PMDS505P Data Mining and Machine Learning Experiment 11

April 2025

1 Work to do today

Note: Make a single pdf file of the work you are doing in jupyter notebook. Upload with proper format. Please mention your name and roll no properly with Experiment number in the first page of your submission.

Q1. **Clustering:** Today, we will look at three clustering algorithms mainly

K-means clustering

Agglomerative hierarchical clustering

DBSCAN

- Let's construct some synthetic data initially. Create synthetic data with make_bloobs function with say 1000 points, 4 centers, and std deviation of cluster to be 1.

  ```
  import numpy as np
  import matplotlib.pyplot as plt
  import seaborn as sns
  from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
  from sklearn.datasets import make_blobs
  from sklearn.preprocessing import StandardScaler


  X, _ = make_blobs(n_samples=300, centers=4, cluster_std=2.0, random_state=42)
  X = StandardScaler().fit_transform(X)
  ```

- Now, do the scatter plot for the data.

- Now use the k-means algorithm and plot the data to see the clusters formed based on K=4. Which means you are looking for 4 clusters in the data that you have.

  ```
  k = 4
  kmeans = KMeans(n_clusters=k, random_state=42)
  kmeans_labels = kmeans.fit_predict(X)
  ```

```
plt.figure(figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap=’viridis’, edgecolors=’k’)
plt.title(’K-Means Clustering’)
plt.show()
```

- Now you can change the number of clusters to 3 and do a scatter plot to see the difference.

- Next, we will go for **Agglomerative hierarchical clustering**. This is a bottom-up approach where each data point starts as its own cluster. Clusters are iteratively merged based on their similarity until a single cluster remains. The dendrogram visually represents how clusters are merged at different distance levels.

- In Agglomerative Hierarchical Clustering, linkage methods determine how distances between clusters are measured when merging them.

  Single Linkage (’single’)

  Merges clusters based on the shortest distance between any two points in different clusters. Forms long, chain-like clusters and is sensitive to outliers. Good for detecting elongated or irregular clusters.

  Ward Linkage (’ward’)

  Minimizes the variance within merged clusters. Commonly used due to its effectiveness in forming well-separated groups.

  Complete Linkage (’complete’)

  Merges clusters based on the maximum distance between points in different clusters. Tends to create compact clusters. Less sensitive to outliers than single linkage.

  Average Linkage (’average’)

  Merges clusters based on the average distance between points in different clusters. Balances between single and complete linkage. Suitable for datasets where clusters have varying densities.

- we will use the following code to determine clusters in the data.

  ```
  linkage_methods = [’single’,’ward’, ’complete’, ’average’]
  fig, axes = plt.subplots(1, 4, figsize=(15, 5))
  for i, method in enumerate(linkage_methods):
  hierarchical = AgglomerativeClustering(n_clusters=4, linkage=method)
  labels = hierarchical.fit_predict(X)
  axes[i].scatter(X[:, 0], X[:, 1], c=labels, cmap=’viridis’, edgecolors=’k’)
  axes[i].set_title(f’Hierarchical (method linkage)’)
  plt.show()
  ```

- Now we can plot the dendrograms in each case. These give an idea of how the clusters are formed step by step from bottom to top.

  ```
  fig, axes = plt.subplots(1, 4, figsize=(20, 5))
  for i, method in enumerate(linkage_methods):
  ```

```
linkage_matrix = linkage(X, method=method)
axes[i].set_title(f'Dendrogram (method linkage)')
dendrogram(linkage_matrix, ax=axes[i])
plt.show()
```

- Next we will see how DBSCAN clustering can be used. DBSCAN is a density-based clustering algorithm that groups together closely packed points, marking points that are in low-density regions as outliers. It is particularly useful for identifying clusters of varying shapes and sizes in a dataset, especially when there is noise or outliers.

  Key Parameters in DBSCAN:
  eps (epsilon):

  Defines the maximum distance between two samples for them to be considered as part of the same neighborhood.

  A small value for eps means only very close points will be clustered together, potentially leading to more noise being classified as outliers.

  A large value for eps might result in multiple clusters being merged into one large cluster.

  min_samples:

  The minimum number of points required to form a dense region (cluster).

  This parameter determines the minimum number of points that must be within a given eps distance for a cluster to be considered valid.

  DBSCAN Clustering Process:
  Core Points:

  A point is considered a core point if it has at least min_samples points (including itself) within its eps neighborhood.

  Border Points:

  A point is considered a border point if it is within the eps neighborhood of a core point but does not have enough points within its own neighborhood to be considered a core point.

  Noise Points:

  A point is considered noise if it is neither a core point nor a border point (i.e., it does not meet the min_samples and eps criteria).

  How DBSCAN Works: Initialization:

  DBSCAN begins by selecting an unvisited point and checking if it is a core point.

  Cluster Expansion:

  If the point is a core point, DBSCAN expands the cluster by recursively adding all reachable points within the eps distance. These reachable points will be assigned to the same cluster.

Border and Noise Points:

Border points are assigned to the cluster of the core point they are reachable from.

Points that are neither core nor border points are classified as noise.

```
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
eps_values = [0.3, 0.5, 0.7]
for i, eps in enumerate(eps_values):
dbscan = DBSCAN(eps=eps, min_samples=5)
labels = dbscan.fit_predict(X)
axes[i].scatter(X[:, 0], X[:, 1], c=labels, cmap=’viridis’, edgecolors=’k’)
axes[i].set_title(f’DBSCAN (eps=eps)’)
plt.show()
```