Homework 5

Ayman Tawaalai

February 2024

1 Introduction

The purpose of the given assignment is to study and implement Hierarchical Clustering through practical means. The assignment states that the student must one hot encode and ngram the patient sequence data from homework 1. Once that is completed, it will serve as the input features of x. It will be used for clustering. Another task will be do the same clustering but with Principal Component Analysis (PCA).

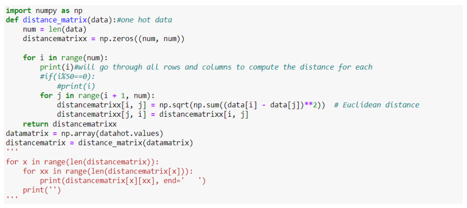
2 Hierarchical Clustering and PCA

Hierarchical clustering is a machine learning model that creates a desired amount of clusters using trees. There are two different approaches, agglomerative, which takes a bottom up approach, and divisive, which takes a top down approach. The main benefit to use this model over K-Means is that it will be easier to recognize outliers.The main distance metric used to create a distance matrix of all the data points is euclidean distance. There are different types of linkages as well for the distance matrix. For example, single linkage will calculate for the smallest distance while avg will calculate for the average. The model will start off with many small clusters which will form together to create larger clusters. This will continue until there are no more clusters that could be merged or if there is a preset k that has been reached.

PCA is used to reduce the amount of dimensions in multi dimensional problems within a dataset. It uses basic concepts from Linear Algebra to accomplish this goal. The first step in PCA is to center the data around the mean for each feature. After a covariance matrix has been created using the mean centered data, the eigenvectors can be found and be ordered in descending order.

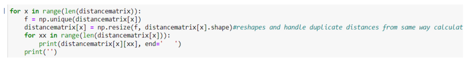
3 Python Implementation

The first task of the assignment is it conduct Clustering analysis on the homework one data set. Loading the data, conducing one hot, and conducting PCA



```python
import numpy as np
def distance_matrix(data):#one hot data
    num = len(data)
    distancematrixx = np.zeros((num, num))

    for i in range(num):
        print(i)#will go through all rows and columns to compute the distance for each
        #if(i%50==0):
            #print(i)
        for j in range(i + 1, num):
            distancematrixx[i, j] = np.sqrt(np.sum((data[i] - data[j])**2))  # Euclidean distance
            distancematrixx[j, i] = distancematrixx[i, j]
    return distancematrixx
datamatrix = np.array(datahot.values)
distancematrix = distance_matrix(datamatrix)
'''
for x in range(len(distancematrix)):
    for xx in range(len(distancematrix[x])):
        print(distancematrix[x][xx], end='   ')
    print('')
'''
```

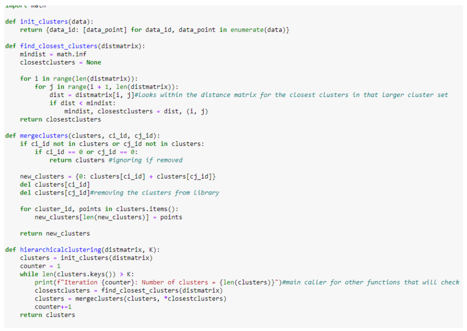
Figure 1: Distance Matrix



```python
for x in range(len(distancematrix)):
    f = np.unique(distancematrix[x])
    distancematrix[x] = np.resize(f, distancematrix[x].shape)#reshapes and handle duplicate distances from same way calculat
    for xx in range(len(distancematrix[x])):
        print(distancematrix[x][xx], end='   ')
    print('')
```

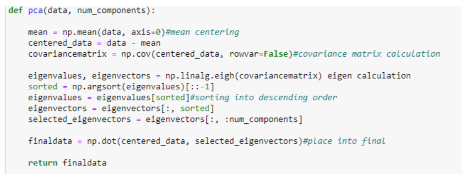
Figure 2: Resize Loop

was done in a similar manner as was done in HW4. The distance matrix function works by taking in the one hot data and performing euclidean distance in a piece wise manner. After the distances have been calculated and placed in the matrix. The matrix is then resized to avoid duplicates within the matrix. For example, 2 to 3 and 3 to 2 will have the same distance. It is for that reason that loop occurs. The hierarchical clustering function will be that main work horse function for this model. It will continue calling functions that call other functions until the desired K amount of clusters have been reached. To make it easier, the distance matrix initializes itself into a dictionary and is returned into clusters. This is used to assist in merging and deleting clusters. The find closest clusters function will use single linkage to get the minimum distance for the closest cluster. One the work horse function is completed, it will return the final clusters back to the user and print it out. The second outlined task is to incorporate PCA into this clustering model. The PCA steps are done in the PCA function. The function follows the outlined steps in section 2, Hierarchical Clustering and PCA, when it is called upon.



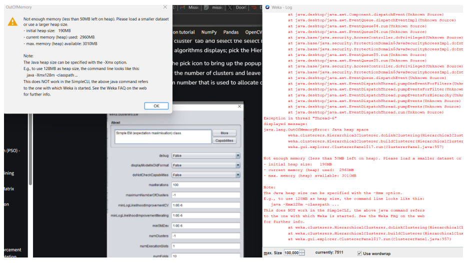
```python
def init_clusters(data):
    return {data_id: [data_point] for data_id, data_point in enumerate(data)}

def find_closest_clusters(distmatrix):
    mindist = math.inf
    closestclusters = None

    for i in range(len(distmatrix)):
        for j in range(i + 1, len(distmatrix)):
            dist = distmatrix[i, j]#looks within the distance matrix for the closest clusters in that larger cluster set
            if dist < mindist:
                mindist, closestclusters = dist, (i, j)
    return closestclusters

def mergeclusters(clusters, ci_id, cj_id):
    if ci_id not in clusters or cj_id not in clusters:
        if ci_id == 0 or cj_id == 0:
            return clusters #ignoring if removed

    new_clusters = {0: clusters[ci_id] + clusters[cj_id]}
    del clusters[ci_id]
    del clusters[cj_id]#removing the clusters from library

    for cluster_id, points in clusters.items():
        new_clusters[len(new_clusters)] = points

    return new_clusters

def hierarchicalclustering(distmatrix, K):
    clusters = init_clusters(distmatrix)
    counter = 1
    while len(clusters.keys()) > K:
        print(f"Iteration {counter}: Number of clusters = {len(clusters)}")#main caller for other functions that will check
        closestclusters = find_closest_clusters(distmatrix)
        clusters = mergeclusters(clusters, *closestclusters)
        counter+=1
    return clusters
```

Figure 3: Hierarchical Clustering python implementation



```python
def pca(data, num_components):

    mean = np.mean(data, axis=0)#mean centering
    centered_data = data - mean
    covariancematrix = np.cov(centered_data, rowvar=False)#covariance matrix calculation

    eigenvalues, eigenvectors = np.linalg.eigh(covariancematrix) eigen calculation
    sorted = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted]#sorting into descending order
    eigenvectors = eigenvectors[:, sorted]
    selected_eigenvectors = eigenvectors[:, :num_components]

    finaldata = np.dot(centered_data, selected_eigenvectors)#place into final

    return finaldata
```

Figure 4: PCA Function



```
OutOfMemory

Not enough memory (less than 50MB left on heap). Please load a smaller dataset
or use a larger heap size.
- initial heap size:   190MB
- current memory (heap) used:  2960MB
- max. memory (heap) available: 3010MB

Note:
The Java heap size can be specified with the -Xmx option.
E.g., to use 128MB as heap size, the command line looks like this:
   java -Xmx128m -classpath ...
This does NOT work in the SimpleCLI, the above java command refers
to the one with which Weka is started. See the Weka FAQ on the web
for further info.
```

```
weka.clusterers.EM
About
Simple EM (expectation maximisation) class.
debug                                  False
displayModelInOldFormat                False
doNotCheckCapabilities                 False
maxIterations                          100
maximumNumberOfClusters                -1
minLogLikelihoodImprovementCV          1.0E-6
minLogLikelihoodImprovementIterating   1.0E-6
```

```
Weka - Log
        at java.desktop/java.awt.Component.dispatchEvent(Unknown Source)
        at java.desktop/java.awt.EventQueue.dispatchEventImpl(Unknown Source)
        at java.desktop/java.awt.EventQueue$4.run(Unknown Source)
        at java.desktop/java.awt.EventQueue$4.run(Unknown Source)
        at java.base/java.security.AccessController.doPrivileged(Unknown Source)
        at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doInt
        at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doInt
        at java.desktop/java.awt.EventQueue$5.run(Unknown Source)
        at java.desktop/java.awt.EventQueue$5.run(Unknown Source)
        at java.base/java.security.AccessController.doPrivileged(Unknown Source)
        at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doInt
        at java.desktop/java.awt.EventQueue.dispatchEvent(Unknown Source)
        at java.desktop/java.awt.EventDispatchThread.pumpOneEventForFilters(Unkn
        at java.desktop/java.awt.EventDispatchThread.pumpEventsForFilter(Unknown
        at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(Unkn
        at java.desktop/java.awt.EventDispatchThread.pumpEvents(Unknown Source)
        at java.desktop/java.awt.EventDispatchThread.pumpEvents(Unknown Source)
        at java.desktop/java.awt.EventDispatchThread.run(Unknown Source)
Exception in thread "Thread-6"
displayed message:
java.lang.OutOfMemoryError: Java heap space
        weka.clusterers.HierarchicalClusterer.doLinkClustering(HierarchicalClust
        weka.clusterers.HierarchicalClusterer.buildClusterer(HierarchicalCluster
        weka.gui.explorer.ClustererPanel$17.run(ClustererPanel.java:957)

Not enough memory (less than 50MB left on heap). Please load a smaller dataset or
- initial heap size:   190MB
- current memory (heap) used:  2960MB
- max. memory (heap) available: 3010MB

Note:
The Java heap size can be specified with the -Xmx option.
E.g., to use 128MB as heap size, the command line looks like this:
   java -Xmx128m -classpath ...
This does NOT work in the SimpleCLI, the above java command refers
to the one with which Weka is started. See the Weka FAQ on the web
for further info.
        at weka.clusterers.HierarchicalClusterer.doLinkClustering(HierarchicalCl
        at weka.clusterers.HierarchicalClusterer.buildClusterer(HierarchicalClus
        at weka.gui.explorer.ClustererPanel$17.run(ClustererPanel.java:957)

max. Size 100,000     currently: 7511     ✓ Use wordwrap
```

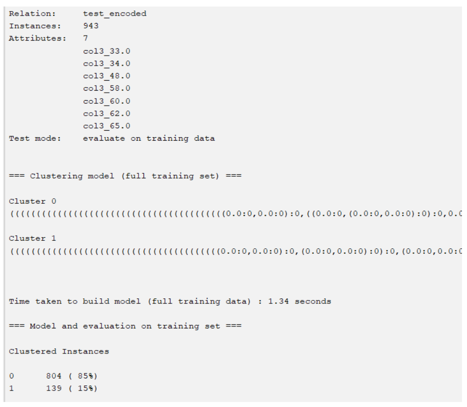
Figure 5: WEKA Crash Log

4 WEKA

There were many issues in trying to get WEKA to run the one hot encoded data. After many hours spent on trying to troubleshoot the issue, WEKA would repeatedly crash. This is because the patient dataset is too large to conduct hierarchical clustering within WEKA. As a result, a smaller subset of the data had to be spliced to get it to run. The subset that will be used is the first patient data. The linkage used in WEKA will be single and it will calculate the distance using Euclidean. The result had all clusters reduced to one.

5 Conclusion

The purpose of the given assignment is to study and implement Hierarchical Clustering and PCA through practical means. The model proved to be highly accurate with the given data and handled outliers very well. However, because of its memory intensive operations it was very time consuming for the given dataset. In the future, it would be important for a machine learning engineer to chose which clustering algorithm it best based on how large the data set is.



```
Relation:     test_encoded
Instances:    943
Attributes:   7
              col3_33.0
              col3_34.0
              col3_48.0
              col3_58.0
              col3_60.0
              col3_62.0
              col3_65.0
Test mode:    evaluate on training data


=== Clustering model (full training set) ===

Cluster 0
(((((((((((((((((((((((((((((((((((((((((0.0:0,0.0:0):0,((0.0:0,(0.0:0,0.0:0):0):0,0.0

Cluster 1
((((((((((((((((((((((((((((((((((((((((0.0:0,0.0:0):0,(0.0:0,0.0:0):0):0,(0.0:0,0.0:0



Time taken to build model (full training data) : 1.34 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      804 ( 85%)
1      139 ( 15%)
```

Figure 6: WEKA output