TDT4171 - Exercise 4

Andreas B. Berg

Task 1 – Decision Trees

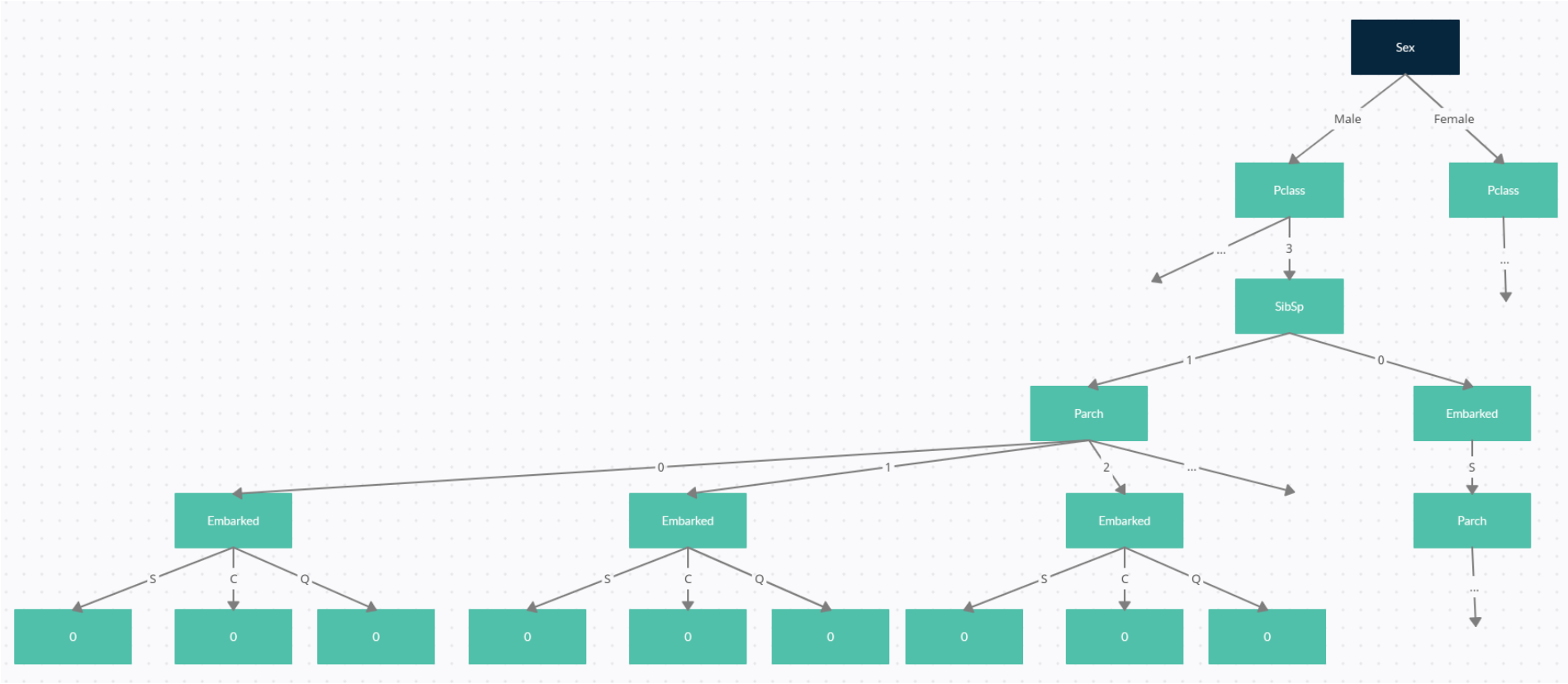
a) *Implement a decision tree that can support categorical variables. Use information gain as the Importance function. Use your implementation to train a decision tree on the Titanic dataset. Some of the columns in the dataset are continuous attributes and cannot be used with this implementation. Which are these? Test your model on the test set and print your accuracy. Add your decision tree to the PDF.*

The following attributes are continuous, and can therefore not be used directly:

- Name
- Age
- Ticket
- Fare
- I have decided to not include "Cabin" as well. Cabin is not very generalizable, and as such it makes for a poor attribute to include. As the dataset is lacking, the chance of someone staying in the same cabin is low, meaning the attribute complicates the tree without adding much to the solution.

It can be argued that SibSp and Parch are continuous as well. While they are numbers, they are realistically limited, and can therefore be used (it is between 0 and 8 in the dataset).

I have sketched the tree below. Note that this is just parts of the final tree. Please see "task1a_tree.txt" for the console-logged version.



This gives the following accuracy:

```
Failed 56 times on 417 runs
Accuracy: 0.8657074340527577
```

b) *Extend your implementation to also support continuous variables. Train this model on the Titanic dataset again, now also using the continuous attributes you left out in a). Test the model again on the test set and print the accuracy of the predictions. Add your decision tree to the PDF*

While there are many values that may be usable, the most realistic (and easiest to implement) will be age and fare. Age is realistically impacting the result, and fare can give an indication as to whether having money helps. Name, Ticket and Cabin are very hard to generalize, as each value will have very few passengers. I will therefore not include them here, as I have skipped them above as well.

The implementation can be found in “task1b.py”, and the tree itself in “task1b_tree.txt”.

I got the following result after running:

```
Failed 68 times on 417 runs
Accuracy: 0.8369304556354916
```

c) *Discuss the results from a) and b); is the performance difference what you expected? What might be done to improve the performance of the decision tree classifier on the test set? Suggest at least 2 changes that might be made to the algorithm that could improve performance.*

The performance was not what I expected, as the accuracy got reduced by approximately 3%, while the learning time increased drastically. I was expecting the accuracy to increase, as we got another range of values to learn from. It seems like that does not hold true here. I suspect this to be because of my implementation, rather than stemming from the fact that we include continuous variables.

To improve the performance, I can see a couple things that can be done:

- Improve the number of splits. When iterating, I only split each value once (in “higher than” and “lower than or equal” to the split). Splitting more than once, and choosing the split(s) with the highest information gain, could increase the performance of the set
- Adding more data. It is safe to assume that throwing more data at the problem could increase the accuracy, as the chances of more relevant values increases. It is not directly related to continuous variables, but still.
- Implementing one of the below mentioned methods for dealing with missing data. The dataset is currently missing many values in ”Cabin”, and dealing with that in a smart way could increase accuracy (and thus performance)
- Implementing different types of learning systems. The book argues that ensembles of models generally outperform a single model (like our decision tree), and it would be unwise to not assume the same to be true for our problem.

Task 2 – Missing Values

One of the columns in the Titanic dataset contained missing values. Suggest at least two methods that we could have used to handle missing data either during training or prediction. Describe your proposed solutions either in text or with pseudo code. Discuss the advantages and disadvantages of your methods. What assumptions are you making?

The column with missing data was "Cabin". To handle this, I would use one of the following methods:

- Have "empty cell" as its own value for attribute "Cabin". To implement this, all I would have to do would be to replace empty cells with, say, "N/A", and use that as its own value.
    - Advantages: Very easy to implement, little code needed. Can use the whole dataset, even with missing information.
    - Disadvantages: Chance to get "fake groupings", as passengers without cabin information gets grouped together – may show connections that are not there in real life.
    - Assumptions: The missing values are few enough that "fake groupings" are avoided.

- Remove the mentioned columns (or rows) from the dataset. This is the assignment told us to do, as we were not required to implement handling of missing information.
    - Advantages: Very easy to implement. When only one column is affected, like in this dataset, the chance of its removal impacting the end result is slim, meaning a full "Cabin"-column might not have given much higher accuracy than we got
    - Disadvantages: If many rows or columns are missing data, then you risk losing out on significant accuracy. If the missing data is skewed (say: it was missing only for lower-class passengers who did not survive), then the result can get biased.
    - Assumptions: Missing data is distributed randomly throughout the dataset, so it does not bias the result to any significant degree.

- Create a separate training set with different combinations of Cabin in the empty cells. This is not very realistic to do, as it would mean a huge number of different training sets. If the dataset contained few missing values, however, this could be feasible. To implement this, I would make a script that copies the training set and fills the missing values with a random value from its domain.
    - Advantages: May end up with a totally realistic decision tree. No need to make conscious decisions, just throw computer power at the problem – meaning it can be implemented if the developer has no idea what to do.
    - Disadvantages: VERY demanding computationally. Not realistic for datasets with large holes, such as this one. Hard to figure out exactly which decision tree is the right one (although one could look at which has the highest accuracy, this might not give an accurate result).
    - Assumptions: The missing values are few enough (and with little enough domain) that multiple training sets can be realistically implemented. The resulting decision trees can be checked to see if they are correct, and it is easy to separate the different trees.