NB_hw_1

September 25, 2023

1 CMPS 320- Machine Learning

1.1 HW1

Nischal Bhandari

[1]:
```
# Load packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import scale

%matplotlib inline
plt.style.use('seaborn-white')
```

[2]:
```
# loading data
expression_df= pd.read_csv("NCI60_data.csv",index_col=0)
expression_df.head()
```

[2]:
```
           1         2         3         4         5             6         7  \
V1  0.300000  1.180000  0.550000  1.140000 -0.265000 -7.000000e-02  0.350000
V2  0.679961  1.289961  0.169961  0.379961  0.464961  5.799610e-01  0.699961
V3  0.940000 -0.040000 -0.170000 -0.040000 -0.605000  0.000000e+00  0.090000
V4  0.280000 -0.310000  0.680000 -0.810000  0.625000 -1.387779e-17  0.170000
V5  0.485000 -0.465000  0.395000  0.905000  0.200000 -5.000000e-03  0.085000

           8         9        10  …      6821      6822      6823      6824  \
V1 -0.315000 -0.450000 -0.654980  … -0.990020  0.000000  0.030000 -0.175000
V2  0.724961 -0.040039 -0.285019  … -0.270058 -0.300039 -0.250039 -0.535039
V3  0.645000  0.430000  0.475019  …  0.319981  0.120000 -0.740000 -0.595000
V4  0.245000  0.020000  0.095019  … -1.240020 -0.110000 -0.160000  0.095000
V5  0.110000  0.235000  1.490019  …  0.554980 -0.775000 -0.515000 -0.320000

        6825      6826      6827      6828      6829      6830
V1  0.629981 -0.030000  0.000000  0.280000 -0.340000 -1.930000
```

```
V2  0.109941 -0.860039 -1.250049 -0.770039 -0.390039 -2.000039
V3 -0.270020 -0.150000  0.000000 -0.120000 -0.410000  0.000000
V4 -0.350019 -0.300000 -1.150010  1.090000 -0.260000 -1.100000
V5  0.634980  0.605000  0.000000  0.745000  0.425000  0.145000

[5 rows x 6830 columns]
```

[3]:
```
# changing the index to integers
expression_df.index = expression_df.index.str[1:].astype(int)
```

[4]:
```
expression_df.tail(3)
```

[4]:
```
       1     2     3     4      5             6     7      8     9        10  \
62  0.21 -0.62 -0.15 -1.33  0.045 -4.000000e-01 -0.39 -0.675 -0.36  0.945020
63 -0.05  0.14 -0.09 -1.26  0.045 -2.710505e-20  0.42 -0.305  0.31  0.065019
64  0.35 -0.27  0.02 -1.23 -0.715 -3.400000e-01 -0.52  0.475  0.23  0.915019

    …     6821  6822  6823   6824      6825  6826     6827  6828  6829  6830
62  … -0.16002 -0.12  0.85 -0.125  0.779980  0.39  0.00000  0.16  2.03  3.94
63  …  0.88998 -0.42 -0.46 -0.855 -0.160020 -0.35 -0.36001 -0.49  0.01 -1.72
64  …  1.62998  3.00  2.86  2.145  0.869981  0.48  0.96999  0.29 -0.15  1.21

[3 rows x 6830 columns]
```

[5]:
```
cancer_type = pd.read_csv("NCI60_labs.csv", index_col= 0, header=0, names=␣
 ↪["cancer type"])
cancer_type.head()
```

[5]:
```
  cancer type
1         CNS
2         CNS
3         CNS
4       RENAL
5      BREAST
```

Checking the dimension of the dataframes

[6]:
```
# Print the dimensions of the dataframes
print("Dimensions of expression data:", expression_df.shape)



# Print the dimensions of the modified Hitters data (263 rows x 20 columns)
print("Dimensions of cancer classification data:", cancer_type.shape)
```

```
Dimensions of expression data: (64, 6830)
Dimensions of cancer classification data: (64, 1)
```

Checking for missing values in both dataframes

```
[7]: # expression df
     print("Missing values in expression_df:")
     print(expression_df.isnull().sum().sum())

     # cancer type df

     print("Missing values in cancer_type: ", cancer_type.isnull().sum().sum())
```

```
Missing values in expression_df:
0
Missing values in cancer_type:  0
```

So we do not have any missing values in both of our dataframes.

2. Data Preprocessing

```
[8]:      # Checking if the data is standardized
     if np.allclose(expression_df.mean(), 0) and np.allclose(expression_df.std(), 1):
         print("Data is standardized")
     else:
         print("Data is not standardized")
```

```
Data is not standardized
```

```
[9]:  # Standardizing the data
     from sklearn import preprocessing

     expression_df = pd.DataFrame(preprocessing.scale(expression_df, axis=0),␣
      ↪index=expression_df.index, columns=expression_df.columns)
     expression_standard = expression_df
```

```
[11]: # joingeing the cancer dataframe
      expression_standard = expression_standard.join(cancer_type)
      expression_standard.set_index("cancer type",inplace=True)
      expression_standard.head()
```

```
[11]:                    1         2         3         4         5         6  \
      cancer type
      CNS         0.728671  1.607220  1.325688  1.355688 -0.604845 -0.220654
      CNS         1.596418  1.753544  0.441686  0.654119  0.911898  1.648748
      CNS         2.190290 -0.016217 -0.349092  0.266465 -1.311310 -0.019322
      RENAL       0.682995 -0.375502  1.628079 -0.444299  1.244434 -0.019322
      BREAST      1.151170 -0.581759  0.965145  1.138767  0.361351 -0.033703

                         7         8         9        10  …      6821      6822  \
      cancer type                                         …
      CNS         0.898137 -0.868741 -1.058612 -1.059174  … -1.030663 -0.358518
      CNS          1.849697  2.226625 -0.095860 -0.477977  … -0.215657 -0.625720
      CNS          0.191185  1.988627  1.007979  0.716019  …  0.452274 -0.251651
      RENAL        0.408709  0.798057  0.045135  0.119051  … -1.313667 -0.456479
      BREAST       0.177590  0.396239  0.550041  2.310550  …  0.718297 -1.048700

                       6823      6824      6825      6826      6827      6828  \
      cancer type
      CNS         -0.238245 -0.392487  0.831370 -0.200286 -0.075668  0.520893
      CNS         -0.489938 -0.800791  0.013818 -1.105413 -1.117676 -0.823652
      CNS         -0.930304 -0.868790 -0.583517 -0.331142 -0.075668  0.008704
      RENAL       -0.409013 -0.086293 -0.709285 -0.494711 -1.034286  1.558075
      BREAST      -0.728079 -0.556925  0.839231  0.492157 -0.075668  1.116312

                       6829      6830
      cancer type
      CNS         -0.836365 -1.384675
      CNS         -0.925425 -1.431446
      CNS         -0.960951 -0.095838
      RENAL       -0.693981 -0.830408
      BREAST       0.525182  0.000992

      [5 rows x 6830 columns]
```

[18]:
```
expression_standard.describe()
```

[18]:
```
                  1          2             3             4             5  \
count  6.400000e+01  64.000000  6.400000e+01  6.400000e+01  6.400000e+01
mean  -8.673617e-18   0.000000 -3.014082e-17  3.989864e-17 -2.428613e-17
std    1.007905e+00   1.007905  1.007905e+00  1.007905e+00  1.007905e+00
min   -2.377270e+00  -2.877193 -3.931262e+00 -2.105826e+00 -1.768435e+00
25%   -8.071713e-01  -0.501898 -4.013951e-01 -9.173725e-01 -5.217310e-01
50%    4.353664e-02   0.037011  4.634208e-02  3.033881e-01 -5.421676e-02
75%    7.515195e-01   0.506077  4.243081e-01  9.426144e-01  3.821298e-01
max    2.190290e+00   3.017748  2.721339e+00  1.687994e+00  3.509280e+00

                  6             7             8             9            10  \
count  6.400000e+01  6.400000e+01  6.400000e+01  6.400000e+01  6.400000e+01
mean  -5.204170e-18  2.428613e-17 -3.816392e-17  1.734723e-18 -4.510281e-17
std    1.007905e+00  1.007905e+00  1.007905e+00  1.007905e+00  1.007905e+00
min   -2.032645e+00 -2.555051e+00 -2.029547e+00 -2.115392e+00 -2.630141e+00
25%   -4.687243e-01 -7.230938e-01 -5.412910e-01 -8.472563e-01 -7.803041e-01
50%   -1.932168e-02 -5.352873e-02  6.883255e-02 -1.832542e-03 -3.022115e-02
75%    5.127569e-01  6.194484e-01  5.451041e-01  6.909453e-01  7.042364e-01
max    3.317043e+00  2.502375e+00  2.226625e+00  2.323083e+00  2.310550e+00

       …          6821          6822       6823       6824          6825  \
count  …  6.400000e+01  6.400000e+01  64.000000  64.000000  6.400000e+01
```

```
mean    … -3.469447e-18  2.081668e-17   0.000000   0.000000  1.040834e-17
std     …  1.007905e+00  1.007905e+00   1.007905   1.007905  1.007905e+00
min     … -2.004194e+00 -1.048700e+00  -1.298802  -2.569869 -2.344267e+00
25%     … -6.570984e-01 -5.165919e-01  -0.559558  -0.566848 -5.245635e-01
50%     …  9.005226e-02 -3.585180e-01  -0.265209  -0.194028 -1.590203e-01
75%     …  4.352942e-01  1.401940e-01   0.114525   0.290779  3.165084e-01
max     …  3.406828e+00  4.272379e+00   4.687052   4.336512  4.400030e+00

               6826          6827          6828          6829          6830
count  6.400000e+01  6.400000e+01  6.400000e+01  6.400000e+01  6.400000e+01
mean   4.510281e-17 -1.387779e-17 -2.081668e-17  4.076600e-17  1.162265e-16
std    1.007905e+00  1.007905e+00  1.007905e+00  1.007905e+00  1.007905e+00
min   -1.617887e+00 -1.767830e+00 -2.014441e+00 -1.850851e+00 -1.985687e+00
25%   -5.274250e-01 -5.779040e-01 -7.115609e-01 -7.206783e-01 -8.304080e-01
50%   -1.675724e-01 -7.566773e-02  1.623610e-01 -2.312333e-01 -9.583766e-02
75%    1.868279e-01  3.686063e-02  5.497099e-01  5.719101e-01  7.389014e-01
max    4.848555e+00  4.375607e+00  3.440368e+00  3.381761e+00  2.535260e+00

[8 rows x 6830 columns]
```

K-Means Clustering to the data

[20]:
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_init=150, random_state=123)
kmeans.fit(expression_standard)
```

[20]:
```
KMeans(n_init=150, random_state=123)
```

[21]:
```python
labels = kmeans.labels_
# not specifying the number of clusters
contingency_table = pd.crosstab(index=expression_standard.index, columns=labels)
contingency_table
```

[21]:
```
col_0         0  1  2  3  4  5  6  7
row_0
BREAST        0  2  1  0  2  0  2  0
CNS           0  0  2  0  3  0  0  0
COLON         0  0  0  1  0  6  0  0
K562A-repro   0  0  0  0  0  0  0  1
K562B-repro   0  0  0  0  0  0  0  1
LEUKEMIA      4  0  0  1  0  0  0  1
MCF7A-repro   0  0  0  0  0  0  1  0
MCF7D-repro   0  0  0  0  0  0  1  0
MELANOMA      0  7  0  1  0  0  0  0
NSCLC         0  0  1  6  2  0  0  0
OVARIAN       0  0  1  4  1  0  0  0
```

```
PROSTATE      0  0  0  2  0  0  0  0
RENAL         0  0  1  2  6  0  0  0
UNKNOWN       0  0  1  0  0  0  0  0
```

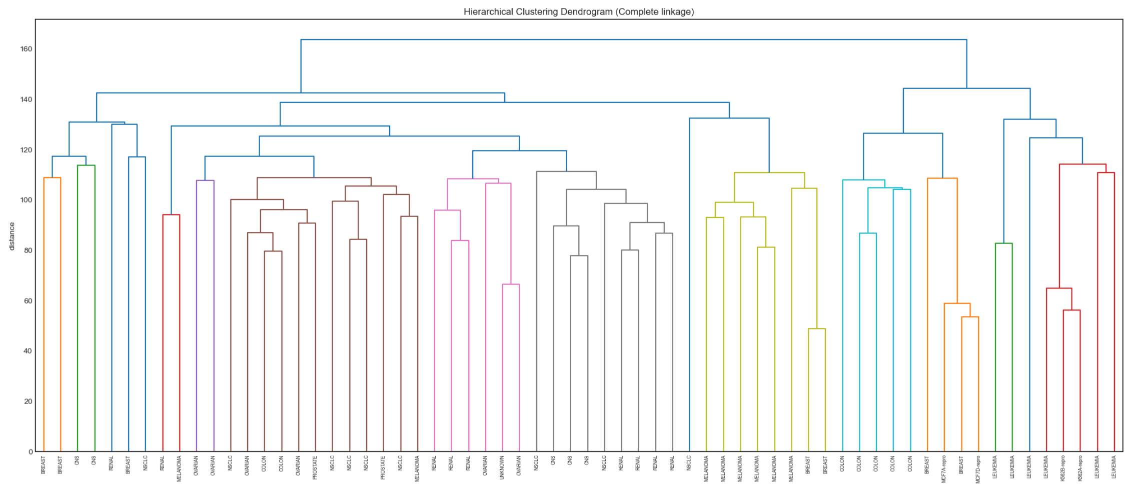
The results of the K-Means clustering are summarized in contingency tables above, which show the number of instances of each cancer type in each cluster.

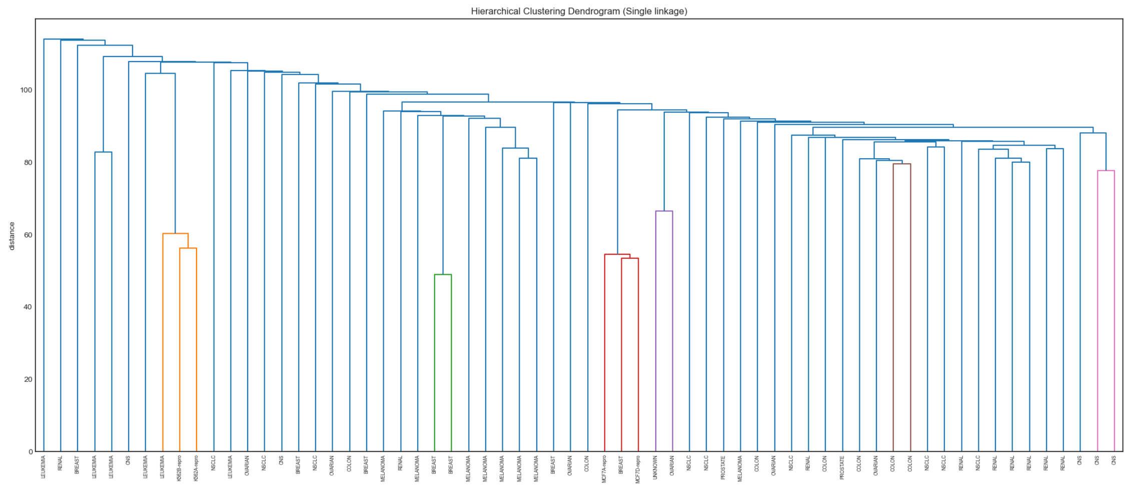
We can see that the algorithm has identified 8 clusters (numbered 0 to 7). For example, the 'BREAST' cancer type has instances in clusters 1, 2, 4, and 6. The 'CNS' cancer type has instances in clusters 2 and 4, and so on.

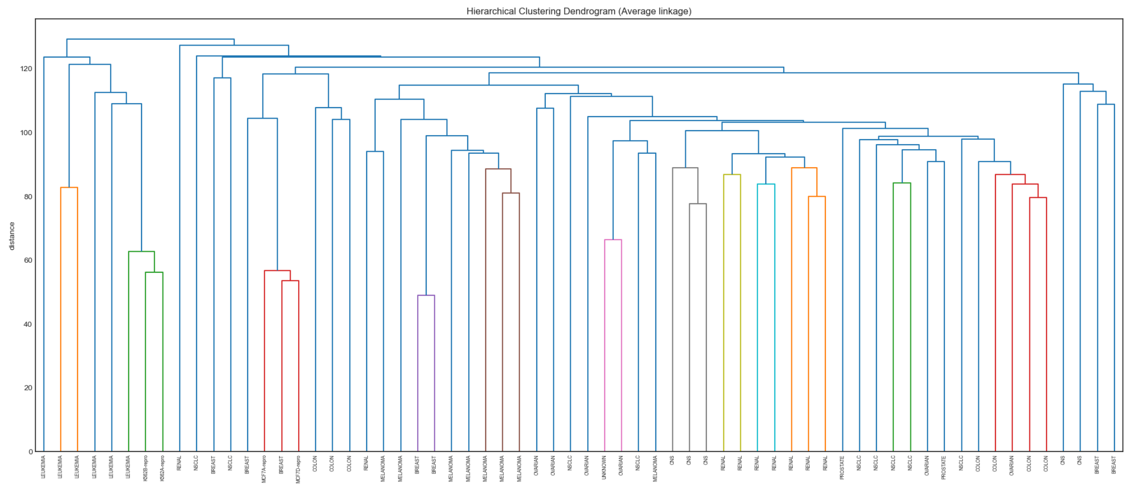
It can be inferred from the repeating instances of same cancers in different clusters that the number of clusters can be minimzed by half–maybe upto 4 or 5.

Hierarchical Clustering

[25]:
```
# Hierarchical Clustering
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from scipy.spatial.distance import pdist

# Generate the linkage matrix
Z = linkage(expression_standard, 'complete', metric='euclidean')

# Plot dendrogram for 'complete' linkage
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (Complete linkage)')
plt.ylabel('distance')
dendrogram(Z, labels = expression_standard.index)
plt.show()

Z = linkage(expression_standard, 'single', metric='euclidean')

# Plot dendrogram for 'single' linkage
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (Single linkage)')
plt.ylabel('distance')
dendrogram(Z, labels = expression_standard.index)
plt.show()

Z = linkage(expression_standard, 'average', metric='euclidean')

# Plot dendrogram for 'average' linkage
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (Average linkage)')
plt.ylabel('distance')
dendrogram(Z, labels = expression_standard.index)
plt.show()
```


Hierarchical Clustering Dendrogram (Complete linkage)
distance


Hierarchical Clustering Dendrogram (Single linkage)
distance


Hierarchical Clustering Dendrogram (Average linkage)
distance

Complete linkage tend to better group the cancer types. Since the maximum distance is allowed, complete linkage minimizes the risk of overfitting all the data. It has the highest distance of all linkage–160. Then the average linkage follows. It has color-coded separation of nearest cancer type. It has eucledian distance of around 125. It tends to improve and more difininf than the single linkage. Single linkage provides somewhat blurry pictures of how different cancers can be clustered together. It's distance is around 115, and that might explain some of the varaibility being underestimated in such linkage.

If we observe, it can be clearly seen that CNS anad CNS, breast and breast, and so on are identically clustered together in complete linkage. Average linkage tends to cluster Lukemia quite impressively. But single linkage massively fails to cluster identical cancers together. So in general complete linkage produced the better results.

[33]:
```
# Using the complete linkage that produced better clustering results
from scipy.cluster.hierarchy import cut_tree

# Cut the dendrogram at a specific height
Z = linkage(expression_standard, 'complete', metric='euclidean')
cut_height = 140 # specify the height to cut the dendrogram
clusters = cut_tree(Z, height=cut_height)

# Print the number of clusters obtained
print(f"Number of clusters obtained: {clusters.max() + 1}")
```

```
Number of clusters obtained: 4
```

[39]:
```
from scipy.cluster.hierarchy import fcluster

# Assign samples to clusters
column_labels = fcluster(Z, t=7, criterion='maxclust')

# Create a cross table
contingency_table = pd.crosstab(expression_standard.index, column_labels)
print(contingency_table)
```

```
col_0        1  2  3  4  5  6  7
row_0
BREAST       3  0  2  0  2  0  0
CNS          2  3  0  0  0  0  0
COLON        0  2  0  0  5  0  0
K562A-repro  0  0  0  0  0  0  1
K562B-repro  0  0  0  0  0  0  1
LEUKEMIA     0  0  0  0  0  2  4
MCF7A-repro  0  0  0  0  1  0  0
MCF7D-repro  0  0  0  0  1  0  0
MELANOMA     0  2  6  0  0  0  0
```

```
NSCLC        1  7  0  1  0  0  0
OVARIAN      0  6  0  0  0  0  0
PROSTATE     0  2  0  0  0  0  0
RENAL        1  8  0  0  0  0  0
UNKNOWN      0  1  0  0  0  0  0
```

While it is obvious that it is the data representaion in the same number of clusters, both k-means and hierarchical clustering distribute same number of instances of each cancer type across clusters. It appears that hierarchical cluster performs slightly in an augmented manner such that a cancer is more likely to be in a certain cluster than in a even possibility across multiple clusters (ref. K-means and fcluster contingency table).