

Foundations of A.I.
Assessment III

Question 1:

This question is considered to be an optimisation problem which means it has many solutions but there is only one optimal solution that can be found. By understanding the problem, it can be concluded that a solution for this problem would be reaching the goal state (final state of delivering all parcels in one day) without using all of the 100 vans (Minimum number of vans) taking into consideration there are 3 different types of parcels with different sizes.

a) **State Representation and Initial State:**

Parcels Types Representation $\rightarrow [A, B, C]$

Each Van Includes $\rightarrow [P1, P2, P3], [P1, P2, P3], [P1, P2, P3], \dots, [P1, P2, P3], \dots$

This means each van include a combination [x of parcels A, x of parcels B, x of parcels C] (for 100 vans)

Initial State (Starting State) $\rightarrow [A = 100 \text{ parcels}, B = 200 \text{ parcels}, C = 500 \text{ parcels}]$
 $[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], \dots$ (until 100 vans)

All the 800 parcels are still at the delivery company and all the vans are empty.

b) **The actions:**

- Actions \rightarrow*
- 1- *[Load x parcels A into vans]*
 - 2- *[Load x parcels B into vans]*
 - 3- *[Load x parcels C into vans]*
 - 4- *[Calculate area in each van \wedge check space is $\leq 200\text{cm}^3$]*
 - 5- *[If space is more than $200\text{cm}^3 \rightarrow$ penalize the agent]*
 - 6- *[If space is $\leq 200\text{cm}^3 \rightarrow$ order the van to leave for delivery]*

c) **Successor Function:**

- Successor Function \rightarrow*
- 1- *[Check the state of van (VAN)]*
 - 2- *[if empty]*
 - 3- *[Load the parcels in the van]*
 - 4- *[State of van changed to be (VAN')]*
 - 5- *[Change state of parcels X to be X -1] (deduce number of parcels left to be loaded)*

d) Goal Test:

Goal to be reached $\rightarrow [A = 0, B = 0, C = 0] \rightarrow$ Make sure no parcels are pending for delivery.

$[V_{USED} < 100] \rightarrow$ Make sure the least vans are used.

$[D < 24 \text{ hours}] \rightarrow$ Make sure to deliver all in one day.

If this goal is reached – this would be optimal for the agent.

e) Path Cost:

The uniform cost is the closest node the search problem would visit in the shortest time possible.

Cost \rightarrow [Cost for each fully loaded van = 1]

[Cost for each parcel loaded will be -1] \rightarrow 1 parcel taken out the warehouse

[Cost for filling a van with more than 200cm^3 high] \rightarrow Meaning it is a penalty on the agent.

The main goal is for the agent to understand what the shortest cost is, so filling a van parcels is only a cost of 1. The agent must fully fill a van before moving to an empty van in order to use the least cost possible.

Question 2:

a) **PDDL Domain:**

```
(define (domain clean-offices))

(:requirements :strips)

(:predicates (chair ?x)
              (desk ?x)
              (computer ?x)
              (office ?x)
              (holding ?h)
              (empty ?x ?y)
              (empty_arm ?x)
              (dirty ?d)
              (clean ?y)
              (clear ?z)
              (on ?x ?y)
              (in ?x ?y))

(:action pickup_computers
  :parameters (?x ?y ?z)
  :precondition (and (clear ?x) (on ?x ?y)(empty_arm)(in ?x ?y))
  :effect (and (not(on ?x ?y)) (holding ?x) (not (clear ?x)) (not (empty_arm)) (clear
?y)))

(:action pickup_objects
  :parameters (?x)
  :precondition (and (clear ?x) (empty_arm)
  :effect (and (holding ?x) (not (clear ?x)) (not (empty_arm)))))

(:action lift
  :parameters (?x ?from ?to)
  :precondition (and (office ?from) (office ?to) (holding ?x) (not (empty_arm))
(not(clear ?x)))
  :effect (and (not (empty_arm)) (not (clear ?x)) (not(empty_arm)) (not (holding
?x))))

(:action move_objects
  :parameters (?from ?to)
  :precondition (and (office ?from) (office ?to) (empty_arm) (not (holding ?x)))
```

```
                :effect (and (office ?to) (not (office ?from))))

(:action drop_objects
  :parameters (?x)
  :precondition (holding ?x)
  :effect (and (clear ?x) (not (clear ?y)) (empty_arm) (not (holding ?x)) (on ?x ?y)))

(:action clean_office
  :parameters (?x ?y)
  :precondition (and (empty ?x) (dirty ?x) (empty_arm) (not(empty ?y)))
  :effect (clean ?x) (empty_arm))
```

- 1- First, we set the requirements for the PDDL problem, : STRIPS and :ADL are used. Where *Strips* is to allows us to add and delete effects in the code. ADL is used to import the function “For All” in the code where in our case, we can move multiple objects at one time from an office to the other.
- 2- Then we specify the predicates in the program, which are the properties that we will use, it includes many so-called variables that are TRUE and will be used later in the program.
- 3- We then have a set of actions that describe our problem and finds a solution for it. In our problem the first step is to remove the PCs from the desks, this it what the first action in the program specifies (To pick up the PCs using an arm and clear it off the desk). The actions include parameters, preconditions, and effect. The precondition describes the state where the problem is before applying the action. The effect is the next state after applying the action.
- 4- Many actions are then performed such as picking up the objects, lifting the object with the arm, moving the objects from the office, dropping the objects, and cleaning the office when it is dirty.

b) **PDDL PROBLEM:**

```
(define (problem clean-offices))
(:domain clean-offices)

(:objects office1 office_nearby2 office_nearby3 office_nearby4
  computer1 chair1 desk1
  computer2 chair2 desk2
  computer3 chair3 desk3
  computer4 chair4 desk4)

(:init
  (office office1) (office office_nearby2) (office office_nearby3) (office office_nearby4)
  (dirty office1) (dirty office_nearby2) (dirty office_nearby3) (dirty office_nearby4)

  (chair chair1) (chair chair2) (chair chair3) (chair chair4)
  (computer computer1) (computer computer2) (computer computer3) (computer
  computer4)
  (desk desk1) (desk desk2) (desk desk3) (desk chair4)

  (on computer1 desk1) (on computer2 desk2) (on computer3 desk3) (on computer4 desk4)

  (clear chair1)(clear computer1)
  (clear chair2)(clear computer2)
  (clear chair3)(clear computer3)
  (clear chair4)(clear computer4)

  (empty_arm)

  (:goal
    (and (empty_arm)
      (in chair1 office1) (in desk1 office1) (in computer1 office1)
      (in chair2 office2) (in desk2 office2) (in computer2 office2)
      (in chair3 office3) (in desk3 office3) (in computer3 office3)
      (in chair4 office4) (in desk4 office4) (in computer4 office4)

      (on computer1 desk1) (on computer2 desk2) (on computer3 desk3) (on computer4 desk4)

      (clear chair1)(clear computer1)
      (clear chair2)(clear computer2)
      (clear chair3)(clear computer3)
      (clear chair4)(clear computer4)

      (not (empty office1))
      (not (empty office2))
      (not (empty office3))
```

```
(not (empty office4))
```

```
(clean office1)
```

```
(clean office2)
```

```
(clean office3)
```

```
(clean office4)))
```

- 1- The code above shows the problem, which introduces all the states and objects that will be used to solve the problem. This shows how many offices we have, number of chairs, desks, and PCs in the office, it also shows the nearby office and the current office.
- 2- The problem also shows the state of every object, for example, stating that the computer can be found on the desk, and the desk is in the office. Also explaining what an empty arm is, and what is the goal

Output:

After running both codes in the PDDL editor (Domain + Problem). At first the errors were missing parenthesis, which was solved by making sure every open parenthesis was closed and removing the unnecessary ones.

The next error was “Malformed Expression” – it meant that some expressions were not understood by the PDDL compiler, meaning they were written in wrong syntax. This problem was found in the domain – actions and it was solved by fixing the syntax.

Finally, the error facing the program right now is:

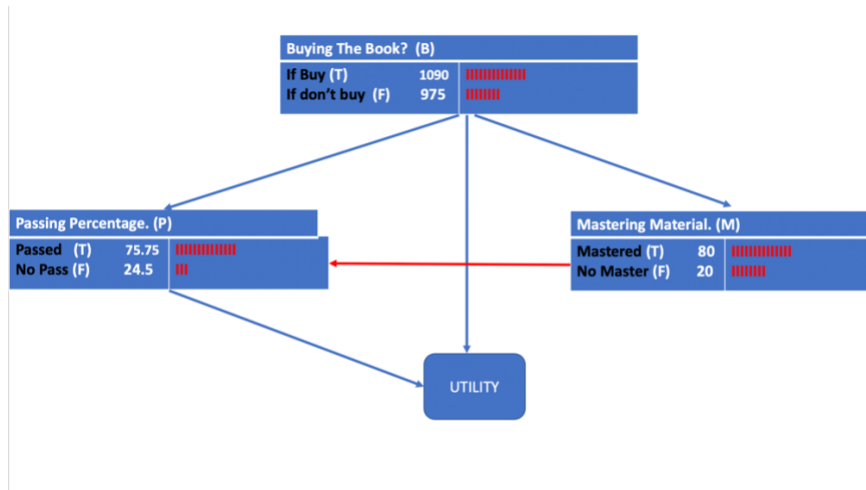
```
syntax error in line 3, '(':  
domain definition expected
```

After researching about this error, it can be seen that the domain definition (where the error is stating) is what contains the predicates and operators of the program, therefore, I have tried changing the syntax in the predicates, changing the variables name from X to different name for each predicate, I have made sure all parenthesis are closed within the domain definition area. But the error kept pointing when solving the problem.

I have attempted to run the program on another PDDL editor online, I did not get the same error, the new error was “Malformed Expression”.

Question 3:

a)



- b) When $B = b$ and $B = \neg b$, we calculate $P(p | B) \rightarrow P(\neg p | B)$ by disparaging M, to compute the expected utility.

$$\begin{aligned}
 P(p | b) &= \sum_m P(p | b, m) P(m | b) \\
 &= 0.9 \times 0.9 \times 0.1 \times 0.1 \\
 &= \mathbf{0.86}
 \end{aligned}$$

$$\begin{aligned}
 P(p | \neg b) &= \sum_m P(p | \neg b, m) P(m | \neg b) \\
 &= 0.3 \times 0.3 \times 0.8 \times 0.7 \\
 &= \mathbf{0.65}
 \end{aligned}$$

The expected utilities are:

$$\begin{aligned}
 EU[b] &= \sum_p P(p | b) U(p | b) \\
 &= 0.86(2000 - 100) + 0.14(-100)
 \end{aligned}$$

$$= 1620$$

$$EU[\neg b] = \sum_p P(p \mid \neg b) U(p \mid \neg b)$$

$$= 0.65 \times 2000 + 0.14 \times 0$$

$$= 1300$$

- c) Yes, the student should buy the book. The reason is because the expected *utility* of buying the book is of high value.

Question 4:

- a) Looking at the dataset provided, it is included of 4 different CSV files, where 2 files are for testing and the other 2 are for training. The features defer from a dataset to the other:

X_Test: Includes 320 features (Known as columns) which describe the dataset, it also has 8 classes (represented as classes) which is also describe the content of this dataset file. Includes a range index of 320 entries (0 to 319) with all data types being Float 64.

X_Train: Includes 744 features (Known as columns) and it also has the same 8 classes (represented as classes) that were found in the X_Test file – Includes a range index of 744 entries (0 to 743) with all data types being Float 64.

The Y files are split from the X files to ease the Train & Test process later. So, looking at the data found in Y_Test it will have the same features as in X_Test. Same goes for Y_Train and X_Train.

Y_Test: Includes 320 Features (Columns) and only 1 class which is split from X_Test - Includes a range index of 320 entries (0 to 319) - with all data types being Int64 (Integers)

Y_Train: Includes 744 features (Columns) and only 1 class which is split from X_Train. Includes a range index of 744 entries (0 to 743) with all data types being Int64 (Integers).

b)

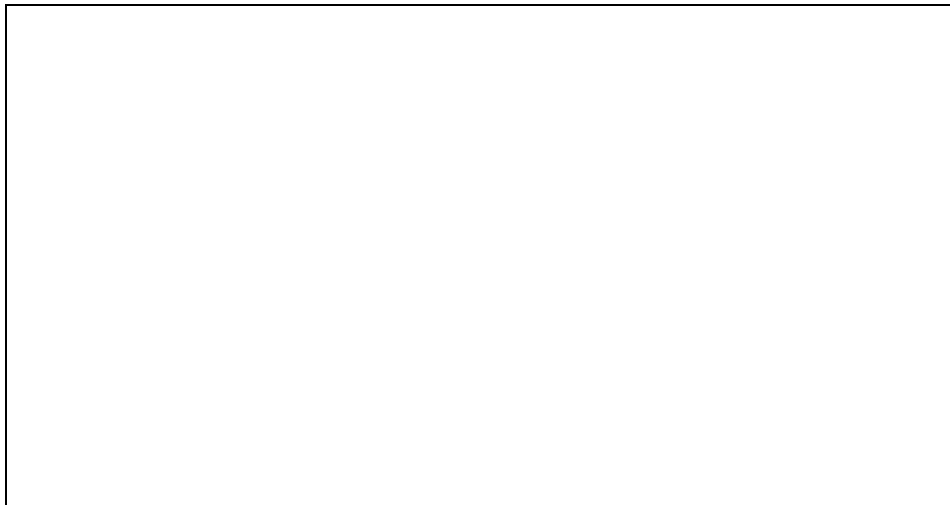
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

x1 = pd.read_csv("x_test.csv")
x2 = pd.read_csv("x_train.csv")
y1 = pd.read_csv("y_test.csv")
y2 = pd.read_csv("y_train.csv")

x_train, y_train = (x2, y2)
x_test, y_test = (x1, y1)

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train)
goal = accuracy_score(y1, tree.predict(x1))

print("Result is ", goal)
```



In the code above the following steps are undertaken:

- 1- We import the *Pandas* library which will allow us to import the dataset using “*pd.read_csv*”
- 2- As specified in the question, we need to use *SciKit* libraries, therefore *DecisionTreeClassifier* library is imported to perform classification on the data set.
- 3- The library *Accuracy_Score* is also imported which is used to predict the accuracy of the model after being trained and tested.
- 4- x1, x2, x3, and x4 are all variable names given to the four files of the dataset being imported.
- 5- As the dataset files are already split into X & Y, we do not need to split them. So we move into training the files (x2, y2) from the dataset. Next we test the other files (x1, y1)
- 6- We then use the “*DecisionTreeClassifier*” to classify the model after being trained using the dataset.
- 7- Finally, we need an output after training and testing the dataset, the output variable is named “goal” and is done using *accuracy_score*. The output that is received after running the code is: 0.80625 – 80%.

c) This code is continuous on the code in part (B):

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

tree_model = DecisionTreeClassifier(criterion="entropy",
                                    random_state=100, max_depth = 4, min_samples_leaf = 5)
tree_model.fit(x_train, y_train)

tree_prediction = tree.predict(x_test)
```

```
print("Predicted values are:")
print(tree_prediction)

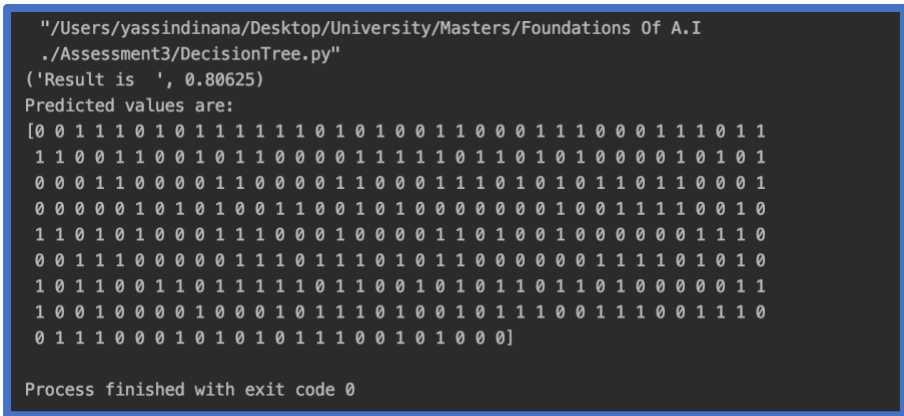
print("Confusion Matrix: ",
      confusion_matrix(y_test, tree_prediction))

print ("Accuracy : ",
       accuracy_score(y_test, tree_prediction) * 100)

print("Report : ",
      classification_report(y_test, tree_prediction))
```

In part C, two more libraries are called from *sklearn*, first one is the classification report, which is used to measure the quality of the tree predictions which will also show the accuracy of the predictions after training and testing the model.

First, we print the prediction values, below is a figure showing the output:



```
"/Users/yassindinana/Desktop/University/Masters/Foundations Of A.I
./Assessment3/DecisionTree.py"
('Result is ', 0.80625)
Predicted values are:
[0 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 1 1
 1 1 0 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 1
 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1
 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0
 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0
 0 0 1 1 1 0 0 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 1 0
 1 0 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 1
 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0
 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0]

Process finished with exit code 0
```

Then, using the confusion matrix library imported, we print a confusion matrix, which is used to describe the performance of a classification model by knowing the True & False values.

We also use the accuracy checking library to check the accuracy of the result, looking at the code provided in the question, it can be seen that the accuracy of the predictions is multiplied by 100, and this is to show the exact accuracy percentage.

Finally, we build a classification report, which shows the main classification given metrics such as measure the total true positives and negatives of the prediction model. The output also shows F1 score – which is a measure of the test's accuracy that is also shown.

The figure below shows the output, including the confusion matrix, accuracy, and more variables. Where the accuracy of the predictions is set to be 80.625%.

Using Entropy, which is the measure of random information being processed, the minimum leaf is processed to be 5 and the maximum leaf number and the maximum depth given to be 4.

```
('Confusion Matrix: ', array([[132, 27],
 [ 35, 126]]))
('Accuracy : ', 80.625)
('Report : ', u'
precision    recall  f1-score   support\n\n
 0.79      0.83      0.81    159\n
 161\n\n micro avg      0.81      0.81      0.81    320\n
 0.81      0.81      0.81    320\n\n weighted avg   0.81      0.81      0.81    320\n\n')
Process finished with exit code 0
```

Finally, we can build an actual decision tree for this problem. This is done by using GraphVis (Graph Visualisation) and PyDotPlus which can generate images using DOT.

The code below is what derives the decision tree using the mentioned libraries – it is also added to the same program mentioned in question 4.

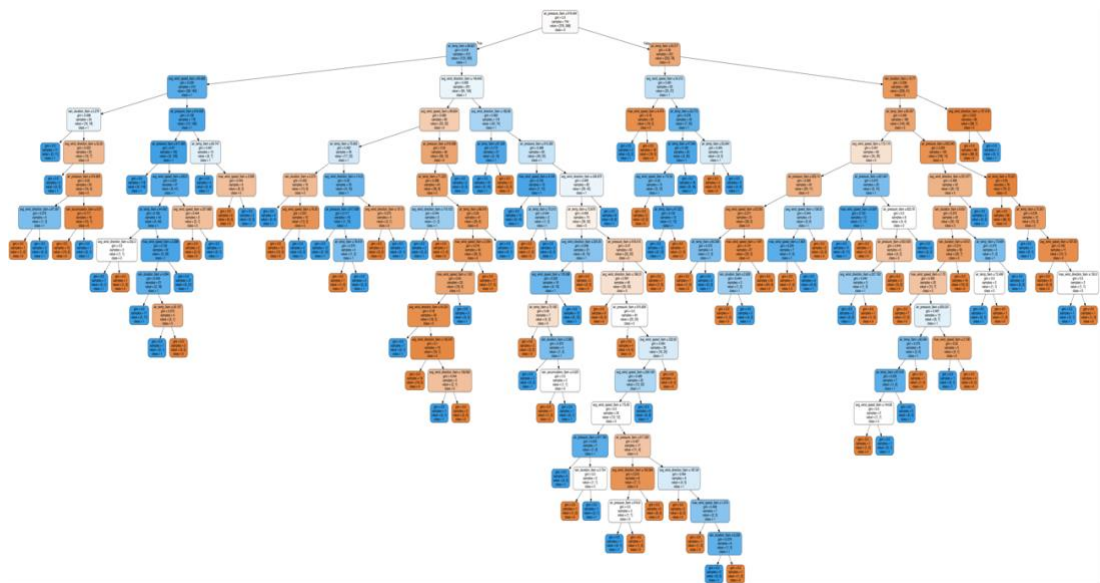
```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

feature_cols = ['air_pressure_9am',
                'air_temp_9am',
                'avg_wind_speed_9am',
                'max_wind_speed_9am',
                'avg_wind_direction_9am',
                'rain_duration_9am',
                'rain_accumulation_9am',
                'max_wind_direction_9am']

dot_data = StringIO()
export_graphviz(tree, out_file=dot_data,
```

```
filled=True, rounded=True,  
special_characters=True, feature_names  
= feature_cols, class_names=['0', '1'])  
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())  
graph.write_png('decision_tree.png')  
Image(graph.create_png())
```

After importing the libraries mentioned above and downloading GraphViz using PIP on the machine, running this program generates a .png version of the decision tree, below is the output:



As it can be seen in the figure above, the tree is overfitting, meaning the model is fed more data than it needs, which might decrease the accuracy. A solution that was tried to avoid overfitting is to split the data further, by using limited samples and testing the data on less features and classes, this process is known as cross validation.

Screenshot of the full code:

```
DecisionTree.py
1 import pandas as pd
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.metrics import accuracy_score
4 from sklearn.metrics import classification_report
5 from sklearn.metrics import confusion_matrix
6 from sklearn.tree import export_graphviz
7 from sklearn.externals.six import StringIO
8 from IPython.display import Image
9 import pydotplus
10
11 x1 = pd.read_csv("x_test.csv")
12 x2 = pd.read_csv("x_train.csv")
13 y1 = pd.read_csv("y_test.csv")
14 y2 = pd.read_csv("y_train.csv")
15
16 feature_cols = ['air_pressure_9am',
17                'air_temp_9am',
18                'avg_wind_speed_9am',
19                'max_wind_speed_9am',
20                'avg_wind_direction_9am',
21                'rain_duration_9am',
22                'rain_accumulation_9am',
23                'max_wind_direction_9am']
24
25 x_train, y_train = (x2, y2)
26 x_test, y_test = (x1, y1)
27
28 tree = DecisionTreeClassifier()
29 tree.fit(x_train, y_train)
30
31 goal = accuracy_score(y1, tree.predict(x1))
32
33 print("Result is ", goal)
34
35 tree_model = DecisionTreeClassifier(criterion="entropy", random_state=100, max_depth=4, min_samples_leaf=5)
36 tree_model.fit(x_train, y_train)
37
38 tree_prediction = tree.predict(x_test)
39 print("Predicted values are:")
40 print(tree_prediction)
41
42 print("Confusion Matrix: ",
43       confusion_matrix(y_test, tree_prediction))
44
45 print("Accuracy : ",
46       accuracy_score(y_test, tree_prediction) * 100)
47
48 print("Report : ",
49       classification_report(y_test, tree_prediction))
50
51 dot_data = StringIO()
52 export_graphviz(tree, out_file=dot_data,
53                filled=True, rounded=True,
54                special_characters=True, feature_names = feature_cols, class_names=['0','1'])
55 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
56 graph.write_png('decision_tree.png')
57 Image(graph.create_png())
58
59
60
61
62
63
```

Question 5:

- a) **Decision Tree:** The decision tree can be presented as a flowchart making it easy for humans to read and understand, where it includes different nodes, each node represents a decision that can help get closer to the goal, or it can also include a consequence which will move us away from the goal, those nodes include information from the dataset. It is used to explain the possible steps of an algorithm.

It is also considered to be part of the machine learning research; the algorithms are used for two separate tasks that can be solved which are *Classification tasks & Regression tasks*.

An example of a decision tree could be the decision tree built in *Question 4*. Another example can be shown below of a decision tree, the example goal is to know if I need to buy groceries.

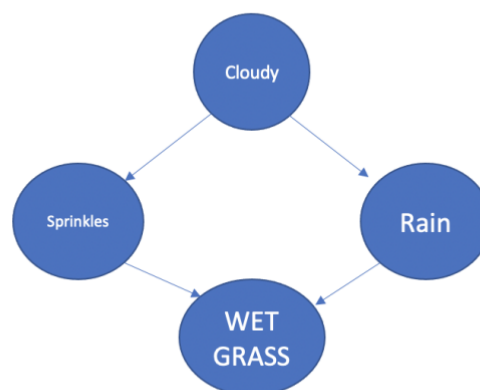


Decision Networks: The decision network also known as the Bayesian Network is also a graphical representation of a progressive decision problem. It uses a directed acyclic graph (DAG), where every path can be defined as decisions that the agent can solve, it mainly solves probabilistic problems. The DAG it is impossible to traverse a full graph from one node, the path has only one way to travel.

An example of decision network could be Apple / Google Maps, where the algorithm allows the agent to arrange the road from the initial state (starting node) till the goal state (goal node) by also applying the possibility of rerouting in case of traffic or road closures.

An example of decision networks could be the problem in question 2, where the decision network can decide whether the student should buy a textbook or no.

A decision network is built below where it can show whether the grass is wet or no.



Where the figure above represents a decision tree, explaining the situation to be:

- If it is cloudy → It might rain or drop sprinkles
- If it rains OR drop sprinkles → The grass will be wet

Therefore, using the given data, the agent can predict if the grass will be wet.

b) **Comparing decision networks and decision trees:**

<i>Decision Trees</i>	<i>Decision Networks</i>
Only two nodes can be found in decision trees (Neighbouring) which are the decision nodes and the leaf nodes (also known as split nodes).	It uses conditional probability tables, where each neighbouring node is a decision (as seen in the figure above). It uses utility nodes and decision nodes.
Used in for two sets of problems which are classification and regression problems.	Cannot solve regression problems – only available for classification problems.
Does not use DAG, tree looks like a flowchart.	Uses DAG (Directed Acyclic Graph)
Current nodes are the only extended nodes (known as parent nodes) by single stage.	Data is split as per conditions from initial stage to leaf nodes, resulting in conditional statements.