Project 1 INF283

Markus Ragnhildstveit, hef005

February 15, 2019

Contents

You will find python documentation in each function in the code files which contains what type the arguments is and what the function returns. To run the program you just run the "test_run.py" file. If you want to see the tree structure, uncomment the last 2 lines.

1 Implementing the ID3 algorithm

To implement ID3 algorithm we needed to represent the data in a structure. I decided to go with a tree, and the class is called "node". You can find the implementation of this tree in "node.py". The rest of implementation of ID3 is in

1.1 learn

I have decided to implement learn with a queue. first thing we need to do is initialize the tree and put the root node in the queue. Then we loop through the queue. We find the information gain for each column with the function "IGs". I will explain what IGs does further down. IGs returns a list of information gain for each column with corresponding index. So when we find the maximum information gain, we use the same index. If the maximum information gain is as close to 0 as possible (not necessarly equal because of floats), then I will not split the current node. Otherwise I will use the split function on "node" and add the children to the queue.

This will continue until the queue is empty.

1.1.1 IG

First I find the correct impurity function, and find the impurity to data set before a potential. Then I loop through the different type of values found on the column and see how many "p" and "e" to find the impurity for the sub group. Then subtract the weighted impurity from the impurity before the potential split.

1.1.2 IGs

Creates a list and loops through the columns of X for finding the information gain on each of them.

1.1.3 entropy

Pretty self explanatory, but takes in number of yes and number of yes and returns the entropy. I have not made it compatible with more than 2 outcomes, Since it were not needed in the dataset.

1.1.4 unique

Unique finds the unique values for each column and returns it as list of list.

1.2 node

1.2.1 split

First Is to split the data between the different values on column with index the same as paramindex. Then initialize the children in the self.children list. The different types of values on the column is saved in num_of_type. If you index a value on num_of_type it will return the same index as the corresponding child in self.children

1.2.2 predict

If it is a leaf, it will return the prediction of the node. Otherwise it will find the correct child to predict on.

1.2.3 prun

Function used for pruning the tree. If the current node is not a leaf, the data has to be split to the correct children. When that is done I prune the children first, but if one of them gets pruned, the whole pruning for current node is done also. If all children are nodes, then we can check the 3 conditions: keep the split, current node gets set to "e", and current node gets set to "p". The one with the smallest error is then chosen.
If the current node is a leaf, we count the number of errors in the prune dataset.

2 Implementing Gini index

implemented "gini" so it takes the same input as "entropy", so in "IG" i just make a variable "impurity_func" which is either "gini" or "entropy"

2.0.1 gini

Same as entropy, but returns the gini index.

3 Implementing Pruning

Pruning is implemented as an own function which uses "prun" in "node", you can find the explanation of "prun" in "node" under the explanation of "node". If pruning is true in learn, it will be done at the end of learn after the tree is finished.

3.1 prun

Loops until the "prun" function in "node" returns false, because when it returns false it means there is no more pruning that can be done. This is done because we only want to do one change of the tree at once, and then start at the top again.

4 Classifying edible and poisonous mushrooms

The classifiction of edible and poisonous mushrooms are done in "test_run.py" on lines 13-41. First i remove the rows which contains "?" with "remove_unknown_elements" function which is defined in "ID3.py". Then the data is split into training and test data.

The output from the classification with using the test data is 0 number of wrongly predicted data points with an error rate of 0.
This is how the tree structure looks like:

```
Node with 7 children, split on feature 4:
{
    Leaf with value "c" on feature 4
    Node with value "n" on feature 4, with 4 children split on feature 19:
    {
        Leaf with value "n" on feature 19
        Leaf with value "k" on feature 19
        Node with value "w" on feature 19, with 6 children split on feature 2:
        {
            Leaf with value "n" on feature 2
            Leaf with value "c" on feature 2
            Leaf with value "g" on feature 2
            Leaf with value "w" on feature 2
            Leaf with value "y" on feature 2
            Leaf with value "p" on feature 2
        }
        Leaf with value "r" on feature 19
    }
    Leaf with value "f" on feature 4
    Leaf with value "a" on feature 4
    Leaf with value "p" on feature 4
    Leaf with value "l" on feature 4
    Leaf with value "m" on feature 4
}
```

If you want to print this you can uncomment the last 2 lines in "test_run.py".

5 Comparison to decision tree in sklearn

The implementation of the decision tree classifier from sklearn is written in "test_run" on lines 44-72. First I have to transform the data values to integers from strings since DecisionTreeClassifier does not accept strings. Then split test and training data and fit on the training data.

Testing with the test data returns 0 wrongly predicted and an error ratio of 0. So the same as my implementation of the decision tree. However the tree structure is different, because DecisionTreeClassifier only uses binary tree, while I have implemented a tree that splits to the amount of different values on a column.