

CSC487: Data Mining - Homework #2

Cason Konzer

1. Find the distance between objects 1 and 3. Notice that we have mixed type of attributes. (25 points)

The following python code has will find the distance between any two objects for this dataset.

```
# Build the dataframe.
```

```
table1 = {  
    "Object Identifier": [1,2,3,4],  
    "test-1 (nominal)" : ["A","B","C","A"],  
    "test-2 (ordinal)" : ["excellent","fair","good","excellent"],  
    "test-3 (numeric)" : [45,22,64,28]  
}
```

```
df1 = pd.DataFrame(table1)
```

```
# Set the index.
```

```
df1.set_index("Object Identifier",inplace=True)  
df1
```

```
##                test-1 (nominal) test-2 (ordinal)  test-3 (numeric)  
## Object Identifier  
## 1                      A          excellent          45  
## 2                      B           fair           22  
## 3                      C           good           64  
## 4                      A          excellent          28
```

```
# Replace nominal values.
```

```
df1["test-1 (nominal)"].replace("A",3,inplace=True)  
df1["test-1 (nominal)"].replace("B",2,inplace=True)  
df1["test-1 (nominal)"].replace("C",1,inplace=True)  
  
t1_range = (df1["test-1 (nominal)"].max()-df1["test-1 (nominal)"].min())  
df1["test-1 (nominal)"] = (df1["test-1 (nominal)"]-1)/t1_range  
df1
```

```
##                test-1 (nominal) test-2 (ordinal)  test-3 (numeric)  
## Object Identifier  
## 1                      1.0          excellent          45  
## 2                      0.5           fair           22  
## 3                      0.0           good           64  
## 4                      1.0          excellent          28
```

```
# Replace ordinal values.
```

```
df1["test-2 (ordinal)"].replace("excellent",3,inplace=True)  
df1["test-2 (ordinal)"].replace("good",2,inplace=True)  
df1["test-2 (ordinal)"].replace("fair",1,inplace=True)
```

```
t2_range = (df1["test-2 (ordinal)"].max()-df1["test-2 (ordinal)"].min())
df1["test-2 (ordinal)"] = (df1["test-2 (ordinal)"]-1)/t2_range
df1
```

```
##                test-1 (nominal)  test-2 (ordinal)  test-3 (numeric)
## Object Identifier
## 1                      1.0              1.0              45
## 2                      0.5              0.0              22
## 3                      0.0              0.5              64
## 4                      1.0              1.0              28
```

Replace numeric values.

```
t3_range = (df1["test-3 (numeric)"].max()-df1["test-3 (numeric)"].min())
df1["test-3 (numeric)"] = (
    df1["test-3 (numeric)"]-df1["test-3 (numeric)"].min()
)/t3_range
df1
```

```
##                test-1 (nominal)  test-2 (ordinal)  test-3 (numeric)
## Object Identifier
## 1                      1.0              1.0              0.547619
## 2                      0.5              0.0              0.000000
## 3                      0.0              0.5              1.000000
## 4                      1.0              1.0              0.142857
```

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:')
    return neum/denom
```

```
# Compute the distance between object 1 & 3  
dist(1,3, df1)
```

```
## test-1 (nominal)  
## feature distance: 1  
##  
## test-2 (ordinal)  
## feature distance: 0.5  
##  
## test-3 (numeric)  
## feature distance: 0.45238095238095233  
##  
## numerator: 1.9523809523809523  
## denominator: 3  
##  
## total distance:  
## 0.6507936507936508
```

We can see that the distance between object 1 and object 3 is [0.651](#) .

2. Write a program in any language which can compute Manhattan and Euclidean distances between any two given vectors with any length. You can pass the length to your function, but please don't limit the dimension to 2. You can test your function on vectors you fill in your code without asking user input. (25 points)

The following python code has independent functions that will find the Manhattan and Euclidean distance between any two given numpy vectors.

```
# Create test vector 1.
test_v1 = np.array([0,1,2,3,5,4])
test_v1

## array([0, 1, 2, 3, 5, 4])

# Create test vector 2.
test_v2 = np.array([1,0,3,2,4,6])
test_v2

## array([1, 0, 3, 2, 4, 6])

# Build our Manhattan Function
def manhattaan(v1,v2):
    return np.sum(np.abs(v2-v1))

# Test our Manhattan Function
manhattaan(test_v1,test_v2)

## 7

# Build our Euclidean function.
def euclidian(v1,v2):
    return np.sqrt(np.sum(np.square(v2-v1)))

# Test our Euclidean function.
euclidian(test_v1,test_v2)

## 3.0
```

We can see that the simple test vectors pass for both functions.

3. In the table below, determine whether passing a class has a dependency on attendance by using Chi-square test. (25 points)

```
# Build the dataframe.
```

```
table2 = {  
    ""      : ["Attended", "Skipped", "Total"],  
    "Pass"  : [25, 8, 33],  
    "Fail"  : [6, 15, 21]  
}
```

```
df2 = pd.DataFrame(table2)  
df2
```

```
##           Pass  Fail  
## 0  Attended    25    6  
## 1   Skipped     8   15  
## 2     Total    33   21
```

```
# Total on attendance status.
```

```
df2["Total"] = df2["Pass"] + df2["Fail"]  
df2
```

```
##           Pass  Fail  Total  
## 0  Attended    25    6     31  
## 1   Skipped     8   15     23  
## 2     Total    33   21     54
```

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

```
expectation_df2 = df2.copy()  
expectation_df2
```

```
##           Pass  Fail  Total  
## 0  Attended    25    6     31  
## 1   Skipped     8   15     23  
## 2     Total    33   21     54
```

```
# Compute probabilities on attendance status
```

```
expectation_df2["Total"] = expectation_df2["Total"].apply(  
    lambda x: x / expectation_df2["Total"].max()  
)  
expectation_df2
```

```
##           Pass  Fail    Total  
## 0  Attended    25    6  0.574074  
## 1   Skipped     8   15  0.425926  
## 2     Total    33   21  1.000000
```

```
# Compute passing expectations.
```

```
expectation_df2["Pass"] = (
    expectation_df2["Pass"].max()*expectation_df2["Total"])
expectation_df2
```

```
##
## 0  Attended  18.944444    6  0.574074
## 1  Skipped   14.055556   15  0.425926
## 2    Total   33.000000   21  1.000000
```

```
# Compute failing expectations
```

```
expectation_df2["Fail"] = (
    expectation_df2["Fail"].max()*expectation_df2["Total"])
expectation_df2
```

```
##
## 0  Attended  18.944444  12.055556  0.574074
## 1  Skipped   14.055556   8.944444  0.425926
## 2    Total   33.000000  21.000000  1.000000
```

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

```
df2 = df2.iloc[:-1,:-1]
df2
```

```
##
## 0  Attended    25    6
## 1  Skipped     8   15
```

```
# Trim our expectation dataframe
```

```
expectation_df2 = expectation_df2.iloc[:-1,:-1]
expectation_df2
```

```
##
## 0  Attended  18.944444  12.055556
## 1  Skipped   14.055556   8.944444
```

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

```
pass_ = np.sum(
    np.square(
        (df2["Pass"] - expectation_df2["Pass"])
        )/expectation_df2["Pass"]
    )
pass_
```

```
## 4.544562029835523
```

```
# Compute failing portion of chi square.
fail_ = np.sum(
    np.square(
        (df2["Fail"] - expectation_df2["Fail"])
        )/expectation_df2["Fail"]
    )
fail_
```

```
## 7.141454618312963
```

```
# Sum for our chi squared value.
chi_squared_ = pass_ + fail_
chi_squared_
```

```
## 11.686016648148486
```

We can see that we yield $\chi^2 = 11.686$.

4. In R, there is a built-in data frame called `mtcars`. Please calculate the correlation between `mpg` and `wt` attributes of `mtcars` by using `cor()` function. Then generate scatter plot based on these two attributes. (25 points)

```
# Preview mtcars dataset.
```

```
mtcars
```

```
##           mpg  cyl  disp  hp  drat    wt   qsec vs  am  gear  carb
## Mazda RX4      21.0    6 160.0 110 3.90 2.620 16.46 0   1    4    4
## Mazda RX4 Wag  21.0    6 160.0 110 3.90 2.875 17.02 0   1    4    4
## Datsun 710      22.8    4 108.0  93 3.85 2.320 18.61 1   1    4    1
## Hornet 4 Drive  21.4    6 258.0 110 3.08 3.215 19.44 1   0    3    1
## Hornet Sportabout 18.7    8 360.0 175 3.15 3.440 17.02 0   0    3    2
## Valiant        18.1    6 225.0 105 2.76 3.460 20.22 1   0    3    1
## Duster 360     14.3    8 360.0 245 3.21 3.570 15.84 0   0    3    4
## Merc 240D      24.4    4 146.7  62 3.69 3.190 20.00 1   0    4    2
## Merc 230       22.8    4 140.8  95 3.92 3.150 22.90 1   0    4    2
## Merc 280       19.2    6 167.6 123 3.92 3.440 18.30 1   0    4    4
## Merc 280C      17.8    6 167.6 123 3.92 3.440 18.90 1   0    4    4
## Merc 450SE     16.4    8 275.8 180 3.07 4.070 17.40 0   0    3    3
## Merc 450SL     17.3    8 275.8 180 3.07 3.730 17.60 0   0    3    3
## Merc 450SLC    15.2    8 275.8 180 3.07 3.780 18.00 0   0    3    3
## Cadillac Fleetwood 10.4    8 472.0 205 2.93 5.250 17.98 0   0    3    4
## Lincoln Continental 10.4    8 460.0 215 3.00 5.424 17.82 0   0    3    4
## Chrysler Imperial 14.7    8 440.0 230 3.23 5.345 17.42 0   0    3    4
## Fiat 128       32.4    4  78.7  66 4.08 2.200 19.47 1   1    4    1
## Honda Civic    30.4    4  75.7  52 4.93 1.615 18.52 1   1    4    2
## Toyota Corolla 33.9    4  71.1  65 4.22 1.835 19.90 1   1    4    1
## Toyota Corona  21.5    4 120.1  97 3.70 2.465 20.01 1   0    3    1
## Dodge Challenger 15.5    8 318.0 150 2.76 3.520 16.87 0   0    3    2
## AMC Javelin    15.2    8 304.0 150 3.15 3.435 17.30 0   0    3    2
## Camaro Z28     13.3    8 350.0 245 3.73 3.840 15.41 0   0    3    4
## Pontiac Firebird 19.2    8 400.0 175 3.08 3.845 17.05 0   0    3    2
## Fiat X1-9      27.3    4  79.0  66 4.08 1.935 18.90 1   1    4    1
## Porsche 914-2  26.0    4 120.3  91 4.43 2.140 16.70 0   1    5    2
## Lotus Europa   30.4    4  95.1 113 3.77 1.513 16.90 1   1    5    2
## Ford Pantera L 15.8    8 351.0 264 4.22 3.170 14.50 0   1    5    4
## Ferrari Dino   19.7    6 145.0 175 3.62 2.770 15.50 0   1    5    6
## Maserati Bora   15.0    8 301.0 335 3.54 3.570 14.60 0   1    5    8
## Volvo 142E     21.4    4 121.0 109 4.11 2.780 18.60 1   1    4    2
```

```
# Set variables for naming.
```

```
mpg = mtcars$mpg
```

```
weight = mtcars$wt
```

```
# Find Correlation.
```

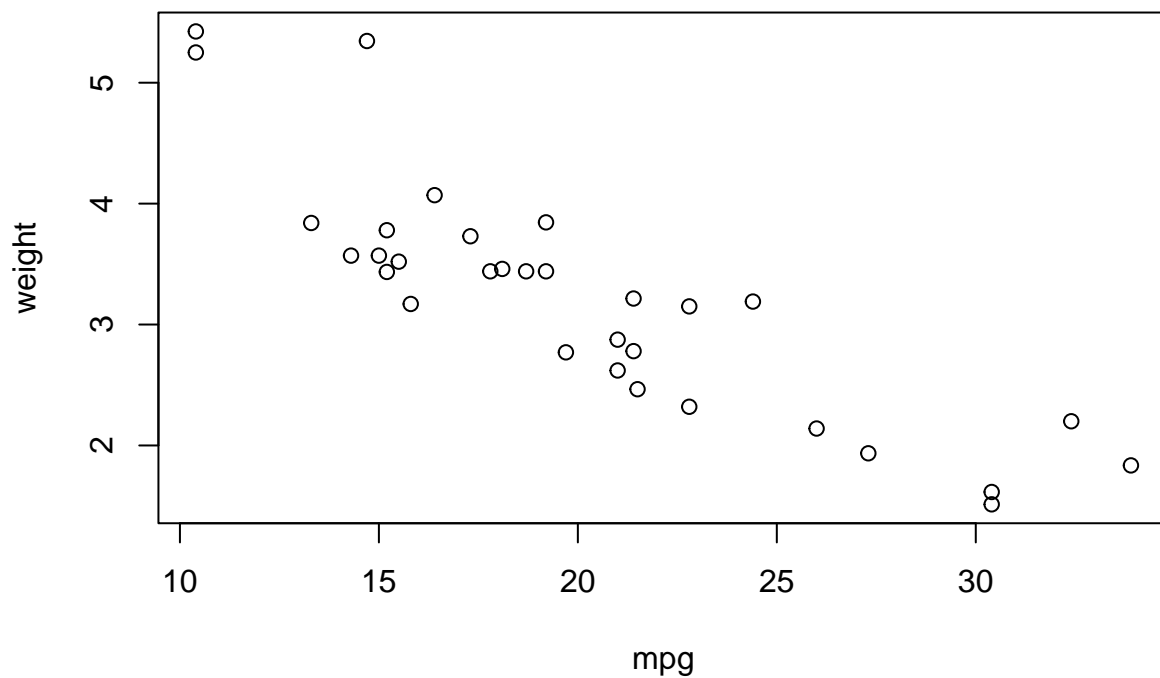
```
cor(mpg, weight)
```

```
## [1] -0.8676594
```

We can see that the correlation between vehicle miles per gallon and vehicle weight in the mtcars dataset is $r_{mpg,weight} = -0.868$. What this explains is that as one variable (weight/mpg) increases, the other (mpg/weight) decreases.

```
# Make scatterplot.
```

```
plot(mpg, weight)
```



```
# To coincide with the example given in the assignment. (:  
plot(weight, mpg)
```

