Minh Tran – HW1: Decision Trees Report

1. **Part 1: Implementation:**

a. Write a program to construct a decision tree based on the idea of splitting by Information Gain:

- Pycharm IDE is used with Python 3.6 as the programming language. Basic libraries such as math and csv are used to facilitate the program
- Algorithm:

ID3 algorithm is implemented with the pseudocode shown below:

```
DecisionTreeLearning(Examples, Features)
- if Examples have the same class, return a leaf with this class
- else if Features is empty, return a leaf with the majority class
- else if Examples is empty, return a leaf with majority class of parent
- else: find the best feature A to split (e.g. based on conditional entropy and information gain)
    Tree  <- a root with test on A
    For each value a of A:
        Child  <- DecisionTreeLearning(Examples with A = a, Features - {A})
        add Child to Tree as a new branch
return Tree
```

- Data structure:

i. Data entry is converted from text to csv by function txtReader so that training data is a list of lists (shown below). The provided file dt-data.txt has been converted to a csv file – dt_data.csv

```
training data:  [['Occupied', 'Price', 'Music', 'Location', 'VIP', 'FavoriteBeer', 'Enjoy'],
 ['High', 'Expensive', 'Loud', 'Talpiot', 'No', 'No', 'No'], ['High', 'Expensive', 'Loud',
 'City-Center', 'Yes', 'No', 'Yes'], ['Moderate', 'Normal', 'Quiet', 'City-Center', 'No', 'Yes',
 'Yes'], ['Moderate', 'Expensive', 'Quiet', 'German-Colony', 'No', 'No', 'No'], ['Moderate',
 'Expensive', 'Quiet', 'German-Colony', 'Yes', 'Yes', 'Yes'], ['Moderate', 'Normal', 'Quiet',
 'Ein-Karem', 'No', 'No', 'Yes'], ['Low', 'Normal', 'Quiet', 'Ein-Karem', 'No', 'No', 'No'],
 ['Moderate', 'Cheap', 'Loud', 'Mahane-Yehuda', 'No', 'No', 'Yes'], ['High', 'Expensive', 'Loud',
 'City-Center', 'Yes', 'Yes', 'Yes'], ['Low', 'Cheap', 'Quiet', 'City-Center', 'No', 'No', 'No'],
 ['Moderate', 'Cheap', 'Loud', 'Talpiot', 'No', 'Yes', 'No'], ['Low', 'Cheap', 'Quiet', 'Talpiot',
  'Yes', 'Yes', 'No'], ['Moderate', 'Expensive', 'Quiet', 'Mahane-Yehuda', 'No', 'Yes', 'Yes'],
 ['High', 'Normal', 'Loud', 'Mahane-Yehuda', 'Yes', 'Yes', 'Yes'], ['Moderate', 'Normal', 'Loud',
 'Ein-Karem', 'No', 'Yes', 'Yes'], ['High', 'Normal', 'Quiet', 'German-Colony', 'No', 'No',
 'No'], ['High', 'Cheap', 'Loud', 'City-Center', 'No', 'Yes', 'Yes'], ['Low', 'Normal', 'Quiet',
 'City-Center', 'No', 'No', 'No'], ['Low', 'Expensive', 'Loud', 'Mahane-Yehuda', 'No', 'No',
 'No'], ['Moderate', 'Normal', 'Quiet', 'Talpiot', 'No', 'No', 'Yes'], ['Low', 'Normal', 'Quiet',
 'City-Center', 'No', 'No', 'Yes'], ['Low', 'Cheap', 'Loud', 'Ein-Karem', 'Yes', 'Yes', 'Yes']]
```

The first list is the attributes: Occupied, Price… Enjoy, of which the last item is the target attribute.

ii. Dictionary is used extensively to represent the complete tree structure as Tree = {attribute1: {Value1.1: {attribute2: {Value2.1…}}}}. An attribute

is associated with different values. For example, attribute “Occupied” is associated with 3 values “Moderate”, “High” and “Low”

iii. A “Node” class is used with two properties (attribute and children) and two methods (setValue and makeChildren). This node will be part of the tree structure when implementing splitting criteria based on information gain.

iv. Files included:

- ✓ Hw3MinhTranPart1.py - Contains the implementation of decision tree
- ✓ dt_data.csv - Training data in csv format.
- ✓ dt_data.txt - Training data in txt format.
- ✓ Hw1MinhTranReport.pdf – pdf report file

- Code-level optimization:
  i. To improve code performance, several functions are implemented: calcIG, calcEntropy, findMajority, getValues, getSubdata, chooseAttr
  ii. In the createTree function, recursion is an input to be able to constrain the depth of the tree. Currently, the tree has no pruning as recursion parameter is not constrained.
  iii. Code can be broken down to smaller .py files. But they are wrapped in 1 python script for easy tracking
- Challenges:
  i. Storing the count of values (“Yes/No”) of the target attribute (“Enjoy”)
  ii. Pruning the tree based on ID3 algorithm is difficult in the top down greedy approach.

b. Run program on data file:

- Final decision tree when using ID3 algorithm on the whole dataset: {'Occupied': {'High': {'Location': {'Talpiot': 'No', 'City-Center': 'Yes', 'Mahane-Yehuda': 'Yes', 'German-Colony': 'No'}}, 'Moderate': {'Location': {'City-Center': 'Yes', 'German-Colony': {'VIP': {'No': 'No', 'Yes': 'Yes'}}, 'Ein-Karem': 'Yes', 'Mahane-Yehuda': 'Yes', 'Talpiot': {'Price': {'Cheap': 'No', 'Normal': 'Yes'}}}}}, 'Low': {'Location': {'Ein-Karem': {'Price': {'Normal': 'No', 'Cheap': 'Yes'}}, 'City-

Center': {'Price': {'Cheap': 'No', 'Normal': {'Music': {'Quiet': {'VIP': {'No': {'FavoriteBeer': {'No': 'No'}}}}}}}}, 'Talpiot': 'No', 'Mahane-Yehuda': 'No'}}}}

- Readable format:

  Occupied = High

  | Location = Talpiot: No

  | Location = City-Center: Yes

  | Location = German-Colony: No

  | Location = Ein-Karem: null

  | Location = Mahane-Yehuda: Yes

  Occupied = Moderate

  | Location = Talpiot

  | | Price = Expensive: null

  | | Price = Normal: Yes

  | | Price = Cheap: No

  | Location = City-Center: Yes

  | Location = German-Colony

  | | VIP = No: No

  | | VIP = Yes: Yes

  | Location = Ein-Karem: Yes

  | Location = Mahane-Yehuda: Yes

  Occupied = Low

  | Location = Talpiot: No

  | Location = City-Center

  | | Price = Expensive: null

  | | Price = Normal:

  | | | Music = Quiet:

  | | | | VIP = No:

  | | | | | Favorite Beer =No: No

| | Price = Cheap: No

| Location = German-Colony: null

| Location = Ein-Karem

| | Price = Expensive: null

| | Price = Normal: No

| | Price = Cheap: Yes

| Location = Mahane-Yehuda: No

c. Make prediction for testData = (occupied = Moderate; price = Cheap; music = Loud; location = City-Center;VIP = No; favorite beer = No):

The prediction made is <u>Enjoy = YES</u>

The predictions made by the algorithm is displayed on the IDE (see below).

```
training data:  [['Occupied', 'Price', 'Music', 'Location', 'VIP', 'FavoriteBeer', 'Enjoy'],
 ['High', 'Expensive', 'Loud', 'Talpiot', 'No', 'No', 'No'], ['High', 'Expensive', 'Loud',
 'City-Center', 'Yes', 'No', 'Yes'], ['Moderate', 'Normal', 'Quiet', 'City-Center', 'No', 'Yes',
 'Yes'], ['Moderate', 'Expensive', 'Quiet', 'German-Colony', 'No', 'No', 'No'], ['Moderate',
 'Expensive', 'Quiet', 'German-Colony', 'Yes', 'Yes', 'Yes'], ['Moderate', 'Normal', 'Quiet',
 'Ein-Karem', 'No', 'No', 'Yes'], ['Low', 'Normal', 'Quiet', 'Ein-Karem', 'No', 'No', 'No'],
 ['Moderate', 'Cheap', 'Loud', 'Mahane-Yehuda', 'No', 'No', 'Yes'], ['High', 'Expensive', 'Loud',
 'City-Center', 'Yes', 'Yes', 'Yes'], ['Low', 'Cheap', 'Quiet', 'City-Center', 'No', 'No', 'No'],
 ['Moderate', 'Cheap', 'Loud', 'Talpiot', 'No', 'Yes', 'No'], ['Low', 'Cheap', 'Quiet', 'Talpiot',
  'Yes', 'Yes', 'No'], ['Moderate', 'Expensive', 'Quiet', 'Mahane-Yehuda', 'No', 'Yes', 'Yes'],
 ['High', 'Normal', 'Loud', 'Mahane-Yehuda', 'Yes', 'Yes', 'Yes'], ['Moderate', 'Normal', 'Loud',
 'Ein-Karem', 'No', 'Yes', 'Yes'], ['High', 'Normal', 'Quiet', 'German-Colony', 'No', 'No',
 'No'], ['High', 'Cheap', 'Loud', 'City-Center', 'No', 'Yes', 'Yes'], ['Low', 'Normal', 'Quiet',
 'City-Center', 'No', 'No', 'No'], ['Low', 'Expensive', 'Loud', 'Mahane-Yehuda', 'No', 'No',
 'No'], ['Moderate', 'Normal', 'Quiet', 'Talpiot', 'No', 'No', 'Yes'], ['Low', 'Normal', 'Quiet',
 'City-Center', 'No', 'No', 'Yes'], ['Low', 'Cheap', 'Loud', 'Ein-Karem', 'Yes', 'Yes', 'Yes']]
target:  Enjoy
Decision Tree generated
FINAL TREE:  {'Occupied': {'High': {'Location': {'Talpiot': 'No', 'City-Center': 'Yes',
 'Mahane-Yehuda': 'Yes', 'German-Colony': 'No'}}, 'Moderate': {'Location': {'City-Center': 'Yes',
 'German-Colony': {'VIP': {'No': 'No', 'Yes': 'Yes'}}, 'Ein-Karem': 'Yes', 'Mahane-Yehuda':
 'Yes', 'Talpiot': {'Price': {'Cheap': 'No', 'Normal': 'Yes'}}}}}, 'Low': {'Location':
 {'Ein-Karem': {'Price': {'Normal': 'No', 'Cheap': 'Yes'}}, 'City-Center': {'Price': {'Cheap':
 'No', 'Normal': {'Music': {'Quiet': {'VIP': {'No': {'FavoriteBeer': {'No': 'No'}}}}}}}},
 'Talpiot': 'No', 'Mahane-Yehuda': 'No'}}}}
test Data:  [['Moderate', 'Cheap', 'Loud', 'City-Center', 'No', 'No']]
test Data Entry 1 = Yes
```

2. Part 2: Software familiarization

a. Python library function:

- ✓ Scikit-Learn offers useful built-in methods to implement decision tree. However, it does not support categorical data, thus needing conversion into binary values using one-hot encoding (0 indicates off, 1 indicates on). The default criterion for calculating information-gain in Scikit-Learn is gini method with an option to switch to entropy
- ✓ Scikit-Learn has a method to export the tree to a dot file which can be visualized easily using GraphViz.
- ✓ Scikit-Learn can handle continuous features in a classification framework by breaking into discrete sub-intervals.
- ✓ Random guess in the known labels are made if it cannot make a prediction due to invalid walk in the built tree

b. Weka Decision Tree:

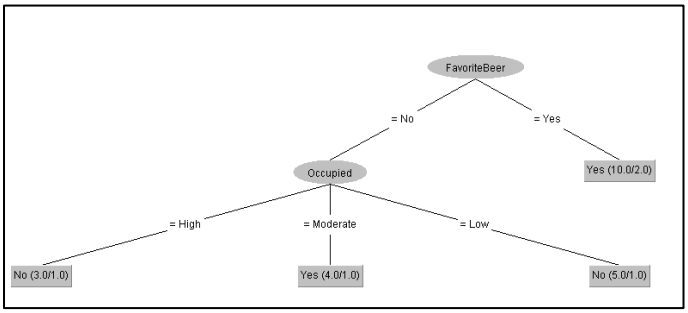
Using the free software Weka and providing input as the given dataset, a similar tree is obtained compared to my code implementing ID3 algorithm. I train different trees using cross-validation of the training data (different folds) and picking the best tree.

```
=== Classifier model (full training set) ==

Id3


Occupied = High
|  Location = Talpiot: No
|  Location = City-Center: Yes
|  Location = German-Colony: No
|  Location = Ein-Karem: null
|  Location = Mahane-Yehuda: Yes
Occupied = Moderate
|  Location = Talpiot
|  |  Price = Expensive: null
|  |  Price = Normal: Yes
|  |  Price = Cheap: No
|  Location = City-Center: Yes
|  Location = German-Colony
|  |  VIP = No: No
|  |  VIP = Yes: Yes
|  Location = Ein-Karem: Yes
|  Location = Mahane-Yehuda: Yes
Occupied = Low
|  Location = Talpiot: No
|  Location = City-Center
|  |  Price = Expensive: null
|  |  Price = Normal: No
|  |  Price = Cheap: No
|  Location = German-Colony: null
|  Location = Ein-Karem
|  |  Price = Expensive: null
|  |  Price = Normal: No
|  |  Price = Cheap: Yes
|  Location = Mahane-Yehuda: No
```

An improvement can be made if I utilize the J48 algorithm, which is based on C4.5 algorithm. The new features (versus ID3) are: (i) accepts both continuous and discrete features; (ii) handles incomplete data points; (iii) solves over-fitting problem by (very clever) bottom-up technique usually known as "pruning"; and (iv) different weights can be applied the features that comprise the training data. The result is a tree with simpler structure:



3. Applications

The biggest advantage of decision-trees is the simplicity of the hypothesis and the interpretability of the tree structure with very little explanation needed. It can be constructed with little training data and easily combined with other decision techniques. Decision tree has become increasingly powerful to unearth existing trends in the data itself instead of making predictions of unseen data.

However, there are disadvantages associated with decision tree method. It is unstable, meaning a small perturbation in the training data may cause a large change in the structure of the optimal tree. Decision tree works better with a random forest of decision trees. In addition, information gain calculation is biased to favor categorical variables with many levels.

There are many applications of decision tree:

- Manufacturing: chemical material evaluation
- Production: process optimization for saving time and labor
- Astronomy: filtering noise from Hubble Space Telescope images
- Pharmacology: analysis of drug efficacy
- Medicine: analysis of SIDS
- Planning: optimal scheduling of assembly line