CS5350 Assignment 1

Corey Schulz
u0945949

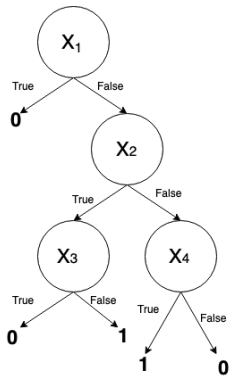
September 4, 2019

1 Decision Trees

1. (a)

$$\neg x_1 \wedge (x_2 \oplus x_3)$$

This Boolean function can be represented by a decision tree as follows:

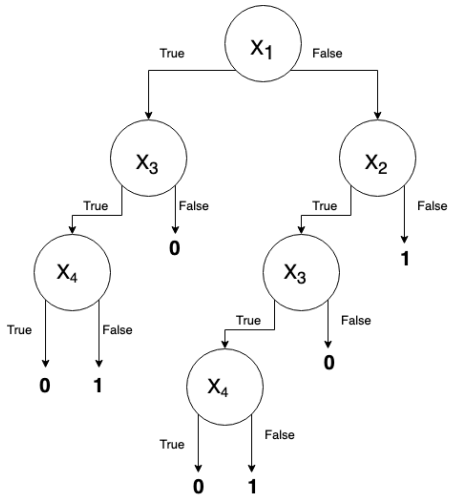


This function cannot be split with a linear classifier as it uses XOR, and when you XOR something it's simply not possible to separate the output space with a single line or plane.

(b)

$$\neg(x_1 \vee x_2) \vee (x_3 \wedge (\neg x_4))$$

This Boolean function can be represented by a decision tree as follows:

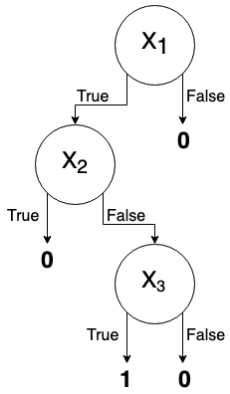


The Boolean function cannot be represented by a linear classification, as the points are scattered in 4D, with points of each potentially in each of the four quadrants, making it unable to be linearly classified.

(c)

$$x_1 \wedge \neg x_2 \wedge x_3$$

This Boolean function can be represented by a decision tree as follows:



This can be split with a linear classifier, as one point is positive and all others are negative in a 3D space, so the other points can easily be partitioned off, apart from the positive point. A plane as such can be defined as:

$$0 = (x_1 - 1) + (x_2 - 0) + (x_3 - 1) - .1$$

This plane will separate the negative points from the positive point.

2. (a) There are four classifications being made:
   i. **number of rooms** : (one, two, three or four rooms) *4 options*
   ii. **apartment condition** : (poor, fair, good, excellent) *4 options*
   iii. **distance from college** : (x $<$ 1 mile, 1 $\leq$ x $\leq$ 5 miles, x $>$ 5 miles) *3 options*
   iv. **price** : (x $<$ 500, 500 $\leq$ x $\leq$ 1000, x $>$ 1000) *3 options*

   So, we can determine the number of possible functions for these four features by the following:

$$4 * 4 * 3 * 3 = 144 \textit{ possible options}$$

$$2^{144} \approx 2.23007 * 10^{43} \textit{ possible functions}$$

The number of functions consistent with the given data set, however, is a smaller number. Since each row of the training data halves the number of possible functions, and there are *16 rows*, the number of functions consistent with the given data can be determined by:

$$\frac{2^{144}}{2^{16}} = 2^{128} \approx 3.4028 * 10^{38} \textit{equations.}$$

(b)

$$\textit{Entropy(S)} = H(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

Where the proportion of positive examples is p_+ and the proportion of negative examples is p_-.
In the given data, the proportion of "like"s is 9/16 and "dislikes" comprise 7/16 proportionally.

$$H(Like) = -(\frac{9}{16}) \log_2(\frac{9}{16}) - (\frac{7}{16}) \log_2(\frac{7}{16})$$

$$H(Like) = 0.9887$$

(c)

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

Information gain for number of rooms is calculated as follows:

$$\textit{H(one room) (T = 3/3, F = 0/3)} = 0.0$$
$$\textit{H(two rooms) (T = 2/4, F = 2/4)} = 1.0$$
$$\textit{H(three rooms) (T = 2/6, F = 4/6)} = 0.9183$$
$$\textit{H(four rooms) (T = 2/3, F = 1/3)} = 0.9183$$

$$\textit{Expected-Entropy(num rooms)} = \frac{3}{16}(0.0) + \frac{4}{16}(1.0) + \frac{6}{16}(0.9183) + \frac{3}{16}(0.9183) = 0.76654$$

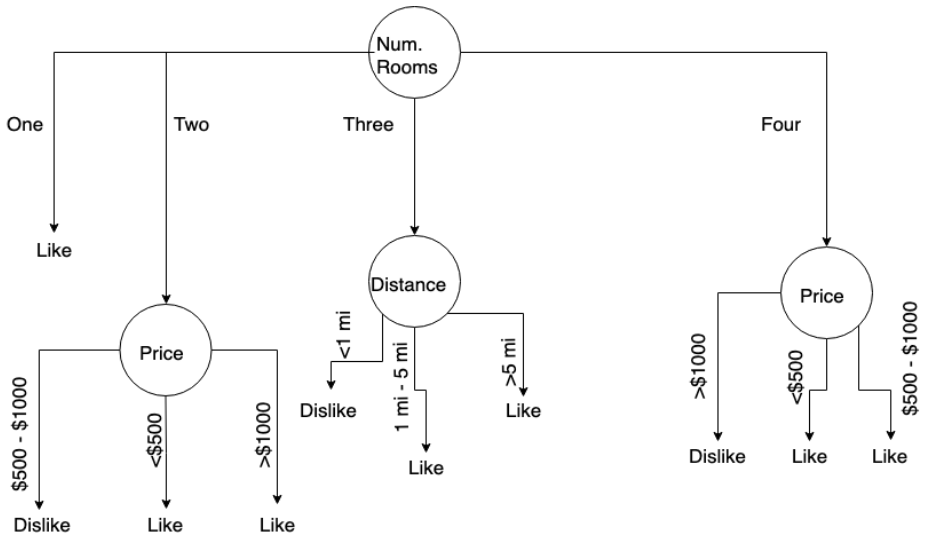
$$\textit{Information-Gain(num rooms)} = \textit{H(Like) - Expected-Entropy(num rooms)}$$
$$\textit{Information-Gain(num rooms)} = .9887 - 0.76654 = 0.222158$$

...The process for calculating information gain for each of the three remaining features (apartment condition, distance, price) is identical to the above, and I wrote a Python script to calculate it, so to save space the calculations have been omitted. Then, the information gain for each feature is as follows:

Feature	Information Gain
Number of rooms	0.222
Apartment condition	0.013
Distance	0.195
Price	0.184

(d) By using the information gain heuristic of ID3, we can determine that it's best to split on ***number of rooms*** as it had the highest information gain out of all the features.

(e) Using *number of rooms* as the root of the tree, a depth limited decision tree that represents the data is as follows:



...The decision tree was not made by ID3 necessarily, but by logical breaks in the data. Predicting the outcomes of all the data in *Table 3: Test Set* yields the same results as in the label section for each of the examples. As such, the accuracy of the learned classifier on the test set is 1.0, though it almost certainly won't stay that high should more examples be introduced into the test set.

3. (a) Since the Gini Index and entropy can be functionally interchanged in computations, an equation that calculates information gain using Gini can be defined as follows:

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

(b) In the case of *num rooms*, the information gain can be calculated as thus:

$$\begin{aligned}
\textit{Gini(one room) (T = 3/3, F = 0/3)} &= 1 - (\frac{3}{3}^2 + \frac{0}{3}^2) \\
&= 0.0 \\
\textit{Gini(two rooms) (T = 2/4, F = 2/4)} &= 0.5 \\
\textit{Gini(three rooms) (T = 2/6, F = 4/6)} &= 0.44444 \\
\textit{Gini(four rooms) (T = 2/3, F = 1/3)} &= 0.44444
\end{aligned}$$

$$\textit{Expected-Entropy(num rooms)} = \frac{3}{16}(0.0) + \frac{4}{16}(0.5) + \frac{6}{16}(0.44444) + \frac{3}{16}(0.44444) = 0.37499$$

$$\textit{Gini(S) (T = 9/16, F = 7/16)} = 1 - \left(\frac{9}{16}^2 + \frac{7}{16}^2\right) = 0.49218$$

Finally,

$$\textit{Information Gain(S, A)} = 0.49218 - 0.37499 = .11718$$

To save space, the calculations for the other features have been omitted, but the final table looks like:

Feature	Information Gain (using Gini Index)
Number of rooms	0.117
Apartment condition	0.009
Distance	0.126
Price	0.121

(c) According to the results by calculating Gini Index, *distance* should be the root for the decision tree. These two measures, then, *do not* lead to the same tree.

2 Experiments

1. **Implementation: Full Trees**

   (a) The hardest part about implementing the ID3 algorithm was storing data in a suitable manner and being able to extract data out of the CSV. I think I'll use Pandas or Numpy moving forward to help with that. The actual implementation of ID3 is routine, following the pseudocode. I ended up making a Node class to house the decision tree, and built it recursively along the course of the ID3 algorithm. The entropy calculations are as you'd expect as well. I started by implementing the entropy calculations (because I wanted to use them on the math portion of the homework) and implemented ID3 thereafter. I won't do that in the future, as I'd like a more solid foundation on the storage of data, which my approach didn't give. Next time I'll make the Node class, for example, first and build it from there. For information gain calculations, I processed each label and then compared to find the greatest information gain after calculating the entropies. I use dictionaries to store most interim values.

   (b) The error of my decision tree for the examples in *data/train.csv* is 0%. This is as you'd expect, because with unrestricted depth, the ID3 algorithm will run until every item in the training data can be placed with 100% certainty.

(c) Running the accuracy calculations on the test data yields 11.712% error, or 88.288% accuracy. I believe this is in line with what you'd expect from a simple model like ID3.

(d) The maximum depth of the decision tree ID3 provides after running on the training data is *9*.

Additionally, The root feature chosen by ID3 was *spore-print-color* with an information gain of 0.424(!).

3 Limiting Depth

(a) The tested hyperparameters, accuracy, and standard deviations are as follows:

Hyperparameter	Avg. Accuracy	Std. Dev
1	66.541	5.168
2	92.707	2.613
3	93.459	1.946
4	93.458	1.755
5	92.632	3.003
10	96.917	6.475
15	97.067	6.556

...I believe that the best chosen depth is *4*. It has a high accuracy, with a low standard deviation. It's not quite as accurate as a maximum depth of 10 or 15, but it keeps the computation time relatively low while maximizing, to some extent, accuracy. Though the "best" accuracy might depend on what you need your ML model to do. If you want to prioritize accuracy with it, for example, then you may choose a larger hyperparameter. But for the average case, I'd say a depth of 4 has a pretty good min(computation time) / max(accuracy).

(b) Using the depth with the *greatest accuracy* from the above table (a depth of 15), with a model trained on the training data and tested on the testing data, an accuracy of **88.288** is the result. This is the same as the unrestrained data because, with no restraints the depth of the decision tree produced by the training data only goes 9 deep.

(c) Limiting depth of a decision tree can increase evaluation performance of the testing data, not necessarily in accuracy, but in completion time. The point of k-fold cross validation testing is that you can find the point where the computation time is reduced, while keeping up the accuracy. So it will depend on the model whether limiting the depth is a good idea. Here, it pays off as the accuracy stays roughly consistent throughout all the validation tests, but when you increase the depth hyperparameter, the computation time increases too. I think that, in the general case, limiting

depth is a good idea. It's an optimization problem that, in general, is worth doing to keep accuracy high and costs low.