IDAI610-PS1-Report

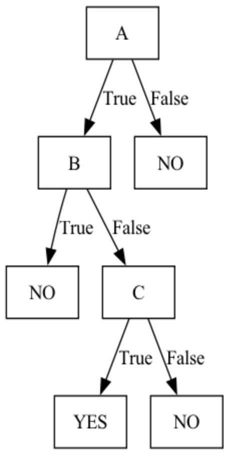
Khawaja Abaid Ullah

September 2024

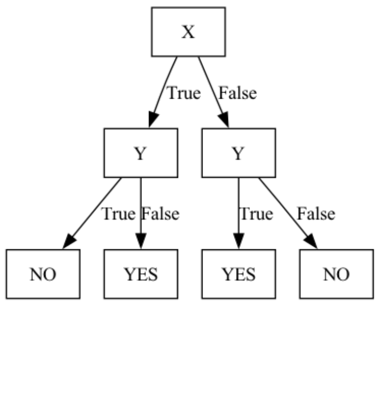
Problem 1: Decision trees for Boolean functions

Q1: Draw a Decision Tree based on Boolean functions.

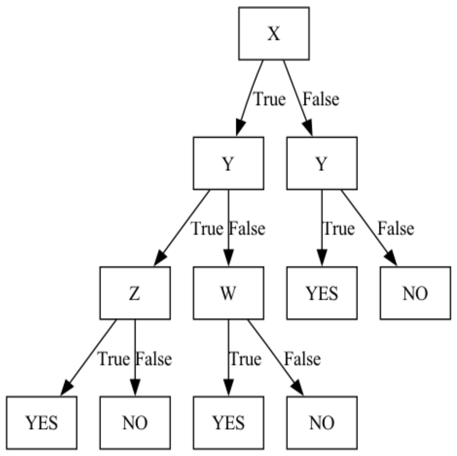
1. $A \wedge \overline{B} \wedge C$



2. $X \wedge \overline{Y} \vee \overline{X} \wedge Y$



2. $X \wedge Y \wedge Z \vee X \wedge \overline{Y} \wedge W \vee \overline{X} \wedge Y$



Q2: Root node selection using Information Gain and Gini

Solution in the accompanying Jupyter notebook.

Problem 2: Implement the decision tree algorithm

Q3: Discuss your implementation in one paragraph.

My implementation of id3 algorithm itself is not that innovative in that it is based on the id3 algorithm pseudocode given in the lecture notes. But is is criterion agnostic, that is, it supports both Information Gain and Gini as node splitting criterion. But but, there's one thing that is the cornerstone of my implementation and that I believe sets my implementation apart from any other (simple ones). It is my custom and especially tailored for this algorithm, the **Node** class. It's not just your simple Node class, it contains attributes that save us a lot of hassle of recomputing things down the road, it provides valuable information about the given node and tree at our fingertips by accessing the relevant attribute, and it also incorporates a *predict* method that recursively finds the outcome based on the given data instance starting from the node it was invoked from. This is not to say my implementation is perfect, in fact far from it, there's so much room for improvement and additions but there's no time for it. But since it gets the job done for the assignment, so I'm happy with it.

Q4: Any challenges you faced.

The biggest challenge for me was the development of the **Node** class as the implementation of algorithm otherwise was pretty straightforward. I knew I needed some kind of a node class when I began my implementation but I didn't know how much value such a class had and how big of a challenge it was. I faced issues with making distinctions among the node types, the correct parent-child relations, and the way a way to evaluate the node after the connections had been established in the "training"/tree building process.

Q4: How you overcame them.

By trial and error. I tried several different configurations for the Node class and used them to build trees to see how they fare. I liberally used print statements and tree visualizations to gather information and identify flaws. To distinguish among the nodes I used python enum and included an attribute in the class named *node_type*. To make sure the parent-child relationships among nodes are consistent and error-free I implemented a method named *add_child* and for evaluation of the node (and tree in general) I implemented a recursive method named *predict*.

Problem 3: Use your decision tree to develop a model for WDBC

Q7: Provide the performance results on the test set

I consider the M (malign) class as positive as that's what we're interested in.

Method	Accuracy	Log Loss	Precision	Recall
Gini DT	0.91	3.16	0.86	0.90
IG DT	0.91	3.16	0.86	0.90
Majority Classifier	0.63	13.27	0	0

Q8: Discuss if precision or recall is most critical (most important) as a performance metric for this problem

Although both precision are recall are important for this problem, but when compared to one another in this specific context, we value precision more. Sure, we'd like to detect cancer in more people (higher recall) but more importantly we want to make sure that the cases we do detect as positive are indeed positive (higher precision).

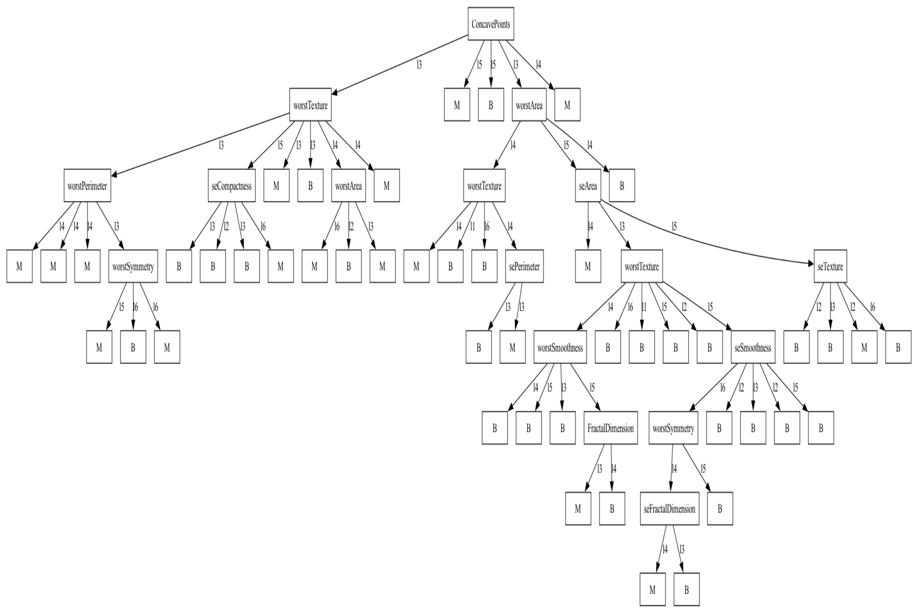
Q9: Explain your tuning procedure with the dev (tuning) set and what you learned from this part of the problem.

Well, the tuning process was super simple and straightforward for this problem as we were only exploring 4 different sets of configurations i.e. two for pruning (on or off) and two for criterion (gini or information gain). But the annoying part was all the trees ended up having the exact same structure and hence results, which made me learn a new fact that this is not something totally unexpected occurrence when dealing with small datasets and tree building algorithms like id3.

Problem 4: Compare with an implementation in an ML library

Q10: Decision tree visualization.

Visualization of my best performing tree on the WDBC dataset using graphviz (as my custom developed visualization function failed for bigger trees like the one below).



Q11: Discuss how your best-performing informativeness metric compared with the best-performing result in your experiment.

As you can see in the table above, although my best performing tree has decent performance. It achieves an accuracy of 0.91, a precision of 0.86 and and recall of 0.90. But when compared with the best performing decision tree in the included notebook that uses sklearn which has an accuracy of 0.96, precision of 0.95 and recall of 0.93, my tree looks poor.

Q12: elaborate on the similarities and differences between your implementation and that of scikit-learn.

In terms of similarities, my implementation and sklearn both allow the freedom to choose either of gini and information gain as the criterion for node splitting. But sklearn also allows another criterion called the 'log loss'. Additionally, sklearn comes with a host of various knobs that can be tweaked and restrictions that can be put on the development of the tree. For instance, we can chose the strategy used to choose the split at each node, control maximum depth, specify the minimum number of samples required to split an internal node and set the minimum number of samples required to be at a leaf node, among other myriad hyperparameters. Sklearn's implementation is also more robust and efficient.

Q13: Compare and contrast the performance of the binned versus non-binned (normalized) data and report if there is a difference in performance between the two.

For the sake of this comparison, I used the discretized datasets and converted the categorical labels to integers using sklearn's OridnalEncoder, and trained the decision tree classifier from the sklearn library using the exact same hyperparameters as the best performing tree for the normalized dataset that is already provided in the notebook. It resulted in a precision of 0.95, recall of 0.90 and accuracy of 0.95. When compared with best performing tree on normalized dataset, it has the same precision, but lacks in recall (0.90 vs 0.93) and also has a little less accuracy (0.95 vs 0.96). The results aren't that surprising as binning the data into categorial values ends up in some information loss as compared to continuous values.

Q14: Speculate why that may be?

The reason is most likely information loss as mentioned above. Then there's the fact that our thresholds for binning the dataset perhaps aren't as good as the threshold employed by the decision tree when splitting nodes or making decisions, which would make sense in this context.

Q15: Discuss observations.

The maximum depth hyperparameter helps us fight the overfitting problem, as otherwise the decision tree just basically memorizes the whole dataset and altought it shows close to or perfect performance on training set it performs poorly on test set. But setting the maximum depth too low also can results in underfitting as the tree stays too simple to learn the complex patterns in the dataset. The important thing is to find a good balanced value either through trial and error or hyperparameter tuning. Like maximum depth, the minimum number of instances required for a split in a decision tree classifier also helps fight overfitting. It makes it so the tree is more conservative in making splits. But as maximum depth, finding a good value for this hyperparameter is another imporantant thing for have a decision tree that has good performance and generalization capability.