objects. The collection of the decision trees is called tree ensemble.

Bagging, also known as bootstrap aggregation, is the learning method utilizing several different models that are built by training on different sample subset. Then either the average, aggregation, or majority of the predictions from different models is calculated to reduce variance. In the context of Decision Tree, tree ensemble and majority decision is used for bagging. Bagging help soften the over-fitting problems for Decision Tree that are resulted from the high variance.

2.2 Sample With Replacement

In order to build tree ensemble, we can utilize a technique called sampling with replacement. The way sampling with replacement works is that first, we randomly select a sample from the dataset. The term ”with replacement” indicates that after recording the sample that was picked, we put in back to the dataset. Then we pick another random sample from the dataset,

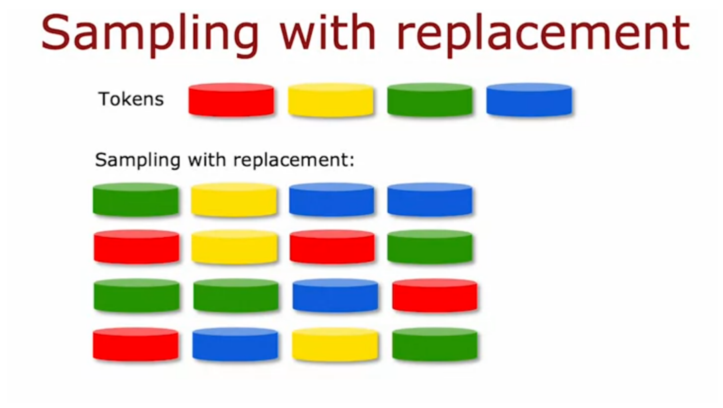


Figure 5: Sampling with Replacemnt. Ng, Andrew. Sampling With Replacemnt.

record it, and put it back again. This means that the same item may be randomly selected more than once. We repeat the process until the amount of samples needed is met.

It is used for Bagging becasue it allows us to use the same dataset multiple times to build models. We can even have the sampled dataset as the same size of the original dataset by using this technique, and the different datasets generated will still appear different to one another most of the time. The new datasets will be very similar to the original dataset in terms of size and memebers, but it's not completely the same.

2.3 Random Forest: Bootstrapping and Random Splits

A popular machine learning technique called Random Forest is used for classification, regression, and other applications. Several decision trees are used in this ensemble learning technique to increase accuracy and decrease overfitting.
The Random Forest algorithm's fundamental principle is to construct a huge number of decision trees, each of which is trained using a unique part of the training data and a random sample of characteristics. Each tree in the decision forest makes a forecast for an output given an input, and the final prediction is formed by collecting the majority vote of all the trees in the forest. By preventing the decision trees from being overly specialized to the training data, the random feature selection and data sampling contribute to reducing overfitting. The technique also permits feature importance analysis, which may be used to pinpoint the features that are most crucial for foretelling the result.

$$RandomForest = DecisionTree + Bagging + Boostrapping + RandomSplits$$

```
    Given traing set of size m
For b = 1 to B:
    Use sampling with replacemnt to create a new traing set of size m
    Train a decision tree on the new dataset
```

B stands for bagging and denotes the number of decision tree we are going to train, and is often recommended to be picked as 64 or 128. Different generated decision tree may result in different split across tree. This is the formal algorithm for bagging. But one simple modification will make this algorithm even better and change it to Random Forest.

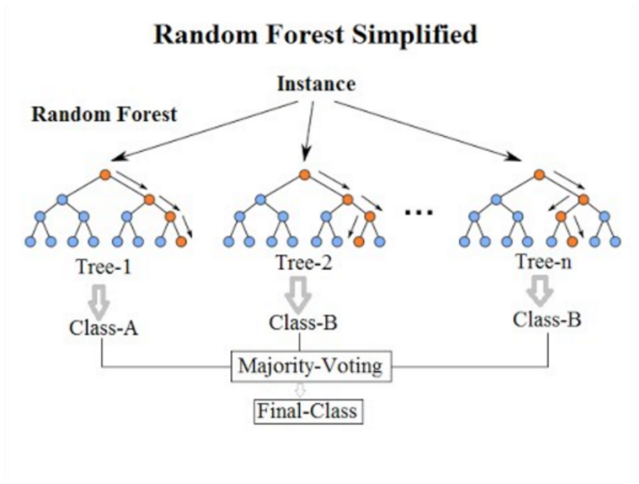


Figure 6: Random Forest. Koehrsen, Will. Random Forest Simple Explanation.

The key idea is that even with sampling with replacement, you always ended up with same root nodes and similar splits. Random Forest improve this by randomizing the feature choice. At each node, when choosing a feature to use to split, if n features are available, pick a random subset of k<n features and allow the algorithm to only choose from that subset of features[1].

The main applications of Random Forest are:
Classification: Data may be divided into several groups using Random Forest, such as spam vs non-spam emails or benign versus malignant tumors. Both binary and multi-class classification issues may be handled by the approach.
Regression: When attempting to forecast a continuous output variable, Random Forest may also be utilized to solve the problem. The method may be used in fields like finance to forecast stock prices or other financial parameters.
Data Anomaly Detection: Random Forest may be employed to find data abnormalities. Random Forest may be used to find anomalies, which are data points that are noticeably different from the majority of the data.

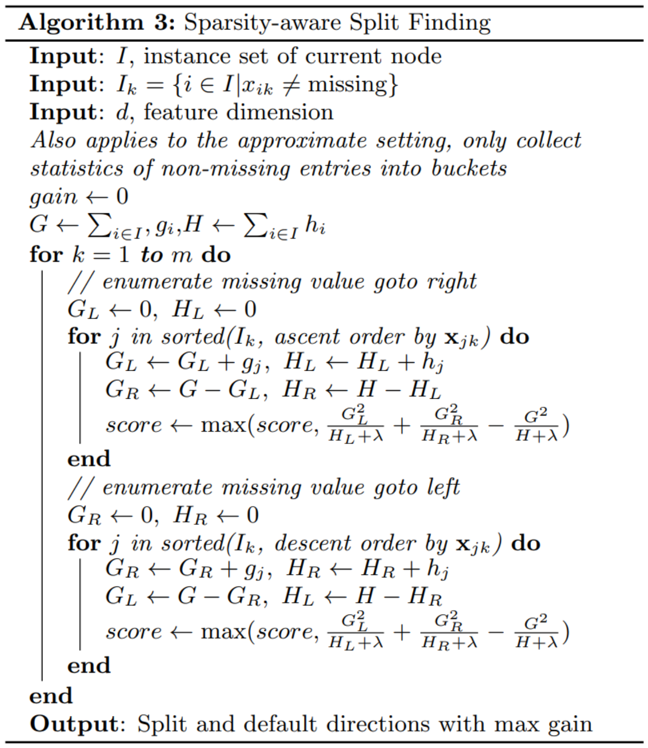
3 Boosting

3.1 Boosted Tree

Boosting is a method of combining many weak learners into a strong classifier. For example, instead of sampling all items in the dataset with an equal probability, we give a larger weight to the items that are misclassfied from previously trained trees so that they are more likely to be picked compared to other ones.

3.2 XGBoost

XGBoost, which stands for "extreme gradient boosting", is also meant to be an improvement over the bagging algorithm that we showed above. It's an open source implementation of boosted trees that are very popular. It has built in regularization that help prevent overfitting.



```
Algorithm 3: Sparsity-aware Split Finding
Input: $I$, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: $d$, feature dimension
Also applies to the approximate setting, only collect
statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to $m$ do
    // enumerate missing value goto right
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for $j$ in sorted($I_k$, ascent order by $\mathbf{x}_{jk}$) do
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
    // enumerate missing value goto left
    $G_R \leftarrow 0,\ H_R \leftarrow 0$
    for $j$ in sorted($I_k$, descent order by $\mathbf{x}_{jk}$) do
        $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
        $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split and default directions with max gain
```

Figure 7: XGBoost algorithm

4 References

[1] Ng, Andrew. 2022. Random Forest Algorithm. https://www.youtube.com/watch?v=VfKwWWrSnq4