

CSC487: Data Mining - Homework #3

Cason Konzer

1. Suppose that we have age data including the following numbers in sorted order. (25 points total)

[13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70]

Let us first load in the data...

```
# load in the data...
nums = [13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25,
        25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70]
df1 = pd.DataFrame()
df1['nums'] = nums

# preview the data...
df1
```

```
##      nums
## 0       13
## 1       15
## 2       16
## 3       16
## 4       19
## 5       20
## 6       20
## 7       21
## 8       22
## 9       22
## 10      25
## 11      25
## 12      25
## 13      25
## 14      30
## 15      33
## 16      33
## 17      35
## 18      35
## 19      35
## 20      35
## 21      36
## 22      40
## 23      45
## 24      46
## 25      52
## 26      70
```

a.) Use smoothing by bin means to smooth the above data, using a bin depth of 3. Illustrate your steps. Comment on the effect of this technique for the given data. (5 points)

```
# length of data.
```

```
l = df1.size
```

```
l
```

```
## 27
```

```
# binning & averaging.
```

```
df1['3bin_mean'] = 0
```

```
for i in range(0, l, 3):
```

```
    df1['3bin_mean'].iloc[i:i+3] = df1['nums'].iloc[i:i+3].mean()
```

```
df1
```

##	nums	3bin_mean
## 0	13	14.666667
## 1	15	14.666667
## 2	16	14.666667
## 3	16	18.333333
## 4	19	18.333333
## 5	20	18.333333
## 6	20	21.000000
## 7	21	21.000000
## 8	22	21.000000
## 9	22	24.000000
## 10	25	24.000000
## 11	25	24.000000
## 12	25	26.666667
## 13	25	26.666667
## 14	30	26.666667
## 15	33	33.666667
## 16	33	33.666667
## 17	35	33.666667
## 18	35	35.000000
## 19	35	35.000000
## 20	35	35.000000
## 21	36	40.333333
## 22	40	40.333333
## 23	45	40.333333
## 24	46	56.000000
## 25	52	56.000000
## 26	70	56.000000

This technique [generalizes](#) our data into groups of 3, increasing our previous minimum and decreasing our previous maximum.

b.) How can you determine outliers in the data? (5 points)

Outliers are typically determined by the Interquartile Range (IQR). If a value is outside 1.5 times the IQR we call it an outlier.

```
# summary statistics.
```

```
df1.describe()
```

```
##           nums  3bin_mean
## count  27.000000  27.000000
## mean   29.962963  29.962963
## std    12.942124  12.343839
## min    13.000000  14.666667
## 25%    20.500000  21.000000
## 50%    25.000000  26.666667
## 75%    35.000000  35.000000
## max    70.000000  56.000000
```

```
# IQR.
```

```
q1 = df1.describe()['nums']['25%']
```

```
q2 = df1.describe()['nums']['75%']
```

```
# 1.5 IQR.
```

```
min_allowed = q1/1.5
```

```
min_allowed
```

```
## 13.666666666666666
```

```
# 1.5 IQR.
```

```
max_allowed = q2*1.5
```

```
max_allowed
```

```
## 52.5
```

```
# Mask the dataframe.
```

```
outliers = df1['nums'][(df1['nums'] < min_allowed) | (df1['nums'] > max_allowed)]
```

```
outliers
```

```
## 0      13
```

```
## 26     70
```

```
## Name: nums, dtype: int64
```

As seen in part a.) $l = 27$, thus our smallest value, idx 0, and our largest value, idx 27, are shown here. 13 & 17 would generally be accepted as outliers.

c.) Use min-max normalization to transform the value 35 for age onto the range [0.0, 1.0]. (5 points)

```
# Create min max scaling function.
def min_max_scaler(val):
    r = df1['nums'].max() - df1['nums'].min()
    return (val - df1['nums'].min()) / r
```

```
# Scale 35.
min_max_scaler(35)
```

```
## 0.38596491228070173
```

We can see min-max normalization transforms 35 into 0.386 .

d.) Use z-score normalization to transform the value 35 for age? (5 points)

```
# Create z score scaling function.
def z_score_scaler(val):
    return (val - df1['nums'].mean()) / df1['nums'].std()
```

```
# Scale 35.
z_score_scaler(35)
```

```
## 0.3891970907527787
```

We can see z-score normalization transforms 35 into 0.389 .

e.) Use normalization by decimal scaling to transform the value 35 for age. (5 points)

```
# Create decimal scaling function.
def decimal_scaler(val):
    return val/100
```

```
# Scale 35.
decimal_scaler(35)
```

```
## 0.35
```

We can see normalization transforms by decimal scaling 35 into 0.35 .

2. Write a function in your preferred language which can take a data vector and do min-max normalization by transforming data onto a desired range. (25 points)

```
# Use data from q1 as example.
a = np.array(nums)

# preview the data...
a

## array([13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33,
##        35, 35, 35, 35, 36, 40, 45, 46, 52, 70])

# Define variable min max scaler.
def min_max_mapper(np_array, new_min, new_max):
    mn = np_array.min()
    mx = np_array.max()
    r = mx - mn
    a = (np_array - mn) / r
    new_r = new_max - new_min
    return (a * new_r) + new_min

# Test with data.
min_max_mapper(a, -10, 10)

## array([-10.          , -9.29824561, -8.94736842, -8.94736842,
##        -7.89473684, -7.54385965, -7.54385965, -7.19298246,
##        -6.84210526, -6.84210526, -5.78947368, -5.78947368,
##        -5.78947368, -5.78947368, -4.03508772, -2.98245614,
##        -2.98245614, -2.28070175, -2.28070175, -2.28070175,
##        -2.28070175, -1.92982456, -0.52631579,  1.22807018,
##         1.57894737,  3.68421053, 10.          ])
```

Solution writted in [python](#) dependent on [numpy](#) .

3. Using information gain on the data in Table 1, do calculations for two levels of a decision tree which decides whether a person is senior or junior. (25 points)

```
# load in the data...
table1 = {
    'department' : ['sales', 'sales', 'sales', 'systems',
                    'systems', 'systems', 'systems', 'marketing',
                    'marketing', 'secretary', 'secretary'],
    'age' : ['31_35', '26_30', '31_35', '21_25',
             '31_35', '26_30', '41_45', '36_40',
             '31_35', '46_50', '26_30'],
    'salary' : ['46K_50K', '26K_30K', '31K_35K', '46K_50K',
                '66K_70K', '46K_50K', '66K_70K', '46K_50K',
                '41K_45K', '36K_40K', '26K_30K'],
    'status' : ['senior', 'junior', 'junior', 'junior',
                'senior', 'junior', 'senior', 'senior',
                'junior', 'senior', 'junior'],
    'count' : [30, 40, 40, 20, 5, 3, 3, 10, 4, 4, 6]
}
df2 = pd.DataFrame(table1)

# preview the data...
df2
```

##	department	age	salary	status	count
## 0	sales	31_35	46K_50K	senior	30
## 1	sales	26_30	26K_30K	junior	40
## 2	sales	31_35	31K_35K	junior	40
## 3	systems	21_25	46K_50K	junior	20
## 4	systems	31_35	66K_70K	senior	5
## 5	systems	26_30	46K_50K	junior	3
## 6	systems	41_45	66K_70K	senior	3
## 7	marketing	36_40	46K_50K	senior	10
## 8	marketing	31_35	41K_45K	junior	4
## 9	secretary	46_50	36K_40K	senior	4
## 10	secretary	26_30	26K_30K	junior	6

```
# Create information gain function.
def gainer(df, label, counts):
    # Create DataFrame for storing gain.
    gainz = pd.DataFrame(index=['entropy', 'gain'])

    # Find total observations.
    total = df[counts].sum()
    print('{} total observations'.format(total))
    print()

    # Find total observations per label.
    sum0 = pd.DataFrame(
```

```

    df.groupby(by=label)[counts].apply(
        lambda x: x.sum()
    ))
print('observations per label')
print(sum0)
print()

# Find label probabilities.
p_label = sum0 / total
print('label probabilities')
print(p_label)
print()

# Find total entropy.
total_entrop = float((-p_label * np.log2(p_label)).sum())
print('total entropy')
print(total_entrop)
print()

# Find attribute columns.
cols = df.columns.drop(label)
cols = cols.drop(counts)
print('attribute columns')
print(cols)
print()

for col in cols:
    # Find total observations per bin.
    sum1 = pd.DataFrame(
        df.groupby(by=col)[counts].apply(
            lambda x: x.sum()
        ))

    # Find label totals per bin.
    sum2 = pd.DataFrame(
        df.groupby(by=[col, label])[counts].apply(
            lambda x: x.sum()
        ))

    # Solve for entropy per class
    p_label_class = sum2 / sum1
    H = -p_label_class * np.log2(p_label_class)

    # Solve for expected entropy per class
    entrop = pd.DataFrame(
        H.unstack().apply('sum', axis=1), columns=['H']
    )
    entrop['P[class]'] = sum1 / total

```



```

    entrop['E[H]'] = entrop['H'] * entrop['P[class]']
    entropy = entrop['E[H]'].sum()

    gain = total_entrop - entropy
    gainz[col] = [entropy, gain]

    return gainz

```

```

# calculate information gain on status at level 1.
g1 = gainer(df2, 'status', 'count')

```

```

## 165 total observations
##
## observations per label
##      count
## status
## junior   113
## senior    52
##
## label probabilities
##      count
## status
## junior  0.684848
## senior  0.315152
##
## total entropy
## 0.899030771238222
##
## attribute columns
## Index(['department', 'age', 'salary'], dtype='object')

```

```
g1.T
```

```

##      entropy    gain
## department  0.850424  0.048607
## age        0.474296  0.424735
## salary     0.361513  0.537518

```

As [salary](#) has the most information gain we will use it for the level 1 split.

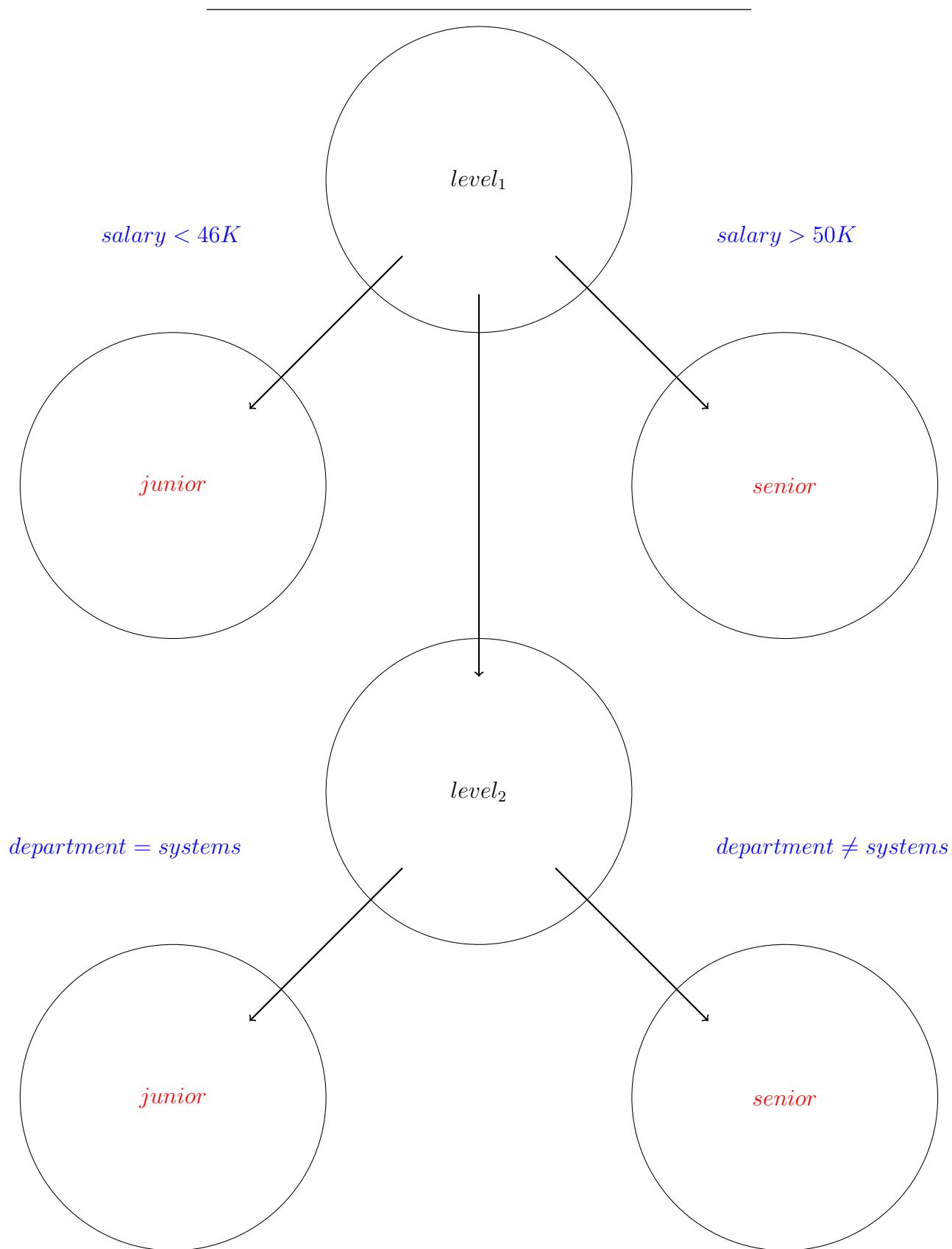
```
g2 = df2.groupby('salary').apply(  
    lambda x: gainer(x.drop('salary', axis=1), 'status', 'count')  
)
```

g2

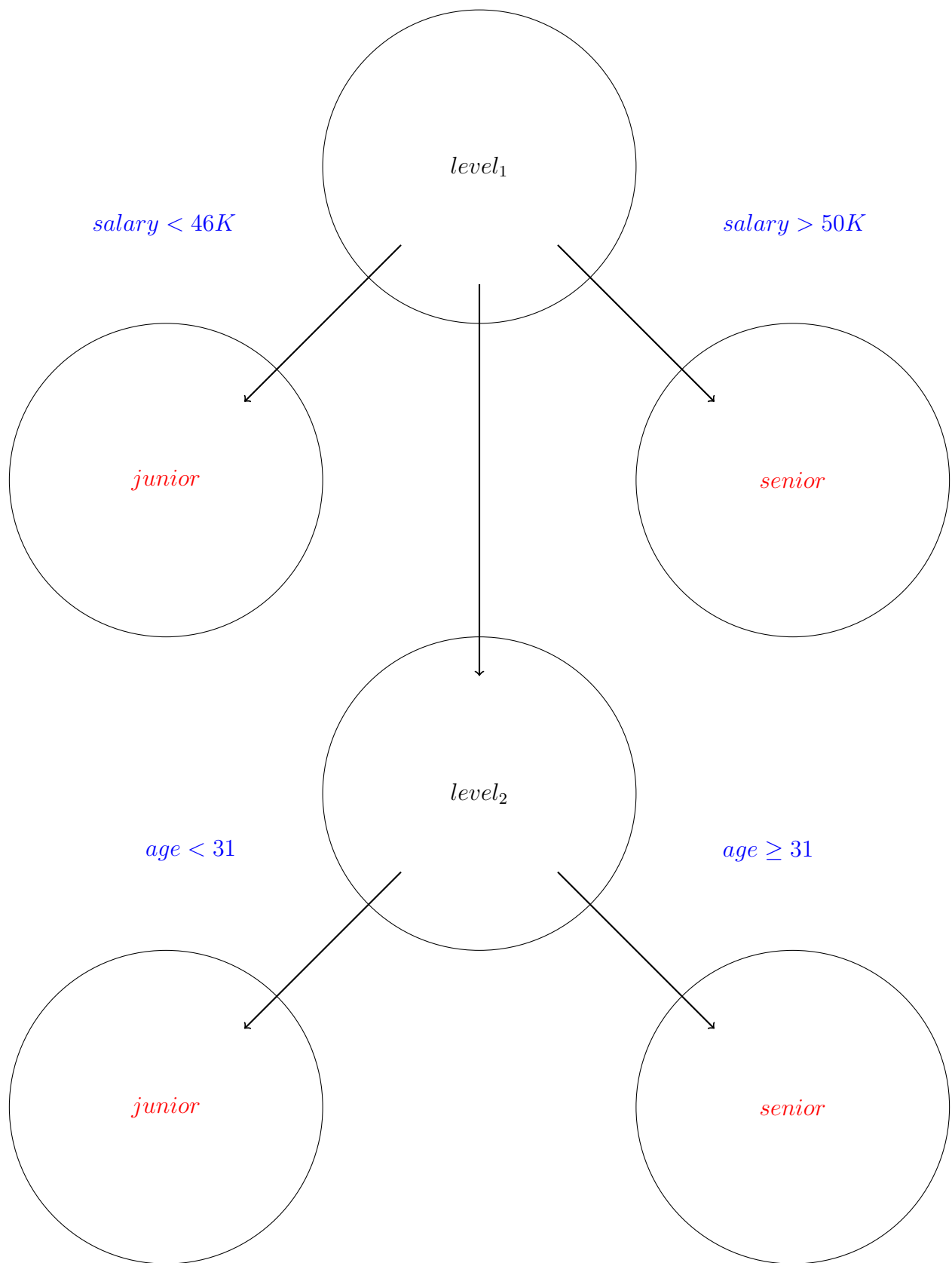
##		department	age
##	salary		
##	26K_30K	entropy	0.000000
##		gain	0.000000
##	31K_35K	entropy	0.000000
##		gain	0.000000
##	36K_40K	entropy	0.000000
##		gain	0.000000
##	41K_45K	entropy	0.000000
##		gain	0.000000
##	46K_50K	entropy	0.000000
##		gain	0.946819
##	66K_70K	entropy	0.000000
##		gain	0.000000

As [department](#) and [age](#) have equal information gain we can use [either](#) for the level 2 split.

4. Using the decision tree, generate if-then rules. (25 points)



Tree 1: Level 2 Split on Department



Tree 2: Level 2 Split on Age

For both trees we have $level_1$ if-then rules as follows ...

- *If salary < 46K then label = junior*
- *If salary > 50K then label = senior*
- *Else continue*

For Tree 1 we have $level_2$ if-then rules as follows ...

- *If department = systems then label = junior*
- *Else then label = senior*

For Tree 2 we have $level_2$ if-then rules as follows ...

- *If age < 31 then label = junior*
 - *Else then label = senior*
-