CSCE 478 Homework 1

Jakob Snyder(jsnyde) and Clayton Henderson(chenders)

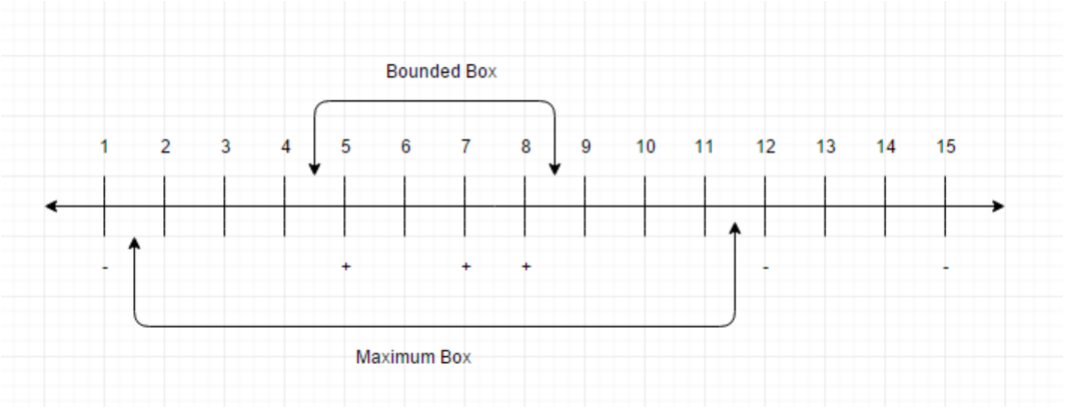
26 september 2016

Problem 1 *Assume that you are given a set of training data {(5, +), (8, +), (7, +), (1, -), (12, -), (15, -)}, where each instance consists of a single, integer-valued attribute and a binary label. The label comes from a function C, which is represented as a single interval. Formally, the interval is represented by two points a and b, and an instance x is labeled as positive if and only if* $a \leq x \leq b$.

1a) Describe a hypothesis that is consistent with the training set.

The hypothesis we came up with as consistent with the given dataset was $5 \leq x \leq 8$

1b) Describe the version space and compute its size.



The size of the version space is 17.

1c) Suppose that your learning algorithm is allowed to pose queries, in which the algorithm can ask a teacher the label C(x) for any instance x. Specify a query x_1 that is guaranteed to reduce the size of the version space, regardless of the answer. Specify a query x_2 that is guaranteed to not change the size of the version space, regardless of the answer.

A query of $x_2 = 6$ will not change the size of the version space because we already know it will be a + response

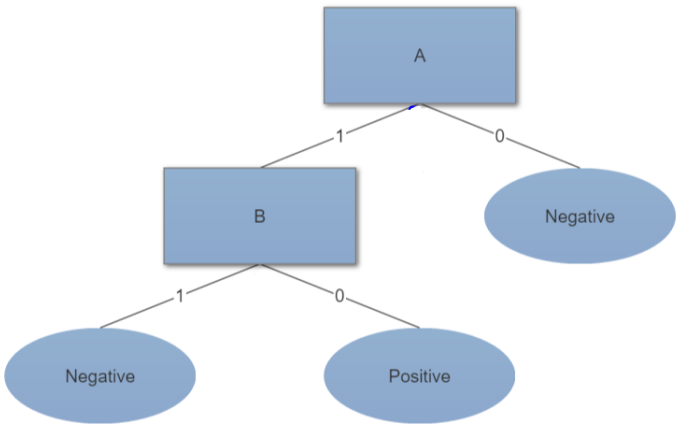
1d) Suppose that you start with a size-three training set $\{(x_1, -), (x_2, +), (x_3, -)\}$, where $0 < x_1 < x_2 < x_3$. Describe a series of fewer than $2(log_2 x_3)$ queries that will reduce the size of the version space to a single hypothesis.

Pick either the left or right side of the bounded box. Once picked find the midpoint between the + and - labels and query that point. If it is positive then find the next midpoint between it a the - label and query it. If it is a negative point then query the midpoint between the the 1st query and the original + value. Repeat this until you get a + and - directly next to each other then repeat for the opposite side of the bounded box.

Problem 2 *Give a decision tree to represent each of the following boolean functions ($\oplus$ is exclusive OR, $\sim$ is negation):*

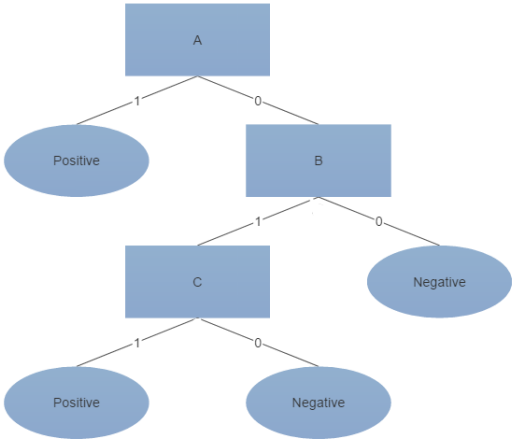
2a)

A	*B*	*F*
0	*0*	*0*
0	*1*	*0*
1	*0*	*1*
1	*1*	*0*



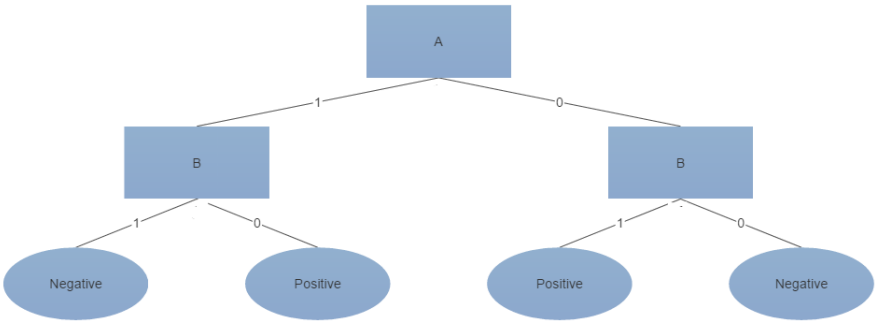
2b)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



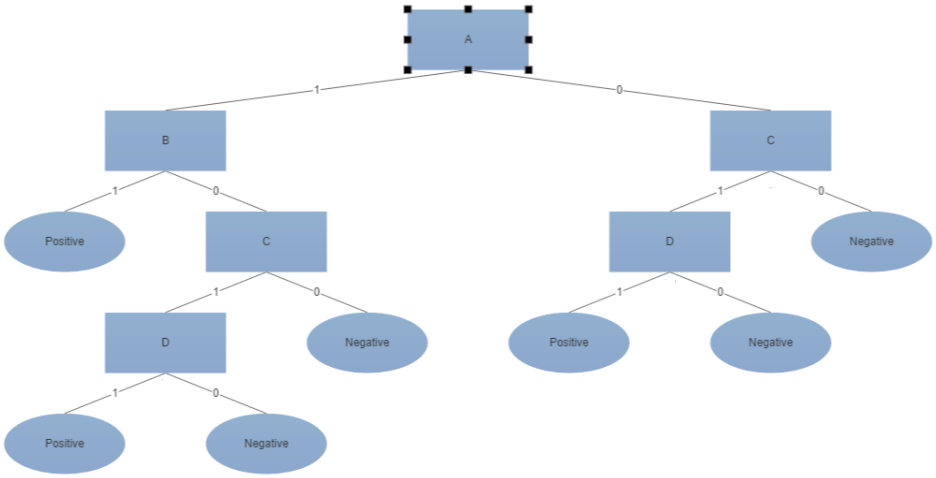
2c)

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



2d)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



Problem 3 *Consider the following set of training examples and answer the questions below. Show your work.*

Instance	*Label*	a_1	a_2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

3a)
$I_m \equiv -p_m^{\oplus} log_2(p_m^{\oplus}) - p_m^{\ominus} log_2(p_m^{\ominus})$
$I_m = -.5log_2(.5) - .5log_2(.5) = 1$
3b)
$I_{a1} = -(-.5log_2(.5) - .5log_2(.5)) = 1$
$I_{a2} = -(-.667log_2(.667) - .333log_2(.333)) = .918$
So you would split on a2

Problem 4 *Implementing the ID3 Algorithm*

1 Introduction

With this problem we want to Implement an ID3 decision tree. We were given some key restrictions while solving this problem. The first restriction was we needed three distinct data sets. Two of these, Monks and Voter Data, were given to the class and the third was selected by us. The next restriction was that we needed to use large data sets. In the long run this actually helped us because a larger data set resulted in a more accurate algorithm. The third restriction was for us to avoid using data sets with unspecified values. This also helped us because it kept our algorithm simpler.

2 Implementation

Our implementation begins with simply reading from three data sources: monks.data, poker.data, and votes.data. Our three parsing methods are made specifically for each data set, and the types of attributes that are associated with them. After the data is properly parsed into lists, a randomly selected training and testing set are chosen for the creation and validation of our tree. Once the training and testing data was set aside, the method generateTree() is called with the training set as its only parameter.

The generateTree() method is a recursive function which first analyzes the training data and identifies all possible attribute values for each attribute in the training set. Next, the algorithm identifies the ideal attribute to split the training set on by computing the entropy for each attribute value. This is accomplished by use of two functions: splitAttribute() and calculateEntropy().

At this point, the most suitable attribute to split the training set on has been chosen. The split attribute is associated with the root node, and the training set is properly split into a number of lists that reflect their relationship to the root. Those smaller lists then become the new training sets as the generateTree() method is recursively called until all of the remaining datapoints have the same label.

3 Evaluation

While evaluating our algorithm we used the thirty data points that we set aside before training our tree. As mentioned before these data points were different every time but sometimes our trees did differ slightly when running because of the randomness of the training data. To evaluate our algorithm we first printed it out in order to see what it looked like. After this we tested each data point using a while loop and waiting to hit a leaf. If the leaf label matched the value of the label on the data point than the tree worked correctly. If the labels did not match or if there was an error while traversing then we added 1 to an error count. At the end we divided the number of errors by 30 to see what percentage of the data set was labeled correctly by the tree.

One key error that we ran into was that we did not properly handle the '?' in the Voting data set. This leads to a high error and a tree that is not optimal.

4 Conclusion

Our implementation of the ID3 algorithm produced generally intelligent trees based on the input data. We had a few significant struggles that forced us to run out of time for the next portion of the project which was rule post pruning. The first large struggle we encountered was finding a data set that was similar to the others so we would not have to change our parse algorithm extensively. We eventually found the 'Poker' data set. With this set, the attributes were somewhat similar but the labels were much different. To get around this we changed how the labels were parsed for this data set. Originally the data set has labels with values 0-9 but we changed this to 0-1 by taking any labels with values $<= 4$ as 0 and > 4 as 1. This made it so we did not have to change our algorithm for this data set at all. We were going to try to figure out how to change our algorithm for it to be able to handle more than binary labels. Another key struggle we had was trying to calculate the entropy correctly. We had many errors where the program was dividing by 0 and the entropy was returning NaN. We fixed this by adding some if statements that checked for these cases.

Problem 5 *Implementing rule post pruning*

Unfortunately we were not able to implement this due to time restrictions and the fact that the ID3 algorithm took much longer than expected. Although we were not able to complete this we did create a plan on how we would implement it. The first step of this process would be to turn our tree into a set of rules. We would achieve this by traversing down to every

leaf and creating 'and' statements that would result in a leaf value. After this is completed we would remove parts of the rules in order to make them more simple but to also keep the error very low. It was very confusing to figure this out because the lecture slides talk about increasing the accuracy when the rules should have complete accuracy if the data set is large enough. Also based on our testing our tree had 100% accuracy so this also made it more confusing.