# Tutorial 2: Decision Trees

**Universidad Carlos III de Madrid**
**Grado en Ingeniería Robótica**
**Machine Learning**

Made by Ignacio González Meléndez (NIA 100522976)
& Lucas Beltrán Rúa (NIA 100523114)

**Exercise 1:**

1. No, the taxi is not able to get to the passenger because he produces movements that change the state in the environment. These movements made the taxi stuck in a sequence of positions, which we could not solve.
2. The main advantages of implementing machine learning in controlling the agent are:
    - Less human intervention, specific rules for complicated situations do not have to be defined by the human, as the agent would learn itself.
    - Better management for signals if we have to handle continuous signals such as velocity, position, etc. Simple hand rule models would not be enough as the agent could get stuck in an operation.
    - Lastly, the main advantage is adaptability. With ML our agent can be continuously improving among the process.

The behavior implemented in this exercise consists of an agent that decides the next movement based on the distance to the destination position. For example, if the taxi is in position (0, 0) and the passenger in position (3,2), the agent will decide to move right as the distance in the horizontal axis is greater than the distance in the vertical axis. Also, once the passenger is picked up the destination would change, this means that the (row, column) destination is edited. However, this agent cannot evade the walls of the environment, which is why the taxi gets stuck.

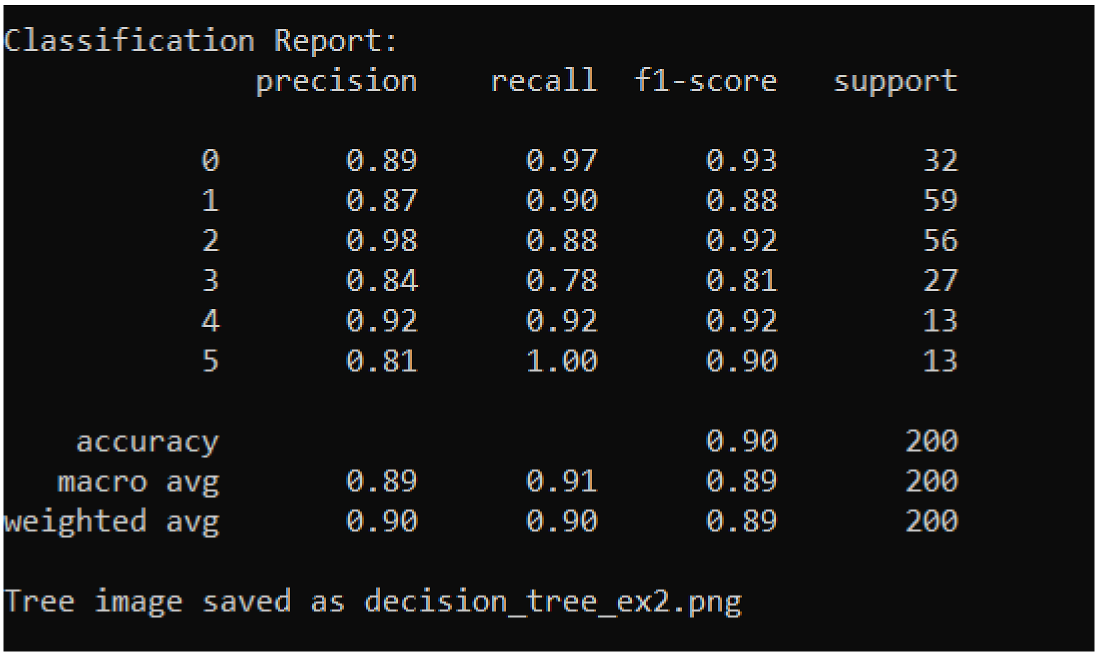
**Exercise 2:**

1. With the weighted average precision, which is 0.90, we can calculate the number of correct and incorrect predictions, in this case it is 180 correct and 20 incorrect. This model predicts going east with the most precision (0.98) and dropping off operation with the least (0.81). The weighted recall and accuracy of the model are 0.90 and 0.895, respectively.

2. In the tree we can see that each action corresponds to a color. Also, the tree has learned which action to reproduce regarding the state, this is based on our plays on the first tutorial.
   It starts by checking the most important feature (X[4]), if this value is low (≤ 1.5), it follows one path and checks another feature (X[1]) to decide the action. If the value is high (> 1.5), it follows a different, more complex path that needs to check several more features (like X[3]) before making a decision.

**Exercise 3:**

1. This model has predicted 180 actions correct and 20 incorrect. The weighted precision, recall and accuracy of this model are 0.90, 0.90 and 0.895, respectively.

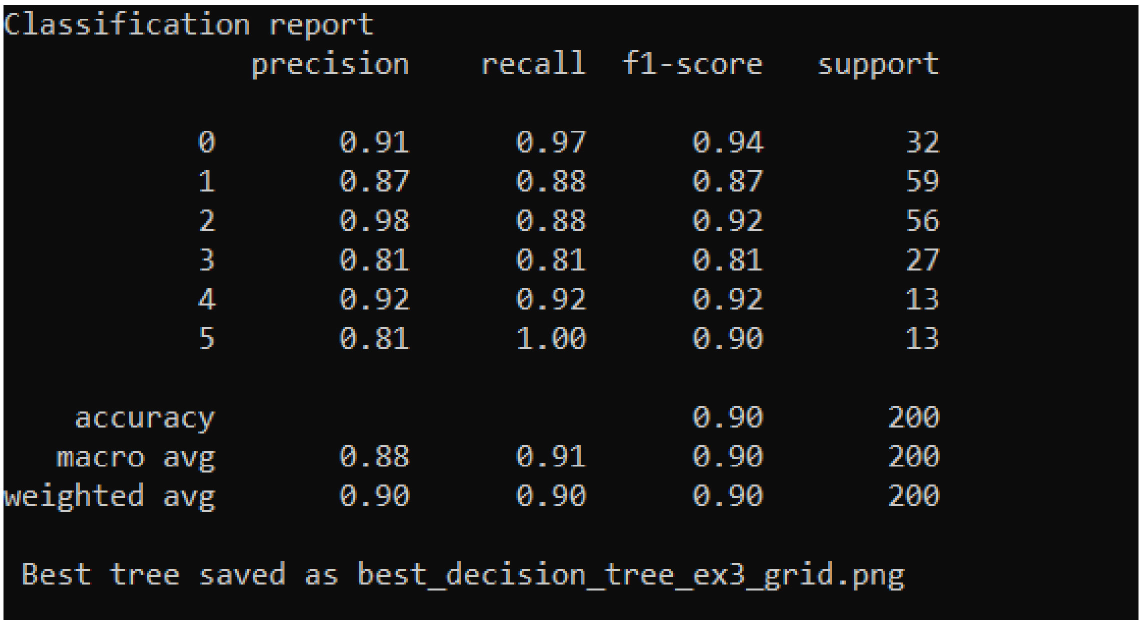


```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        32
           1       0.87      0.90      0.88        59
           2       0.98      0.88      0.92        56
           3       0.84      0.78      0.81        27
           4       0.92      0.92      0.92        13
           5       0.81      1.00      0.90        13

    accuracy                           0.90       200
   macro avg       0.89      0.91      0.89       200
weighted avg       0.90      0.90      0.89       200

Tree image saved as decision_tree_ex2.png
```



```
Classification report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        32
           1       0.87      0.88      0.87        59
           2       0.98      0.88      0.92        56
           3       0.81      0.81      0.81        27
           4       0.92      0.92      0.92        13
           5       0.81      1.00      0.90        13

    accuracy                           0.90       200
   macro avg       0.88      0.91      0.90       200
weighted avg       0.90      0.90      0.90       200

 Best tree saved as best_decision_tree_ex3_grid.png
```

If we compare the classification report from the default model with the one from the best model we can see that the second one predicts correctly the action move south in more occasions, which means that it will have less errors when controlling the taxi towards the destination.

2. The best parameters for this model are criterion= 'gini', splitter= 'best', max_depth= 10, min_samples_split= 2, class_weight= 'balanced'. This is because we have compared 96 possible combinations of parameters and selected the one with the most accuracy.

3. This tree shows a more controlled and balanced structure compared to the default model. The tree maintains a moderate depth, which suggests that the optimal parameters prevent overfitting while still capturing important decision patterns. The structure is more symmetrical and organized than a fully grown tree, indicating that the model focuses on the most relevant features.
   Starting from the root, the tree makes an initial split based on the most discriminative feature, then continues to branch into left and right subtrees. The leaf nodes at the bottom represent the final action predictions, and their distribution shows that the tuned

model has learned to generalize better, creating decision rules that balance accuracy with simplicity. The optimized parameters (such as min_samples_split or criterion selection) have helped the tree avoid creating overly specific rules that only work for training data.

4. As we have discussed in questions 1 and 3, this best-parameters model works better than the default one because it focuses on the most important actions, for example predicting more correctly south movement. Also, the parameters make the model to be more stable and create rules that will work on our implementation.

**Exercise 4:**

1. Yes, the model is capable of picking up the passenger and dropping him at the desired position. This is produced in an infinite loop till the user decides to stop the program.
2. Yes, it works better than the one in exercise 1 because the model from exercise 3 does not get stuck with the walls of the environment and can complete the task by following the train experience from the first tutorial.

**Parameters:**

criterion = 'gini'

- Measures how mixed the classes are at each node. A lower Gini means the node contains mostly one type of action, which is good. The alternative is 'entropy', which uses information gain

splitter = 'best'

- This determines how the tree chooses which feature to split on at each node. 'best' means it evaluates all features and picks the one that gives the best split. The alternative is 'random', which picks a random feature - but 'best' gives better accuracy, which is why it was selected.

max_depth = 10

- This limits how deep the tree can grow - it can only have 10 levels maximum. This is important to prevent overfitting: without a limit, the tree would keep splitting.

min_samples_split = 2

- This is the minimum number of samples required to split a node. With min_samples_split = 2, a node will split even if it only has 2 examples. This is the default and most permissive setting.

class_weight = 'balanced'

- This automatically adjusts the importance of each action class based on how often it appears in your training data. This prevents the model from only learning the most common actions and helps it learn all actions equally well.