**Name: ____Anthony Sharp___________** **40 points possible**

# CS 5402 – Intro to Data Mining
# Fall 2019
# HW #5

**Submit as a <u>single</u> pdf file via Canvas by 11:59 p.m. on Oct. 26, 2020**

1. Write a **Python** function which, given a dataframe, a rule, and a confidence level (0-100), uses the method discussed in class to determine whether **any conditions in the rule can be dropped without adversely affecting the predicted error rate**. The function should (output and) return the "optimized" version of the rule.

   Assume that a **rule** is specified as a **list [*antecedent*, *consequent*]**, where the *antecedent* and the *consequent* are each lists with entries of the form **[*attributeIndexNumber*, *operatorCode*, *value*]**. The ***attributeIndexNumber*** is a 0-based index number of an attribute in the dataframe. The ***operatorCode*** is: 0 for '<', 1 for '<=', 2 for '>', 3 for '>=', 4 for '==', or 5 for '!='. The ***value*** can be of type integer, float, or string (you can assume that it is appropriate for the attribute's type). The conditions in an *antecedent* or a *consequent* should be treated as **conjunctions** (i.e., *and*'d). **<u>Note</u>**: It's possible for the antecedent list to be empty, but the consequent will never be "empty" in a rule!

   Here's an example of what you have to do. Suppose you have the following rule:

   **If AIT203 ≤ 420 and FIT201 > 0.5 and AIT402 > 250 then P205 == 2**

   If **AIT203** is attribute 0, **FIT201** is attribute 5, **AIT402** is attribute 16, and **P205** is attribute 21 in the dataframe, then the rule would be encoded as:

   **[ [[0, 1, 420], [5, 2, 0.5], [16, 2, 250]], [[21, 4, 2]] ]**

   You would have to count the instances in the dataframe to obtain the **Y1, Y2, E1,** and **E2** values for <u>each</u> condition in the antecedent. For this example, you'd need values for the entries shown in the table below.

| **Condition** | **Y1** | **Y2** | **E1** | **E2** |
|---|---|---|---|---|
| AIT203 <= 420 | … | … | … | … |
| FIT201 > 0.5 | … | … | … | … |
| AIT402 > 250 | … | … | … | … |

   To simplify things a bit, let **A1** be ***AIT203 ≤ 420***, **A2** be ***FIT201 > 0.5***, **A3** be ***AIT402 > 250***, and **C** be ***P205 == 2***.

   **In <u>every</u> row of the table:**
   **Y1** is the number of instances that satisfy A1, A2, A3, and C
   **E1** is the number of instances that satisfy A1, A2, A3, and !C

   **In the 1st row of the table (which is for A1):**
   **Y2** is the number of instances that satisfy !A1, A2, A3, and C
   **E2** is the number of instances that satisfy !A1, A2, A3, and !C

   **In the 2nd row of the table (which is for A2):**
   **Y2** is the number of instances that satisfy A1, !A2, A3, and C
   **E2** is the number of instances that satisfy A1, !A2, A3, and !C

**Name: ____Anthony Sharp___________** **40 points possible**

**In the 3rd row of the table (which is for A3):**
**Y2** is the number of instances that satisfy A1, A2, !A3, and C
**E2** is the number of instances that satisfy A1, A2, !A3, and !C

You then have to compute **$U_{CF}$(E1+E2, Y1+Y2+E1+E2)** for each condition in the antecedent (where **CF** is the specified confidence level) to see what the predicted error rate is if that condition is dropped. If the error rate improves (from what it was before dropping that condition), then you drop that condition. If a condition is dropped, you need to continue trying to drop conditions, until the rule can't be further improved.

Note that this also requires that you compute the **predicted error for the rule before any condition is dropped**. That is computed as **$U_{CF}$(E1, Y1 + E1**), where **Y1** is the number of instances that satisfy **all** the conditions in the antecedent and consequent (e.g., A1, A2, and A3, and C), and **E1** is the number of instances that satisfy **all** the conditions in the antecedent and **not** the condition in the consequent (e.g., A1, A2, A3, and !C).

Test your function by running it on each rule that can be generated from the **breast-cancer dataset** decision tree shown below using a **confidence level of 25% and then a confidence level of 95%**:

```
node-caps = yes
|  deg-malig = 1: recurrence-events
|  deg-malig = 2
|  |  tumor-size = 15-19: no-recurrence-events
|  |  tumor-size = 20-24
|  |  |  age = 30-39: no-recurrence-events
|  |  |  age = 40-49: recurrence-events
|  |  |  age = 50-59: no-recurrence-events
|  |  tumor-size = 25-29: no-recurrence-events
|  |  tumor-size = 30-34
|  |  |  breast-quad = left_low: no-recurrence-events
|  |  |  breast-quad = right_up: no-recurrence-events
|  |  |  breast-quad = right_low: recurrence-events
|  |  |  breast-quad = central: recurrence-events
|  |  tumor-size = 35-39: no-recurrence-events
|  |  tumor-size = 40-44: no-recurrence-events
|  |  tumor-size = 50-54: no-recurrence-events
|  deg-malig = 3
|  |  breast-quad = left_up
|  |  |  age = 40-49
|  |  |  |  irradiat = yes: no-recurrence-events
|  |  |  |  irradiat = no: recurrence-events
|  |  |  age = 50-59: no-recurrence-events
|  |  |  age = 60-69: recurrence-events
|  |  breast-quad = left_low
|  |  |  inv-nodes = 0-2: no-recurrence-events
|  |  |  inv-nodes = 3-5: recurrence-events
|  |  |  inv-nodes = 6-8: recurrence-events
|  |  |  inv-nodes = 9-11: recurrence-events
|  |  |  inv-nodes = 15-17: recurrence-events
|  |  |  inv-nodes = 24-26: recurrence-events
|  |  breast-quad = right_up: recurrence-events
|  |  breast-quad = right_low: recurrence-events
|  |  breast-quad = central: no-recurrence-events
node-caps = no
|  inv-nodes = 0-2
|  |  tumor-size = 0-4: no-recurrence-events
|  |  tumor-size = 5-9: no-recurrence-events
|  |  tumor-size = 10-14: no-recurrence-events
|  |  tumor-size = 15-19: no-recurrence-events
|  |  tumor-size = 20-24
|  |  |  menopause = lt40: recurrence-events
|  |  |  menopause = ge40
|  |  |  |  breast = left: no-recurrence-events
|  |  |  |  breast = right
|  |  |  |  |  age = 40-49: recurrence-events
|  |  |  |  |  age = 50-59: no-recurrence-events
|  |  |  |  |  age = 60-69: recurrence-events
|  |  |  menopause = premeno: no-recurrence-events
|  |  tumor-size = 25-29
|  |  |  deg-malig = 1: no-recurrence-events
|  |  |  deg-malig = 2
|  |  |  |  breast-quad = left_up: no-recurrence-events
|  |  |  |  breast-quad = left_low
|  |  |  |  |  age = 30-39: no-recurrence-events
|  |  |  |  |  age = 40-49: recurrence-events
|  |  |  |  |  age = 50-59: no-recurrence-events
|  |  |  |  |  age = 60-69: no-recurrence-events
|  |  |  |  breast-quad = right_up: recurrence-events
|  |  |  |  breast-quad = right_low: no-recurrence-events
|  |  |  |  breast-quad = central: no-recurrence-events
|  |  |  deg-malig = 3
|  |  |  |  breast = left
|  |  |  |  |  age = 40-49: recurrence-events
|  |  |  |  |  age = 50-59: no-recurrence-events
|  |  |  |  |  age = 60-69: recurrence-events
|  |  |  |  breast = right: no-recurrence-events
|  |  tumor-size = 30-34
|  |  |  deg-malig = 1
|  |  |  |  age = 30-39: recurrence-events
|  |  |  |  age = 40-49: no-recurrence-events
|  |  |  |  age = 50-59: no-recurrence-events
|  |  |  |  age = 60-69: no-recurrence-events
|  |  |  deg-malig = 2: no-recurrence-events
|  |  |  deg-malig = 3
|  |  |  |  irradiat = yes: recurrence-events
|  |  |  |  irradiat = no
|  |  |  |  |  breast-quad = left_up: no-recurrence-events
|  |  |  |  |  breast-quad = left_low: no-recurrence-events
|  |  |  |  |  breast-quad = right_up: recurrence-events
|  |  |  |  |  breast-quad = central: recurrence-events
|  |  tumor-size = 35-39
|  |  |  age = 10-19: no-recurrence-events
|  |  |  age = 20-29: no-recurrence-events
|  |  |  age = 30-39: recurrence-events
|  |  |  age = 40-49: no-recurrence-events
|  |  |  age = 50-59: no-recurrence-events
|  |  tumor-size = 40-44
|  |  |  age = 30-39: no-recurrence-events
|  |  |  age = 40-49: no-recurrence-events
|  |  |  age = 50-59: no-recurrence-events
|  |  |  age = 60-69: recurrence-events
|  |  |  age = 70-79: no-recurrence-events
|  |  tumor-size = 45-49: no-recurrence-events
|  |  tumor-size = 50-54: no-recurrence-events
|  inv-nodes = 3-5
|  |  breast = left: recurrence-events
|  |  breast = right
|  |  |  tumor-size = 10-14: no-recurrence-events
|  |  |  tumor-size = 20-24: no-recurrence-events
|  |  |  tumor-size = 25-29: no-recurrence-events
|  |  |  tumor-size = 30-34: recurrence-events
|  |  |  tumor-size = 40-44: no-recurrence-events
|  inv-nodes = 6-8: no-recurrence-events
|  inv-nodes = 9-11: recurrence-events
|  inv-nodes = 12-14: no-recurrence-events
|  inv-nodes = 15-17: no-recurrence-events
```

The **breast-cancer.csv** file is posted on Canvas along with this assignment; that is the file you'll use to supply a dataframe to your function for the tests. You will note that it contains some **missing values** (designated with ?'s). Do **not** modify the file to change the missing values into a "valid" value for the attribute. If you're counting whether an instance has *node-caps* == "yes", then a value of "?" for *node-caps* simply is not a match (just as a value "no" for *node-caps* would not be a match for "yes").

Submit a listing of your source code **as well as** the result for each of the above specified tests. **You will NOT get full credit for your solution if you hard-code your code to work just for the breast-cancer dataset and its rules; your solution must be generalized to work for any dataframe, any rule (with any kind of operators), and any confidence level! (35 pts.)**

**Note**: See the link below for a **Python** function that does a **Clopper-Pearson confidence interval** (although a beta distribution, not a binomial distribution**)**
https://www.statsmodels.org/stable/generated/statsmodels.stats.proportion.proportion_confint.html

**Another note**: Although not required for this problem, you might find it useful to write a function/program that can parse a text file of (Weka) decision tree output and generate rules in the [*antecedent*, *consequent*] format. You then could use that instead of writing (by hand) the 80+ rules for this problem. Additionally, for your comprehensive project later this semester, you will be required to "extract" rules from decision trees. So you'll be doing this again (and likely won't want to generate rules by hand!).

**Name: ____Anthony Sharp___________** **40 points possible**

2. Consider a dataset with attributes ***x***, ***y***, and ***z,*** where the decision attribute is ***z***. Suppose that we have determined that there are two **support vectors**: the **2D** point **(3, 0)** which corresponds to an instance in the dataset that has ***x*** = 3, ***y*** = 0, ***z*** = -1, and the **2D** point **(3, 1)** which corresponds to an instance in the dataset that has ***x*** = 3, ***y*** = 1, ***z*** = 1.

   The equations for the support vector machine are shown below where $\mathbf{s_1} = \mathbf{(3\ 0\ 1)}$ is the augmented support vector for (3, 0), $\mathbf{s_2} = \mathbf{(3\ 1\ 1)}$ is the augmented support vector for (3, 1), and $\alpha_1$ and $\alpha_2$ are the respective parameters for the support vectors that will be used to define the 2D hyperplane.

$$\alpha_1 s_1 \bullet s_1 + \alpha_2 s_2 \bullet s_1 = -1$$
$$\alpha_1 s_1 \bullet s_2 + \alpha_2 s_2 \bullet s_2 = 1$$

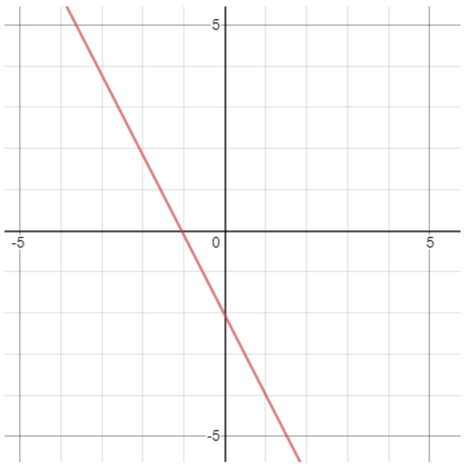
   a. **Solve for each $\alpha_i$** showing ALL of your work! **(3 pts.)**

   b. Using your results from part a., **define the discriminating 2D hyperplane** for this dataset; that is, give an **equation** for the **2D hyperplane**. Show your work! **(2 pts.)**

**a)**
**a1(3 0 -1) . (3 0 -1) + a2(3 1 1) . (3 0 -1) = -1**
**a1(10) + a2(8) = -1**

**a1(3 0 -1) . (3 1 1) + a2(3 1 1) . (3 1 1) = 1**
**a1(8) + a2(11) = 1**
**rref ([[10 8 -1] , [8 11 1]]) == [[-19/46],[9/23]]**
**a1 = -19/46**
**a2 = 9/23**

**w' = a1s1 + a2s2**
**= -19/46 (3 0 -1) + 9/23 (3 1 1)**
**= (-57/46, 0, 19/46) + (27/23, 9/23, 9/23)**
**= (35/46 , 9/23, 37/46)**
**2D : ax + by + c = 0**



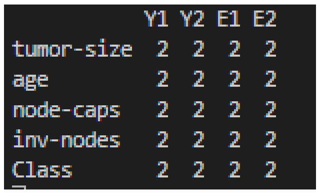
**Name: ____Anthony Sharp____________** **40 points possible**

1.

```
import copy
import statsmodels.api as sm
from statsmodels.stats.proportion import proportion_confint
import pandas as pd
import numpy as np

df = pd.read_csv("breast-cancer1.csv")
f = open('rules_in_List.txt','r+')

# 50-59,lt40,20-24,    0-2,?,1,    left,left_low,no,recurrence-events

#    int  int float , int int float, int int float
# [[[4, 4, 'yes'], [5, 4, '1']], [[9, 4, 'recurrence-events']]]

# [attributeIndexNumber, operatorCode, value]
# [[[AIT203 = 0, operatorCode = 1, AIT203 <= 420]  ,[FIT201 = 5, 2, FIT201 = 0.5]]]

'''
Every row of the table
Y1 = A1, A2, A3, A4, AND C
E1 = A1, A2, A3, A4, AND !C

Row 1
Y2 = !A1, A2, A3, A4, AND C
E2 = !A1, A2, A3, A4, AND !C

Row 2
Y2 = A1, !A2, A3, A4, AND C
E2 = A1, !A2, A3, A4, AND !C

...
calc y1 and e1 once
loop the y2 and e2...

#for the fails only
Rule original
1 - U(cf)(y1, y1+e2)

Rules with condition drop
1 - U(cf)(y1+y2, y1 + e1 + y2 + e2)
'''



outer_row_list = []
```

```
for row in df:
    outer_row_list.append(row)
# outer_row_list = ['age', 'menopause', 'tumor-size', 'inv-nodes', 'node-caps', 'deg-
malig', 'breast', 'breast-quad', 'irradiat', 'Class']
#print(outer_row_list)


""" print(df.loc[df["tumor-size"] < '0-4',:])
print(df.loc[df["age"] < '40-49',:]) """

#Data_slicer(df,'==',"tumor-size",'0-4')
def Logic_Rule(atribute,_logic,value):
    if(_logic == "<"):
        return df.loc[df[atribute] < value,:]

    elif(_logic == "<="):
        return df.loc[df[atribute] <= value,:]

    elif(_logic=='>'):
        return df.loc[df[atribute] > value,:]

    elif(_logic == ">="):
        return df.loc[df[atribute] >= value,:]

    elif(_logic == "=="):
        return df.loc[df[atribute] == value,:]

    elif(_logic == "!="):
        return df.loc[df[atribute] != value,:]

# A must to merge dataframe together
def merge_frame(f):
    _size = len(f)
    newDataFrame = pd.merge(f[0],f[1], how='inner')
    for i in range(_size):
        newDataFrame = pd.merge(newDataFrame,f[i], how='inner')
    return newDataFrame


# START HERE!!!!
# hardcode
test_case = [[0, 1, 420], [5, 2, 0.5], [16, 2, 250]], [[21, 4, 2]]

test_rule = [[2,4,"0-4"],[0,4,"40-49"],[4,4,"no"],[3,4,"0-2"],[9,4,"no-recurrence-
events"]]
```

```
test1 = Logic_Rule('tumor-size','==','0-4')
test2 = Logic_Rule('age','==','40-49')
test3 = Logic_Rule('inv-nodes','==','0-2')
test4 = Logic_Rule('node-caps','==','no')

test_rule2 = [[[4, 4, 'yes'], [5, 4, '1']], [[9, 4, 'recurrence-events']]]
""" test5 = Logic_Rule('node-caps','==','yes')
test6 = Logic_Rule('deg-malig','==','1')
test7 = Logic_Rule('class','==','recurrence-events') """

ruleLength = len(test_rule)

operatorCode = ["<","<=",">",">=","==","!="]

#print(len(test_case))

frame1 = [test1, test2, test3, test4]
#fram2 = [test5, test6, test7]

#   Call to merge frame
M = merge_frame(frame1)
attribute = []
for i in df:
    attribute.append(i)

attributeRow = []

for i in test_rule:
    attributeRow.append(attribute[i[0]])

change_R_and_C = pd.DataFrame(columns = ["Y1","Y2","E1","E2"], index = attributeRow)

for i in change_R_and_C.index:
    _Y = []
    _E = []

    count1 = 0

    for j in test_rule:
        if(count1 == ruleLength-1):
            #Logic_Rule('age','==','40-49')
            _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

        else:
            _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

        _Y.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

        count1 += 1

    mergeY = merge_frame(_Y)
    mergeE = merge_frame(_E)

    change_R_and_C.loc[i,"Y1"] = mergeY.index.size
    change_R_and_C.loc[i,"E1"] = mergeE.index.size

count2 = 0
for i in change_R_and_C.index:
    _Y = []
    _E = []

    count1 = 0

    for j in test_rule:
        if(count1 != count2):
            if(count1 == ruleLength-1):
                _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            else:
                _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            _Y.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            count1 += 1
        else:
            if(count1 == ruleLength-1):
                _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            else:
                _E.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            _Y.append(Logic_Rule(attribute[j[0]],operatorCode[j[1]],j[2]))

            count1 += 1
    count1 += 1

    mergeY = merge_frame(_Y)
    mergeE = merge_frame(_E)
    change_R_and_C.loc[i,"Y2"] = mergeY.index.size
    change_R_and_C.loc[i,"E2"] = mergeE.index.size

print(change_R_and_C)
```



```
            Y1 Y2 E1 E2
tumor-size   2  2  2  2
age          2  2  2  2
node-caps    2  2  2  2
inv-nodes    2  2  2  2
Class        2  2  2  2
```