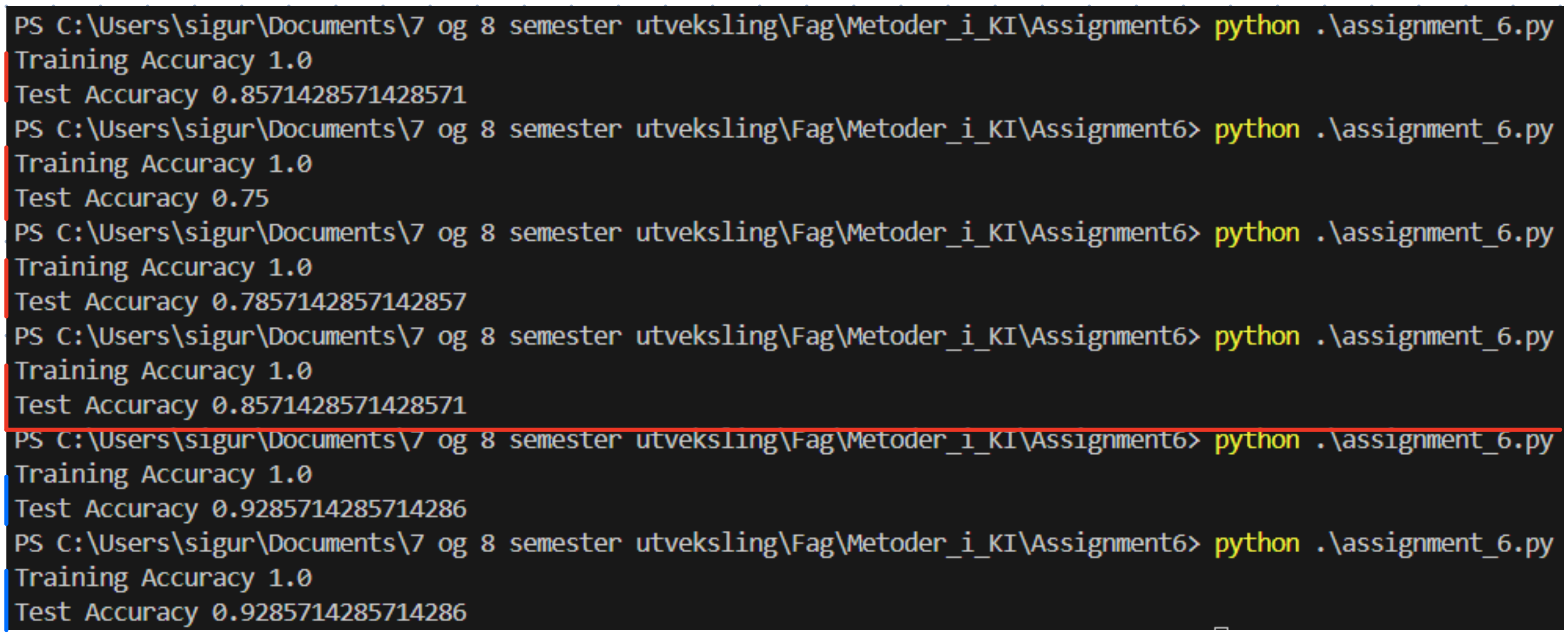
Decision tree algorithm allocating importance two different ways

Screen shot from printing out training and test accuracy.



```
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.8571428571428571
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.75
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.7857142857142857
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.8571428571428571
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.9285714285714286
PS C:\Users\sigur\Documents\7 og 8 semester utveksling\Fag\Metoder_i_KI\Assignment6> python .\assignment_6.py
Training Accuracy 1.0
Test Accuracy 0.9285714285714286
```

The first 4 training/testing accuracies are with random allocation of importance to each attribute. The last 2 training/testing accuracies are with allocating importance based on the expected information gains.

Discussion;

The implementation I have made seem to overfit to the data because all runs, independent of importance allocation, gave a training accuracy of 1.

The accuracy when testing shows that allocating importance based on the expected information gain works better and gives more consistent results than the case in which we allocate randomly. This makes sense as information gain is a measure of how well an attribute separates the training examples according to their target classifications. Using this to allocate performance will always be better or, perhaps in worst case, equal to allocating the weights randomly.