TOP-DOWN DECISION TREE INDUCERS

Lev Dubinets

A decision tree is a tree where each internal node specifies a test on an attribute of the data in question and each edge between a parent and a child represents a decision or outcome based on that test. Leaf nodes, sometimes called terminal nodes, contain classes or labels of the data. Tests are encoded as numerical or categorical rules such as "X > 5" or "X is 'female'". A decision tree can also be thought of as an apparatus that accepts as a set of input values and follows decision rules to get to a leaf of the tree, which corresponds to an output value. As such, a decision tree is a classifier. Decision trees are a widely used technique in statistical learning, where they are constructed to fit an existing set of data, and then used to predict outcomes on new data. This paper is about one of the most common ways to grow a decision tree based on a dataset, called "Top-Down Induction" [1].

We start with N labeled "training records" of the form $(\boldsymbol{X}, Y)$ where $\boldsymbol{X}$ is a k-dimensional vector of features describing the data we have, and Y is a label we give this record. For example, our data may be characteristics about humans and our label may be their shoe size, so an example dataset could look like this:

Gender	Height	Shoe Size
F	70	10
M	64	10.5
M	68	9
…	…	…

Figure 1: An example set of labeled records. In this case $\boldsymbol{X}$ is a 2-dimensional vector of $(gender, height)$ and Y is shoe size.

Each component of $\boldsymbol{X}$ is called an "input variable", Y is called the "dependent variable" or "target variable", and each row in such a table is called a "training example". Although this is not necessary in decision tree learning, we will simplify our decision rules for now by restricting them to have binary outcomes. This means that a given decision will split the remaining dataset into exactly two portions. Suppose we have two input variables, so that $\boldsymbol{X} = (X_1, X_2)$. Further, let's assume there is an interesting value of X_1 that we can split the dataset around, and three interesting values of X_2. Then, an example partitioning of our space of (X_1, X_2) values is depicted in the left side of Figure 2, and a decision tree corresponding to such a partitioning is shown in the right side of the figure. Given an unlabeled vector $\boldsymbol{X} = (X_1, X_2)$, we first test whether $X_1 > a$. Then, if that turns out to be true, we test whether $X_2 > d$. This allows us to classify $\boldsymbol{X}$ into the region R_4 or the region R_5. If we initially had that $X_1 \leq a$, then we will test X_2 against c and then against b, which allows us to further classify $\boldsymbol{X}$ into one of the regions $R_1, R_2,$ or R_3.

Next, we let Y take on a single constant value for each of the regions $R_1, .., R_5$. Let Y_i be the value we choose for Y for the region R_i, and let $I_i(\boldsymbol{X})$ be an indicator function that equals 1 when $\boldsymbol{X} \in R_i$. This allows us to obtain a model that can predict Y based on $\boldsymbol{X}$:

$$\hat{Y}(\boldsymbol{X}) = \sum_{i=1}^{5} Y_i \times I_i(\boldsymbol{X})$$

Obtaining such a model is the ultimate goal of training a decision tree. This particular model is represented in Figure 2 as a partition of 2D space and as a decision tree.

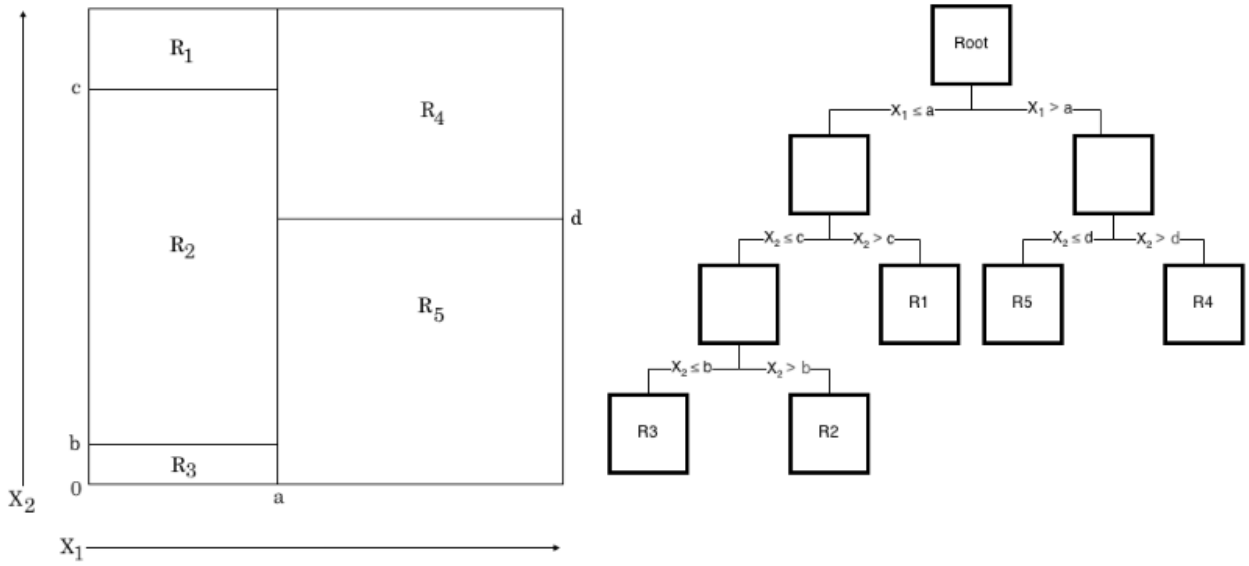

Figure 2: On the left is a partition of a 2D space into five regions, on the right is a decision tree capable of classifying a point into one of the five regions.

We can also interpret this decision tree as being represented by a set of lines in the 2D plane, namely the lines drawn in the left side of Figure 2. In the higher-dimensional case, when there are more than two input variables, a decision tree would correspond to a set of hyperplanes that divide the space [4]. In the case that an input variable is categorical rather than continuous, we can assign each category (such as "male" or "female") a unique discrete value and then split between those unique values. Each resulting region corresponds to an output of the classifier. In practice, the result of the classification is usually output alongside a vector of probabilities indicating the likelihood of the target variable having each of the possible classification values.

The most basic process of training a decision tree on a dataset involves these three elements: the selection of attribute splits in the tree, the decision of when to stop splitting a node and mark it terminal, and the assignment of a label to each terminal node. Some algorithms add a fourth element, called pruning. There are many different ways of implementing splitting criteria, stopping criteria, and pruning methods. Splitting criteria are algorithmic rules that decide which input variable to split the dataset around next. In the preceding example, we could have chosen to test $X_2 > d$ at the root instead of testing $X_1 > a$. However, this may have resulted in a more complicated decision tree. Stopping criteria are rules that determine when to stop splitting the dataset and instead output a classification. Stopping criteria are actually optional, but in their absence a trained tree would have a separate region for each training record. As this is undesirable, stopping criteria are used as a method of deciding when to stop growing the tree. Lastly, pruning methods are ways to reduce the size and complexity of an already trained tree by combining or removing rules that do not significantly increase classification accuracy. All three of these things directly affect the complexity of a tree, which can be measured according to

various metrics such as tree height, tree width, and number of nodes. It is desirable to train trees that are not overly complex, and such a preference is backed primarily by Occam's Razor and the fact that simpler trees require less storage. However, it has been shown that for some measures of tree complexity, training an optimal tree is an NP-complete problem or is otherwise computationally inefficient [2]. As a result, many decision tree training algorithms use heuristics for splitting criteria and we consider them greedy algorithms.

We now begin a description of several splitting criteria. In particular, we will discuss splitting criteria that only consider the values of a single input variable. Such criteria are called univariate splitting criteria. For a given dataset of training records D of the form $(\boldsymbol{X}, Y)$, we let T_i be a test on a record that considers the attribute X_i and has n outcomes $O_1, O_2, \ldots, O_n$. An example of such a test would be $X_1 > a$ which has two outcomes. Many different tests could exist for each input attribute. A given test will split the dataset D into n different subsets $D_1, D_2, \ldots, D_n$. The following is a generic tree training algorithm.

Algorithm: TrainTree

Input: D, a dataset of training records of the form (**X**,Y).
Output: Root node R of trained decision tree

1) Create a root node R
2) If a stopping criterion has been reached then label R with the most common value of Y in D and output R
3) For each input variable X_i in **X**
    a. Find the test T_i whose partition $D_1, D_2, \ldots, D_n$ performs best according to the chosen splitting metric.
    b. Record this test and the value of the splitting metric
4) Let T_i be the best test according to the splitting metric, let V be the value of the splitting metric, and let $D_1, D_2, \ldots, D_n$ be the partition.
5) If $V < threshold$
    a. Label R with the most common value of Y in D and output R
6) Label R with T_i and make a child node C_i of R for each outcome O_i of T_i.
7) For each outcome O_i of T_i
    a. Create a new child node C_i of R, and label the edge O_i
    b. Set C_i = TrainTree(D_i)
8) Output R

Before we consider any individual splitting criterion, we need to define the notion of an impurity function, which is a start to understanding how well a given split organizes the dataset.

An impurity function of a probability distribution P is a function $\phi: P \to \mathbb{R}$ that satisfies a few constraints:

1) ϕ attains its maximum if and only if P is a uniform distribution.
2) ϕ attains its minimum if and only if some $p_j = 1$ for some $p_j \in P$
3) ϕ is symmetric about the components of P
4) $\phi(P) \geq 0$ for all P

The application to decision trees arises from the fact that at each node, when considering a split on a given attribute we have a probability distribution P with a component p_j for each class j of the target variable Y. Hence we see that a split on an attribute it most impure if P is uniform, and is pure if some $p_j = 1$, meaning all records that past this split are definitely of class j. Once we have an impurity function, we can define an impurity measure of a dataset D node n as so:

If there are k possible values $y_1, y_2, \ldots, y_k$ of the target variable Y, and σ is the selection operator from relational algebra then the probability distribution of S over the attribute Y is

$$P_Y(D) = \left(\frac{\left|\sigma_{Y=y_1}(D)\right|}{|D|}, \frac{\left|\sigma_{Y=y_2}(D)\right|}{|D|}, \ldots, \frac{\left|\sigma_{Y=y_k}(D)\right|}{|D|} \right)$$

$$\sigma_{\varphi}(D) = set\ of\ all\ X \in D\ s.t.\ the\ expression\ \varphi\ holds\ true\ for\ X$$

And the impurity measure of a dataset D is denoted as

$$impurity_Y(D) = \phi(P_Y(D))$$

Lastly, we define the goodness-of-split (or change in purity) with respect to an input variable X_i that has m possible values $v_1, .. v_m$ and a dataset D as $\Delta i(X_i, D)$

$$\Delta i_Y(X_i, D) = impurity_Y(D) - \sum_{j=1}^{m} \frac{\left|\sigma_{X_i=v_j}(D)\right|}{|D|} impurity_Y(\sigma_{X_i=v_j}(D))$$

Impurity based splitting criteria use an impurity function ϕ plugged into the general goodness-of-split equation defined above.

Information Gain

Information gain is a splitting criterion that comes from information theory. It uses information entropy as the impurity function. Instead of the usual definition of information entropy as a function of a discrete random variable, we use a very similar definition of information entropy as a function of a probability distribution. This allows us to use it as our impurity function. Given a probability distribution $P = (p_1, p_2, .., p_n)$, where p_i is the probability that a point is in the subset D_i of a dataset D, we define the entropy H:

$$Entropy(P) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Plugging in $Entropy$ as our function ϕ gives us $InformationGain_Y(X_i, D)$:

$$InformationGain_Y(X_i, D) = Entropy(P_Y(D)) - \sum_{j=1}^{m} \frac{\left|\sigma_{X_i=v_j}(D)\right|}{|D|} Entropy\left(P_Y(\sigma_{X_i=v_j}(D))\right)$$

$$InformationGain_Y(X_i, D) = EntropyBeforeSplit - EntropyAfterSplit$$

We can also define information gain as a function of conditional entropy. Suppose P and Q are two distributions with n and m elements, respectively. The outcomes of the splits on P and Q form a table T where the entry T_{ij} contains the number of records in the dataset that survive both the P_i split and the Q_j split. We let $p_{ij} = \frac{T_{ij}}{|D|}$.

Then conditional entropy is the entropy of a record surviving the split Q given that it survived P as follows. We restrict the Q distribution to the column of T that contains the result of the split on P, and then average over all possible splits on P.

$$Entropy(Q|P) = -\sum_i p_i \sum_j \frac{p_{ij}}{p_i} \log_2\left(\frac{p_{ij}}{p_i}\right) = -\sum_{i,j} p_{ij} \log_2\left(\frac{p_{ij}}{p_i}\right)$$

$$Entropy(P|Q) = -\sum_j q_j \sum_i \frac{p_{ij}}{q_i} \log_2\left(\frac{p_{ij}}{q_j}\right) = -\sum_{i,j} p_{ij} \log_2(\frac{p_{ij}}{q_i})$$

$$InformationGain_Y(X_i, D) = Entropy\big(P_Y(D)\big) - Entropy(P_Y(D)|P_{X_i}(D))$$

Plugging in the definition for conditional entropy into the above immediately yields the equation for information gain presented earlier. We can now prove an interesting property of information gain, showing that when using this splitting criterion we cannot make negative progress in classifying data under our target attribute.

PROPOSITION 1: Information Gain is non-negative.

PROOF: This follows pretty easily from Jensen's inequality since $-log(x)$ is convex.

Jensen's inequality states that $\varphi(E[X]) \leq E[\varphi(X)]$, where X is a random variable, E is the expectation, and φ is a convex function.

$$Entropy(Q) - Entropy(Q|P) = -\sum_j q_j \log_2(q_j) + \sum_{i,j} p_{ij} \log_2\left(\frac{p_{ij}}{p_i}\right) \quad (1)$$

$$= \sum_{i,j} p_{ij} \log_2\left(\frac{p_{ij}}{p_i}\right) - \sum_j \log_2(q_j)(\sum_i p_{ij}) \quad (2)$$

$$= \sum_{i,j} p_{ij} \log_2\left(\frac{p_{ij}}{p_i}\right) - \sum_{i,j} \mathrm{p_{i,j}} \log_2(q_j) \quad (3)$$

$$= \sum_{i,j} p_{ij} \log_2\left(\frac{p_{ij}}{p_i}\right) - \sum_{i,j} \mathrm{p_{i,j}} \log_2(q_j) \quad (4)$$

$$= \sum_{i,j} p_{ij} \log_2(\frac{p_{ij}}{p_i q_j}) \quad (5)$$

$$= -\sum_{i,j} p_{ij} \log_2(\frac{p_i q_j}{p_{ij}}) \quad (6)$$

$$\geq -\log_2\left(\sum_{i,j} p_{ij}\left(\frac{p_i q_j}{p_{ij}}\right)\right) \quad (by\ Jensen's\ inequality) \tag{7}$$

$$= -\log_2\left(\sum_{i,j} p_i q_j\right) \tag{8}$$

$$= -\log_2\left(\sum_i p_i\left(\sum_j q_j\right)\right) \quad \left(\sum_{p\in P} p = 1\ for\ any\ probability\ distribution\ P\right) \tag{9}$$

$$= -\log_2\left(\sum_i p_i\right) = -\log_2(1) = 0 \tag{10}$$

Letting $P = P_{X_i}(D)$ and $Q = P_Y(D)$ we get that splitting a dataset D on an input variable X_i yields non-negative information gain with respect to a target variable Y. □

The result of this proof seems to say that splitting on any variable should not make the model any worse, since information gain is non-negative. However, this is not entirely true because over fitting may occur.

PROPOSITION 2: Information Gain is symmetric, meaning that if we switch the split variable and target variable, we still get the same amount of information gain. Expressed in our notation as:

$$InformationGain_Y(X, D) = InformationGain_X(Y, D)$$

PROOF:

In step (5) of the proof of Proposition 1 we showed that

$$Entropy(Q) - Entropy(Q|P) = \sum_{i,j} p_{ij}\log_2\left(\frac{p_{ij}}{p_i q_j}\right) \tag{1}$$

Since this expression is symmetric around p_i and q_j it follows that

$$\sum_{i,j} p_{ij}\log_2\left(\frac{p_{ij}}{p_i q_j}\right) = Entropy(P) - Entropy(P|Q) \tag{2}$$

Letting $P = P_X(D)$ and $Q = P_Y(D)$ we immediately obtain

$$InformationGain_Y(X, D) = InformationGain_X(Y, D) \quad \square \tag{3}$$

One issue with using information gain as a splitting criterion is that it is biased towards tests that have many outcomes [9]. For example, suppose we are building a decision tree on a dataset of customers, to be able to predict some form of customer behavior in the future. If one of the input attributes is something unique to each customer, such as a credit card number, phone number, social security number, or any other form of ID, then the information gain for such an attribute will be very high. As a result, a test on this attribute may be placed near the root of the tree. However, testing on such attributes will not extend well to records outside of the training set, which in this case are new customers, since their unique identifiers will not be an outcome of the test.

Gain Ratio

As a result, Quinlan proposes a related splitting criterion, called Gain Ratio or Uncertainty Coefficient [9]. This serves to normalize information gain on an attribute X_i relative how much entropy this attribute has.

$$GainRatio_Y(X_i, D) = \frac{InformationGain_Y(X_i, D)}{Entropy\left(P_{X_i}(D)\right)}$$

We can clearly see that in the preceding example with customer data, the denominator will be very large. Since the identifiers are unique, the denominator would be on the order of the number of customers in the dataset, whereas the numerator is at most the number of possible values for Y. However, it must be noted that if $Entropy\left(P_{X_i}(D)\right)$ is very small, then this gain ratio will be large but X_i may still not be the best attribute to split on. Quinlan suggests the following procedure when using gain ratio as the criterion:

(1) Calculate information gain for all attributes, and compute the average information gain.
(2) Calculate gain ratio for all attributes whose information gain is greater or equal to the average information gain, and select the attribute with the largest gain ratio.

Quinlan claims that this gain ratio criterion tends to perform better than the plain information gain criterion, creating smaller decision trees. He also notes that it may tend to favor attributes that create unevenly sized splits where some subset D_i of D in the resulting partition is much smaller than the other subsets.

Gini Index

The Gini Index is another function that can be used as an impurity function. It is a variation of the Gini Coefficient, which is used in economics to measure the dispersion of wealth in a population [10]. It was introduced as an impurity measure for decision tree learning by Breiman in 1984. It is given by this equation

$$Gini(P) = \sum_{i=1}^{n} p_i(1 - p_i) = 1 - \sum_{i=1}^{n} (p_i)^2$$

$$where\ P = (p_1, \dots, p_n).$$

$Gini(D)$ will be zero if some $p_i = 1$ and it since each $p_i < 1$, it will be maximized if all p_i are equal. Hence we see that the Gini Index is a suitable impurity function.

Remembering that in our definition of $P_Y(D)$, $p_i = \frac{\left|\sigma_{Y=y_i} D\right|}{|D|}$, the Gini Index measures the expected error if we randomly choose a single record and use it's value of Y as the predictor, which can be seen by looking at the second formulation of the Gini Index, $1 - \sum_{i=1}^{n} (p_i)^2$. This can be interpreted as a difference between the norms of the vectors $(1, \dots, 1)$ and $P_y(D)$. Also, in the first formulation, $\sum_{i=1}^{n} p_i(1 - p_i)$, it is simply the sum of n variances of n

Bernoulli random variables, corresponding to each component of the probability distribution $P_Y(D)$. [5]

We define the goodness-of-split due to the Gini Index similar to what we did with information gain, by simply plugging in $\phi(D) = Gini_Y(D)$ to obtain

$$GiniGain_Y(X_i, D) = Gini\big(P_y(D)\big) - \sum_{j=1}^{m} \frac{\left|\sigma_{X_i=v_j}(D)\right|}{|D|} Gini\left(P_Y(\sigma_{X_i=v_j}(D))\right)$$

It turns out that the Gini Index and information entropy are very related concepts. Figure 3 shows that they graphically look similar:

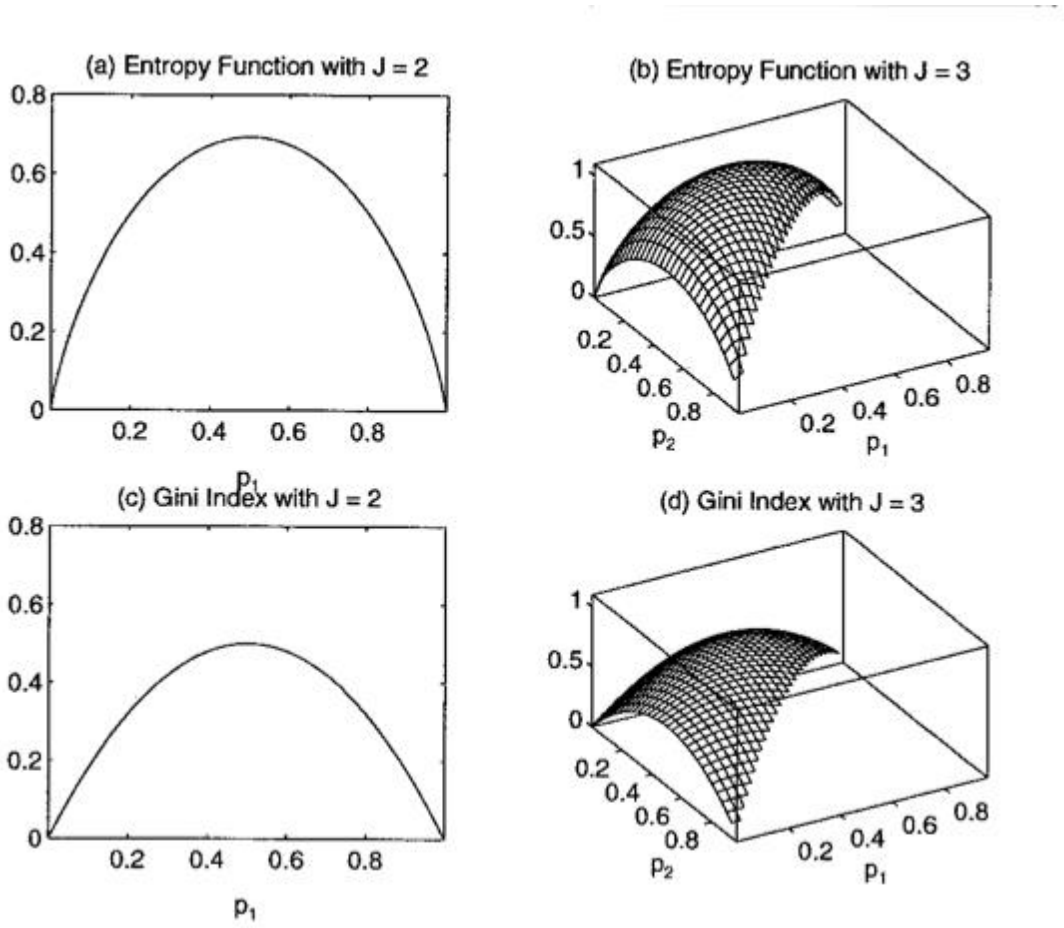

Figure 3: Entropy and Gini Index graphed for two different values of J, which is the total number of classes of the target variable [8].

In fact, in 1967 Havrda and Charvat defined a generalization of Shannon entropy that they called α-entropy [5].

$$H_\alpha(P) = \frac{1}{1-\alpha} \sum_i^n p_i^\alpha - 1$$

For $\alpha = 2$, the Havdra-Charvat entropy immediately yields the Gini Index, and for $\alpha = 1$, Havdra and Charvat specifically defined $H_1(P) = Entropy(P) = -\sum_i^n p_i \log(p_i)$, yielding the Shannon entropy.

Misclassification Rate

Given a probability distribution P we define the misclassification rate to be

$$Misclassification(P) = 1 - \max_j p_j$$

$$where\ P = (p_1, \dots, p_n)$$

Again, it is easy to see that $Misclassification$ is indeed an impurity function. It is maximized when P is uniform and it is zero when some $p_j = 1$. We then define $MisclassificationGain$ by plugging in $\phi(P) = Misclassification(P)$ into the goodness-of-split equation.

$$MisclassificationGain_Y(X_i, D)$$
$$= Misclassification\big(P_Y(D)\big) - \sum_{j=1}^{m} \frac{\left|\sigma_{X_i=v_j}(D)\right|}{|D|} Misclassification\left(P_Y(\sigma_{X_i=v_j}(D))\right)$$

An interesting fact is that the Gini Index and the Misclassification Rate have the same maximum. Given that P has n elements, meaning that there are n possible values of the target variable, the maximum Misclassification Rate is $1 - \frac{1}{n}$ and the maximum Gini Index is $1 - n * \left(\frac{1}{n}\right)^2 = 1 - \frac{1}{n}$.

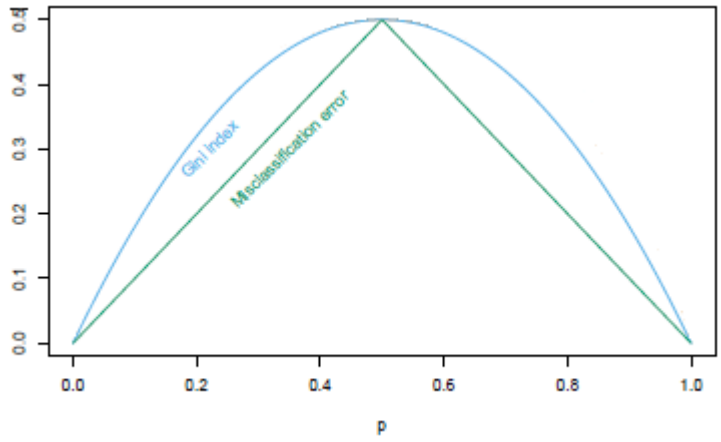

Figure 4: Gini Index and Misclassification Rate plotted in the case when the target variable has 2 possible values. [4]

Since the Gini Index and Entropy impurity functions are both differentiable, they are preferred over the Misclassification Rate, which is not [4]. Differentiability helps with numerical optimization techniques.

Other Splitting Criteria

There are many other splitting criteria that one can use, and a good overview of these is in chapter 9 of the Data Mining and Knowledge Discovery Handbook [3]. While we have only considered univariate splitting criteria, this book has a brief description of multivariate splitting criteria as well.

Stopping Criteria

Stopping criteria are usually not as complicated as splitting criteria. Common stopping criteria include:

- Tree depth exceeds a predetermined threshold
- Goodness-of-split is below a predetermined threshold
- Each terminal node has less than some predetermined number of records

Generally stopping criteria are used as a heuristic to prevent over fitting. Over fitting is when a decision tree begins to learn noise in the dataset rather than structural relationships present in the data. An over-fit model still performs very well in classifying the dataset it was trained on, but would not generalize well to new data, just like the example with credit card numbers or other unique identifiers. If we did not use stopping criteria, the algorithm would continue growing the tree until each terminal node would correspond to exactly one record.

Approximation Algorithms

As mentioned, Rivest and Hyafil proved in 1976 that training an optimal decision tree is an NP-complete problem. In their formulation of the problem, the cost of a tree is the sum of the number of tests that must be performed to reach a terminal node for each of the records in the dataset. It must be noted that when considering the decision tree problem, it is no longer a machine learning task, in the sense that we only care about classifying the existing dataset correctly, and don't worry about performance on new data.

The decision tree problem $DT(w)$ is to determine whether there exists a decision tree that correctly classifies a given dataset with cost less than or equal to w. Also, in Hyafil's formulation of the problem, the tree is a binary decision tree and the distribution $P_Y(D)$ is uniform. Heeringa and Adler showed that obtaining a $(1+\epsilon)$-approximation to the decision tree problem is an NP-hard task [6]. They also gave a $(1+\ln(N))$-approximation algorithm, where N is the number of records in the dataset. Chakaravarthy et al. built on this analysis to show some interesting results [6]. First, they named the aforementioned decision tree problem 2-UDT, where the 2 stands binary decision trees and the U stands for uniform distribution, and improved on Heeringa's result, showing that a $(2-\epsilon)$-approximation for 2-UDT is NP-hard. They then showed an $O(\log(N))$-approximation algorithm for the 2-DT problem, which is the same problem but without the requirement that $P_Y(D)$ be uniform. Both Heeringa and Chakaravarthy extend the problem slightly to give weights to performing each test, which means that the cost is now a weighted sum. Chakaravarthy et al. also show a connection between Ramsey numbers and the K-UDT problem. Lastly, Gupta

et al. showed that the optimal decision tree problem is a special case of the adaptive travelling salesman problem [7].

Conclusion

We have studied a general method of growing a decision tree using a greedy algorithm, with interesting analyses of several splitting criteria that can be used with this algorithm. A brief discussion of approximation algorithms for the NP-complete decision tree problem shows that decision trees play a role in the fundamentals of theoretical computer science.

References

[1] Quinlan, J. R. "Induction of Decision Trees." Machine Learning 1.1 (1986): 81-106. Web.

[2] Hyafil, Laurent, and Ronald L. Rivest. "Constructing Optimal Binary Decision Trees Is NP-complete." Information Processing Letters 5.1 (1976): 15-17. Web.

[3] Maimon, Oded, and Lior Rokach. Data Mining and Knowledge Discovery Handbook. New York: Springer, 2005. Chapter 9. http://www.ise.bgu.ac.il/faculty/liorr/hbchap9.pdf

[4] Hastie, Trevor, Robert Tibshirani, and J. H. Friedman. "9.2 Tree-Based Methods." The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2009. N. pag. Print.

[5] Gras, Régis, and Pascale Kuntz. "Reduction of Redundant Rules in Statistical Implicative Analysis." Studies in Classification, Data Analysis, and Knowledge Organization. *Selected Contributions in Data Analysis and Classification (2007)*: 367-76. Web.

[6] Chakaravarthy, Venkatesan T., Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh Mohania. "Decision Trees for Entity Identification." Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS '07 (2007): n. pag. Web.

[7] Gupta, Anupam, Viswanath Nagarajan, and R. Ravi. "Approximation Algorithms for Optimal Decision Trees and Adaptive TSP Problems." Automata, Languages and Programming Lecture Notes in Computer Science (2010): 690-701. Web.

[8] Jugal Kalita, Decision and Regression Tree Learning. Slides. Used for figure 3. http://www.cs.uccs.edu/~jkalita/work/cs586/2013/DecisionTrees.pdf

[9] Quinlan, J. R. "Section 2.2.2." C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann, 1993. N. pag. Print.

[10] González Abril, Luis, et al. "The similarity between the square of the coefficient of variation and the Gini index of a general random variable." Revista de métodos cuantitativos para la economía y la empresa 10 (2010): 5-18.