```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import numpy

import pandas
from sklearn.model_selection import train_test_split
from itertools import combinations

# Number 4
a = pandas.read_csv("/Users/gyucheonheo/Downloads/claim_history.csv")

# a) Please provide the frequency table (i.e., counts and proportions) of the
target field in the Training partition?

train, test = train_test_split(a, test_size = 0.3, random_state = 60616,
stratify = a['CAR_USE'])

train_freq = pandas.DataFrame(train.groupby('CAR_USE').size(), columns =
['Count'])
train_freq['Proportion'] = train_freq / train.shape[0]

print(train_freq)



# b)  What is the probability that an observation will be assigned to the Test
partition given that CAR_USE is Private?

test_freq = pandas.DataFrame(test.groupby('CAR_USE').size(), columns =
['Count'])
test_freq['Proportion'] = test_freq / test.shape[0]
print(test_freq)
# Calculate the Prob(CAR_USE | Partition) * Prob(Partition)

p_train = train_freq['Proportion'] * (train.shape[0]/a.shape[0])
p_test = test_freq['Proportion'] * (test.shape[0]/a.shape[0])

# Calculate the Prob(CAR_USE | Training) * Prob(Training) + Prob(CAR_USE | Test)
* Prob(Test)

answer = p_test / (p_train + p_test)


print("The probability of when car is priviate", answer['Private'])

# What is the root node entropy of the given data set?
import math
import itertools


print(train['CAR_USE'])
total = train['CAR_USE']
private = train.loc[train['CAR_USE'] == 'Private']
commercial = train.loc[train['CAR_USE'] == 'Commercial']


root_entropy = (-private.shape[0] / total.shape[0])*math.log2(private.shape[0] /
total.shape[0]) - (commercial.shape[0] /
total.shape[0])*math.log2(commercial.shape[0] / total.shape[0])
```

```
print(root_entropy)


ch_ds = pandas.read_csv("/Users/gyucheonheo/Downloads/claim_history.csv")

cross_tab = pandas.crosstab(ch_ds['EDUCATION'], ch_ds['CAR_USE'], margins=False)

print(cross_tab)

# Reference : https://stackoverflow.com/questions/58908934/find-all-
combinations-to-split-a-single-list-into-two-lists
def two_partitions(S):
    res_list = []
    for l in range(1,int(len(S)/2)+1):
        combis = set(itertools.combinations(S,l))
        for c in combis:
            res_list.append((sorted(list(c)), sorted(list(S-set(c)))))
    return res_list



print(cross_tab.shape[0])
print(cross_tab.shape[1])
def getEntropies(two_p, index):
    entropies = {}
    for p in two_p:
        left = p[0]
        right = p[1]
        left_commercial_sum = 0
        left_private_sum = 0
        left_total = 0
        right_commercial_sum = 0
        right_private_sum = 0
        right_total = 0
        for n,k in enumerate(left):
            left_commercial_sum +=cross_tab.iloc[index[k]]['Commercial']
            left_private_sum += cross_tab.iloc[index[k]]['Private']
        left_total = left_commercial_sum + left_private_sum
        for n,k in enumerate(right):
            right_commercial_sum +=cross_tab.iloc[index[k]]['Commercial']
            right_private_sum += cross_tab.iloc[index[k]]['Private']
        right_total = right_commercial_sum + right_private_sum
        if(left_commercial_sum == 0 or left_private_sum == 0):
            left_entropy = 0
        else:
            left_entropy = -
((left_commercial_sum/left_total)*math.log2(left_commercial_sum/left_total) +
(left_private_sum/left_total)*math.log2(left_private_sum/left_total))

        if(right_commercial_sum == 0 or right_private_sum ==0 ):
            right_entropy = 0
        else:
            right_entropy = -
((right_commercial_sum/right_total)*math.log2(right_commercial_sum/right_total)
+ (right_private_sum/right_total)*math.log2(right_private_sum/right_total))

        entropies[str(p)] = (left_total)/(left_total + right_total) *
left_entropy + (right_total)/(left_total+right_total)*right_entropy
    return entropies
cross_tab = pandas.crosstab(ch_ds['OCCUPATION'], ch_ds['CAR_USE'],
margins=False)
index = { "Blue Collar" : 0, "Clerical" :1, "Doctor":2, "Home Maker":3,
"Lawyer":4, "Manager": 5, "Professional": 6, "Student": 7, "Unknown": 8}
two_p = two_partitions({"Blue Collar", "Clerical", "Doctor", "Home Maker",
```

```
"Lawyer", "Manager", "Professional", "Student", "Unknown"})
occ_entropies = getEntropies(two_p, index)


cross_tab = pandas.crosstab(ch_ds['EDUCATION'], ch_ds['CAR_USE'], margins=False)
index = {
    'Below High School':1,
    'High School':3,
    'Bachelors':0,
    'Masters':4,
    'Doctors':2
    }
two_p = two_partitions({"Below High School", "High School", "Bachelors",
"Masters", "Doctors"})
edu_entropies = getEntropies(two_p, index)


cross_tab = pandas.crosstab(ch_ds['CAR_TYPE'], ch_ds['CAR_USE'], margins=False)
index= {
        'Minivan' : 0,
        'Panel Truck': 1,
        'Pickup' : 2,
        'SUV':3,
        'Sports Car':4,
        'Van' : 5
        }

two_p = two_partitions({"Minivan", "Panel Truck", "Pickup", "SUV", "Sports Car",
"Van"})
car_type_entropies = getEntropies(two_p, index)


# Q5 (b)

print(min(occ_entropies.values()), min(edu_entropies.values()),
min(car_type_entropies.values()))

for key, value in occ_entropies.items():
    if(value == min(occ_entropies.values())):
        print(key, value)


# Q5 (c)

train_left= ch_ds[ch_ds["OCCUPATION"].isin({'Blue Collar', 'Student',
'Unknown'})]
cross_tab = pandas.crosstab(train_left['OCCUPATION'], train_left['CAR_USE'],
margins=False)

index = { "Blue Collar" : 0,
         "Student": 1,
         "Unknown":2}
two_p = two_partitions({"Blue Collar", "Student", "Unknown"})
occ_entropies = getEntropies(two_p, index)

cross_tab = pandas.crosstab(train_left['EDUCATION'], train_left['CAR_USE'],
margins=False)

index = {
    'Below High School':1,
    'High School':3,
    'Bachelors':0,
    'Masters':4,
    'Doctors':2
```

```
    }
two_p = two_partitions({"Below High School", "High School", "Bachelors",
"Masters", "Doctors"})
edu_entropies = getEntropies(two_p, index)

cross_tab = pandas.crosstab(train_left['CAR_TYPE'], train_left['CAR_USE'],
margins=False)
index= {
        'Minivan' : 0,
        'Panel Truck': 1,
        'Pickup' : 2,
        'SUV':3,
        'Sports Car':4,
        'Van' : 5
        }

two_p = two_partitions({"Minivan", "Panel Truck", "Pickup", "SUV", "Sports Car",
"Van"})
car_type_entropies = getEntropies(two_p, index)
print(min(occ_entropies.values()), min(edu_entropies.values()),
min(car_type_entropies.values()))

for key, value in edu_entropies.items():
    if(value == min(edu_entropies.values())):
        print("Edu", key, value)


train_right= ch_ds[ch_ds["OCCUPATION"].isin({'Clerical', 'Doctor', 'Home Maker',
'Lawyer', 'Manager', 'Professional'})]
cross_tab = pandas.crosstab(train_right['OCCUPATION'], train_right['CAR_USE'],
margins=False)

index = { "Blue Collar" : 0,
         "Student": 1,
         "Unknown":2}
two_p = two_partitions({"Blue Collar", "Student", "Unknown"})
occ_entropies = getEntropies(two_p, index)

cross_tab = pandas.crosstab(train_right['EDUCATION'], train_right['CAR_USE'],
margins=False)

index = {
    'Below High School':1,
    'High School':3,
    'Bachelors':0,
    'Masters':4,
    'Doctors':2
    }
two_p = two_partitions({"Below High School", "High School", "Bachelors",
"Masters", "Doctors"})
edu_entropies = getEntropies(two_p, index)

cross_tab = pandas.crosstab(train_right['CAR_TYPE'], train_right['CAR_USE'],
margins=False)
index= {
        'Minivan' : 0,
        'Panel Truck': 1,
        'Pickup' : 2,
        'SUV':3,
        'Sports Car':4,
        'Van' : 5
        }

two_p = two_partitions({"Minivan", "Panel Truck", "Pickup", "SUV", "Sports Car",
```

```
"Van"})
car_type_entropies = getEntropies(two_p, index)
print(min(occ_entropies.values()), min(edu_entropies.values()),
min(car_type_entropies.values()))

for key, value in car_type_entropies.items():
    if(value == min(car_type_entropies.values())):
        print("car type", key, value)


train_left['EDUCATION'] = train_left['EDUCATION'].map({
    'Below High School': 0,
    'High School':1,
    'Bachelors':2,
    'Masters':3,
    'Doctors':4
    })


ll = train_left[train_left['EDUCATION'] >= 2.5]
t = ll.shape[0]
commercial = ll[ll['CAR_USE'] == 'Commercial'].shape[0]
private = ll[ll['CAR_USE'] == 'Private'].shape[0]
com = commercial / t
pri = private / t
entropy = -((com * math.log2(com)) + (pri * math.log2(pri)))
class_name = 'Commercial' if com > pri else 'Private'
print("leave0",entropy, class_name)

rl = train_left[train_left['EDUCATION'] < 2.5]
t = rl.shape[0]
commercial = rl[rl['CAR_USE'] == 'Commercial'].shape[0]
private = rl[rl['CAR_USE'] == 'Private'].shape[0]
com = commercial / t
pri = private / t
entropy = -((com * math.log2(com)) + (pri * math.log2(pri)))
class_name = 'Commercial' if com > pri else 'Private'
print("leave1",entropy, class_name)
lr = train_right[
    train_right['CAR_TYPE'].isin({'Minivan', 'SUV', 'Sports Car'})]
t = lr.shape[0]
commercial = lr[lr['CAR_USE'] == 'Commercial'].shape[0]
private = lr[lr['CAR_USE'] == 'Private'].shape[0]
com =  commercial/ t
pri = private / t
entropy = -((com * math.log2(com)) + (pri * math.log2(pri)))
class_name = 'Commercial' if com > pri else 'Private'
print("leave2",entropy, class_name)

rr = train_right[
    train_right['CAR_TYPE'].isin({'Panel Truck', 'Pickup', 'Van'})]
t = rr.shape[0]
commercial = rr[rr['CAR_USE'] == 'Commercial'].shape[0]
private = rr[rr['CAR_USE'] == 'Private'].shape[0]
com = commercial / t
pri = private / t
entropy = -((com * math.log2(com)) + (pri * math.log2(pri)))
class_name = 'Commercial' if com > pri else 'Private'
print("leave\3",entropy, class_name)

a = pandas.read_csv("/Users/gyucheonheo/Downloads/claim_history.csv")

threshold = a.groupby("CAR_USE").size()["Commercial"]/a.shape[0]
```

```
print("threshold", threshold)

a = pandas.read_csv("~/Downloads/claim_history.csv")
a = a[["CAR_TYPE", "OCCUPATION", "EDUCATION", "CAR_USE"]].dropna()

train, test = train_test_split(a, test_size=0.3, random_state=60616)
train_data = train
test_data = test
traindx = train_data[["CAR_TYPE", "OCCUPATION", "EDUCATION"]]
traindy= train_data['CAR_USE']
testdx= test_data[["CAR_TYPE", "OCCUPATION", "EDUCATION"]]
testdy= test_data["CAR_USE"]

testdx['EDUCATION'] = testdx['EDUCATION'].map(
    {'Below High School': 0, 'High School': 1, 'Bachelors': 2, 'Masters': 3,
'Doctors': 4})

ppy = numpy.ndarray(shape=(len(testdx), 2), dtype=float)
counter = 0
for index, row in testdx.iterrows():
    if row['OCCUPATION'] in ('Blue Collar', 'Student', 'Unknown'):
        if row['EDUCATION'] <= 0.5:
            probability = [0.6832518880497557, 0.3167481119502443]
        else:
            probability = [0.8912579957356077, 0.10874200426439233]
    else:
        if row['CAR_TYPE'] in ('Minivan', 'SUV', 'Sports Car'):
            probability = [0.006838669567920423, 0.9931613304320795]
        else:
            probability = [0.5306122448979592, 0.46938775510204084]
    ppy[counter] = probability
    counter += 1

ppy = ppy[:, 0]

ty = testdy
num_y = ty.shape[0]

pred_y = numpy.empty_like(ty)
for i in range(num_y):
    if ppy[i] <= threshold:
        pred_y[i] = 'Private'
    else:
        pred_y[i] = 'Commercial'

import sklearn.metrics as metrics
accuracy = metrics.accuracy_score(ty, pred_y)
misclassification_rate = 1 - accuracy
print('Accuracy', accuracy, 'Misclassification Rate', misclassification_rate)

RASE = 0.0
for target, pred zip(ty, pred_prob_y):
    v = 1 if target =='Commercial' else 0
    RASE = RASE + ( v - pred) **2
RASE = numpy.sqrt(RASE / num_y)
print('RASE', RASE)

nConcordant = 0.0
nDiscordant = 0.0
nTied = 0.0

ppE_oE, freq_pE_oE = numpy.unique(pred_prob_y[target_y == 'Commercial'],
return_counts = True)
ppE_oNE, freq_pE_oNE = numpy.unique(pred_prob_y[target_y == 'Private'],
```

```
return_counts = True)



for eProb, eFreq in zip(ppE_oE, freq_pE_oE):
    _thisC = eFreq * numpy.sum(freq_pE_oNE[ppE_oNE < eProb])
    _thisD = eFreq * numpy.sum(freq_pE_oNE[ppE_oNE > eProb])
    _thisT = eFreq * numpy.sum(freq_pE_oNE[ppE_oNE == eProb])
    nConcordant = nConcordant + _thisC
    nDiscordant = nDiscordant + _thisD
    nTied = nTied + _thisT


y_true = 1.0 * numpy.isin(target_y, ['Commercial'])
print("y_true", y)
print("pred_prob_y", pred_prob_y)
AUC = metrics.roc_auc_score(y_true, pred_prob_y)
print('Area Under Curve',AUC)
import matplotlib.pyplot as plt



one_minus_specificity, sensitivity, thresholds = metrics.roc_curve(target_y,
pred_prob_y, pos_label='Commercial')

one_minus_specificity = numpy.append([0], one_minus_specificity)
sensitivity = numpy.append([0], sensitivity)

one_minus_specificity = numpy.append(one_minus_specificity, [1])
sensitivity = numpy.append(sensitivity, [1])

plt.figure(figsize=(6, 6))
plt.plot(one_minus_specificity, sensitivity, marker='x', color='blue',
linestyle='solid', linewidth=2, markersize=6)
plt.plot([0, 1], [0, 1], color='red', linestyle=':')
plt.grid(True)
plt.xlabel("1 - Specificity")
plt.ylabel("Sensitivity ")
ax = plt.gca()
ax.set_aspect('equal')
plt.show()

true_positive = sensitivity

false_positive = one_minus_specificity

# KS_STAT

cutoff = numpy.where(thresholds > 1.0, 1.0, thresholds)
true_positive = list(filter(lambda x : x > 0, true_positive))
false_positive = list(filter(lambda x : x > 0, false_positive))

print(cutoff)
print(true_positive)
plt.plot(cutoff, true_positive, marker = 'o', label = 'True Positive', color =
'blue', linestyle = 'solid')

plt.plot(cutoff, false_positive, marker = 'o', label = 'False Positive', color =
'red', linestyle = 'solid')

plt.grid(True)

plt.xlabel("Probability Threshold")
plt.ylabel("Positive Rate")
```

```
plt.legend(loc = 'upper right', shadow = True)
plt.show()

ks_stat = 0
for i in range(len(true_positive)):
    ks_stat = max(ks_stat, abs(true_positive[i] - false_positive[i]))
    print(i, ks_stat)
print("ks_stat", ks_stat)


gini = 2*AUC-1

print(gini)

gk_gamma = (nConcordant - nDiscordant) / (nConcordant + nDiscordant)

print(gk_gamma)
```