CNSCC.361 Artificial Intelligence

Lab session 6 Clustering Methods

AIM OF THE SESSION:

- K-means Clustering and Its Application

Clustering of numerical data forms the basis of the data analysis /mining, machine learning, pattern recognition (and, thus, of the AI). It can also be used in many classification and system modeling algorithms. The purpose of clustering is to identify natural groupings of data from large data set by partitioning the data to produce a concise representation of the raw data.

In this lab session we will learn how to identify groups of data with similar characteristics.

1. Kmeans Clustering

1.1. Algorithm Summary

Kmeans algorithm is one of the most widely used clustering algorithms. Kmeans algorithm minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances.
The algorithm starts with initial estimates for the *K* centroids, which can either be randomly generated or randomly selected from the data set. The algorithm, then, iterates the following two steps:

1) Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance.

2) Centroid update step:

In this step, the centroids are recalculated. This is done by taking the mean of all data points assigned to that centroid's cluster.

Kmeans algorithm can generate the locally optimal data partitioning result. However, there are some disadvantages as well:

- *K* is required to be defined in advance (by the user)
- different proximity measures can lead to different results

Proximity Measures

The `pdist2` function can be used to calculate the pairwise distance between two sets of observations (two vectors/multidimensional points).
This calculation is commonly known as distance/ dissimilarity matrix. By default, the `pdist2` function calculates the Euclidean distance between different observations; however you can specify one of several other options (e.g. Mahalonobis, cosine distance etc.).

For example, let us consider the data set, `Inp`, made up of four observations/data items where each one of them is a 2D point (A,B,C,D).

Inp

Data sample	x	y

A	3	2
B	2	4
C	4	2
D	6	1

One can think of `x` and `y` as two features. For example, temperature (T, ^oC) and pressure (P, kPa) or size (L, m) and weight (W,kg), etc.

Let us consider another data set, `Out` of three other observations/2D data vectors, where each one of them is a 2D data point (E,F,G).

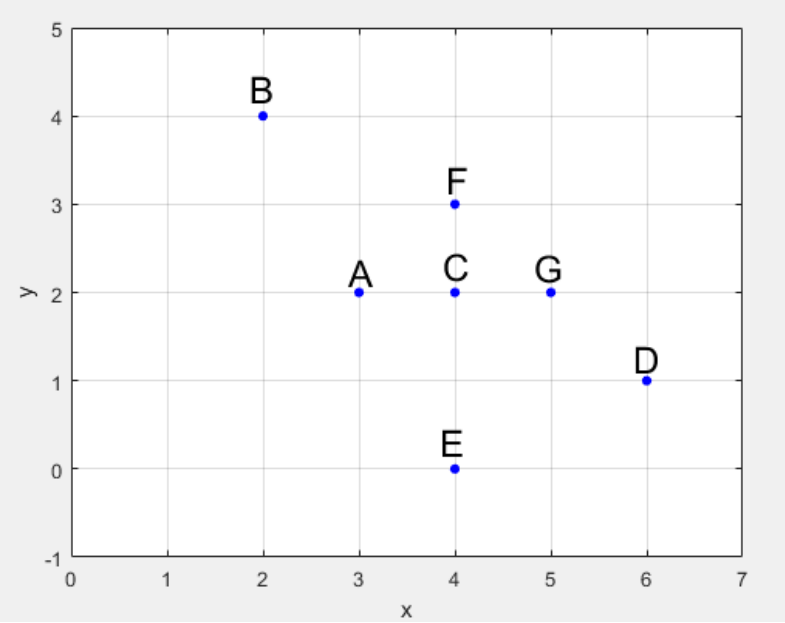
Out

Data sample	x	y
E	4	0
F	4	3
G	5	2

In Matlab, you can define these two datasets as a matrix

```
Int = [3    2;                  A(3;2)
       2    4;                  B(2;4)
       4    2;                  C(4;2)
       6    1;];                D(6;1)

Out = [4    0;                  E(4;0)
       4    3;                  F(4;3)
       5    2;];                G(5;2)
```

and then pass it to `pdist2`.



The `pdist2` function will calculate the Euclidean distances between each pair of observations in the data matrix Inp and data matrix Out.

```
ProxMat = pdist2(Inp,Out)
```

ProxMat

	E	F	G
A	2.2361	1.4142	2.0000
B	4.4721	2.2361	3.6056
C	2.0000	1.0000	1.0000
D	2.2361	2.8284	1.4142

This means that the distance between points A and E is 2.2361:

$d_{AE} = 2.2361$ because $d_{AE} = |A - E| = \sqrt{(3-4)^2 + (2-0)^2}$;

This means that the distance between points A and F is 1.4142:

$d_{AF} = 1.4142$ because $d_{AF} = |A - F| = \sqrt{(3-4)^2 + (2-3)^2}$;

This means that the distance between points A and G is 2.0000:

$d_{AG} = 2.0000$ because $d_{AG} = |A - G| = \sqrt{(3-5)^2 + (2-2)^2}$;

Task 1:

Try to calculate the Euclidean distance bwteen other points manually.

One can further specific the type of distance calculated by the `pdist2` function in the following form:

```
ProxMat = pdist2(Inp,Out,Distance)
```

The chocies of `Distance` can be:

'euclidean' - Euclidean distance (default)

'cityblock' - City Block distance

'cosine' - One minus the cosine of the included angle between observations (treated as vectors)

'hamming' - Hamming distance, percentage of coordinates that differ

For more information, use:

```
help pdist2
```

1.2. Algorithm Implementation

In this section, we will implement the kmeans algorithm step-by-step. We use the Flame dataset (http://cs.joensuu.fi/sipu/datasets/) as an example.

```
load Flame
[L,W] = size(Data); % Getting the size of the dataset
```

Step 0: Initialize K centres

Randomly generate K centres from the dataset (in this example, K=2):

```
K = 2; % Defining the value of K
seq = randperm(L); % Generating a random permutation of the
integers from 1 to L;
Centres = Data(seq(1:K),:); % Selecting randomly two data samples
as centres
```

Step 1: Data assigment

In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance.

```
ProxMat = pdist2(Data, Centres).^2; % Calculating the pairwise
square Euclidean distances between centres and data samples [~,
idx] = min(ProxMat,[],2); % Finding the nearest centre for
each data sample
```

Step 2: Centroid update

In this step, the mean of all data points assigned to each centroid's cluster is calculated to update the centirods.

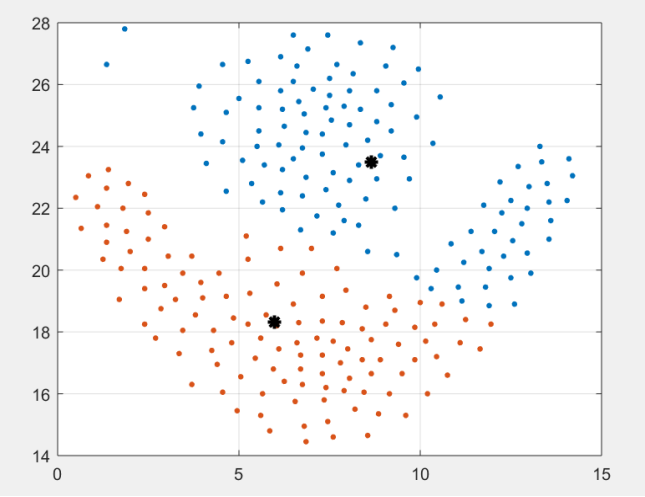
```
Centres_new=zeros(K,W); % Initializing a variable storing new
centres
for i=1:1:K
    Centres_new(i,:)=mean(Data(idx==i,:)); % Calculating the new
centre based on the assigined data samples per centre
end
```

The kmeans algorithm, then, iterates between the steps 1 and 2 until the centres stop changing:

```
while ~isequal(Centres_new,Centres) % The loop continues
until centres remain the same after each iteration

    Centres = Centres_new; % Assign the new centres derived at
the previous iteration as the centres for this iteration

    %% Step 1
    ProxMat = pdist2(Data,Centres).^2; % Calculating the pairwise
square Euclidean distances between centres and data samples
    [~, idx] = min(ProxMat,[],2); % Finding the nearest centre
for each data sample

    %% Step 2
    for i=1:1:K
    Centres_new(i,:)=mean(Data(idx==i,:)); % Calculating the new
centre based on the assigined data samples per centre
    end

end
```

Use the following commands to visualize the clustering result:

```
for i=1:1:K
    plot(Data(idx==i,1),Data(idx==i,2),'.','markersize',10); %
plotting the data samples of each cluster
    hold on
end
grid on
plot(Centres(:,1),Centres(:,2),'k*','markersize',8,'linewidth',2)
; % plotting the cluster centres
hold off
```



Task 2:

Try the following options and visualize the results.

- distance measure: cosine, Hamming
- number of clusters set to: 3, 4

Task 3:

Run the same code for several times to see whether the centres remain the same each time. And explain why.

1.3. Build-in Function available in Matlab

Kmeans algorithm has a build-in function as a part of MATLAB. To call/use it, you can write:

```
[IDX, C, SUMD, Proxmat] = kmeans(Data, K)
```

`Data` is a data matrix. The rows of Data correspond to the data points; columns correspond to features/attributes.

`IDX` is a vector containing the cluster indices for each data point.

`C` is a matrix containing the K cluster centroid locations.

`SUMD` is a vector containing within-cluster sums of point-to-centroid distances.

`Proxmat` is a matrix containing distances from each point to every centroid.

One can further specify the distance measure, in P-dimensional space, that kmeans algorithm aims to minimize:

```
[IDX, C, SUMD, Proxmat] = kmeans(Data, K, 'Distance',val)
```

'Distance' - Distance measure, in P-dimensional space, that kmeans algorithm aims to minimize. The choices are:

'sqeuclidean' - Squared Euclidean distance (the default)

'cityblock' - Sum of absolute differences, a.k.a. L_1 distance

'cosine' - 1- the cosine of the included angle
between points (treated as vectors)

'correlation' - 1- the sample correlation between data points
(treated as sequences of values)

'hamming' - Percentage of bits that differ (only suitable for binary data)

By default, kmeans uses squared Euclidean distances.

For other information, use the following command:

```
help kmeans
```

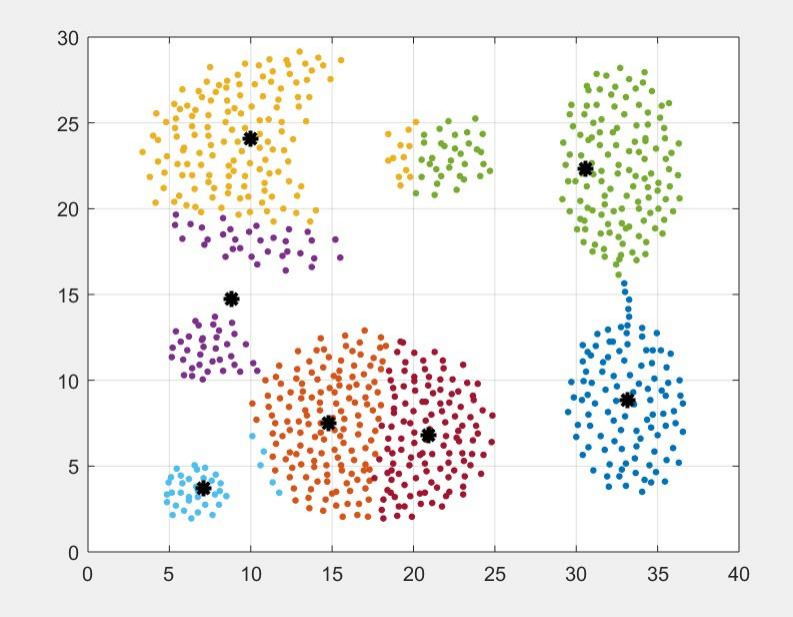
For illustrating the build-in kmeans function, we can use the Aggregation dataset (http://cs.joensuu.fi/sipu/datasets/).

```
[idx,c]=kmeans(Data,7); % conduting kmeans clustering, using the
squared Euclidean distance as a default setting.


centres =

   21.2872   22.9989
   33.1426    8.6193
   32.8231   19.2604
   18.8620    7.1104
    9.2946   22.9527
   32.5813   24.8000
    9.5050    7.5562
```

Task 4:

Visualize the partitioning result using the code given in the previous section

You should get the following result:



Task 5:

Try the following options with kmeans and visualize the results.

- number of clusters set to: 5, 6, 8, 9
- distance measure: cosine, Hamming