```
# programmer - Sophia Quinton
# date - 12-1-21
# class - DSC -540
# assignment - Assignment 5

#libraries
import pandas as pd
import numpy as np
import statsmodels.tools.tools as stattools
from tabulate import tabulate
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.ensemble import RandomForestClassifier

#convert table
Temperature = ['hot', 'hot', 'hot', 'mild', 'cool', 'cool', 'cool', 'mild',
'cool', 'mild', 'mild', 'mild', 'hot', 'mild']
Wind = ['weak', 'strong', 'weak', 'weak', 'weak', 'strong', 'strong', 'weak',
'weak', 'weak', 'strong', 'strong', 'weak', 'strong']
Traffic = ['long', 'long', 'long', 'long', 'short', 'short', 'short', 'long',
'short', 'short', 'short', 'long', 'short', 'long']
Driving = ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'yes',
'yes', 'yes', 'yes', 'no']

table_driving = pd.DataFrame({'Temperature_X1': Temperature,
                    'Wind_X2': Wind,
                    'Traffic-Jam_X3': Traffic,
                    'Car_Driving_y': Driving})


table_driving.head()
```

| | Temperature_X1 | Wind_X2 | Traffic-Jam_X3 | Car_Driving_y |
|---|---|---|---|---|
| 0 | hot | weak | long | no |
| 1 | hot | strong | long | no |
| 2 | hot | weak | long | yes |
| 3 | mild | weak | long | yes |
| 4 | cool | weak | short | yes |

```
#calculate information gain (AskPython, nd)
##temperature
H_temp_hot = -1 * ((2/4) * np.log2((2/4)) + (2/4) * np.log2((2/4)))
H_temp_mild = -1 * ((4/6) * np.log2((4/6)) + (2/6) * np.log2((2/6)))
H_temp_cool = -1 * ((3/4) * np.log2((3/4)) + (1/4) * np.log2((1/4)))
Net_Entropy = (4/14) * H_temp_hot + (6/14) * H_temp_mild + (4/14) *
H_temp_cool
Total_Reduction = 1-Net_Entropy
print("The information gain for temperature is: ", Total_Reduction)

The information gain for temperature is:  0.08893660698832373
```

```
##wind
H_wind_weak = -1 * ((6/8) * np.log2((6/8)) + (2/8) * np.log2((2/8)))
H_wind_strong = -1 * ((3/6) * np.log2((3/6)) + (3/6) * np.log2((3/6)))
Net_Entropy_wind = (8/14) * H_wind_weak + (6/14) * H_wind_strong
Total_Reduction_Wind = 1 - Net_Entropy_wind
print("The information gain for wind is: ", Total_Reduction_Wind)
```

```
The information gain for wind is:  0.10784107173763835
```

```
##traffic
H_traffic_long = -1 * ((3/7) * np.log2((3/7)) + (4/7) * np.log2((4/7)))
H_traffic_short = -1 * ((6/7) * np.log2((6/7)) + (1/7) * np.log2((1/7)))
Net_Entropy_traffic = (7/14) * H_traffic_long + (7/14) * H_traffic_short
Total_Reduction_traffic = 1 - Net_Entropy_traffic
print("The information gain for traffic is: ", Total_Reduction_traffic)
```

```
The information gain for traffic is:  0.21154954269171045
```

```
##categorize the dataset (Larose & Larose, 2019)
temp_np = np.array(table_driving['Temperature_X1'])
(temp_cat, temp_cat_dict) = stattools.categorical(temp_np, drop = True,
dictnames = True)
temp_cat_pd = pd.DataFrame(temp_cat)
X = temp_cat_pd

wind_np = np.array(table_driving['Wind_X2'])
(wind_cat, wind_cat_dict) = stattools.categorical(wind_np, drop = True,
dictnames = True)
wind_cat_pd = pd.DataFrame(wind_cat)
X = pd.concat((X, wind_cat_pd), axis=1)

traffic_np = np.array(table_driving['Traffic-Jam_X3'])
(traffic_cat, traffic_cat_dict) = stattools.categorical(traffic_np, drop =
True, dictnames = True)
traffic_cat_pd = pd.DataFrame(traffic_cat)
X = pd.concat((X, traffic_cat_pd), axis=1)

X_name = ['hot', 'cold', 'mild', 'weak', 'strong', 'long', 'short']
X.columns = X_name

drive_np = np.array(table_driving['Car_Driving_y'])
(drive_cat, drive_cat_dict) = stattools.categorical(drive_np, drop = True,
dictnames = True)
drive_cat_pd = pd.DataFrame(drive_cat)
Y = drive_cat_pd
Y_name = ['F', 'T']
Y.columns = ['F', 'T']
```

```
C:\Users\sophi\AppData\Local\Programs\Python\Python39\lib\site-
packages\statsmodels\tools\tools.py:158: FutureWarning: categorical is
deprecated. Use pandas Categorical to represent categorical data and can
```

```
get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
```

```
##short C5 tree
C5_1 = DecisionTreeClassifier(criterion="entropy", splitter="best",
max_depth=1).fit(X,Y)
export_graphviz(C5_1, out_file = "C:/Users/sophi/OneDrive/Desktop/Graduate
Classes/DSC - 540 Machine Learning/Week 5/C5_1.dot", feature_names=X_name,
class_names=Y_name)
```

```
###(graphviz, n.d.)
with open("C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540 Machine
Learning/Week 5/C5_1.dot") as f:
    dotC5_graph = f.read()
gtrainC5 = graphviz.Source(dotC5_graph)
gtrainC5
```

```
gtrainC5.render("C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540
Machine Learning/Week 5/C5_1.jpg", view=True)
```

```
'C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540 Machine
Learning/Week 5\\C5_1.jpg.pdf'
```

```
##full C5 tree
C5_2 = DecisionTreeClassifier(criterion="entropy").fit(X,Y)
export_graphviz(C5_2, out_file = "C:/Users/sophi/OneDrive/Desktop/Graduate
Classes/DSC - 540 Machine Learning/Week 5/C5_2.dot", feature_names=X_name,
class_names=Y_name)
```

```
###(graphviz, n.d.)
with open("C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540 Machine
Learning/Week 5/C5_2.dot") as f:
    dotC5_graph = f.read()
gtrainC5 = graphviz.Source(dotC5_graph)
gtrainC5
```

```
gtrainC5.render("C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540
Machine Learning/Week 5/C5_2.jpg", view=True)
```

```
'C:/Users/sophi/OneDrive/Desktop/Graduate Classes/DSC - 540 Machine
Learning/Week 5\\C5_2.jpg.pdf'
```

```
C5_2
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
#Part 2 Fuzzy decision
print("Net Entropy for temp: ", Net_Entropy)
print("Net Entropy for wind: ", Net_Entropy_wind)
print("Net Entropy for traffic: ", Net_Entropy_traffic)
```

```
Net Entropy for temp:  0.9110633930116763
Net Entropy for wind:  0.8921589282623617
Net Entropy for traffic:  0.7884504573082896

print("The information gain for temperature is: ", Total_Reduction)
print("The information gain for wind is: ", Total_Reduction_Wind)
print("The information gain for traffic is: ", Total_Reduction_traffic)

The information gain for temperature is:  0.08893660698832373
The information gain for wind is:  0.10784107173763835
The information gain for traffic is:  0.21154954269171045

##create a fuzzy tree by scratch
##using the above info gains, the traffic amount has the highest gain: this
is the head node
##next select sub nodes
short_traffic_table = table_driving[table_driving['Traffic-Jam_X3'] ==
"short"]
short_traffic_table
H_temp_cool = -1 * ((3/4) * np.log2(3/4) + (1/4) * np.log2(1/4))
Net_Entropy_temp_2 = (4/7) * H_temp_cool
Total_Reduction_temp_2 = 1 - Net_Entropy_temp_2
print(Total_Reduction_temp_2)

0.536412500309067

H_wind_strong = -1 * ((2/3) * np.log2(2/3) + (1/3) * np.log2(1/3))
Net_Entropy_wind_2 = (3/7) * H_wind_strong
Total_Reduction_wind_2 = 1-Net_Entropy_wind_2
print(Total_Reduction_wind_2)

0.606444642548076

long_traffic_table = table_driving[table_driving['Traffic-Jam_X3'] == "long"]
long_traffic_table
H_temp_hot = -1 * ((1/3) * np.log2(1/3) + (2/3) * np.log2(2/3))
H_temp_mild = -1 * ((2/4) * np.log2(2/4) + (2/4) * np.log2(2/4))
Net_Entropy_temp_3 = (4/7) * H_temp_mild + (3/7) * H_temp_hot
Total_Reduction_temp_3 = 1-Net_Entropy_temp_3
print(Total_Reduction_temp_3)

0.035016071119504555

h_wind_weak = -1 * ((2/4) * np.log2(2/4) + (2/4) * np.log2(2/4))
h_wind_strong = -1 * ((1/3) * np.log2(1/3) + (2/3) * np.log2(2/3))
Net_Entropy_wind_3 = (4/7) * h_wind_weak + (3/7) * h_wind_strong
Total_Reduction_wind_3 = 1 - Net_Entropy_wind_3
print(Total_Reduction_wind_3)

0.035016071119504555

short_traffic_table
```

```
   Temperature_X1 Wind_X2 Traffic-Jam_X3 Car_Driving_y
4            cool    weak          short           yes
5            cool  strong          short            no
6            cool  strong          short           yes
8            cool    weak          short           yes
9            mild    weak          short           yes
10           mild  strong          short           yes
12            hot    weak          short           yes

long_traffic_table

   Temperature_X1 Wind_X2 Traffic-Jam_X3 Car_Driving_y
0             hot    weak           long            no
1             hot  strong           long            no
2             hot    weak           long           yes
3            mild    weak           long           yes
7            mild    weak           long            no
11           mild  strong           long           yes
13           mild  strong           long            no
```

```
#pydot (Carrera, nd)
import pydot
graph = pydot.Dot('fuzzy_tree', graph_type='graph', bgcolor='yellow')

x = pydot.Node("First", label = "Traffic = short")
graph.add_node(x)
x = pydot.Node("Second True", label = "True; wind weak?")
graph.add_node(x)
x = pydot.Node("Third True hot", label = "True; Temperature hot?")
graph.add_node(x)
x = pydot.Node("Third True mild", label = "True; Temperature mild?")
graph.add_node(x)
x = pydot.Node("Third True cool", label = "True; Temperature cold?")
graph.add_node(x)
x = pydot.Node("Third False hot", label = "False; Temperature hot?")
graph.add_node(x)
x = pydot.Node("Third False mild", label = "False; Temperature mild?")
graph.add_node(x)
x = pydot.Node("Third False cool", label = "False; Temperature cool?")
graph.add_node(x)

y = pydot.Node("Second False", label = "False; wind weak?")
graph.add_node(y)
y = pydot.Node("F - Third True hot", label = "True; Temperature hot?")
graph.add_node(y)
y = pydot.Node("F - Third True mild", label = "True; Temperature mild?")
graph.add_node(y)
y = pydot.Node("F - Third True cool", label = "True; Temperature cold?")
graph.add_node(y)
y = pydot.Node("F - Third False hot", label = "False; Temperature hot?")
```

```
graph.add_node(y)
y = pydot.Node("F - Third False mild", label = "False; Temperature mild?")
graph.add_node(y)
y = pydot.Node("F - Third False cool", label = "False; Temperature cool?")
graph.add_node(y)

z = pydot.Node("Drive Car", label = "drive")
graph.add_node(z)
z = pydot.Node("Dont drive", label = "dont drive")
graph.add_node(z)

edge = pydot.Edge("First", "Second True")
graph.add_edge(edge)
edge = pydot.Edge("First", "Second False")
graph.add_edge(edge)

edge = pydot.Edge("Second True", "Third True hot")
graph.add_edge(edge)
edge = pydot.Edge("Second True", "Third True mild")
graph.add_edge(edge)
edge = pydot.Edge("Second True", "Third True cool")
graph.add_edge(edge)
edge = pydot.Edge("Second True", "Third False hot")
graph.add_edge(edge)
edge = pydot.Edge("Second True", "Third False mild")
graph.add_edge(edge)
edge = pydot.Edge("Second True", "Third False cool")
graph.add_edge(edge)

edge = pydot.Edge("Second False", "F - Third True hot")
graph.add_edge(edge)
edge = pydot.Edge("Second False", "F - Third True mild")
graph.add_edge(edge)
edge = pydot.Edge("Second False", "F - Third True cool")
graph.add_edge(edge)
edge = pydot.Edge("Second False", "F - Third False hot")
graph.add_edge(edge)
edge = pydot.Edge("Second False", "F - Third False mild")
graph.add_edge(edge)
edge = pydot.Edge("Second False", "F - Third False cool")
graph.add_edge(edge)

edge = pydot.Edge("Third True hot", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("Third True mild", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("Third True cool", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("Third False mild", "Drive Car")
```

```
graph.add_edge(edge)
edge = pydot.Edge("Third True cool", "Drive Car")
graph.add_edge(edge)

edge = pydot.Edge("F - Third True hot", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("F - Third True hot", "Dont drive")
graph.add_edge(edge)
edge = pydot.Edge("F - Third True mild", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("F - Third True mild", "Dont drive")
graph.add_edge(edge)

edge = pydot.Edge("F - Third False hot", "Dont drive")
graph.add_edge(edge)
edge = pydot.Edge("F - Third False mild", "Drive Car")
graph.add_edge(edge)
edge = pydot.Edge("F - Third False mild", "Dont drive")
graph.add_edge(edge)

graph.write_png("graph.png")

#choose root node with decision partial trees
##split data
from sklearn.model_selection import train_test_split
driving_train, driving_test = train_test_split(table_driving, test_size=0.2,
random_state=25)
driving_train.reset_index(level=0, inplace=True)
driving_test.reset_index(level=0, inplace=True)
print("length of train: ", len(driving_train))
print("length of test: ", len(driving_test))
```

```
length of train:  11
length of test:  3
```

```
driving_train
```

| | index | Temperature_X1 | Wind_X2 | Traffic-Jam_X3 | Car_Driving_y |
|---|---|---|---|---|---|
| 0 | 5 | cool | strong | short | no |
| 1 | 11 | mild | strong | long | yes |
| 2 | 1 | hot | strong | long | no |
| 3 | 12 | hot | weak | short | yes |
| 4 | 13 | mild | strong | long | no |
| 5 | 8 | cool | weak | short | yes |
| 6 | 2 | hot | weak | long | yes |
| 7 | 7 | mild | weak | long | no |
| 8 | 6 | cool | strong | short | yes |
| 9 | 10 | mild | strong | short | yes |
| 10 | 4 | cool | weak | short | yes |

```
driving_test
```

```
   index Temperature_X1 Wind_X2 Traffic-Jam_X3 Car_Driving_y
0      0            hot    weak           long            no
1      9           mild    weak          short           yes
2      3           mild    weak           long           yes
```

```
temp_np = np.array(driving_train['Temperature_X1'])
(temp_cat, temp_cat_dict) = stattools.categorical(temp_np, drop = True,
dictnames = True)
temp_cat_pd = pd.DataFrame(temp_cat)
X = temp_cat_pd

wind_np = np.array(driving_train['Wind_X2'])
(wind_cat, wind_cat_dict) = stattools.categorical(wind_np, drop = True,
dictnames = True)
wind_cat_pd = pd.DataFrame(wind_cat)
X = pd.concat((X, wind_cat_pd), axis=1)

traffic_np = np.array(driving_train['Traffic-Jam_X3'])
(traffic_cat, traffic_cat_dict) = stattools.categorical(traffic_np, drop =
True, dictnames = True)
traffic_cat_pd = pd.DataFrame(traffic_cat)
X = pd.concat((X, traffic_cat_pd), axis=1)

X_name = ['cold', 'hot', 'mild', 'strong', 'weak', 'long', 'short']
X.columns = X_name

drive_np = np.array(driving_train['Car_Driving_y'])
(drive_cat, drive_cat_dict) = stattools.categorical(drive_np, drop = True,
dictnames = True)
drive_cat_pd = pd.DataFrame(drive_cat)
Y = drive_cat_pd
Y_name = ['F', 'T']
Y.columns = Y_name
```

```
C:\Users\sophi\AppData\Local\Programs\Python\Python39\lib\site-
packages\statsmodels\tools\tools.py:158: FutureWarning: categorical is
deprecated. Use pandas Categorical to represent categorical data and can
get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
```

```
cold_test =pd.DataFrame([0,0,0])
temp_np_test = np.array(driving_test['Temperature_X1'])
(temp_cat_test, temp_cat_dict_test) = stattools.categorical(temp_np_test,
drop = True, dictnames = True)
temp_cat_pd_test = pd.DataFrame(temp_cat_test)
X_test = pd.concat((cold_test, temp_cat_pd_test), axis=1)

strong_test = pd.DataFrame([0,0,0])
X_test = pd.concat((X_test, strong_test), axis=1)
wind_np_test = np.array(driving_test['Wind_X2'])
```

```
(wind_cat_test, wind_cat_dict_test) = stattools.categorical(wind_np_test,
drop = True, dictnames = True)
wind_cat_pd_test = pd.DataFrame(wind_cat_test)
X_test = pd.concat((X_test, wind_cat_pd_test), axis=1)

traffic_np_test = np.array(driving_test['Traffic-Jam_X3'])
(traffic_cat_test, traffic_cat_dict_test) =
stattools.categorical(traffic_np_test, drop = True, dictnames = True)
traffic_cat_pd_test = pd.DataFrame(traffic_cat_test)
X_test = pd.concat((X_test, traffic_cat_pd_test), axis=1)

X_name = ['cold', 'hot', 'mild', 'strong', 'weak', 'long', 'short']
X_test.columns = X_name

drive_np_test = np.array(driving_test['Car_Driving_y'])
(drive_cat_test, drive_cat_dict_test) = stattools.categorical(drive_np_test,
drop = True, dictnames = True)
drive_cat_pd_test = pd.DataFrame(drive_cat_test)
Y_test = drive_cat_pd_test
Y_name = ['F', 'T']
Y_test.columns = Y_name
```

```
C:\Users\sophi\AppData\Local\Programs\Python\Python39\lib\site-
packages\statsmodels\tools\tools.py:158: FutureWarning: categorical is
deprecated. Use pandas Categorical to represent categorical data and can
get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
```

```
##test train
C5_train = DecisionTreeClassifier(criterion="entropy").fit(X,Y)
ypred = C5_train.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_test = accuracy_score(Y_test, ypred)
print("accuracy: ", accuracy_test)
```

```
accuracy:  0.3333333333333333
```