

# CSC487: Data Mining - Midterm

*Cason Konzer*

---

1. Suppose you have these data points: 29, 75, 13, 20, 168, 163, 140, 52, 4, 37, 36, 123, 120, 31, 111. (35 points total)

---

Let us first load in the data...

```
nums = [29, 75, 13, 20, 168, 163, 140, 52, 4, 37, 36, 123, 120, 31, 111]
nums = sorted(nums)
df1 = pd.DataFrame()
df1['nums'] = nums
df1['nums']
```

```
## 0      4
## 1     13
## 2     20
## 3     29
## 4     31
## 5     36
## 6     37
## 7     52
## 8     75
## 9    111
## 10   120
## 11   123
## 12   140
## 13   163
## 14   168
## Name: nums, dtype: int64
```

a.) If you draw a histogram with a bin size of 25, how many bars will you have in your chart? Please justify your answer. (7 points)

---

```
# find range.
r = (df1['nums'].max() - df1['nums'].min())
r
```

```
## 164
```

```
# min bin lower bound.
min_bins = r / 25
min_bins
```

```
## 6.56
```

As 6.56 bins are needed to span the whole range of the data, we need 7 bins in total.

---

b. What's the value at the 60<sup>th</sup> percentile. (7 points)

---

```
# 60th percentile.
df1.quantile(0.6)
```

```
## nums      89.4
## Name: 0.6, dtype: float64
```

We can see that the 60<sup>th</sup> percentile cutoff is 89.4 .

---

c. Use z-score normalization to transform the value 36. (7 points)

---

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

# Scale 36.
z_score_scaler(36)
```

```
## -0.6791180070064667
```

We can see that z-score normalization transforms the value 36 to -0.679 .

---

d. Use min-max normalization to transform the value 13 onto the range [1, 10]. (7 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 36.
min_max_scaler(36)
```

```
## 0.1951219512195122
```

We can see that min-max normalization transforms the value 36 to 0.1959 .

---

e. Suppose you have a bin depth 3 and use smoothing by bin median to smooth the first bin. (7 points)

---

```
# length of data.
l = df1.size
l

## 15

# cut idx.
q = int(l/3)
q

## 5

# binning & averaging.
df1['3bin_median'] = 0
df1['3bin_median'].iloc[0:q] = df1['nums'].iloc[0:q].median()
df1['3bin_median'].iloc[q:2*q] = df1['nums'].iloc[q:2*q].median()
df1['3bin_median'].iloc[2*q:l] = df1['nums'].iloc[2*q:l].median()

df1
```

```
##      nums  3bin_median
## 0        4           20
## 1       13           20
## 2       20           20
## 3       29           20
## 4       31           20
```

```
## 5      36      52
## 6      37      52
## 7      52      52
## 8      75      52
## 9     111      52
## 10     120     140
## 11     123     140
## 12     140     140
## 13     163     140
## 14     168     140
```

```
df1.iloc[0:q]
```

```
##      nums  3bin_median
## 0        4           20
## 1       13           20
## 2       20           20
## 3       29           20
## 4       31           20
```

We can see that with depth 3 the first bin's median is [20](#) .

---

2. Compute the distance between objects 3 and 4 in the table below. (15 points)

```
# Build the dataframe.
table1 = {
    "Object": [1,2,3,4],
    "test-1 (nominal)" : ["A","B","A","A"],
    "test-2 (ordinal)" : ["excellent","fair","good","excellent"],
}

df2 = pd.DataFrame(table1)

# Set the index.
df2.set_index("Object",inplace=True)
df2

##          test-1 (nominal) test-2 (ordinal)
## Object
## 1                A          excellent
## 2                B              fair
## 3                A              good
## 4                A          excellent

# Replace nominal values.
df2["test-1 (nominal)"].replace("A",3,inplace=True)
df2["test-1 (nominal)"].replace("B",2,inplace=True)

t1_range = (df2["test-1 (nominal)"].max()-df2["test-1 (nominal)"].min())
df2["test-1 (nominal)"] = (df2["test-1 (nominal)"]-1)/t1_range
df2

##          test-1 (nominal) test-2 (ordinal)
## Object
## 1                2.0          excellent
## 2                1.0              fair
## 3                2.0              good
## 4                2.0          excellent

# Replace ordinal values.
df2["test-2 (ordinal)"].replace("excellent",3,inplace=True)
df2["test-2 (ordinal)"].replace("good",2,inplace=True)
df2["test-2 (ordinal)"].replace("fair",1,inplace=True)

t2_range = (df2["test-2 (ordinal)"].max()-df2["test-2 (ordinal)"].min())
df2["test-2 (ordinal)"] = (df2["test-2 (ordinal)"]-1)/t2_range
df2
```

	test-1 (nominal)	test-2 (ordinal)
## Object		
## 1	2.0	1.0
## 2	1.0	0.0
## 3	2.0	0.5
## 4	2.0	1.0

*# Define our distance function.*

```
def dist(i,j, df):
    neum = 0
    denom = 0
    for col in df.columns:
        print(col)
        if 'nominal' in col:
            if df[col].iloc[j-1] != df[col].iloc[i-1]:
                dist = 1
            else:
                dist = 0
        else:
            dist = np.abs(df[col].iloc[j-1] - df[col].iloc[i-1])
        print('feature distance:',dist)
        neum += dist
        denom += 1
    print()
    print('numerator:',neum)
    print('denominator:',denom)
    print('\ntotal distance:',neum/denom)
    return neum/denom
```

```
# Compute the distance between object 1 & 3  
d_1_3 = dist(3,4, df2)
```

```
## test-1 (nominal)  
## feature distance: 0  
##  
## test-2 (ordinal)  
## feature distance: 0.5  
##  
## numerator: 0.5  
## denominator: 2  
##  
## total distance: 0.25
```

We can see that the distance between objects 3 and 4 is [0.25](#) .

---



3. Use chi-square for the data below to find out whether there's a relation between playing basketball and eating cereal. Based on your result describe the relation (15 points)

---

```
# Build the dataframe.
```

```
table2 = {
  "idx": ["Cereal", "Not cereal", "Sum(col.)"],
  "Basketball" : [213, 138, 351],
  "Not basketball" : [203, 110, 313],
  "Sum (row)" : [416, 248, 664],
}
```

```
df3 = pd.DataFrame(table2)
```

```
# Set the index.
```

```
df3.set_index("idx", inplace=True)
df3
```

```
##           Basketball  Not basketball  Sum (row)
## idx
## Cereal             213             203         416
## Not cereal         138             110         248
## Sum(col.)          351             313         664
```

```
# Create an expectation dataframe as a copy of the original.
```

```
expectation_df3 = df3.copy()
expectation_df3
```

```
##           Basketball  Not basketball  Sum (row)
## idx
## Cereal             213             203         416
## Not cereal         138             110         248
## Sum(col.)          351             313         664
```

```
# Compute probabilities on basketball status
```

```
expectation_df3["Sum (row)"] = expectation_df3["Sum (row)"].apply(
  lambda x: x / expectation_df3["Sum (row)"].max()
)
expectation_df3
```

```
##           Basketball  Not basketball  Sum (row)
## idx
## Cereal             213             203    0.626506
## Not cereal         138             110    0.373494
## Sum(col.)          351             313    1.000000
```

```
# Compute basketball expectations.
```

```
expectation_df3["Basketball"] = (  
    expectation_df3["Basketball"].max()*expectation_df3["Sum (row)"]  
)/  
expectation_df3
```

```
##           Basketball  Not basketball  Sum (row)  
## idx  
## Cereal      219.903614           203    0.626506  
## Not cereal  131.096386           110    0.373494  
## Sum(col.)   351.000000           313    1.000000
```

```
# Compute not basketball expectations
```

```
expectation_df3["Not basketball"] = (  
    expectation_df3["Not basketball"].max()*expectation_df3["Sum (row)"]  
)/  
expectation_df3
```

```
##           Basketball  Not basketball  Sum (row)  
## idx  
## Cereal      219.903614      196.096386    0.626506  
## Not cereal  131.096386      116.903614    0.373494  
## Sum(col.)   351.000000      313.000000    1.000000
```

```
# Trim our original dataframe.
```

```
df3 = df3.iloc[:-1,:-1]  
df3
```

```
##           Basketball  Not basketball  
## idx  
## Cereal           213           203  
## Not cereal       138           110
```

```
# Trim our expectation dataframe
```

```
expectation_df3 = expectation_df3.iloc[:-1,:-1]  
expectation_df3
```

```
##           Basketball  Not basketball  
## idx  
## Cereal      219.903614      196.096386  
## Not cereal  131.096386      116.903614
```

```
# Compute passing portion of chi squared.
```

```
basketball_ = np.sum(  
    np.square(  
        (df3["Basketball"] - expectation_df3["Basketball"])  
    )/expectation_df3["Basketball"]  
    )  
basketball_
```

```
## 0.5802793153909802
```

```
# Compute failing portion of chi square.
```

```
not_basketball_ = np.sum(  
    np.square(  
        (df3["Not basketball"] - expectation_df3["Not basketball"])  
    )/expectation_df3["Not basketball"]  
    )  
not_basketball_
```

```
## 0.6507285613489868
```

```
# Sum for our chi squared value.
```

```
chi_squared_ = basketball_ + not_basketball_  
chi_squared_
```

```
## 1.231007876739967
```

A  $\chi^2$  chart will clearly show that a test statistic of 1.231 given 1 degree of freedom is not a statistically significant. Thus we settle on the null hypothesis that playing basketball and eating cereal are not correlated.

---

4. Using the data table below, calculate the information gain for gender and age.

---

```
# Build the dataframe.
```

```
table3 = {
    "gender": ["male","male","female","female",
               "male","female","female","male","female"],
    "age" : ["young","young","young","teenager",
             "young","young","elder","middle age","elder"],
    "income" : ["medium","low","low","medium",
                "high","medium","high","medium","medium"],
    "play golf?" : ["yes","no","no","no",
                    "yes","no","yes","yes","yes"],
    "count" : [30,20,30,20,15,30,13,10,4],
}
```

```
df4 = pd.DataFrame(table3)
```

```
df4
```

```
##   gender      age  income play golf?  count
## 0   male    young  medium         yes    30
## 1   male    young   low          no    20
## 2  female    young   low          no    30
## 3  female  teenager  medium         no    20
## 4   male    young   high         yes    15
## 5  female    young  medium         no    30
## 6  female    elder   high         yes    13
## 7   male  middle age  medium         yes    10
## 8  female    elder   medium         yes     4
```

```
def gainer(df, label, counts):
```

```
    # Create DataFrame for storing gain.
```

```
    gainz = pd.DataFrame(index=['entropy', 'gain'])
```

```
    # Find total observations.
```

```
    total = df[counts].sum()
```

```
    # Find total observations per label.
```

```
    sum0 = pd.DataFrame(df.groupby(by=label)[counts].apply(lambda x: x.sum()))
```

```
    # Find label probabilities.
```

```
    p_label = sum0 / total
```

```
    # Find total entropy.
```

```
    total_entrop = float((-p_label * np.log2(p_label)).sum())
```

```

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

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

return gainz

g1 = gainer(df4, 'gender', 'count')
g1.T

```

```

##          entropy    gain
## age      0.725905  0.262261
## income   0.982178  0.005988
## play golf? 0.749801  0.238365

```

We can now see that the information gain for age when using gender as labels is [0.262](#) .