

School of Computer Science Engineering and Technology
Assignment-10

Course- B.Tech

Code-23CS301PC206

Year- 2024-2025

Date- 23-09-2024

Type- Core

Course Name- Artificial Intelligence & Machine Learning

Semester- Even, **Instructor:** Prof. E.L.N. Kiran

Batch- AIML-A,B

1 Implement K-Means Clustering using Synthetic Data

1 – Import the required Python, Pandas, Matplotlib, Seaborn packages. [CO1]

Problem: you have a multidimensional set of data (such as a set of hidden unit activations) and you want to see which points are closest to others. The k-means algorithm searches for a pre-determined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:

- The "cluster center" is the arithmetic mean of all the points belonging to the cluster.
- Each point is closer to its own cluster center than to other cluster centers.

2 –Create synthetic dataset of unlabeled blobs. [CO2]

The dataset would be synthesized using `sklearn.datasets.samples_generator` from the `sklearn` package. You will import *binary large objects- blobs* to form clusters from the synthetic dataset.

3 –Import KMeans class from Scikit-learn and fit the data. [CO3]

Verify the syntentic dataset and fit the data to the K-Means model.

4: Visualize the fitted data by coloring the blobs by assigned label numbers. [CO3]

We will use the `c` argument in the `plt.scatter()` function. We will also try to make the cluster centers prominent.

How K-Means is a special case of Expectation-maximization (EM) algorithm. [CO4]

Expectation-maximization (EM) is a powerful algorithm that comes up in a variety of contexts within data science. k-means is a particularly simple and special case of this more general algorithm. The basic algorithmic flow of k-means is to,

- Guess some cluster center (initialization)

- Repeat following steps untill converged,
- E-step: assign points to the nearest cluster center
- M-Step: set the cluster centers to the mean

Here the "E-step" or "Expectation step" involves updating our expectation of which cluster each point belongs to.

The "M-step" or "Maximization step" involves maximizing some fitness function that defines the location of the cluster centers. In the case of k-means, that maximization is accomplished by taking a simple mean of the data in each cluster.

Part 2: Implementing K-Means from Scratch. [CO4]

By taking from `sklearn.metrics import pairwise_distances_argmin` we can implement the K-Means from scratch in 3 steps as mentioned earlier.

- Randomly choose the clusters
- Assign Labels based on closest cluster center
- Find new centers by computing the means of points close to the cluster center.
- Check for convergence point.
- Plot the centers and labels using scatter plot.

Code to Implement K-Means from Scratch

```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
    # 1. Randomly choose clusters
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
        # 2a. Assign labels based on closest center
        labels = pairwise_distances_argmin(X, centers)

        # 2b. Find new centers from means of points
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])

        # 2c. Check for convergence
        if np.all(centers == new_centers):
            break
        centers = new_centers
```

```

return centers, labels

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels,
s=50, cmap='viridis');

```

Although the E–M procedure is guaranteed to improve the result in each step, there is no assurance that it will lead to the global best solution. The initialization is important and particularly bad initialization can sometimes lead to clearly sub-optimal clustering.

How many Number of clusters?

A common challenge with k-means is that you must tell it how many clusters you expect. It cannot learn the number of clusters from the data.

If we force the k-means to look for 6 clusters instead of 4, it will come back with 6 but they may not be what we are looking for!

Some methods like **elbow** and **silhouette analysis** can be used to gauge a good number of clusters.

Limitation of K-Means Clustering

k-means algorithm will often be ineffective if the clusters have complicated geometries. In particular, the boundaries between k-means clusters will always be linear, which means that it will fail for more complicated boundaries. Verify the limitation using the code:

```

from sklearn.datasets import make_moons
X, y = make_moons(200, noise=.05, random_state=0)
labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');

```

Part 3: Kernel Transformation Trick. [CO4]

The situation above is reminiscent of the Support Vector Machines, where we use a kernel transformation to project the data into a higher dimension where a linear separation is possible. We might imagine using the same trick to allow k-means to discover non-linear boundaries.

One version of this kernelized k-means is implemented in **Scikit-Learn** within the **SpectralClustering** estimator. It uses the graph of nearest neighbors to compute a higher-dimensional representation of the data, and then assigns labels using a k-means algorithm using the following code:

```

from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',

```

```
assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```