INF 552 PROGRAMMING ASSIGNMENT – 1 : DECISION TREES

SUDEEPTHA MOUNI GANJI USC ID:2942771049 sganji@usc.edu

Part 1: Implementation

a: Code attached.
b: The program was run on the given data and decision tree thus obtained is attached to page:3
c: Prediction for (occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No) is 'Yes'

Data structures used:

Set:

- Sets in python is an unordered and unindexed collection of data.
- I have used them to count and store labels as it doesn't allow duplicates and is iterable.

File:

- Python allows for reading, writing and handling of files in various formats.
- I have used file objects in python to store the input data after cleaning in CSV format.

Dictionary:

- Dictionaries in python is a mapping of unique keys to values.
- The values can be a set, a list or another dictionary as well.
- I have used dictionaries to store and manipulate the input data throughout the program.
- For the decision tree, I have specifically used nested dictionaries whose depth varies based on the attributes and their values.

Dataframe:

- Dataframes in python is used to store tabular data in the forms of rows and columns.
- I have used dataframes to store and manipulate input data throughout the program as they allow for the storage of heterogenous data which can be accessed and manipulated in a variety of ways.

My Decision Tree structure:

- My decision tree is a dictionary of nested dictionaries.
- The attribute with the high information gain becomes the first key.
- The labels of this attribute are its values which are again in the form of nested dictionaries.
- These labels will also in turn become keys whose values will have the attributes.
- The cycle continues until we get a final string which becomes our output.
- In our case, it is 'Yes'/ 'No' which are the labels of Enjoy attribute.

Code-level optimizations:

- I have used Pandas and NumPy for calculations and in-built functions like log2 and unique() for getting labels to improve the efficiency of the code rather than extract and calculate everything.
- My decision tree doesn't add a node from which no useful data is derived to reduce calculations and save memory and time.
- Also, when there is a case where the information gain is same for different attributes (this has occurred in our case because of having quite less input data), I just take one of the attributes and prune the remaining instead of building the tree on those which have not much use. This results in a reduction in computation. When asked to predict about such attributes where there is no sufficient data, I have handled it in the prediction function by giving the output as 'can't be determined due to lack of information'.
- It also outputs a possibility of not being able to give the right output because of lack of input data or noisy data rather than give an arbitrary value.
- I have used a dictionary of dictionaries in Python to store the tree rather than use lists as it would increase the efficiency and make it easier to find attributes and their labels and also makes it easier to traverse the tree.
- I have used pprint to print the decision tree as it provides a proper indent for the decision tree, which is in the form of a nested dictionary, thereby improving readability of solution.

Challenges faced:

- Deciding the format to store input data and cleaned data was a challenge until I came across the csv package and DataFrames in Pandas along with nested dictionaries.
- The MacOS has python 2.7 as a default version and some of the functionalities of python 3 wouldn't work in 2.7.
- Also, different versions of Python would implement same in-built functions in different manner. For example, DictReader would return an unordered dictionary in Python 2.7 and an ordered dictionary in Python 3. These issues were solved by separately extracting the right order of fields from the files and rearranging the columns in the dataframe.
- While calculating entropy, Python 2.7 wouldn't convert the terms implicitly and would give a 0 instead when divided by total count. So, I converted the numerator which was count of each label to float.
- When the number of labels would sometimes be zero, log values would return Nan. This was solved by adding epsilon value to denominators and logarithmic values when calculating attribute entropy. This also helped in pruning nodes without useful information.
- On following the exact ID3(decision tree) algorithm, my program would go into a loop because of the contradicting data on lines 18 & 21. So, I wrote a termination condition which checks the dataframe and issues a label saying, "can't be determined due to noisy data."

- Similarly, when the decision tree is unable to predict due to lack of data, I have also issued another label “can’t be determined due to lack of data” which gets executed in the predict function when a relevant node is not found.

Prediction obtained:

Input: predict_data = {'Occupied': 'Moderate', 'Price': 'Cheap', 'Music': 'Loud', 'Location': 'City-Center', 'VIP': 'No', 'Favorite Beer': 'No'}

Output: ‘YES’

Printing process:

- The root node of the decision tree (which is a nested dictionary) is the attribute “Occupied” for which the label in the given prediction data is Moderate.
- So, my program extracts the value for “Moderate” which is yet another decision tree in the form of a nested dictionary with the new root node as Location.
- As the result is of the type dictionary it goes deeper into the decision tree.
- The given label for Location is City-Center in the prediction data.
- On extracting the value for City-Center we get a string ‘Yes’.
- As the result is a string, the program gets terminated and prints the prediction: ’Yes ’
- I have attached a screenshot of the tree and how I got the predicted value for your reference.

```
Current dtree:
{'Occupied': {'High': {'Location': {'City-Center': 'Yes',
                                    'German-Colony': 'No',
                                    'Mahane-Yehuda': 'Yes',
                                    'Talpiot': 'No'}},
              'Low': {'Location': {'City-Center': {'Price': {'Cheap': 'No',
                                                             'Normal': {'VIP': {'No': 'cant be determined due to noisy data'}}}},
                                   'Ein-Karem': {'VIP': {'No': 'No',
                                                         'Yes': 'Yes'}},
                                   'Mahane-Yehuda': 'No',
                                   'Talpiot': 'No'}},
              'Moderate': {'Location': {'City-Center': 'Yes',
                                        'Ein-Karem': 'Yes',
                                        'German-Colony': {'VIP': {'No': 'No',
                                                                  'Yes': 'Yes'}},
                                        'Mahane-Yehuda': 'Yes',
                                        'Talpiot': {'Music': {'Loud': 'No',
                                                              'Quiet': 'Yes'}}}}}}
Given prediction data:
{'Price': 'Cheap', 'VIP': 'No', 'Music': 'Loud', 'Location': 'City-Center', 'Occupied': 'Moderate', 'Favorite Beer': 'No'}

Attribute from decision tree:Occupied
Value from prediction data:Moderate

GOING ONE LEVEL DEEPER

Current dtree:
{'Location': {'City-Center': 'Yes',
              'Ein-Karem': 'Yes',
              'German-Colony': {'VIP': {'No': 'No', 'Yes': 'Yes'}},
              'Mahane-Yehuda': 'Yes',
              'Talpiot': {'Music': {'Loud': 'No', 'Quiet': 'Yes'}}}}
Given prediction data:
{'Price': 'Cheap', 'VIP': 'No', 'Music': 'Loud', 'Location': 'City-Center', 'Occupied': 'Moderate', 'Favorite Beer': 'No'}

Attribute from decision tree:Location
Value from prediction data:City-Center

GOING ONE LEVEL DEEPER

City-Center:Yes

FINAL SOLUTION FOUND:
Prediction:Yes

Process finished with exit code 0
```

Part 2: Software Familiarization

I have used the following existing packages and libraries for implementation

csv:

- I have used csv (comma separated values) to read the provided input data and write another csv file(**cleaned_csv_file1.csv**) after cleaning the data.
- The csv module provides classes for reading data in tabular(csv) format and for writing them also.

Pandas:

- I have used Pandas to create a dataframe from a csv file(**cleaned_csv_file1.csv**) which was obtained after cleaning the input.
- Dataframe is a two-dimensional data structure available in Pandas to store tabular data in the form of labelled rows and columns from existing data like csv files, dictionaries etc.

sklearn:

- sklearn is a package which provides efficient implementations for many machine learning algorithms.
- I have used tree module from sklearn package to implement the decision tree by using the criterion as entropy and fitting the give data.
- The tree module provides for the implementation of decision tree models which can be used for classification and regression.
- I have also learned that the sklearn also provides the id3 package which has an id3Estimator() which takes as input the data which is fitted to the estimator and then we can display the tree by using graphiz which develops a dot file which can be used to print the dot file in a graphical format.

Graphviz and Ipython:

- I have used graphviz to convert the decision tree into a graph format by providing the decision tree as input for the Source class which was implemented from graphviz and later used it to convert it to an image in png format.
- Graphviz package takes in structured information and converts them into graphs and also into useful formats like images and SVG (for web pages)
- The program successfully displays decision trees in a graphical format on running the program.

NumPy:

- NumPy is a Python package which supports multidimensional arrays and also provides several mathematical functions.
- I have used NumPy to find unique labels in a dataframe using unique() module and log values during entropy calculation.

Comparing with my implementation and how I could improve my code:

- Python's in-built libraries are very elegant and make it quite easy to implement and develop a decision tree especially when it is a CART algorithm that needs to be implemented(sklearn.tree has the DecisionTreeClassifier method which implements CART algorithm as well as the ID3 algorithm when we give entropy as the criterion).

- However the currently in-built functions in sklearn (I have tried both Id3Estimator from Id3 package and DecisionTreeClassifier from sklearn.tree package) both work only when the attribute labels are integer values.
- In my implementation, I was able to implement directly with String datatype labels which wasn't directly possible with Python's in-built functions unless you perform one-hot-encoding which converts the string labels to binary features but at the same time is computationally expensive thereby reducing the advantage of decision trees which is less computation.
- Finally I implemented the algorithm using construct_decision_tree_classifier() from the DecisionTree 3.4.3 package. This in-built function was able to handle categorial data and build a decision tree based on it and print it using display_decision_tree() function (I have attached this file). It gives the same output as my implementation in a different format.
- The in-built functions provide many features like controlling the max-depth, minimum samples to split on, not using repeating features etc. The in-built python library functions also split and train part of the data and use the remaining data for validation and provides the accuracy score. I haven't implemented these features in my own implementation as the data given to us was very less and I could introduce bias, but I think that would be a great improvement to my current implementation.

```
Constructing a decision tree...
NODE 0:     BRANCH TESTS TO NODE: []
            Decision Feature: Occupied   Node Creation Entropy: 0.976   Class Probs: ['Enjoy=No => 0.409', 'Enjoy=Yes => 0.591']

NODE 1:        BRANCH TESTS TO NODE: ['Occupied=High']
               Decision Feature: Location   Node Creation Entropy: 0.918   Class Probs: ['Enjoy=No => 0.333', 'Enjoy=Yes => 0.667']

NODE 2:           BRANCH TESTS TO NODE: ['Occupied=High', 'Location=City-Center']
                  Decision Feature: VIP   Node Creation Entropy: 0.768   Class Probs: ['Enjoy=No => 0.224', 'Enjoy=Yes => 0.776']

NODE 3:              BRANCH TESTS TO NODE: ['Occupied=High', 'Location=City-Center', 'VIP=Yes']
                     Decision Feature: Price   Node Creation Entropy: 0.392   Class Probs: ['Enjoy=No => 0.077', 'Enjoy=Yes => 0.923']

NODE 4:                 BRANCH TESTS TO NODE: ['Occupied=High', 'Location=City-Center', 'VIP=Yes', 'Price=Normal']
                        Decision Feature: Music   Node Creation Entropy: 0.315   Class Probs: ['Enjoy=No => 0.057', 'Enjoy=Yes => 0.943']

NODE 5:                    BRANCH TESTS TO LEAF NODE: ['Occupied=High', 'Location=City-Center', 'VIP=Yes', 'Price=Normal', 'Music=Loud']
                           Node Creation Entropy: 0.224   Class Probs: ['Enjoy=No => 0.036', 'Enjoy=Yes => 0.964']

NODE 6:           BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Ein-Karem']
                  Decision Feature: VIP   Node Creation Entropy: 0.710   Class Probs: ['Enjoy=No => 0.194', 'Enjoy=Yes => 0.806']

NODE 7:              BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Ein-Karem', 'VIP=Yes']
                     Decision Feature: Price   Node Creation Entropy: 0.347   Class Probs: ['Enjoy=No => 0.065', 'Enjoy=Yes => 0.935']

NODE 8:                 BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Ein-Karem', 'VIP=Yes', 'Price=Normal']
                        Decision Feature: Music   Node Creation Entropy: 0.277   Class Probs: ['Enjoy=No => 0.048', 'Enjoy=Yes => 0.952']

NODE 9:                    BRANCH TESTS TO LEAF NODE: ['Occupied=High', 'Location=Ein-Karem', 'VIP=Yes', 'Price=Normal', 'Music=Loud']
                           Node Creation Entropy: 0.195   Class Probs: ['Enjoy=No => 0.030', 'Enjoy=Yes => 0.970']

NODE 10:          BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Mahane-Yehuda']
                  Decision Feature: VIP   Node Creation Entropy: 0.710   Class Probs: ['Enjoy=No => 0.194', 'Enjoy=Yes => 0.806']

NODE 11:             BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Mahane-Yehuda', 'VIP=Yes']
                     Decision Feature: Price   Node Creation Entropy: 0.347   Class Probs: ['Enjoy=No => 0.065', 'Enjoy=Yes => 0.935']

NODE 12:                BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Mahane-Yehuda', 'VIP=Yes', 'Price=Normal']
                        Decision Feature: Music   Node Creation Entropy: 0.277   Class Probs: ['Enjoy=No => 0.048', 'Enjoy=Yes => 0.952']

NODE 13:                   BRANCH TESTS TO LEAF NODE: ['Occupied=High', 'Location=Mahane-Yehuda', 'VIP=Yes', 'Price=Normal', 'Music=Loud']
                           Node Creation Entropy: 0.195   Class Probs: ['Enjoy=No => 0.030', 'Enjoy=Yes => 0.970']

NODE 14:          BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Talpiot']
                  Decision Feature: VIP   Node Creation Entropy: 0.900   Class Probs: ['Enjoy=No => 0.684', 'Enjoy=Yes => 0.316']

NODE 15:             BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Talpiot', 'VIP=No']
                     Decision Feature: Music   Node Creation Entropy: 0.799   Class Probs: ['Enjoy=No => 0.758', 'Enjoy=Yes => 0.242']
```

```
NODE 15:              BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Talpiot', 'VIP=No']
                      Decision Feature: Music   Node Creation Entropy: 0.799   Class Probs: ['Enjoy=No => 0.758', 'Enjoy=Yes => 0.242']

NODE 16:                 BRANCH TESTS TO NODE: ['Occupied=High', 'Location=Talpiot', 'VIP=No', 'Music=Quiet']
                         Decision Feature: Price   Node Creation Entropy: 0.683   Class Probs: ['Enjoy=No => 0.819', 'Enjoy=Yes => 0.181']

NODE 17:                    BRANCH TESTS TO LEAF NODE: ['Occupied=High', 'Location=Talpiot', 'VIP=No', 'Music=Quiet', 'Price=Cheap']
                            Node Creation Entropy: 0.565   Class Probs: ['Enjoy=No => 0.867', 'Enjoy=Yes => 0.133']

NODE 18:                    BRANCH TESTS TO LEAF NODE: ['Occupied=High', 'Location=Talpiot', 'VIP=No', 'Music=Quiet', 'Price=Expensive']
                            Node Creation Entropy: 0.657   Class Probs: ['Enjoy=No => 0.830', 'Enjoy=Yes => 0.170']

NODE 19:        BRANCH TESTS TO NODE: ['Occupied=Low']
                Decision Feature: VIP   Node Creation Entropy: 0.863   Class Probs: ['Enjoy=No => 0.714', 'Enjoy=Yes => 0.286']

NODE 20:           BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No']
                   Decision Feature: Location   Node Creation Entropy: 0.754   Class Probs: ['Enjoy=No => 0.783', 'Enjoy=Yes => 0.217']

NODE 21:              BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=German-Colony']
                      Decision Feature: Price   Node Creation Entropy: 0.428   Class Probs: ['Enjoy=No => 0.913', 'Enjoy=Yes => 0.087']

NODE 22:                 BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=German-Colony', 'Price=Cheap']
                         Decision Feature: Music   Node Creation Entropy: 0.336   Class Probs: ['Enjoy=No => 0.938', 'Enjoy=Yes => 0.062']

NODE 23:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Low', 'VIP=No', 'Location=German-Colony', 'Price=Cheap', 'Music=Quiet']
                            Node Creation Entropy: 0.260   Class Probs: ['Enjoy=No => 0.956', 'Enjoy=Yes => 0.044']

NODE 24:                 BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=German-Colony', 'Price=Expensive']
                         Decision Feature: Music   Node Creation Entropy: 0.407   Class Probs: ['Enjoy=No => 0.919', 'Enjoy=Yes => 0.081']

NODE 25:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Low', 'VIP=No', 'Location=German-Colony', 'Price=Expensive', 'Music=Quiet']
                            Node Creation Entropy: 0.318   Class Probs: ['Enjoy=No => 0.942', 'Enjoy=Yes => 0.058']

NODE 26:              BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=Talpiot']
                      Decision Feature: Price   Node Creation Entropy: 0.328   Class Probs: ['Enjoy=No => 0.940', 'Enjoy=Yes => 0.060']

NODE 27:                 BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=Talpiot', 'Price=Cheap']
                         Decision Feature: Music   Node Creation Entropy: 0.253   Class Probs: ['Enjoy=No => 0.958', 'Enjoy=Yes => 0.042']

NODE 28:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Low', 'VIP=No', 'Location=Talpiot', 'Price=Cheap', 'Music=Quiet']
                            Node Creation Entropy: 0.193   Class Probs: ['Enjoy=No => 0.970', 'Enjoy=Yes => 0.030']

NODE 29:                 BRANCH TESTS TO NODE: ['Occupied=Low', 'VIP=No', 'Location=Talpiot', 'Price=Expensive']
                         Decision Feature: Music   Node Creation Entropy: 0.310   Class Probs: ['Enjoy=No => 0.944', 'Enjoy=Yes => 0.056']

NODE 30:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Low', 'VIP=No', 'Location=Talpiot', 'Price=Expensive', 'Music=Quiet']
                            Node Creation Entropy: 0.239   Class Probs: ['Enjoy=No => 0.961', 'Enjoy=Yes => 0.039']
```

```
NODE 31:        BRANCH TESTS TO NODE: ['Occupied=Moderate']
                Decision Feature: Location   Node Creation Entropy: 0.764   Class Probs: ['Enjoy=No => 0.222', 'Enjoy=Yes => 0.778']

NODE 32:           BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=City-Center']
                   Decision Feature: VIP   Node Creation Entropy: 0.589   Class Probs: ['Enjoy=No => 0.142', 'Enjoy=Yes => 0.858']

NODE 33:              BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=City-Center', 'VIP=Yes']
                      Decision Feature: Price   Node Creation Entropy: 0.267   Class Probs: ['Enjoy=No => 0.046', 'Enjoy=Yes => 0.954']

NODE 34:                 BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=City-Center', 'VIP=Yes', 'Price=Normal']
                         Decision Feature: Music   Node Creation Entropy: 0.211   Class Probs: ['Enjoy=No => 0.033', 'Enjoy=Yes => 0.967']

NODE 35:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Moderate', 'Location=City-Center', 'VIP=Yes', 'Price=Normal', 'Music=Loud']
                            Node Creation Entropy: 0.146   Class Probs: ['Enjoy=No => 0.021', 'Enjoy=Yes => 0.979']

NODE 36:           BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Ein-Karem']
                   Decision Feature: Price   Node Creation Entropy: 0.532   Class Probs: ['Enjoy=No => 0.121', 'Enjoy=Yes => 0.879']

NODE 37:              BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Ein-Karem', 'Price=Normal']
                      Decision Feature: Music   Node Creation Entropy: 0.438   Class Probs: ['Enjoy=No => 0.090', 'Enjoy=Yes => 0.910']

NODE 38:                 BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Ein-Karem', 'Price=Normal', 'Music=Loud']
                         Decision Feature: VIP   Node Creation Entropy: 0.319   Class Probs: ['Enjoy=No => 0.058', 'Enjoy=Yes => 0.942']

NODE 39:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Moderate', 'Location=Ein-Karem', 'Price=Normal', 'Music=Loud', 'VIP=Yes']
                            Node Creation Entropy: 0.127   Class Probs: ['Enjoy=No => 0.017', 'Enjoy=Yes => 0.983']

NODE 40:           BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Mahane-Yehuda']
                   Decision Feature: Price   Node Creation Entropy: 0.532   Class Probs: ['Enjoy=No => 0.121', 'Enjoy=Yes => 0.879']

NODE 41:              BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Mahane-Yehuda', 'Price=Normal']
                      Decision Feature: Music   Node Creation Entropy: 0.438   Class Probs: ['Enjoy=No => 0.090', 'Enjoy=Yes => 0.910']

NODE 42:                 BRANCH TESTS TO NODE: ['Occupied=Moderate', 'Location=Mahane-Yehuda', 'Price=Normal', 'Music=Loud']
                         Decision Feature: VIP   Node Creation Entropy: 0.319   Class Probs: ['Enjoy=No => 0.058', 'Enjoy=Yes => 0.942']

NODE 43:                    BRANCH TESTS TO LEAF NODE: ['Occupied=Moderate', 'Location=Mahane-Yehuda', 'Price=Normal', 'Music=Loud', 'VIP=Yes']
                            Node Creation Entropy: 0.127   Class Probs: ['Enjoy=No => 0.017', 'Enjoy=Yes => 0.983']
```

Part 3: Applications

Bioinformatics:

- Decision trees can be used in bioinformatics to analyze and classify large amounts of data.
- Qlucore which is a bioinformatics software recently announced its latest version of Qlucore Omics Explorer which includes single-cell data analysis and survival calculations.
- This was possible through the use of gradient boosted decision trees to achieve precision which is a necessity in medical applications.
- The modified decision trees helped to achieve quick, precise and real-time analysis of data.

Customer Relationship Management (CRM):

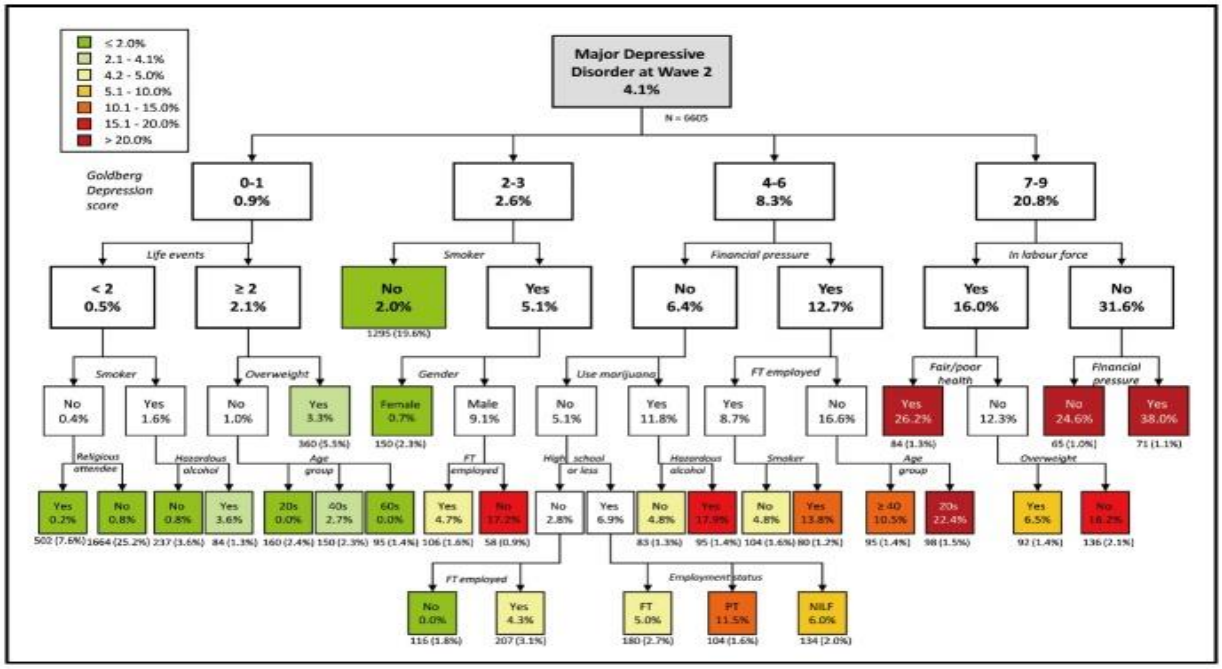
- Decision trees can be used to analyze how individuals analyze use online services and thereby manage and improve the company's relationship with the customer.
- The information obtained can be used to improve sales and make decisions on what products and services can be launched.
- InsideSales which has been funded by Microsoft and Salesforce and caters to American Express uses decision trees to predict customer behavior.
- It uses decision trees along with random forests to predict when a customer will answer a mail or their intent to buy a product even before they buy it by analyzing historical data.

Engineering:

- Decision trees can be used to solve many common problems in various industries involving engineering.
- Decision trees are used to find faults in the bearings of rotary machineries by taking into account only those variables that are necessary and ignoring irrelevant variables.
- Decision trees can be used to determine the expected energy consumption in households based on the number of members, flat size, number of ACs etc taken as attributes.
- Andritz Automation is trying to run its plants autonomously by using decision trees and Markov Decision Problems for training the models.
- CSE Icon has been training models based on decision trees in order to solve common problems in the gas and oil industry like determining the optimal operating point on a well, artificial lift optimization and prediction.

Psychology:

- Decision trees can be used to find out the most important risk factors for psychological diseases like Major Depressive Order(MDD).



- In this study, from 17 risk factors, the most important risk factors were to be identified and those were used as the attributes while individuals were divided into sub-groups from root node to branches. By doing this, researchers identified the highest risk combinations.