Programming Assignment 1: Decision Trees

YiLi Chen

Part 1: Implementation

(a)

- All imported library

```
1. import pandas as pd
2. from math import log
```

- Calculate the entropy of the whole dataset

```
1. def get_orginal_entropy(df, target):
2.     orginal_e = 0
3.     # The probs of enjoy and not enjoy
4.     probs = df.groupby(target).size().div(len(df))
5.     for p in probs:
6.         orginal_e += -p*log(p, 2)
7.     return orginal_e
```

- Calculate the weighted entropy

```
1. def get_feature_entropy(col_name, df, target): #Get entropy of each branch
2.     feature_e = 0
3.     # The probs of each categorized group
4.     probs = df.groupby(col_name).size().div(len(df))
5.     # iterate all groups and sum the entropy
6.     for index_value, p_value in list(probs.items()):
7.         # calculate each group probs
8.         group_df = df[df[col_name] == index_value]
9.         group_probs = group_df.groupby(target).size().div(len(group_df))
10.        group_entropy=0 # remove the previous group's value
11.        # Sum H(C|when x=xi)
12.        for p in group_probs:
13.            group_entropy += -p*log(p,2)
14.        # Sum H(target|col_name)
15.        feature_e += p_value * group_entropy
16.    return feature_e
```

- Get the feature which best splits the dataset by the maximum of information gain

```
def get_max_info_gain(df, target):
    d = {}
    orginal_e = get_orginal_entropy(df, target)
    all_col_name = list(df.columns)
    all_col_name.pop(-1)
    for col_name in all_col_name:
        d[col_name] = orginal_e - get_feature_entropy(col_name, df, target)
    return max(d, key=d.get)
```

- Split the dataset along the value of the feature with the largest information gain and create a dictionary to store the sub dataset

```
def split_df(df, best_feature):
    splitDict = {}
    for k in df[best_feature].unique():
        splitDict[k] = df[df[best_feature]==k].drop(columns=best_feature)
    return splitDict
```

- Create the decision tree and define the stopping criteria

```
def create_decision_tree(df, target):
    # if the original_e less than threshold
    if get_orginal_entropy(df, target) < 0.01:
        return df.iloc[:,-1].value_counts().index[0]
    # if left only one type of feature, then return
    if len(df.columns) == 1:
        last_col_name = df.columns[0]
        return df[last_col_name].value_counts().index[0]
    # find max info gain of feature
    fname = get_max_info_gain(df, target)
    node = {fname: {}}
    #build tree
    df2 = split_df(df, fname)
    for t,d in df2.items():
        node[fname][t] = create_decision_tree(d, d.iloc[:,-1])
    return node
```

(b)

- Read the data as the data structure of DataFrame()

```
1.  df = pd.read_csv('dt_data.txt', sep=",", header=None)
2.  df = df.drop([0])
3.  df.columns = ['Occupied', 'Price', 'Music', 'Location', 'VIP', 'Favorite Beer', 'En
    joy']
4.  df = df.reset_index()
5.  df = df.drop(columns=['index'])
6.  for i in range(22):
7.      df['Occupied'][i] = df['Occupied'][i].lstrip('0123456789: ')
8.      df['Enjoy'][i] = df['Enjoy'][i].rstrip(';').lstrip(' ')
9.      for j in range(6):
10.         df.iloc[:,j][i] = df.iloc[:,j][i].lstrip(' ')
```

- Building a model based on the given data

```
1.  tree_model = create_decision_tree(df, df['Enjoy'])
```

```
{'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': {'Favorite Beer': {'No': 'No'}}}}}}}},
     'Talpiot': 'No',
     'Mahane-Yehuda': 'No'}}}}
```

(c)

- The function of predicting a new data

```
def tree_predict(tree_model, feature_names, data):
    key = list(tree_model.keys())[0]
    tree_model = tree_model[key]
    pred=None
    for k in tree_model:
        # find the corresponding attribute
        if data[key][0] == k:
            # judge is it dict or not
            if isinstance(tree_model[k], dict):
                # if yes, keep finding in the next dict inside
                pred = tree_predict(tree_model[k], feature_names, data)
            else:
                # if no, return the label
                pred = tree_model[k]
    return pred
```

- The new input data

```
data = pd.DataFrame(
    data = [['Moderate','Cheap','Loud','City-Center','No','No']],
    columns=['Occupied', 'Price', 'Music', 'Location', 'VIP', 'Favorite Beer'])
```

	Occupied	Price	Music	Location	VIP	Favorite Beer
0	Moderate	Cheap	Loud	City-Center	No	No

- The result of prediction

```
tree_predict(tree_model, list(data.columns), data)
```

The result of prediction for new input data (occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No) is “Yes”.

The first challenge I faced is that I not really familiar the background knowledge of information gain, not to mention calculate it in programming language. But after I write all the formula of entropy on paper and try to get the information gain of simple example, I finally finished the function of getting the attribute with maximum information gain. Additionally, the process of building the decision tree model with recursive call. I have to specify all the sub dataset carefully, especially when setting the stopping criteria. The function needs to return when there is only the entropy of data is less than the threshold and when there is only one type of attribute in columns left.