CS-33: Machine Learning with Python

Unit - 3 : Unsupervised Learning

P-1 :- k-means (example-1)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Given training and test data
X_train = np.array([
   [1, 2], [2, 3], [3, 3], [6, 6], [7, 8], [8, 8],
   [25, 80], [28, 85], [30, 90],[75, 50], [78, 55], [80, 60]
])

X_test = np.array([
   [5, 7], [27, 81], [77, 57]
])

# Apply K-means clustering (assuming 3 clusters based on the data
structure)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)

# Predict the cluster labels for the training data
y_train_pred = kmeans.predict(X_train)

# Predict the cluster labels for the test data
y_test_pred = kmeans.predict(X_test)
```

```
# Plotting the clusters
plt.figure(figsize=(8, 6))

# Plot training data points with cluster labels
for i in range(3):
    plt.scatter(X_train[y_train_pred == i, 0], X_train[y_train_pred == i, 1],
label=f"Cluster {i+1}")

# Plot test data points
plt.scatter(X_test[:, 0], X_test[:, 1], color='black', marker='x', label='Test
Data')

# Plot centroids
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='*',
label='Centroids')

# Labels and legend
plt.title("K-means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

P-2 :- k-means (example-2 for text)

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
```

```
# Sample documents
documents = [
    "This little kitty came to play when I was eating at a restaurant.",
    "Merley has the best squooshy kitten belly.",
    "Google Translate app is incredible.",
    "If you open 100 tabs in google, you get a smiley face.",
    "Best cat photo I've ever taken.",
    "Climbing ninja cat.",
    "Impressed with google map feedback.",
    "Key promoter extension for Google Chrome."
]

# Convert documents to TF-IDF matrix
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# Perform k-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Display top terms for each cluster
print("Top terms per cluster:")
terms = vectorizer.get_feature_names()  # Use get_feature_names() for
older scikit-learn versions
order_centroids = kmeans.cluster_centers_.argsort()[:, ::-1]  # Sort
terms by importance for each cluster
for i in range(3):
```

```
    print(f"Cluster {i}:")
    for ind in order_centroids[i, :10]:  # Top 10 terms for each cluster
        print(f' {terms[ind]}', end=', ')
    print()  # Move to next line after each cluster's terms

# Predict which cluster new documents belong to
print("\nPredictions for new documents:")
new_docs = [
    "google chrome browser to open.",
    "My cat is hungry."
]
for doc in new_docs:
    Y = vectorizer.transform([doc])
    prediction = kmeans.predict(Y)
    print(f"Prediction for '{doc}': Cluster {prediction[0]}")
```

P-3 :- k-means (example-3 using csv file)

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
```

```
y = iris.target  # True labels (species)

# Step 2: Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Get the cluster centers and labels
centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Step 3: Plot the clustering results using two features: Sepal Length
and Sepal Width
plt.figure(figsize=(8, 6))

# Scatter plot of the data points, colored by the assigned cluster
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.6,
label='Data Points')

# Plot the cluster centers (using the same two features for plotting)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X',
label='Centroids')

plt.title('K-Means Clustering on Iris Dataset (Sepal Length vs Sepal
Width)')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```

P-4 :- Vector Quantization

```
import argparse
import numpy as np
from scipy import misc
from matplotlib.pyplot import imread
from sklearn import cluster
import matplotlib.pyplot as plt

def compress_image(img, num_clusters):
    # Convert input image into (num_samples, num_features)
    # array to run kmeans clustering algorithm
    X = img.reshape((-1, 1))

    # Run kmeans on input data
    kmeans = cluster.KMeans(n_clusters=num_clusters, n_init=4,
random_state=5)
    kmeans.fit(X)
    centroids = kmeans.cluster_centers_.squeeze()
    labels = kmeans.labels_

    # Assign each value to the nearest centroid and
    # reshape it to the original image shape
    input_image_compressed = np.choose(labels,
centroids).reshape(img.shape)
    return input_image_compressed

def plot_image(img, title):
```

```
    vmin = img.min()
    vmax = img.max()
    plt.figure()
    plt.title(title)
    plt.imshow(img.astype('uint8'))
def main():
    input_file = "flower_image.jpg"
    num_bits = 2
    if not 1 <= num_bits <= 8:
        raise TypeError('Number of bits should be between 1 and 8')
    num_clusters = np.power(2, num_bits)

    # Print compression rate
    compression_rate = round(100 * (8.0 - num_bits) / 8.0, 2)
    print("\nThe size of the image will be reduced by a factor of",
8.0/num_bits)
    print("\nCompression rate = " + str(compression_rate) + "%")

    # Load input image
    input_image = imread(input_file, True).astype(np.uint8)

    # original image
    plot_image(input_image, 'Original image')

    # compressed image
    input_image_compressed = compress_image(input_image,
num_clusters)
```

```
    plot_image(input_image_compressed, 'Compressed image;
compression rate = '
          + str(compression_rate) + '%')
    plt.show()
main()
```

P-5 meanshift ex-1

```
import numpy as np
from sklearn.cluster import MeanShift
import matplotlib.pyplot as plt

# Create some data points
X = np.array([[1, 2], [1, 3], [2, 2], [5, 8],[5, 7],[4, 9],
            [8, 8], [8, 7],[8, 6], [2, 3],[3,2],[3, 1]])

# Apply Mean Shift
mean_shift = MeanShift()
mean_shift.fit(X)

# Get cluster centers and labels
cluster_centers = mean_shift.cluster_centers_
labels = mean_shift.labels_

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=20, c='red',
          marker='X')
plt.title('Mean Shift Clustering')
```

```
plt.show()
```

P-6 meanshift ex-2

```
# Importing the necessary libraries
from sklearn.cluster import MeanShift
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
# Loading the Iris dataset
iris = load_iris()
X = iris.data
# Creating an instance of MeanShift clustering
ms = MeanShift()
# Fitting the model to the data
ms.fit(X)
# Extracting the cluster labels and cluster centers
labels = ms.labels_
centers = ms.cluster_centers_
# Visualizing the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='x', color='red', s=200)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Mean-Shift Clustering on Iris Dataset')
plt.show()
```

P-7 meanshift ex-3

```
import numpy as np
from sklearn.cluster import MeanShift, estimate_bandwidth

# Load multivar data in the input file
def load_data(input_file):
    X = []
    f=open(input_file, 'r')
    for line in f.readlines():
        data = [float(x) for x in line.split(',')]
        X.append(data)
    return np.array(X)

# Load data from input file
X = load_data('data_multivar.txt')

# Estimating the bandwidth
bw = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))

# Compute clustering with MeanShift
meanshift_estimator = MeanShift(bandwidth=bw, bin_seeding=True)
meanshift_estimator.fit(X)
labels = meanshift_estimator.labels_
centroids = meanshift_estimator.cluster_centers_
num_clusters = len(np.unique(labels))

print ("Number of clusters in input data =", num_clusters)
```

```
# Plot the points and centroids
import matplotlib.pyplot as plt
plt.figure()

# specify marker shapes for different clusters
markers = '.*xv'
for i, marker in zip(range(num_clusters), markers):
    # plot the points belong to the current cluster
    plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker, color='k')
    # plot the centroid of the current cluster
    centroid = centroids[i]
    print("x=",centroid[0],'y=',centroid[1])
    plt.plot(centroid[0], centroid[1], marker='o', markersize=15)
plt.title('Clusters and their centroids')
plt.show()
```

P-8 Linkage

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import dendrogram, linkage

# Generating random data for demonstration
np.random.seed(42)
X = np.random.randn(50, 2)  # 50 data points, 2 features
```

```
# Generating linkage matrices for different methods
linkage_single = linkage(X, method='single')
linkage_complete = linkage(X, method='complete')
linkage_average = linkage(X, method='average')
linkage_ward = linkage(X, method='ward')

# Plotting the dendrograms for each linkage method
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Single Linkage
dendrogram(linkage_single, ax=axes[0, 0])
axes[0, 0].set_title('Single Linkage')

# Complete Linkage
dendrogram(linkage_complete, ax=axes[0, 1])
axes[0, 1].set_title('Complete Linkage')

# Average Linkage
dendrogram(linkage_average, ax=axes[1, 0])
axes[1, 0].set_title('Average Linkage')

# Ward Linkage
dendrogram(linkage_ward, ax=axes[1, 1])
axes[1, 1].set_title('Ward Linkage')

plt.tight_layout()
plt.show()
```

P-9 agglomerative clustering ex-1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster

# Predefined array (2D data points)
X = np.array([
    [1, 2], [1, 3], [1, 4], [2, 2], [2, 3],  # Cluster 1
    [8, 7], [8, 8],[8, 9], [9, 8], [9, 9],  # Cluster 2
    [5, 1], [6, 1], [5, 2], [6, 2]   # Cluster 3
])

# Perform hierarchical/agglomerative clustering using scipy's linkage
function
linked = linkage(X, method='ward')
# 'ward' minimizes the variance within clusters

# Plotting the dendrogram
plt.figure(figsize=(10, 6))
dendrogram(linked)
plt.title('Dendrogram of Agglomerative Clustering')
plt.xlabel('Sample index')
plt.ylabel('Distance')
```

```
plt.show()
# Define the number of clusters
n_clusters = 3

#perform AgglomerativeClustering using sklearn

sklearn_model = AgglomerativeClustering(n_clusters=n_clusters)
sklearn_labels = sklearn_model.fit_predict(X)

# Scatter plot with sklearn Agglomerative Clustering results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=sklearn_labels, cmap='viridis', marker='o')
plt.title('Agglomerative Clustering with sklearn')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

P-10 agglomerative clustering ex-2

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the Iris dataset
iris = load_iris()
```

```
X = iris.data
y = iris.target
Z = linkage(X, 'ward')

# Plot the dendogram
plt.figure(figsize=(7.5, 3.5))
plt.title("Iris Dendrogram")
dendrogram(Z)
plt.show()

# create an instance of the AgglomerativeClustering class
model = AgglomerativeClustering(n_clusters=5)

# fit the model to the dataset
model.fit(X)
labels = model.labels_

# Plot the results
plt.figure(figsize=(7.5, 3.5))
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.title("Agglomerative Clustering Results")
plt.show()
```

P-11case study-product recommendation

```
import pandas as pd
import numpy as np
```

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Dataset with product features such as price, rating, and category
code.
data = {
   'product_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
   'price': [100, 200, 150, 300, 250, 350, 200, 120, 180, 210],
   'rating': [4.2, 3.9, 4.8, 4.5, 3.7, 4.0, 4.3, 4.7, 3.8, 4.1],
   'category_code': [1, 1, 2, 2, 3, 3, 1, 2, 3, 1]
}

# Create a DataFrame
df = pd.DataFrame(data)



# We need to standardize the data before applying K-means (feature
scaling)

features = df[['price', 'rating', 'category_code']]  # Select features for
clustering

# Normalize the features (important for K-means clustering)
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Applying K-means Clustering (with k=3)
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(features_scaled)

# Recommendation Logic
# Let's assume the user likes product with ID = 103.
liked_p_id = 103

# Find the cluster of the liked product
liked_product_cluster = df[df['product_id'] ==
liked_p_id]['cluster'].values[0]

# Step 6: Recommend similar products (from the same cluster)
recommended_products = df[df['cluster'] == liked_product_cluster]

# Display recommended products
print("Recommended Products for Product ID", liked_p_id)
print(recommended_products[['product_id', 'price', 'rating',
'category_code']])
```

P-12case study-customer segment

```
import pandas as pd
import numpy as np
from sklearn.cluster import MeanShift
import matplotlib.pyplot as plt

# Sample data: grocery items (price in rupees, quantity)
data = {
```

```
    'price': [220, 50, 30, 450, 500, 180, 100,
              70, 550, 150,170,180,300,350],  # price in rupees
    'quantity': [10, 5, 6, 15, 2, 6, 8, 25,
                 10, 15,1,3,5,6]  # quantity in units
}

# Create DataFrame
df = pd.DataFrame(data)

# Features: price and quantity
X = df[['price', 'quantity']].values

# Initialize MeanShift model
meanshift = MeanShift()

# Fit the model
meanshift.fit(X)

# Get the cluster labels (which items belong to the same cluster)
df['cluster'] = meanshift.labels_

# Get the cluster centers
cluster_centers = meanshift.cluster_centers_

# Plot the clusters
plt.figure(figsize=(8, 6))

# Scatter plot of the points with colors according to the cluster
```

```
plt.scatter(df['price'], df['quantity'], c=df['cluster'], s=100, label='Items')

# Plot the cluster centers with a different marker (e.g., a red 'X')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red',
marker='*', s=80, label='Cluster Centers')

# Annotate each cluster center
for i, center in enumerate(cluster_centers):
   plt.text(center[0], center[1], f'Cluster {i}', fontsize=12, ha='center',
color='red')

# Labeling the axes and the title
plt.xlabel('Price (rs)')
plt.ylabel('Quantity')
plt.title('MeanShift Clustering of Grocery Items with Cluster Centers')

# Adding a color bar for the clusters
plt.colorbar(label='Cluster ID')

# Display the legend
plt.legend()

# Show the plot
plt.show()
```