# Machine Learning HW05

*Siffi Singh – 0660819*

Program to code k-means clustering, kernel k-means and spectral clustering in MATLAB.

## OUTLINE

In this report, I will be explaining:

**1. Details of Video Submitted**
- **a. 1_Kmeans_clustering.mp4**
- **b. 2_Kernel_kmeans.mp4**
- **c. 3_Spectral_Clustering.mp4**

**2. Explanation of Code Logic**
- **a. K-means Clustering**
- **b. Kernel k-means Clustering**
- **c. Spectral Clustering**

**3. Additional Initializations and Results**

**4. Inferences from Results**

**5. Conclusion**

---

**1. Details of Video Submitted**

---

### 1.a K-means.mp4

The video shows step-by-step clustering procedure visualized for seven different cases:

1. **Input data**: test1_data.txt, **No. of Clusters**: 2
2. **Input data**: test1_data.txt, **No. of Clusters**: 2, **with different centroid initialization.**
3. **Input data**: test2_data.txt, **No. of Clusters**: 2
4. **Input data**: test1_data.txt, **No. of Clusters**: 3
5. **Input data**: test2_data.txt, **No. of Clusters**: 3
6. **Input data**: test1_data.txt, **No. of Clusters**: 5
7. **Input data**: test2_data.txt, **No. of Clusters**: 5
8. **Input data**: generated_data.txt, **No. of Clusters**: 3, this file k-means clustering performed for 3 clusters on **randomly generated data**.

The initialization of the centroid, every time you run the code is done randomly. So, the no. of iterations, every time you execute the code would be different. The k-means clustering, is a well-known partitioning algorithm, and in this code, we are able to visualize ‘k’ no. of clusters(given by the user) and initial centroids(new initial centroid generated every time) for a given set of data points. Additionally the algorithm has also been used to generate data

randomly and the results have been shown in the video.
Following is the centroid updating process using k-means for the first two cases.

Table 1: test1_data.txt for 2 clusters with Centroid initialized in row1

| Iterations | Centroid 1 | | Centroid 2 | |
|---|---|---|---|---|
| Initial Centroid | 1.5353 | 2.6597 | 7.1001 | 3.5083 |
| 1 | 2.3242 | 3.1314 | 2.9074 | -1.4982 |
| 2 | 2.2493 | 2.6995 | 3.0826 | -2.0497 |
| 3 | 2.2180 | 2.5273 | 3.1734 | -2.2335 |
| 4 | 2.1077 | 2.4127 | 3.3259 | -2.3087 |
| 5 | 1.9599 | 2.3524 | 3.4937 | -2.2937 |
| 6 | 1.7361 | 2.2754 | 3.7393 | -2.2359 |
| 7 | 1.5108 | 2.2161 | 3.9534 | -2.1282 |
| 8 | 1.2614 | 2.0745 | 4.2281 | -2.0015 |
| 9 | 0.9697 | 1.9110 | 4.4985 | 0.9078 |
| 10 | 0.8481 | 1.8126 | 4.6238 | -1.6903 |
| 11 | 0.7359 | 1.7528 | 4.6794 | -1.578 |
| 12 | 0.5454 | 1.5976 | 4.8060 | -1.3781 |
| 13 | 0.4176 | 1.3413 | 5.0207 | -1.179 |
| 14 | 0.2181 | 1.1293 | 5.1236 | -0.9237 |
| 15 | 0.0615 | 0.9269 | 5.1749 | -0.6953 |
| 16 | 0.0267 | 0.7527 | 5.2578 | -0.5454 |
| 17 | -0.0029 | 0.7008 | 5.2603 | -0.4902 |
| 18 | 0.0006 | 0.6558 | 5.2826 | -0.4532 |
| 19 | 0.0242 | 0.5725 | 5.3379 | -0.388 |
| 20 | 0.0154 | 0.5261 | 5.3466 | -0.3426 |
| 21 | 0.0328 | 0.4690 | 5.3825 | -0.2942 |
| Converge Point | 0.0169 | 0.4548 | 5.3717 | -0.2764 |
| Converge Point | 0.0169 | 0.4548 | 5.3717 | -0.2764 |

*Figure 1: Table for test1_data for 2clusters*

Now, for the same data, test1_data.txt, we find the labels using differently initialized centroids, and we observe that the same data is not being clustered within 7 iterations only. Following table shows the differently initialized centroids in the first row and the various updations in each iteration until it finally converges in step 7.
Table 2: test1_data.txt for 2 clusters with different Centroid initialized in row1

| Iterations | Centroid 1 | | Centroid 2 | |
|---|---|---|---|---|
| Initial Centroid | 3.2494 | -3.0119 | 6.9491 | -3.0568 |
| 1 | 6.0645 | 1.0648 | 0.8601 | -0.4505 |
| 2 | 6.1086 | 0.5889 | 0.6452 | -0.2167 |
| 3 | 6.0617 | 0.3303 | 0.5407 | -0.0695 |
| 4 | 6.0021 | 0.2542 | 0.4884 | -0.0249 |
| 5 | 5.9568 | 0.1831 | 0.4498 | 0.021 |
| 6 | 5.8949 | 0.1082 | 0.3996 | 0.0724 |
| Converge Point | 5.8631 | 0.1017 | 0.3753 | 0.0769 |
| Converge Point | 5.8631 | 0.1017 | 0.3753 | 0.0769 |

*Figure 2: Table for test1_data for 2 clusters with new initial centroids*

Above is the centroid updating process for each step, we see, at convergence the centroid does not update itself, and that is when we stop the iteration. test2_data.txt for 2 clusters below:

| Iterations | Centroid 1 | | Centroid 2 | |
|---|---|---|---|---|
| Initial Centroid | -8.9181 | -5.1248 | 11.7955 | -0.6237 |
| 1 | -6.6839 | -1.9659 | 2.3806 | 1.9872 |
| 2 | -6.5295 | -3.8555 | 2.6424 | 1.9835 |
| 3 | -6.2724 | -3.6233 | 2.8214 | 2.0606 |
| 4 | -6.1106 | -3.404 | 2.8214 | 2.0904 |
| 5 | -5.8608 | -3.1733 | 3.1917 | 2.1675 |
| 6 | -5.7197 | -2.8689 | 3.4398 | 2.1569 |
| 7 | -5.5243 | -2.5244 | 3.7782 | 2.1524 |
| 8 | -5.3422 | -2.0246 | 4.2826 | 2.0253 |
| 9 | -5.2717 | -1.5982 | 4.6984 | 1.7915 |
| 10 | -5.221 | -1.314 | 4.9561 | 1.587 |
| 11 | -5.1900 | -1.1644 | 5.0835 | 1.4646 |
| 12 | -5.1984 | -1.0369 | 5.1481 | 1.3334 |
| 13 | -5.2198 | -0.8419 | 5.2284 | 1.1218 |
| 14 | -5.2957 | -0.7254 | 5.2028 | 0.9695 |
| 15 | -5.4063 | -0.5376 | 5.1589 | 0.7374 |
| 16 | -5.5095 | -0.3591 | 5.1051 | 0.5282 |
| 17 | -5.5485 | -0.2294 | 5.0917 | 0.3873 |
| 18 | -5.5046 | -0.1228 | 5.1549 | 0.2793 |
| 19 | -5.4814 | -0.0622 | 5.1852 | 0.2157 |
| Converge Point | -5.5086 | -0.0115 | 5.1592 | 0.1599 |
| Converge Point | -5.5086 | -0.0115 | 5.1592 | 0.1599 |

*Figure 3: test2_data.txt for 2 clusters*

The function k-means partitions data into k mutually exclusive clusters, and returns the index of the cluster to which it has assigned each observation.
Unlike hierarchical clustering, k-means clustering operates on actual observations (rather than the larger set of dissimilarity measures), and creates a single level of clusters. The code has a straightforward logic and performs well in comparison to the built-in function in MATLAB.

### 1.b Kernel k-means.mp4

The video shows step-by-step clustering procedure visualized for seven different cases:
1. **Input data**: test1_data.txt, **No. of Clusters**: 2
2. **Input data**: test2_data.txt, **No. of Clusters**: 2
3. **Input data**: test1_data.txt, **No. of Clusters**: 3
4. **Input data**: test2_data.txt, **No. of Clusters**: 4
5. **Input data**: generated_data.txt, **No. of Clusters**: 3, this file k-means clustering performed for 3 clusters on **randomly generated data**.

The kernel k-means is a simple video visualizing the data points based on the algorithm's analysis of label for each data set. Kernel K-means is an efficient algorithm to separate non-linear data with optimal calculations. The results and the step-by-step clustering process has been shown in the video.

### 1.c Spectral.mp4

The video shows step-by-step clustering procedure visualized for seven different cases:
1. **Input data**: test1_data.txt, **No. of Clusters**: 3
2. **Input data**: test2_data.txt, **No. of Clusters**: 2
3. **Input data**: test1_data.txt, **No. of Clusters**: 4
4. **Input data**: test2_data.txt, **No. of Clusters**: 3
5. **Input data**: test2_data.txt, **No. of Clusters**: 4
6. **Input data**: generated_data.txt, **No. of Clusters**: 3, this file k-means clustering performed for 3 clusters on **randomly generated data**.

In spectral clustering video, in all the cases we can see, that the step-by-step visualization has been projected into 3D space(because of the high-dimension being reduced to low-dimension), and after evaluation of final labels for all data points we have plotted the 2D diagram of each data correspondingly colored according to its label.

## 2. Explanation of Code Logic

### 1.a K-means Clustering

The code for K-means has the following files:

**Main.m**: This is the main function where all the other functions are called, and the plotting of initial data points has been done.

Within this functions, we perform tasks like reading the .txt file into an array and using that array to pass it to k-means function for clustering the data points.

```
X = dlmread('test1_data.txt');
label = dlmread('test1_ground.txt');
[label,C] = kmeans(data, 2); %calling kmeans function
```

Test1_data and Test2_data are both non-linear data, considering that, k-means is definitely not the best method to cluster data.

**kmeans.m**: This is k-means function that takes in two inputs as parameters, the data points and the no. of clusters. Although, there is a built-in function for k-means but for this homework I have used a customized version of it, so as to plot each step for the **Visualizer.**
After you pick some value of "k" to test and initial centroids are selected, the algorithm iterates between two steps as it converges to an answer:

1. A data assignment step, where each data point is assigned to the nearest cluster centroid
2. A centroid update step, where the centroids are updated to reflect the data in each cluster

until stopping criteria is met.

```
function [label, mu] = kmeans(X, m)
```

As an output this function returns two values, the labels for the data, and the co-ordinates of the centroids.

```
while any(label ~= last)
    [~,~,last(:)] = unique(label);                  % remove empty clusters
    mu = X*normalize(sparse(idx,last,1),1);         % compute cluster centers
    [val,label] = min(dot(mu,mu,1)'/2-mu'*X,[],1);  % assign sample labels
end
```

In the above part of code, we compute 'mu', that is the centroid, in every iteration and we calculate the distance of each point, and take mean again and we repeat this process until convergence. The convergence criteria is that the labels of any two steps becomes exactly same, we stop the process!

```
        for i = 1:c
            idc = label==i;
            scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1));
        end
```

And after getting the final labels we plot the data, filling different colors based on its label.

**kseeds.m**: This function is to assign value to centroid in every iteration and to ensure that the code works with good initialization.

```
mu = X(:,ceil(n*rand));
```

```
for i = 2:k
    D = min(D,sum((X-mu(:,i-1)).^2,1));
    mu(:,i) = X(:,randp(D));
end
```

**normalize.m**: This function is used to compute cluster centres, which updates the centroid in each iteration.

```
Y = X./sum(X,dim);  % Normalize the vectors
```

**plotClass.m**: The function plotClass is used to visualize data points after clusterterization. This function is specifically designed for only 2D and 3D data points, for data of higher dimension, you need to reshape the data before plotting it.

```
color = 'brgmcyk';
m = length(color);
c = max(label);

switch d
    case 2
        view(2);
        for i = 1:c
            idc = label==i;
%             plot(X(1,label==i),X(2,label==i),['.'
color(i)],'MarkerSize',15);
            scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1));
            mu(i,1,1) = mean(X(1,idc));
            mu(i,1,2) = mean(X(2,idc));
        end
    case 3
        view(3);
        for i = 1:c
            idc = label==i;
%             plot3(X(1,idc),X(2,idci),X(3,idc),['.' idc],'MarkerSize',15);
            scatter3(X(1,idc),X(2,idc),X(3,idc),36,color(mod(i-1,m)+1));
        end
end
```

The above plotClass.m is called for each iteration to plot step-by-step visualization of the clustering procedure.

**kmeansRnd.m**: And, at last the function that has been used to randomly generate data, for the last case shown in video, for given no. of clusters('k'), we find from the generated data, that which data belongs to which cluster.

```
X = randn(d,n); % Generate samples from a Gaussian mixture distribution with
common variances (kmeans model).
mu = randn(d,k)*beta;
```

## 1.b Kernel k-means Clustering

**main.m**: This is the main function, which the base method from where we call all other necessary functions. The kernel method used here is the RBF kernel, and the result has been computed for different cases to see how efficiently can kernel kmeans cluster data with given no of clusters 'k' and data read from the given input files(or randomly) generated. Similar to

above method, reading the data from .txt file into an array is the first step.

```
X = dlmread('test2_data.txt');
X = X';
```

**Initializing the labels**: In the next step, we initialize the labels, of the data points by using 'rand' function as shown below, hence, after this step, init has the initial labels which would be helpful in the convergence condition.

```
init = ceil(k*rand(1,n));
```

**knkmeans.m**: This function is the kernel kmeans function. It takes in three arguments, the original unlabeled data points, the init, that is the initial randomly assigned labels of the data and the kind of kernel method you wish to use(linear, guassian(RBF) or polynomial) kernel.

```
[y,model,mse] = knKmeans(X,init,@knGauss); %Calling kernel kmeans function
```

The function returns the array of labels and a structure that contains three fields, the labels, the data, the label, and the kernel method used.

**Computing the Kernel matrix**: Inside the knKmeans function, we at first validate the dimensions of data sent as parameter, and next we find the 'K' the gram or the kernel matrix.

```
K = kn(X,X);
```

where,

```
kn = @knGauss;
```

**knGuass.m**: This function is responsible for computing the gram matrix using the RBF kernel.

```
% Gaussian (RBF) kernel K = exp(-|x-y|/(2s));
```

This function takes in two input parameters the data matrix and the data matrix and returns a single matrix 'K' the kernel matrix to the main function.

```
D = bsxfun(@plus,dot(X,X,1)',dot(Y,Y,1))-2*(X'*Y);
K = exp(D/(-2*s^2));
```

**Updating the centroids**: After computation of the kernel matrix we begin with the process of clustering. The convergence condition is taken to the point where our last computed labels matches the current computed labels for the data.

```
while any(label ~= last)      % convergence condition
    [~,~,last(:)] = unique(label);   % remove empty clusters
    E = sparse(last,1:n,1);
    E = E./sum(E,2);     % computing the centroids to find labels
    T = E*K;
    [val, label] = max(T-dot(T,E,2)/2,[],1);     % labels updation
end
```

**plotClass.m:** Plotting in kernel k-means is fairly easy and similar to - process, we pass two arguments the data and the label and get the visualization of data points clustered into different sets based on its label, represented by different colors.

```
scatter(X(1,idc),X(2,idc),36,color(mod(i-1,m)+1));
```

```
            mu(i,1,1) = mean(X(1,idc));
            mu(i,1,2) = mean(X(2,idc));
```

Just a small updation from k-means here is that to plot the centroid in every step we calculate the mean of all points of a particular label and plot it on the visualizer.

```
plot(mu(:,1),mu(:,2),'kx','MarkerSize',9,'LineWidth',3);
```

## 1.c Spectral Clustering

The code for spectral clustering has the following files:

**main.m:** This is the main function from where all the other functions have been called, the functionality and detailed analysis of the different functions have been done in the following section. In main, we read the input data file(or call a generate_data function), using dlmread we read the file into an array. This array is further, sent as a parameter to functions for clustering the data.

```
data = dlmread('test1_data.txt');
data = data;
```

Although, Spectral clustering has a number of computations, it still outplays other algorithms in non-linear data clustering, which is clear from the video results. The given data test1_data and test2_data are both non-linear and a to obtain a proper clustering result is difficult in other cases, but with spectral clustering we get nice result.

**CalculateAffinity.m**: On the matrix of data from above which is d*n, we compute the similarity matrix, that is the difference between every point to every other point in the data space. The similarity matrix is also the weight matrix which is used to compute the Laplacian.

```
% calculate the affinity / similarity matrix (W)
affinity = CalculateAffinity(data);
Inside the function:
for i=1:size(data,1)
    for j=1:size(data,1)
        dist = sqrt((data(i,1) - data(j,1))^2 + (data(i,2) - data(j,2))^2);
        affinity(i,j) = exp(-dist/(2*sigma^2));
    end
end
```

**Computing degree matrix**: Next we compute the degree matrix ‘D’ which is the count of number of nodes outgoing for each node.

```
% compute the degree matrix (D)
for i=1:size(affinity,1)
    D(i,i) = sum(affinity(i,:));
end
```

**Computing Laplacian**: Next we compute the normalized laplacian from the already computed degree matrix and affinity or similarity matrix, the formula for computing the normalized graoh laplacian is:

```
                        NL1 = D^(-1/2) .* L .* D^(-1/2);
for i=1:size(affinity,1)
    for j=1:size(affinity,2)
        NL1(i,j) = affinity(i,j) / (sqrt(D(i,i)) * sqrt(D(j,j)));
    end
end
```

**Eigen Value Decomposition**: In the next step, we do the Eigen value decomposition, using the built-in 'eig' function of MATLAB. The advantage of MATLAB is that it has many useful pre-defined functions, which makes coding convenient and faster.

```
% perform the eigen value decomposition
[eigVectors,eigValues] = eig(NL1);
```

**Selection of Eigen Vectors**: In the next step, we select the no. of eigen vectors, to perform kmeans clustering on each row of the matrix made of eigen vectors.

```
% select k largest eigen vectors
k = 3;
nEigVec = eigVectors(:,(size(eigVectors,1)-(k-1)): size(eigVectors,1));
```

**Performing k-means on each row**: In the next step, we construct the normalized matrix 'U' which is nothing but the compilation of all the different eigen vectors, or rather the first 'k' largest eigen vectors that we have chosen in the previous step.

```
% construct the normalized matrix U from the obtained eigen vectors
for i=1:size(nEigVec,1)
    n = sqrt(sum(nEigVec(i,:).^2));
    U(i,:) = nEigVec(i,:) ./ n;
end
```

On each row we perform clustering, if there are two data points that belong to the same cluster then they will be mapped to the same(or closer) data point in the feature space.

NOTE: the kernel k-means function called here is 'not' the built-in function, rather it is the same function that we have written in the first part where we compute the k-means clustering.

```
% perform kmeans clustering on the matrix U
[IDX,C] = kmeans(U,3);
```

**Plotting**: This step in Spectral clustering process is not a usual plotting process because of the dimension of the data after no. of eigen vector selection. So, here we see, that we cannot use the regular plotClass that we have been using till now, so we transform or squeeze the high-dimension data to low-dimension space. And plot the diagram in 3D for each step, as shown in the video, and finally after all the index i.e. the label has been computed, we can simply plot it using the plot function, which is why the final result is in 2D, while the intermediate steps are plotted in 3D.

```
for i=1:size(IDX,1)
    if IDX(i,1) == 1
        plot(data(i,1),data(i,2),'bo');
    elseif IDX(i,1) == 2
        plot(data(i,1),data(i,2),'ro');
    elseif IDX(i,1) == 3
        plot(data(i,1),data(i,2),'go');
    else
        plot(data(i,1),data(i,2),'yo');
    end
end
```

Hence, above is the code explanation for each of the three algorithms, for any other further clarification, the code files attached can be referred.

## 3. Additional Initializations and Results

Different number of clusters: In all of the above three parts, the data has been clustered for 2, 3, 4, 5… number of clusters and the result is added to the video. Trying different number of clusters has given more insight on how correctly is the algorithm working.

Initialization of centroids: As shown in the video, good data initialization technique is followed to get best results. As an example, in the kmeans clustering video, the test1_data has been initialized with different initial centroid points and the result has been shown in the table above as well as in the video.

Initialization of data points: For the visualization of the algorithms implemented above on data points other than the ones provided, a separate function has been added to generate data randomly as and when needed.

## 4. Inferences from results

Kmeans:

The k-means algorithm is a versatile and simple technique to find clusters or groups in your data. Often, you may hypothesize that groups exist in your data, but those groups haven't been identified or labeled ahead of time.

The algorithm is composed of the following steps:

a. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
b. Assign each object to the group that has the closest centroid.
c. When all objects have been assigned, recalculate the positions of the K centroids.
d. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

### Kernel K-means:

K-means will **fail** to effectively cluster these, even when the true number of clusters K is known to the algorithm.

In some cases, it is more efficient to use weighted kernel k-means by transforming the problem as if in this case we have used the RBF kernel to cluster data.

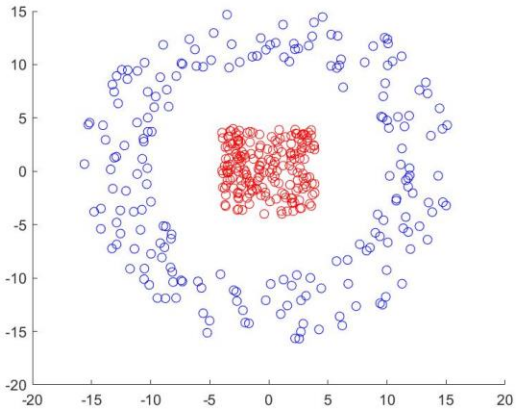


*Figure 4: Non-Linear data clustered*

### Spectral Clustering:

Spectral clustering works by first transforming the data from Cartesian space into similarity space and then clustering in similarity space.

The mapping from Cartesian space to similarity space is facilitated by the creation and diagonalization of the similarity matrix.

When you stack the lowest k-eigenvectors of this matrix as columns in a new matrix and normalize them, the rows of the matrix are the new coordinates for each data point in the new space.The K-means part is run because the eigenvectors can be degenerate, and the clusters do not have to be so cleanly separated.

The similarity transformation reduces the dimensionality of space and, loosely speaking, pre-clusters the data into orthogonal dimensions.

## 5. Conclusion

In most of the cases, the kind of algorithm that would be the best for clustering depends on the nature of the problem you are trying to solve.

The kmeans algorithm is significantly sensitive to the initial randomly selected cluster centres, which is why, we need to run the k-means clustering algorithm for a range of k values and compare the results to find the value of k that best represents the number of clusters in your data.

In spectral clustering, if we do not use a kernel, we get linearly separated data. Non-linear clusters cannot be separated this way.

Although, Spectral clustering requires you to do some additional steps and provide a similarity matrix, it still is the most powerful and used algorithm for clustering data.