TDT4171 Artificial Intelligence Methods
Assignment 4

Danilas Miscenko

March 17, 2021

1 Decision Trees

a)

The attributes that I believe to be continuous in this dataset are 'Age', 'SibSp', 'Parch', 'Ticket' and 'Fare'. The reasons for these attributes being continuous vary however.

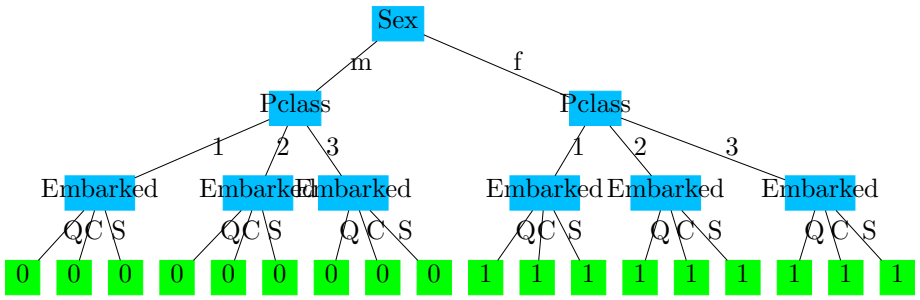
These attributes are clearly continuous because they are numerical in nature. People can be of differing ages in a large range, as well as having a range of amounts of children/parents or siblings/spouses on board. The ticket number is in most cases a numerical value which increases with each ticket sold, and in some cases, the number is prefixed with lettering, probably indicating some special features of the cruise. The fare seems to vary in accordance with these variables as well, meaning that fare also ranges strongly, and since it is a numerical value, I argue that it is continuous.

Removing these attributes leaves 'Pclass', 'Name', 'Sex', 'Cabin' and 'Embarked'. 'Pclass', 'Sex' and 'Embarked' are clearly categorical values, as they have clearly defined categories that split all data. The attribute 'Name' can also be looked at as categorical, as typically, any person will have a unique combination of characters that identifies them and puts then into a "sepparate category" that encompasses only that person. Lastly, the category I'm unsure of, 'Cabin'. It is a category, it the sence that 'Name' is a category, as it puts people that were registered in the same cabin into categories. However, the cabin names are numerical in nature, as they are comprised of letters A-D followed by the cabin number, which can reach into the hundreds (comprised of up to 3 digits).

I've ran the decision tree learning algorithm four times for this part of the assignment with different permutations of attributes, as I was curious about how that would affect the accuracy of the model. The attributes used were:

- The clearly categorical attributes: ['Pclass', 'Sex', 'Embarked']
- The 'Name' attribute included: ['Pclass', 'Name', 'Sex', 'Embarked']
- The 'Cabin' attribute included: ['Pclass', 'Sex', 'Cabin', 'Embarked']
- All attributes together: ['Pclass', 'Name', 'Sex', 'Cabin', 'Embarked']

Running the algorithm with only the categorical attributes:

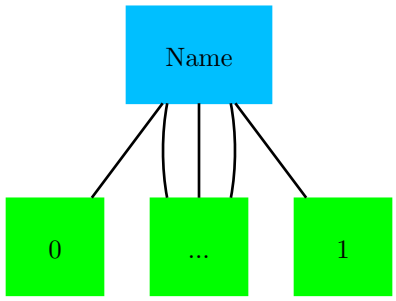


The graph is kind of hard to read, as it is too big for the page, so I will refrain from making full graphs and will just abbreviate where necessary to keep readability.

This tree performed with an accuracy of 87%. This is to be expected if you think about the fact that when the titanic was sinking, women and children were prioritized, which is why all females have a value of 1, while all males have a value of 0 (which were the most popular values for both 'Sex' values). They were however not uniform (not absolutely all men died, and not all women survived), which is why all of the attributes have been used.

Running the algorithm with the 'Name' attribute included:

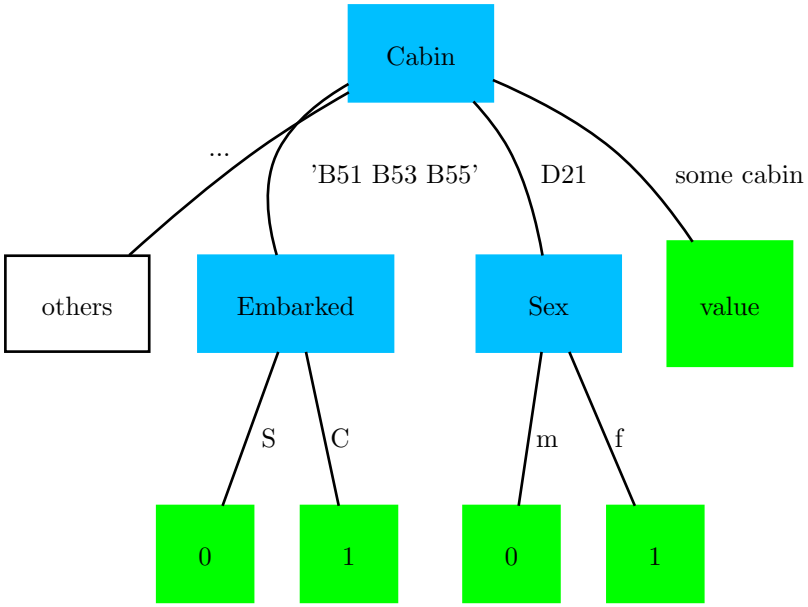
The tree produced by these attributes was quite bad, since it only had used the 'Name' attribute.



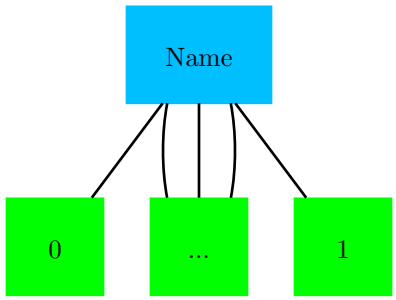
The reason it only used the 'Name' attribute was because all examples have a unique name, meaning that it would map each of the examples to a value directly. This tree did however perform with an accuracy that ranged between 50% and 57%. The reason it ranges so much is because of my implementation of the test function. If all branches of a node have been seen, but no matching value was found, then it selects one branch at random from the branch list.

Running the algorithm with the 'Cabin' attribute included:

The 'Cabin' attribute is similar in nature to the 'Name' attribute, in the sense that few instances in the datasets share a cabin. The tree performed with an accuracy ranging from 47% to 55%, which is slightly worse than the previous tree, but not really by much.

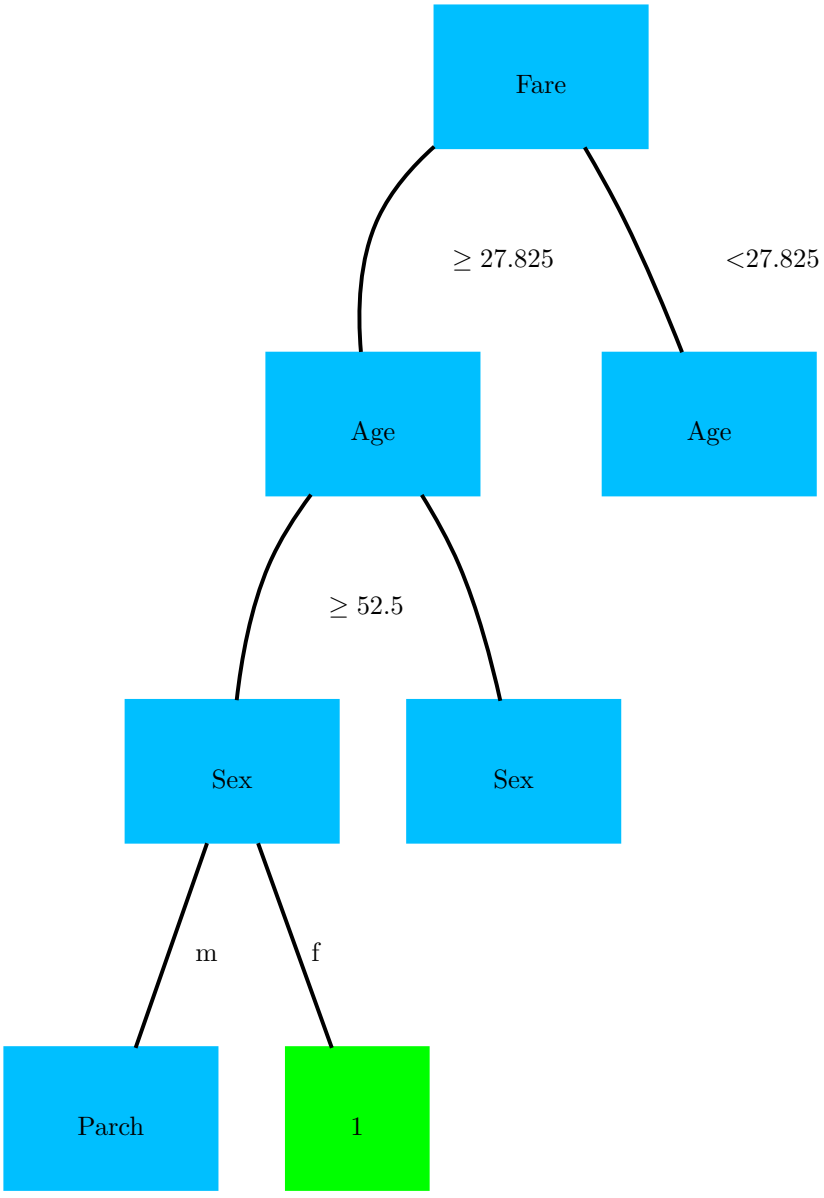


Running the algorithm with all attributes together: Running all of the variables together didn't really do anything, as the 'Name' attribute maps to all values directly, so no other attributes are selected. It performed the same as when I ran the algorithm with only the 'Name' attribute included, which was between 50% and 57% accuracy. The tree also looks the same as it dit with only the 'Name' attribute included.



b)

After the algorithm has been extended to accommodate continuous values, I ran it including every variable. What happened was pretty predictable, which was that the 'Name' attribute took over as before. I therefore removed the 'Name' attribute from the attribute list before running the algorithm again. I also removed the 'Ticket' and 'Cabin' attributes, as they would only map to unique values in the dataset, acting just like the 'Name' attribute. The resulting tree was quite extensive, and I was unable to draw it out fully, as I didn't have the time, but here is what I had time for:



The full tree can be viewed by running the code with no variables sent into the program. This tree performed with an accuracy that ranged between 70% and 78%, which is a bit worse than the first tree which was with only the categorical values.

c)

The performance of the tree with the continuous attributes was expected to be honest. This is in my opinion because the dataset does not really include any examples that had a pattern that could be used to determine whether a passenger had died or survived other than the "women and children first" principle which was perfectly illustrated by the first tree which had only the categorical variables. The newest tree, the one with the continuous attributes might be overfit, and therefore would probably benefit from pruning to remove attributes that don't really do much for information gain. Another tactic would be to include less of the attributes all together, keeping specifically the 'Age' and 'Sex' attributes (because women and children were more likely to survive), and then testing the tree with some other attributes as well to see if any other attributes would increase the accuracy at all.

2 Missing Values

In my code, the cabin missing values were either skipped entirely, or they were included as "nan", and actually contributed to the overall accuracy of the model, whether it was a positive contribution or negative remains to be tested. These ways of handling missing values are however not particularly intelligent, as they either misinform the model or lose information. A slightly more intelligent method (although not by much) is to assign a random value out of the available ones to the attribute. This in turn would preserve the example for the other attributes, while potentially sabotaging the attribute that the value is missing for. An even more intelligent solution is to assign a value to the missing value based on other attributes. One can for example find an attribute that is the most similar to the attribute with the missing values and guess the missing value based on that. By similar, I mean that it corresponds to other instances of the attribute with missing values quite well. Say it is true for all or many other instances of the attribute with missing values that are true, and false for all or many of the attributes false cases. One can also assign a probability where the missing value should be based on the probability of that value being true, false or any other value.