

# Linear Algebra Applications in Data Clustering: A Python Implementation

*Sara Farahani, Tanaz Ghahremani, Mohammad Mirzaee*

*Directed by Mahdi Lotfi*

In this exploration, we exploit fundamental linear algebra principles to implement and validate clustering algorithms, specifically focusing on K-means and Spectral Clustering.


For this project, you are permitted to use Python and the NumPy library. Usage of any machine learning libraries is prohibited, unless we explicitly specify in the task description.

# 1. Implementation of K-means Algorithm

In this task, your objective is to finalize the `kmeans.py` file, completing the implementation of the K-means clustering algorithm.

```
def k_means_clustering(data, k, max_iterations=100):  
    # TODO: Randomly initialize centroids  
    # TODO: Iterate until convergence and update centroids  
    # TODO: Return labels and centroids
```

Where `data` is a  $m \times n$  numpy array representing  $m$  data points each of dimension  $n$ , and  $k$  denotes the desired number of clusters. There is an optional parameter `max_iterations` that determines the maximum number of iterations. The function is expected to return an  $m$ -dimensional vector of labels and a  $k \times n$  array of centroids representing the cluster centers.

 **Implementation note:** You need to vectorize your code using concepts from the labs. Avoid using loops for vector calculations; only use loops for the outermost iteration of the algorithms, alternating between computing the centroid and assigning points to clusters. Violating this incurs a **50% deduction from the related task score**.

## 2. Clustering by Spectral Methods

This task involves using spectral clustering to utilize the data's similarity matrix spectrum for dimensionality reduction before clustering by completing the `spectral.py` file.

### 2.1. Calculate Laplacian matrix

The initial step in spectral clustering involves computing the Laplacian matrix from the affinity (similarity) matrix. We recommend using the symmetric normalized Laplacian, which can be obtained through the following formula:

$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

Where:

- $I$  is the identity matrix,
- $D$  is the degree matrix, a diagonal matrix with  $D_{ii}$  representing the sum of weights for the  $i$ -th row (or column) in the affinity matrix  $A$ ,
- $A$  is an  $m \times m$  symmetric affinity matrix representing a pairwise measure of similarity between data points.

Complete the `laplacian` function to compute the Laplacian of the affinity matrix  $A$ .

```
def laplacian(A):  
    # TODO: Calculate degree matrix  
  
    # TODO: Calculate the inverse square root of the symmetric matrix  
  
    # TODO: Return symmetric normalized Laplacian matrix
```

## 2.2. Implement Spectral clustering

We will finalize the algorithm implementation by completing the `spectral_clustering` method.

```
def spectral_clustering(affinity, k):  
    # TODO: Compute Laplacian matrix  
  
    # TODO: Compute the first k eigenvectors of the Laplacian matrix  
  
    # TODO: Apply K-means clustering on the selected eigenvectors  
  
    # TODO: Return cluster labels
```

Employ your `laplacian` function for Laplacian calculation. It is imperative to utilize your self-implemented `k_means_clustering` algorithm for k-means clustering. If facing challenges in the previous section, consider using the `sklearn.cluster.KMeans`. You may use the `linalg` package from NumPy for computing eigenvalues and eigenvectors.

## 3. Evaluate Your Clustering Algorithms!

Now, it's time to compare and test your clustering algorithms. However, before you start, let's implement some prerequisites by writing some stuff in the `validate.py` file.



### 3.1. Construct the Affinity matrix

Spectral clustering requires an affinity matrix to function. Initially, we will create an affinity matrix from our data by completing the `construct_affinity_matrix` function.

There are several methods available for constructing the affinity matrix, and in this context, you are tasked with implementing the following methods.

- **Gaussian Kernel (RBF Kernel):** Define a similarity measure based on the Gaussian (Radial Basis Function - RBF) kernel:

$$A_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

where  $x_i$  and  $x_j$  are data points,  $\| \cdot \|$  denotes the Euclidean distance, and  $\sigma$  is a bandwidth parameter controlling the width of the Gaussian.

- **k-Nearest Neighbors (KNN):** Connect each data point to its  $k$  nearest neighbors. The affinity matrix  $A$  is binary, with  $A_{ij} = 1$  if  $x_j$  is among the  $k$  nearest neighbors of  $x_i$ , and  $A_{ij} = 0$  otherwise. Make the matrix symmetric afterward.

```
def construct_affinity_matrix(data, affinity_type, *, k=3, sigma=1.0):  
    # TODO: Compute pairwise distances  
  
    if affinity_type == 'knn':  
        # TODO: Find k nearest neighbors for each point  
        # TODO: Construct symmetric affinity matrix  
        # TODO: Return affinity matrix  
  
    elif affinity_type == 'rbf':  
        # TODO: Apply RBF kernel  
        # TODO: Return affinity matrix  
  
    else:  
        raise Exception("invalid affinity matrix type")
```

### 3.2. Implement evaluation metrics

Congratulations! You are now ready to execute your clustering algorithms. Imagine running these algorithms on the following incredibly straightforward dataset:

Feature vector	Ground truth label
$p_1$	0
$p_2$	0
$p_3$	1

What outcomes can be expected from clustering algorithms? Possible results include:

Feature vector	Predicted label 1	Predicted label 2
$p_1$	0	1
$p_2$	0	1
$p_3$	1	0

Observe that both predicted labels exhibit accurate clustering with 100% precision!

Your objective is to finalize the function `clustering_score` to quantify the accuracy of algorithms, reflecting the ratio of correctly clustered points, while addressing the described issue. You could employ simple techniques like brute-force and greedy algorithms, or implement well-known metrics such as *Normalized Mutual Information (NMI)* or *Adjusted Rand Index (ARI)* to receive **extra credit**.

```
def clustering_score(true_labels, predicted_labels):  
    # TODO: Calculate and return clustering score
```

### 3.3. Compare & Visualize algorithms

Now, let's bring everything together and run the `validate.py` file.

Find three diverse toy datasets in the `datasets` directory. Your objective is to complete the code, creating a 3×4 grid with 12 scatter plots. Rows depict different datasets, and columns showcase the results of each algorithm alongside the ground truth labels.

Keep in mind to use different colors for separate clusters in each scatter plot.

```
if __name__ == "__main__":  
    datasets = ['blobs', 'circles', 'moons']  
  
    # TODO: Create and configure plot  
  
    for ds_name in datasets:  
        dataset = np.load("datasets/%s.npz" % ds_name)  
  
        X = dataset['data']      # feature points  
  
        y = dataset['target']    # ground truth labels  
  
        n = len(np.unique(y))    # number of clusters  
  
        k = 3  
  
        sigma = 1.0  
  
        y_km, _ = k_means_clustering(X, n)  
  
        Arbf = construct_affinity_matrix(X, 'rbf', sigma=sigma)  
  
        y_rbf = spectral_clustering(Arbf, n)  
  
        Aknn = construct_affinity_matrix(X, 'knn', k=k)  
  
        y_knn = spectral_clustering(Aknn, n)  
  
        print("K-means on %s:" % ds_name, clustering_score(y, y_km))  
  
        print("RBF affinity on %s:" % ds_name, clustering_score(y, y_rbf))  
  
        print("KNN affinity on %s:" % ds_name, clustering_score(y, y_knn))  
  
        # TODO: Create subplots  
  
    # TODO: Show subplots
```

Besides visualization, you need to compare the accuracy of the algorithms.

Finally, you should end up with the following accuracy report:

	K-means	RBF + Spectral	KNN + Spectral
Blobs			
Circles			
Moons			

💡 **Note:** Explore different `sigma` and `k` values to find the suitable accuracy. Observe how adjusting these values influences clustering results and why.

## 4. Cluster Non-Euclidean Data

### References & Resources

- [K-means clustering](#)
- [Spectral clustering](#)
- [Point cloud](#)
- [Linear Algebra and Learning from Data, by Gilbert Strang, 2019](#) IV.7.
- [Lecture 35](#) of [MIT 18.065, 2018](#)