1 Overview

In this exercise sheet, you will implement the Decision Tree machine learning model to solve classification problems. Edit the Jupyter notebook `Decision_Tree.ipynb`. A simple class hierarchy for a tree is already provided. Your task is to implement the construction of the tree step by step. This is a test.

Ultimately, we want to implement a function `build_tree` that accepts a data set and returns a complete tree. Our data set generally has the form of a list of data points. Each data point consists of a tuple (x, y), where x is a list of the individual features and y is the class (coded as an integer or string).

1.1 Auxiliary Functions (10 points)

We need two auxiliary functions: `count_occurance_of_class` and `get_unique_classes`.

- `count_occurance_of_class` accepts a data set in the above format and a class, and counts the number of occurrences of this class in the data set. The return value is the count.
- `get_unique_classes` also accepts a data set in the above format and returns a list containing all occurring classes. Each class should only be included once, even if it occurs several times in the data set.

a) Argue whether unit tests are required for this function. Which cases should be tested?

- In our use case these two auxilar functions get called after lots of other functions are already called beforehand. All test regarding data format, integrity and should be done by now, probably by the `load_csv_in_sklearn` function. For this reason no test cases to implement regaring the input. There could be made an argument that the output should be tested, but I think this is not necessary as the functions are very simple. One also has to consider the overhead, ressource constraint.

b) Add the required unit tests.

c) Implement the functions.

I needed the following time to complete the task:

1.2 Gini Index (10 points)

Next, we need a function that calculates the Gini index for a data set in the above format.

a) Argue whether unit tests are needed for this function. Which cases should be tested?

- I think unit tests for checking the correctness of the formula are necessary. Errors in the definition of formulas can happen quickly, as no error will be raised if the formula is codewise correct but mathematically wrong leading to false conclusions in the decision for decision boundaries. Also the Gini-Index is not always intuitive, so wrong results can be hard to spot.
- Test the edge cases: all same class, all different classes. Also a third case with a mix of classes can be implemented.

b) Add the required unit tests.

c) Implement the functions.

I needed the following time to complete the task:

1.3 Data Set Split (5 points)

Next, we need a function that performs a split on a data set. The function `split_data` accepts a data set in the above-mentioned format, the index of a feature f at which is to be split and a split value. It returns a tuple $(left, right)$. Here, $left$ and $right$ are each subsets of the data set. For all data points in $left$, feature f is less than or equal to the limit value; for all data points in $right$, feature f is greater than the limit value.

a) Argue whether unit tests are needed for this function. Which cases should be tested?

- In our use case these two auxilar functions get called after lots of other functions are already called beforehand. All test regarding data format, integrity and should be done by now. The function is simple. I wouldnt create testcases for the same reason as in the first task.

b) Add the required unit tests.

c) Implement the functions.

I needed the following time to complete the task:

1.4 Select Split (10 points)

Now let's look at selecting good splits. Implement the function `select_best_split`. This first generates a list of possible splits for a data set in the above format. with possible splits. This contains the split at the median for at least every feature in the data set. A split is defined as a tuple $(feature, value)$, where feature is the index of the features of the best split, and value is the threshold value from which the split is to be made. The function executes the splits on a trial basis and evaluates the result: useless splits (a subset is empty) are sorted out, the remaining splits are evaluated using the Gini index. The function returns a split.

a) Argue whether unit tests are needed for this function. Which cases should be tested?

- In our use case these two auxilar functions get called after lots of other functions are already called beforehand. All test regarding data format, integrity and should be done by now, especially the function testing if the recursion in the decision tree can be stopped is important to handle edge cases that shouldnt go into this function. Examples are datasets containing only one value or all feature values being the same. These functionality of `select_best_split` can be tested with the following edge case:
  - All values of first but not the second feature are the same

b) Add the required unit tests.

c) Implement the functions.

d) Optional: Implement the optimization that a split is evaluated based on a sample of the data set. Determine experimentally from what size of data set this is faster.

e) Optional: Argue in which cases it could be useful to split at points other than the median of the values. Implement an additional split strategy.

- You could also split at the mean of the values. This could be useful if the data is not evenly distributed. For example a lot of points in Class A all haave the value 1 and only one Value of Class B exists and has the value 10. The median would be 1, but the mean would be higher. This could be a better split.

I needed the following time to complete the task:

1.5 Build Tree (20 points)

With the help of the auxiliary functions already implemented, we can now build a Decision Tree.

a) Argue whether unit tests are required for this function. Which cases should be tested? What role can visualizations play?

- Visualizations allow an intuitive understanding of the decision tree. They can be used to check if the tree is built correctly, and let the user understand the decision making. This testing would be qualitative and not quantitative.
- The function should be tested for the following cases:
  - Base case: The function should stop and return a LeafNode if the data set is empty or only contains one class, or all features have the same value.
  - Recursive case: The base case is not fullfilled. Return InnerNode. It can also be checked if the children are correctly assigned as LeafNodes assuming the base case can be reached by a one iteration.

b) Add the required unit tests.

c) Implement the functions.

d) Test your tree using the synthetic data set and the iris data set.

Recompile the latex to display the results.

I needed the following time to complete the task: