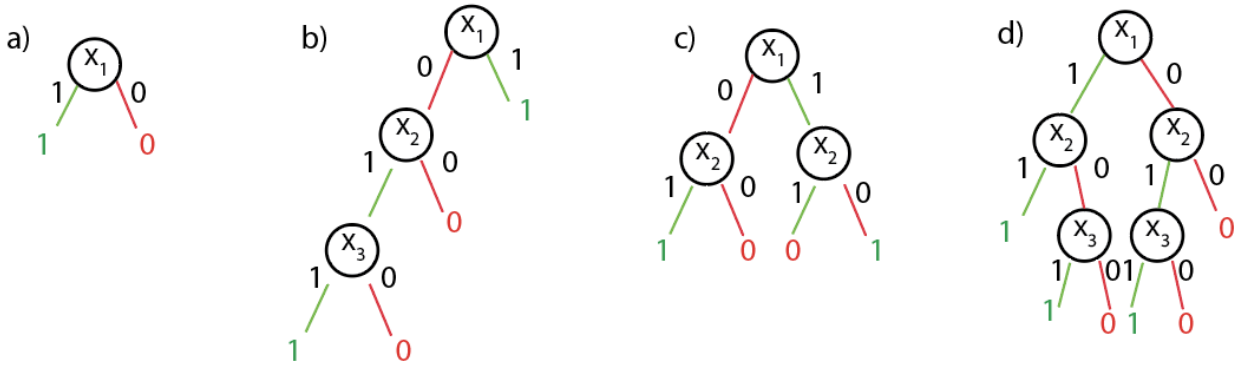
CS 6350: Machine Learning Fall 2020

Homework 1

Michael Young / u0742759

1. Decision trees

1. Boolean functions -> decision trees



d continued) The efficiency of using a decision tree to represent m-of-n functions depends on the relationship between m and n. If m is either very low, or very high with respect to n, then there could be some reasonable information gain happening between levels of the tree. But if m is somewhere in the middle, the tree can balloon with several nodes at each level offering not much in the way of information. So I would say that it depends, but in general using decision trees to represent m-of-n functions is not a great idea.

2. To wait or not to wait?

a) How many possible functions are there to map these four features to a boolean decision? How many functions are consistent with the given training dataset?

There are $2 * 2 * 3 * 4 = 48$ possible outputs over these four features, and each output is binary, meaning there are a total of $2^{48} = 281,474,976,710,656$ functions possible to map these four features to a boolean decision. We are given 9 instances with their corresponding labels (meaning we know 9 rows of the 48 rows of the truth table). Given that we know these 9 rows, that means that there are $2^{48-9} = 2^{39} = 549,755,813,888$ functions that are still consistent with the given training

dataset.

b) What is the entropy of the labels in this data? When calculating entropy, the base of the logarithm should be base 2.

There are 5 Yes labels in this dataset, and 4 No labels. Therefore,

$$\begin{aligned} Entropy(S) = H(S) &= -p_+ log_2(p_+) - p_- log_2(p_-) \\ &= -\frac{5}{9} log_2(\frac{5}{9}) - \frac{4}{9} log_2(\frac{4}{9}) \\ &\approx 0.991 \end{aligned}$$

c) What is the information gain of each of the features?

Information gain is defined as:

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

where S is the entire set, and A is the attribute we are splitting on. Starting first with the "Friday" feature, we see that there are two values it could take: Yes or No. $\frac{6}{9}$ examples belong to"No", and within those 6, they are split between 4 Yes and 2 No. $\frac{3}{9}$ examples belong to "Yes", and within those 3, 2 are labeled Yes and 1 is labeled No. Thus:

$$\begin{aligned} Gain(S, Friday) &= Entropy(S) - \frac{3}{9} Entropy(S_{Friday->Yes}) + \frac{6}{9} Entropy(S_{Friday->No}) \\ &= Entropy(S) - \frac{3}{9}(-\frac{1}{3} log_2(\frac{1}{3}) - \frac{2}{3} log_2(\frac{2}{3})) + \frac{6}{9}(-\frac{4}{6} log_2(\frac{4}{6}) - \frac{2}{6} log_2(\frac{2}{6})) \\ &\approx 0.0728 \end{aligned}$$

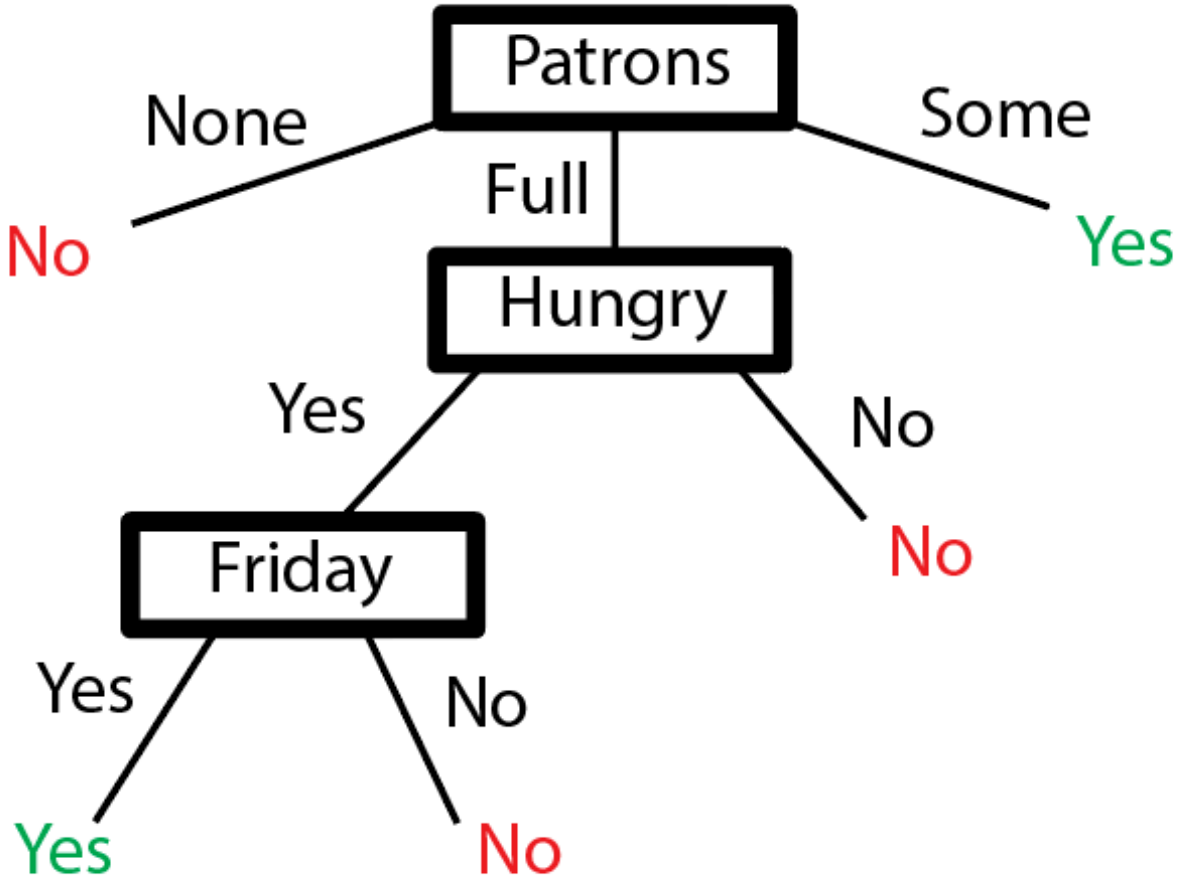
Proceeding in a similar fashion for all features, we see:

$$\begin{aligned} Gain(S, Hungry) &\approx 0.2294 \\ Gain(S, Patrons) &\approx 0.6305 \\ Gain(S, Type) &\approx 0.1567 \end{aligned}$$

d) Which attribute will you use to construct the root of the tree using the ID3 algorithm?

The ID3 algortithm will select the root attribute based on which one reduces the label entropy the most, i.e. the one that has the highest information gain. Based on the calculations made in the previous part, this attribute is *Patrons*.

e) Using the root that you selected in the previous question, construct a decision tree that represents the data. You do not have to use the ID3 algorithm here, you can show any tree with the chosen root.



f) Suppose you are given three more examples, listed in table 2. Use your decision tree to predict the label for each example. Also report the accuracy of the classifier that you have learned.

Example 1: Patrons: Full -> Hungry: Yes -> Friday: Yes -> Label: **YES**

Example 2: Patrons: None -> Label: **NO**

Example 3: Patrons: Full -> Hungry: Yes -> Friday: Yes -> Label: **YES**

$Accuracy = \frac{2}{3}$

3. Exploring other impurity measures

a) Missclassification

i) Write down the definition of the information gain heuristic that uses the misclassification rate as its measure of impurity instead of entropy.

Misclassification is defined as $Misclassification(S) = 1 - \max_i p_i$ where p_i is the fraction of examples that have label i and the maximization is over all labels. The information gain heuristic using misclassification as an impurity measure instead of entropy would be:

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

ii) Use your new heuristic to identify the root attribute for the data in Table 1.

In order to identify the root, we must compute the information gain across each split as we did when entropy was our heuristic. Starting first with the "Friday" feature, we see that there are two values it could take: Yes or No. $\frac{6}{9}$ examples belong to"No", and within those 6, they are split between 4 Yes and 2 No. $\frac{3}{9}$ examples belong to "Yes", and within those 3, 2 are labeled Yes and 1 is labeled No. Thus:

$$\begin{aligned} Gain(S, Friday) &= Misclassification(S) - \frac{3}{9} Misclassification(S_{Friday->Yes}) + \frac{6}{9} Misclassification(S_{Friday->No}) \\ &= 1 - \max(\frac{5}{9}, \frac{4}{9}) - \frac{3}{9}(1 - \max(\frac{1}{3}, \frac{2}{3})) + \frac{6}{9}(1 - \max(\frac{4}{6}, \frac{2}{6})) \\ &= \frac{1}{9} \end{aligned}$$

Likewise for splitting on other features we get:

$$\begin{aligned} Gain(S, Hungry) &\approx 0.222 \\ Gain(S, Patrons) &\approx 0.333 \\ Gain(S, Type) &\approx 0.111 \end{aligned}$$

Based on this heuristic, we would select "patrons" as our root attribute, just as in the case when entropy was our heuristic.

b) Another heuristic that is used to define impurity is the Gini coefficient, which is defined as $Gini(S) = \sum_i p_i(1 - p_i)$. Use Gini coefficient to identify the root attribute for the training data in Table 1.

Similarly for misclassification and entropy, using Gini gives us the gain formula:

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

For splitting on Friday, this looks like:

$$= \frac{5}{9}(1-\frac{5}{9}) + \frac{4}{9}(1-\frac{4}{9}) - (\frac{3}{9}(\frac{1}{3}(1-\frac{1}{3}) + \frac{2}{3}(1-\frac{2}{3})) + \frac{6}{9}(\frac{2}{6}(1-\frac{2}{6}) + \frac{4}{6}(1-\frac{4}{6})))$$
$$= \frac{4}{81}$$
$$\approx 0.0494$$

Proceeding this way for the rest we get:

$$Gain(S, Hungry) \approx 0.149$$
$$Gain(S, Patrons) \approx 0.327$$
$$Gain(S, Type) \approx 0.086$$

Once again, according to this metric, we would use "Patrons" as our root node because it gives us the highest information gain.

2. Experiments

1. Full Trees

Discuss what approaches and design choices you had to make for your implementation and what data structures you used.

The first big design choice here was how to store and access the data. I chose to create a "Data" class that would take the data path as an input, and load/store the LIBSVM data line by line in a dictionary where the key was the index and then the value was another dictionary composed like: {"label": +1, "features": [3, 4, 5]}. I chose to dispose of the "index:value" format of the features and instead stored just the index, because all of the values were 1. I then created several helper functions like "get_common_label( )" or "split_on_attr( )" to make the tasks needed for this data in the decision tree pipeline straightforward.

Once I had my data/data tasks figured out, I defined the entropy, gini, and information gain functions for usage in the ID3 algorithm. To tackle the decision tree itself, I defined a Node class that had level, attribute, information gain, label, right and left as attributes. Leaf nodes in the tree would only have level and label values. I then defined the DecisionTreeClassifier class which had methods to build a tree via the ID3 algorithm, get prediction accuracy and error, and track the depth of the tree.

In retrospect, I would've handled loading and storing my data differently. Storing the data in a numpy array would've made things much more efficient and generalizable, and in future homeworks I'll try to take that approach.

Here are the results on the training and test sets using a full tree:

1. Root feature: 40
2. Information Gain: 0.15394533474074124
3. Max Depth: 34
4. Training set error: 0
5. Test set error: 0.22615971055691952

2. Gini index

Implementation: Made sure to specify a generic "purity function" as an input to my information gain function, that made it very simple to simply pass in the "compute gini" function instead of my "compute entropy" function to accomplish this part of the homework. I implemented both the gini and entropy functions according to the definitions supplied earlier in this homework.

Results:

1. Root feature: 40
2. Information Gain: 0.07374129050815775
3. Max Depth: 30
4. Training set error: 0
5. Test set error: 0.2193435844424344

Using the Gini index actually produced a slightly smaller test error than using entropy as my purity measure. The Gini index split on the same root feature, but given that it gave a different test error and it's depth ended up being different, I have to conclude that it ultimately produced a different tree. This is interesting, as based on the graphs of the gini index compared with entropy I would have assumed they would produce identical results.

3. Limiting Depth

a) After running 5-fold cross validation and experimenting with depths ranging from 1-5, here are the results:

Depth: 1 - Accuracy: 0.7554590570719603 +/- 0.011270573868374292
Depth: 2 - Accuracy: 0.8074648469809759 +/- 0.019260252042547867
Depth: 3 - Accuracy: 0.8100413564929694 +/- 0.02057577837499194
Depth: 4 - Accuracy: 0.8049131513647643 +/- 0.018073372276576235
Depth: 5 - Accuracy: 0.8055541770057898 +/- 0.01680903915088519

Based on these results, I think the winner here is using 3 as the depth, mainly because it has the highest accuracy. This isn't necessarily the clearest cut choice though because it also has the highest standard deviation. Even still, the standard deviation at 3 is very near the standard deviations of its closest competitors (2,4, and 5) so I think choosing 3 despite the standard deviation is a reasonable move.

b) Retraining my tree using a depth of 3 yields the following results:

DEPTH LIMITED - training accuracy: 0.8164313222079589
DEPTH LIMITED - training error: 0.1835686777920411
DEPTH LIMITED - test accuracy: 0.8221992505491665
DEPTH LIMITED - test error: 0.17780074945083346

c) Discuss performance of depth limited tree vs full tree and talk about if limiting depth is a good idea:

With a full tree, our training error was 0, which means that a full tree can perfectly conform to given training set. Unfortunately this doesn't mean anything for future performance - in fact it actually means a full tree will probably do worse on future sets of data, because it's memorized a very specific set of data. Examining the test error bears this out: the test error for a full tree was 0.226, while the test error for our depth limited tree was 0.177. This demonstrates that limiting depth for decision trees is actually critical for preventing overfitting and ensuring better performance on unseen data. It's likely that with this dataset, there are only a few features that were super relevant to predicting the labels, which is why such a short depth performed better than the full tree. Limiting depth helps the tree to filter out the signal from the noise.

4. Fairness concerns

Even a perfect classifier may incorporate societal biases because it's making the predictions based on certain features which themselves may be biased. Say for instance that this decision tree is used to decide whether or not a candidate will be approved for a loan or not. Of course such things like gender, race, sexual orientation etc. should not be factored into a decision like that. Someone using this Adult dataset may filter out those features and build a perfect decision tree without them, but it's completely possible that there are residual features which correlate highly with the features they chose to initially not consider, which means that there must be some inherent bias still present in the tree.