CS 6140: Machine Learning **Spring 2018**
College of Computer and Information Science
Northeastern University
Lecture 1 **January, 8**
Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Virgil Pavlu**

Learning Decision Trees

From a learning perspective we are interested in learning the "optimal" tree from a given training dataset. There are a number of concerns that we need to address before we develop an algorithm to search for the optimal tree.

1 Number of possible decision trees:

The first and foremost problem is to find the number of possible trees that can be constructed to fit our training data. If we consider only binary features and a binary classification task, then for a training dataset having m features there are at most 2^m possible instances. In this case there are 2^{2^m} different ways of labeling all instances, and each labeling corresponds to a different underlying boolean function that can be represented as a decision tree. This means that we have a very large search space, for example given ten features we have about 2^{1024} possible decision trees. Therefore, searching for the best tree becomes an intractable task, and we resort to different heuristics that are able to learn 'reasonably' good trees from training data.

2 What is a good decision tree?

Given a choice between two decision trees, which one should we choose? In other words how can we define the "goodness" of a decision tree? Consistency with training data i.e., how accurately does the decision tree predict the labels of the training instances, is one such measure. But, we can always build a decision such that each instance has its own leaf node, in which case the decision tree will have 100% accuracy. The problem with such a tree is that it will not be able to *generalize* to unseen data. This shows that training error by itself is not a suitable criterion for judging the effectiveness of a decision tree. To address this problem, we define a good decision tree as the one that is as small as possible while being consistent with the training data.

3 Inducing decision trees from training data:

Finding the smallest decision tree consistent with training data is an NP-complete problem and therefore we use a heuristic greedy approach to iteratively grow a decision tree from training data. Algorithm-1 outlines the general set of steps. We start the explanation of the proposed algorithm by first defining what is meant by a split and how to choose the "best" feature to create a split.

```
Data: Training data with $m$ features and $n$ instances
Result: Induced Decision Tree
Begin with an empty tree;
while stopping criterion is not satisfied do
    select best feature;
    split the training data on the best feature;
    repeat last two steps for the child nodes resulting from the split;
end
```
Algorithm 1: A greedy algorithm for inducing a decision tree from given training data

3.1 Tree splitting:

In Figure-1, each internal node of the decision tree defines a split of the training data based on one single feature. Splits based on only one feature are also known as *univariate* splits. For example the root node splits the training data into three subsets corresponding to the different values that the *Patrons* feature can take, so that the left-most child has all the instances whose feature value is 'None'. The type of split that a feature defines depends on how many possible values the feature can take. A binary feature will split the data into two sub-groups while a feature with k distinct values will define a k-ary (multiway) split resulting in multiple sub-groups.

3.1.1 Which feature is the best?

Let the training data for a classification task consist of N instances defined by m features, and let y denote the output/response variable, so that the training can be represented as (x_i, y_i) with $x_i = [x_{i1}, x_{i2}, \ldots, x_{im}]$, and $y_i \in \{1, 2, \ldots, k\}$ for $i = 1, 2, \ldots, N$. Consider a node q in the tree, that has N_q instances, then the proportion of the instance belonging to class k can be estimated as:

$$p_{qk} = \frac{1}{N_q} \sum_{i=1}^{N_q} \mathbb{I}(y_i = k) \tag{1}$$

where $\mathbb{I}(.)$ is an indicator function which is 1 when the arguments evaluate to true and 0 otherwise. In the above equation the indicator function is used simply to count the number of instances belonging to class k. We classify all the instances in q to the majority class i.e., $class(q) = \arg\max_k p_{qk}$.

At node q we need to select the next best feature which will define the next split. One possible criterion for feature selection would be to measure the classification error of the resulting split.

$$Error(q) = \frac{1}{N_q} \sum_{i=1}^{N_q} \mathbb{I}(y_i \neq class(q)) \tag{2}$$

A feature that minimizes the classification error would be considered as the best feature to split node q.

For example consider the training data shown in Figure-**??**, and we decide that *Bar* is the feature that we should use to define our first split (root of the decision tree). This would produce a binary split resulting in two new nodes that we will represent by $\{x_1, x_2, x_4, x_5, x_8, x_{11}\} \in Bar_{no}$ and $\{x_3, x_6, x_7, x_9, x_{10}, x_{12}\} \in Bar_{yes}$. Before splitting the data the misclassification error was 50%

and for both Bar_{no} and Bar_{yes} it is still 50%. So Bar is a bad choice as it has no effect on the classification error.

Although, classification error can be used to grow a decision tree, it has some shortcomings: it does not favor pure nodes i.e., nodes that have instances belonging to only one class. As an example assume that we have a binary classification task with each class having 50 instances, represented as $(50, 50)$. Consider two splits, one which results in $(10, 40)$ and $(40, 10)$ and another that produces $(50, 20)$ and $(0, 30)$. In both cases the classification error is 20%, but the latter split produces a pure node which should be preferred over the former. Therefore, instead of looking at classification error we use other measures that characterize the *purity/impurity* of a node, so that our goal is to prefer splits that produce pure nodes. It should be noted here that the *impurity of a node is measured in terms of the target variable i.e., the class labels and not the feature itself.*

3.1.2 Node Impurity Measures

Entropy: The entropy for a node q having N_q instances is defined as:

$$H(q) = -\sum_{i=1}^{k} p_{qk} \log_2 p_{qk} \tag{3}$$

where k is the number of classes. Entropy measures the impurity of a given node: if all the instances belong to one class then entropy is zero (for this we would need to assume that $0 \log_2 0 = 0$). Similarly, for a uniform distribution of class labels i.e., there is an equal number of instances belonging to each class entropy reaches its maximum value. This can be seen in the case of binary classification in Figure-1a, where the entropy function has a maximum at $p = 0.5$.

In order to select the best feature to further split a node we want the maximum reduction in entropy, i.e., the child nodes should be as pure as possible (or less impure than their parent). If we split node q based on feature V, that has $|V|$ distinct values (resulting in a $|V|$-way split) the reduction in entropy is calculated as:

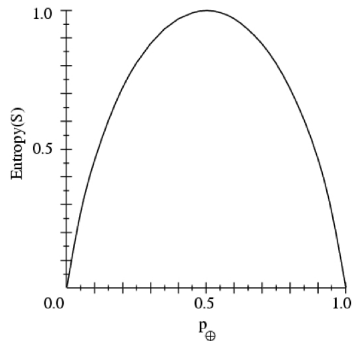
$$IG(q, V) = H(q) - \sum_{i=1}^{|V|} \frac{N_i}{N_q} H(i) \tag{4}$$

where N_i is the number of instances in the i^{th} child node and $H(i)$ is its entropy calculated using (3). (4) is known as information gain, and we would select the feature with the maximum information gain to split node q.

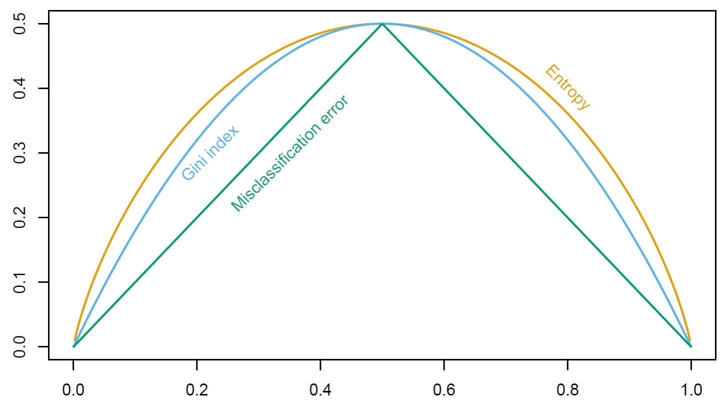
Gini Index: The Gini index for a node q having N_q instances is defined as:

$$Gini(q) = \sum_{i=1}^{k} p_{qk}(1 - p_{qk}) \tag{5}$$

where k is the number of classes. Gini index is another impurity measure that is extensively used and is similar to entropy as can be seen from Figure-1b. It reaches its maximum value when the instances are equally distributed among all classes, and has a value of zero when all instances belong



(a)



(b)

Figure 1: (a) The entropy and its comparison to other node impurity measures (b) for binary classification tasks. The x-axis in both cases is the probability that an instance belongs to class 1, and in (b) the entropy function has been scaled down for ease of comparison.

to one class. Similar to information gain (4), we can measure the gain in Gini index when splitting on feature A:

$$Gain_{Gini}(q,V) = Gini(q) - \sum_{i=1}^{|V|} \frac{N_i}{N_q} Gini(i) \tag{6}$$

where N_i is the number of instances in the i^{th} child node and $Gini(i)$ is calculated using (5). The feature with the maximum gain will be used to split node q.

3.1.3 Binary Splits vs Multiway Splits:

Recursive binary splits are more desirable than multiway splits, because multiway splits can fragment the training data during the early tree growing process and leave less data for the later stages. Since, any k-ary split can be represented equivalently as a sequence of binary splits (*homework problem*), using a binary splitting strategy is preferable.

- Ordinal Features: The values taken by an ordinal feature are discrete but can be ordered. The features *Price*, *Patrons* and *WaitEstimate* are examples of ordinal features. To create a binary split based on an ordinal feature $x_{(i)}$ with 'k' distinct values, we can select one of the feature values as the threshold θ and then split the data into two subsets corresponding to $x_{(i)} < \theta$ and $x_{(i)} \geq \theta$. Figure-2 shows an example of creating a binary split for an ordinal feature.

- Numerical Features: A multiway split for a continuous feature, can result in an N-ary split, where N is the total number of training instances, so that each child has only one instance. To avoid this situation we transform a continuous feature into a binary feature by defining a suitable threshold θ, and then splitting the data as $x_{(i)} < \theta$ and $x_{(i)} \geq \theta$. Let $x_{1i}, x_{2i}, \ldots, x_{Ni}$ be the sorted values for the i^{th} feature. Then the optimal threshold can be found by setting the threshold as each of the $N-1$ mid-points i.e., $\frac{x_{ji}+x_{(j+1)i}}{2}$; $\forall j \in \{1, 2, \ldots, N-1\}$, and identifying the threshold value that maximizes the gain.

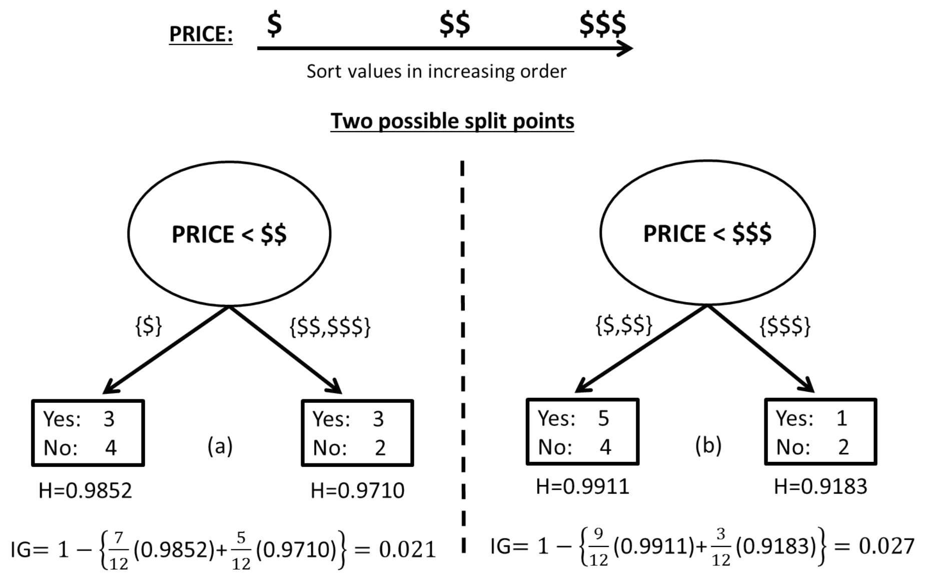

PRICE:
$
$$
$$$
Sort values in increasing order
Two possible split points
PRICE < $$
{$}
{$$,$$$}
Yes: 3
No: 4
(a)
Yes: 3
No: 2
H=0.9852
H=0.9710
IG= 1 − {7/12 (0.9852)+5/12 (0.9710)} = 0.021
PRICE < $$$
{$,$$}
{$$$}
Yes: 5
No: 4
(b)
Yes: 1
No: 2
H=0.9911
H=0.9183
IG= 1 − {9/12 (0.9911)+3/12 (0.9183)} = 0.027

Figure 2: Defining a binary split for an ordinal variable using the *Price* feature from Example-1. The left branch contains feature values that are $<$ the testing value, and the right branch has all values that are $\geq$ the testing value.

- Nominal (Categorical) Features: Defining a binary split for nominal features entails partitioning the set of all possible feature values in two non-empty and non-overlapping subsets. For a nominal feature $A \in \{a_1, a_2, \ldots, a_v\}$ this requires evaluating $2^{v-1} - 1$ possible binary partitions. For example consider the *Type* feature from Example-1. An exhaustive search over the possible binary partitions would entail evaluating the following splits:

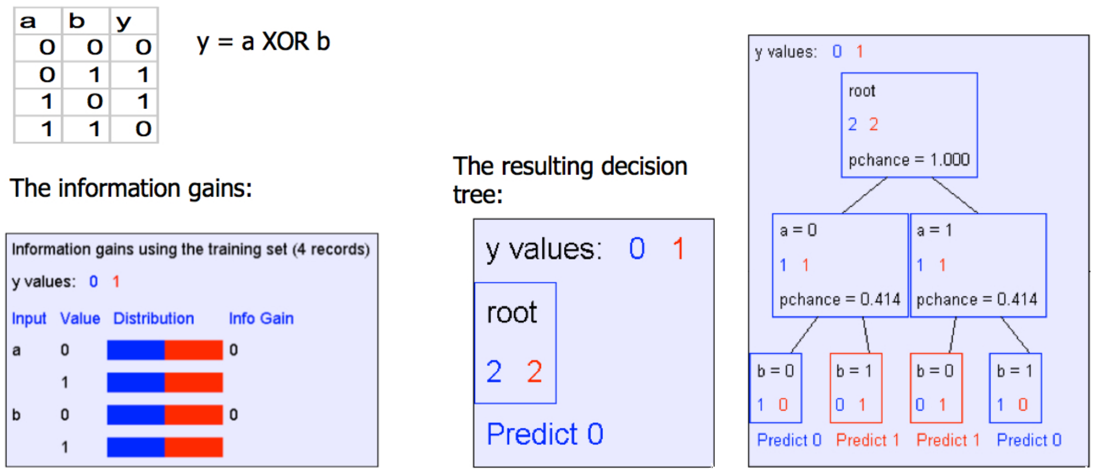
  1. {Burger},{French,Thai}
  2. {French},{Burger,Thai}
  3. {Thai},{Burger,French}

  One possible strategy to avoid an exhaustive search is to replace the original feature by v dummy boolean features, where the i^{th} dummy feature represents the indicator function $\mathbb{I}(x_{ji} = v_i);\ \forall j \in \{1, 2, \ldots, N\}$.

4 When to stop splitting?

- All training instances at a given node have the same target/decision value. REMARK: $H(Y) = 0$
- No attributes can further distinguish records. REMARK : $H(Y|X) = H(Y) \rightarrow IG = 0$ for all features X

Is it a good idea to stop splitting if the information gain is zero?



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a XOR b

The information gains:

Information gains using the training set (4 records)

y values: 0 1

Input	Value	Distribution	Info Gain
a	0	🟦🟥	0
	1	🟦🟥	
b	0	🟦🟥	0
	1	🟦🟥	

Figure 3: "Do not split" VS "split" when Information gain is 0