Decision Trees

Aaron Barnett
acba242@uky.edu

Abstract

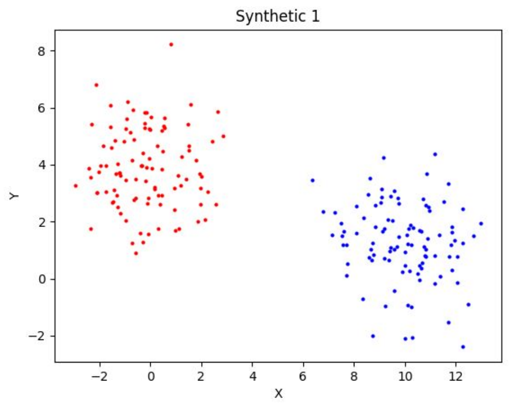
Decision trees for this paper are created using the ID3 algorithm. This paper will outline accuracy seen in tree prediction. Certain implementation details will be covered as well.

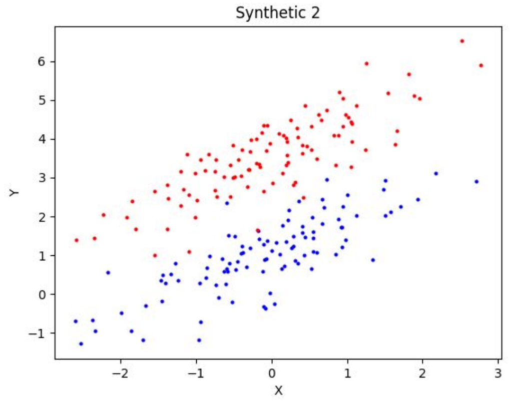
Beginning with implementation details, the tree structure was created using a node class. The node class includes a children list, the feature that node splits the data on, and the prediction of the node if it's a leaf. The discretization of the data is done in two separate ways. The synthetic data is discretized using the pandas cut() function to find the correct interval to place all examples into four bins. Four bins were chosen for simplicity. The pokemon data uses the minimum and maximum value in each column to find a proper interval to split the data into six bins. Six bins were chosen because the generation column is nominal and can take on six possible values. All trees were limited to a depth of three for accuracy calculations. A struggle I had when generalizing the ID3 code to the pokemon data was finding the values a column could take on once the data was split. The possible values in a column would change as the data was purified causing nodes to have incorrect numbers of children. To solve this I created a global variable to hold the original data so unique values in a column would be static throughout the tree creation process. With more time, I would implement a different way of tracking the unique values in a column during tree creation.

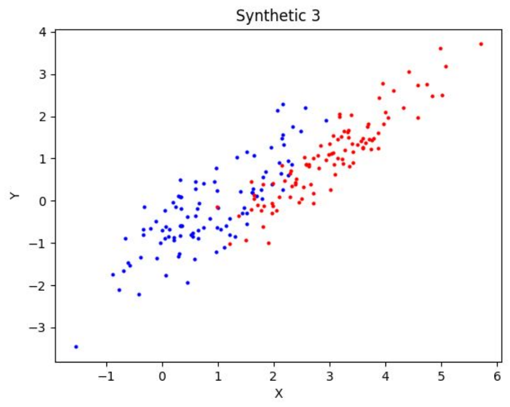
Training set error for all synthetic datasets varied by almost 20%. The tree for the third dataset saw the most error in prediction, leading me to believe a different number of bins might be necessary for better accuracy in that space. The accuracy for synthetic dataset one was 100%. The two classes of data points were distanced well leading the tree to classify the data using only the 'x' feature. Accuracy for synthetic dataset two was 93.5%. The two classes were still quite separate leading to a higher accuracy than future trees. For synthetic dataset three, the tree was only able to reach 83.5% accuracy. I think this was due to the binning technique. Lastly for synthetic dataset four, the tree had an accuracy of 91%. Higher accuracy was achieved on datasets where the two classes were clearly separated. I think with a different binning strategy, higher accuracy could be possible.

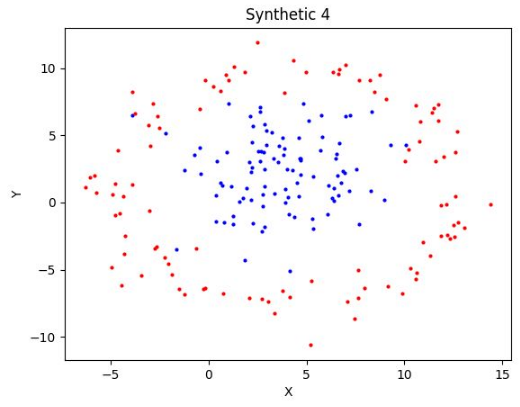
I think my discretization of the pokemon data might have led to inconsistencies in the feature bin numbers produced. Some seem to start at zero and others at one. However, the tree was still able to reach 90.34% accuracy. If I had more time, I would experiment with different binning methods to see how it could affect accuracy.

I've graphed the synthetic data as well as the rough decision boundary of each tree.









The synthetic data was plotted using a matplotlib scatter plot. The decision boundaries were created by sampling each tree prediction across all possible feature values. Those predictions were colored red for a prediction of zero and blue for a prediction of one. Excel was used to create the boundary graphs.

Decision boundaries:

		Synthetic 1			
	4	🟥	🟥	🟦	🟦
	3	🟥	🟥	🟦	🟦
Y	2	🟥	🟥	🟦	🟦
	1	🟥	🟥	🟦	🟦
		1	2	3	4
			X		

		Synthetic 2			
	4	🟥	🟥	🟥	🟥
	3	🟥	🟥	🟥	🟦
Y	2	🟥	🟦	🟦	🟦
	1	🟦	🟦	🟦	🟦
		1	2	3	4
			X		

		Synthetic 3			
	4	🟦	🟦	🟥	🟥
	3	🟦	🟦	🟥	🟥
Y	2	🟦	🟦	🟥	🟥
	1	🟦	🟦	🟥	🟥
		1	2	3	4
			X		

		Synthetic 4			
	4	🟥	🟥	🟥	🟥
	3	🟥	🟦	🟦	🟥
Y	2	🟥	🟦	🟦	🟥
	1	🟥	🟥	🟥	🟥
		1	2	3	4
			X		

References

To view the trees, I used a pretty print function I found at

https://vallentin.dev/2016/11/29/pretty-print-tree

Viewing the trees was helpful during debugging. The snippet from the website is clearly marked in the source code.