Assignment 6 – Jakub Rysiak

Results

When running my algorithm using the importance_gain function, it consistently got a score of 92.9%

```
C:\Users\Kuba\AppData\Local\Program
Training Accuracy 1.0
Test Accuracy 0.9285714285714286

Process finished with exit code 0
```

When running the random version, I observed two things:

- The performance was as expected poorer and varied a lot
- Sometimes there was an error and the tree could not classify the example

```
Traceback (most recent call last):
  File "C:\Projects\School\TDT4171-Metoder-i-AI\Assignment 6\assignment_6.py", line 152, in <module>
    print(f"Test Accuracy {accuracy(tree, test)}")
                           ^^^^^^^^^^^^^^^^^^^^
  File "C:\Projects\School\TDT4171-Metoder-i-AI\Assignment 6\assignment_6.py", line 118, in accuracy
    pred = tree.classify(example[:-1])
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Projects\School\TDT4171-Metoder-i-AI\Assignment 6\assignment_6.py", line 22, in classify
    return self.children[example[self.attribute]].classify(example)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Projects\School\TDT4171-Metoder-i-AI\Assignment 6\assignment_6.py", line 22, in classify
    return self.children[example[self.attribute]].classify(example)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Projects\School\TDT4171-Metoder-i-AI\Assignment 6\assignment_6.py", line 22, in classify
    return self.children[example[self.attribute]].classify(example)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  [Previous line repeated 3 more times]
KeyError: 2
```

I believe the error was due to the fact that when creating a random tree, not all possibilities were accounted for, such that a combination of attributes was not classified, and hence the error in some cases. Furthermore, if my understanding of the algorithm is correct I believe that this same error can arise when using the information_gain version, although in this particular case it works. To counter it I

added this part to the code:

```
possible_values = (1,2)
# iterate over the values of the best attribute and create a subtree f
for value in possible_values:
    subtree = learn_decision_tree(
        # examples where the best attribute is equal to the value
        examples[examples[:, best_attr] == value],
        # remove the best attribute from the list of attributes
        np.delete(attributes, np.where(attributes == best_attr)),
        # parent examples
        examples,
        # parent
        node,
        # branch
        value,
        measure
    )
    node.children[value] = subtree

return node
```

Whereas before I iterated over all unique examples, which might have been just one. I'm not sure if this is the correct way to solve it, or if it violates the algorithm as it creates more branches, please let me know.

Anyhow the results were the following for 10 different runs:

```
C:\Users\Kuba\AppData\Local\Program
Training Accuracy 1.0
Test Accuracy 0.8571428571428571
Training Accuracy 1.0
Test Accuracy 0.7142857142857143
Training Accuracy 1.0
Test Accuracy 0.8214285714285714
Training Accuracy 1.0
Test Accuracy 0.7857142857142857
Training Accuracy 1.0
Test Accuracy 0.6785714285714286
Training Accuracy 1.0
Test Accuracy 0.8928571428571429
Training Accuracy 1.0
Test Accuracy 0.9285714285714286
Training Accuracy 1.0
Test Accuracy 0.8571428571428571
Training Accuracy 1.0
Test Accuracy 0.7857142857142857
Training Accuracy 1.0
Test Accuracy 0.7142857142857143

Process finished with exit code 0
```

Comparison

The importance function using the information gain works better, since it uses the attributes that simply provide most information about the class. This is as we learnt because it uses the entropy calculation, which provides information about how much variance there is between examples. If an attribute split has a low entropy in the different branches, it means that it provides a high probability of classifying the examples based solely on that attribute. On the other hand I had some rare runs on the random version that had a higher score than the information_gain. I suppose this is due to the fact that the algorithm is generally the most optimal for finding a good decision tree, not the correct formula for always finding the best decision tree, which the random algorithm can stumble upon.