Lab-4: Clustering

Introduction:

This lab focuses on the implementation and analysis of clustering algorithms, a key area of unsupervised machine learning. Clustering involves grouping data points so that those within the same cluster exhibit high similarity, while being distinct from those in other clusters. The algorithms explored include K-means, Mini-batch K-means, K-means++, K-Medoids, and Agglomerative Clustering.

DataSets

● Synthetic 2-D data points: Randomly generated 2-D data points within specified ranges (0-100 or 0-200) for K-means, Mini-batch K-means, and K-means++.

● Iris Dataset: A well-known multivariate dataset used for KMedoids and Agglomerative Clustering.

4.1 K-Means Clustering

Implementation Code

```
[ ]  import numpy as np
     import time
     from sklearn.cluster import KMeans
     print("Kishor Lab-4.1")
     # Generate 10,000 2-D data points in the range 0-100
     data = np.random.rand(10000, 2) * 100
     kmeans = KMeans(n_clusters=5, random_state=42)\
     # Measure the time taken by the algorithm
     start_time = time.time()
     kmeans.fit(data)
     end_time = time.time()
     # Calculate the time taken
     time_taken = end_time - start_time
     print(f"Time taken by K-means to find clusters: {time_taken:.4f} seconds")
     # Get the cluster centers
     print("Cluster Centers:\n", kmeans.cluster_centers_)
```

Output SnapShot

```
Kishor Lab-4.1
Time taken by K-means to find clusters: 0.1003 seconds
Cluster Centers:
 [[21.98350264 21.31650293]
 [77.52757929 77.40219545]
 [22.27859544 77.64243537]
 [78.12378742 22.4370641 ]
 [48.81252649 48.76943724]]
```

4.2 Mini-Batch K-means

Implementation Code

```
import numpy as np
import time
from sklearn.cluster import MiniBatchKMeans
print("Kishor Lab-4.2")
# Generate 10,000 2-D data points in the range 0-100
data = np.random.rand(10000, 2) * 100
batch_sizes = [100, 300, 500, 1000, 1500]
times = []
for batch_size in batch_sizes:
    # Initialize Mini-batch K-means algorithm
    minibatch_kmeans = MiniBatchKMeans(n_clusters=5, batch_size=batch_size, random_state=42)
    # Measure the time taken by the algorithm
    start_time = time.time()
    minibatch_kmeans.fit(data)
    end_time = time.time()
    # Calculate the time taken
    time_taken = end_time - start_time
    times.append(time_taken)
    print(f"Time taken with batch size {batch_size}: {time_taken:.4f} seconds")
# Determine the best batch size
best_batch_size = batch_sizes[np.argmin(times)]
print(f"\nBest batch size: {best_batch_size}")
```

Output SnapShot

```
Kishor Lab-4.2
Time taken with batch size 100: 0.0127 seconds
Time taken with batch size 300: 0.0132 seconds
Time taken with batch size 500: 0.0100 seconds
Time taken with batch size 1000: 0.0129 seconds
Time taken with batch size 1500: 0.0192 seconds

Best batch size: 500
```

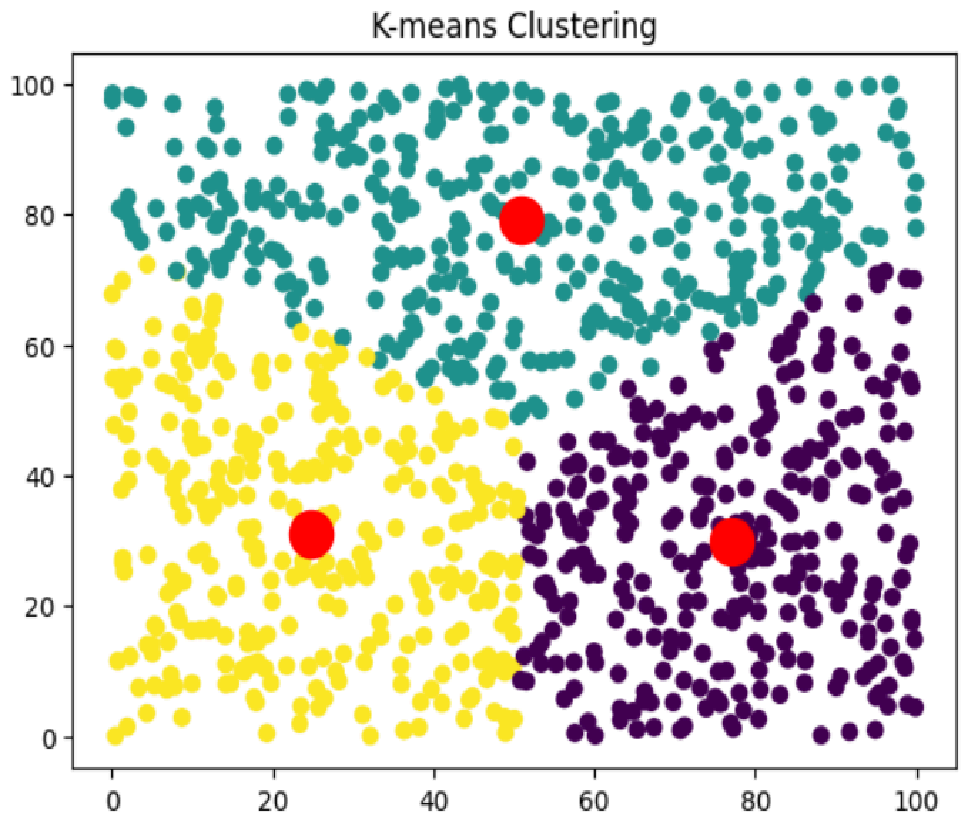
4.3 K-Means Clustering Algorithm

Implementation Code

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
print("Kishor Lab-4.3")
# Generate 1,000 2-D data points in the range 0-100
data = np.random.rand(1000, 2) * 100# Initialize K-means algorithm with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
# Fit the model
kmeans.fit(data)
# Plot the data points and cluster centers
plt.scatter(data[:, 0], data[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.title("K-means Clustering")
plt.show()
```

Output SnapShot

```
Kishor Lab-4.3
```



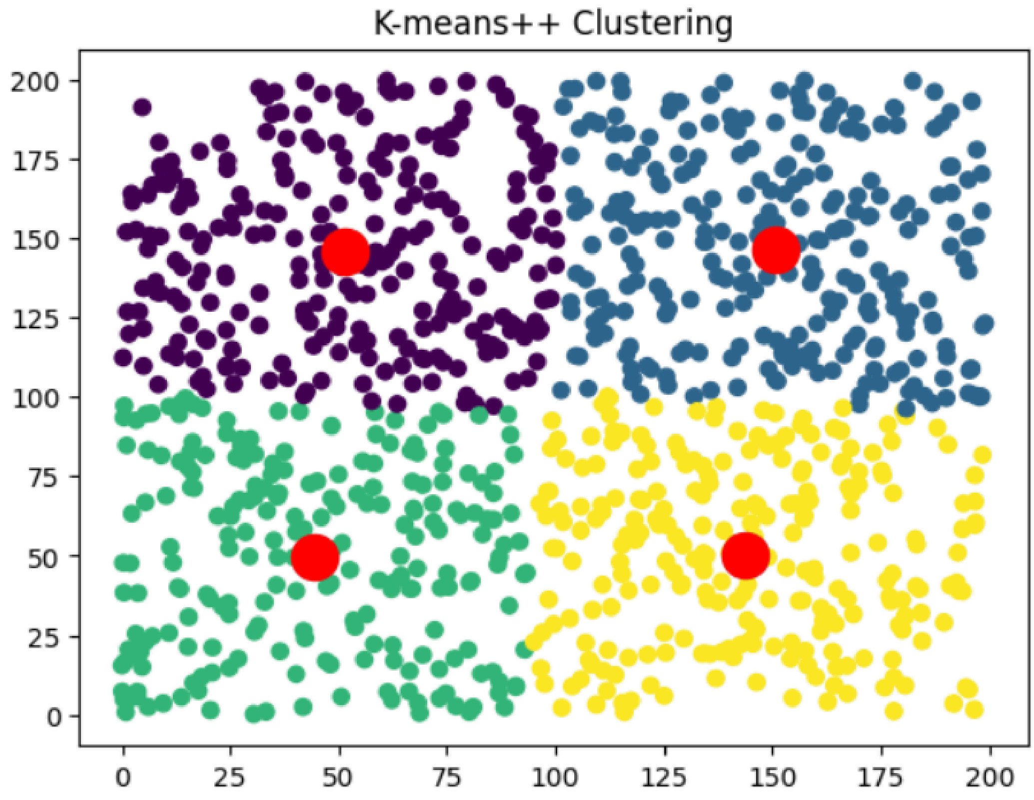
4.4 K-Means++ Algorithm

Implementation Code

```python
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
print("Kishor Lab-4.4")
# Generate 1,000 2-D data points in the range 0-200
data = np.random.rand(1000, 2) * 200# Initialize K-means++ algorithm with 4 clusters
kmeans_plus = KMeans(n_clusters=4, init='k-means++', random_state=42)
# Fit the model
kmeans_plus.fit(data)
# Plot the data points and cluster centers
plt.scatter(data[:, 0], data[:, 1], c=kmeans_plus.labels_, cmap='viridis')
plt.scatter(kmeans_plus.cluster_centers_[:, 0], kmeans_plus.cluster_centers_[:, 1], s=300, c='red')
plt.title("K-means++ Clustering")
plt.show()
```

Output SnapShot

Kishor Lab-4.4



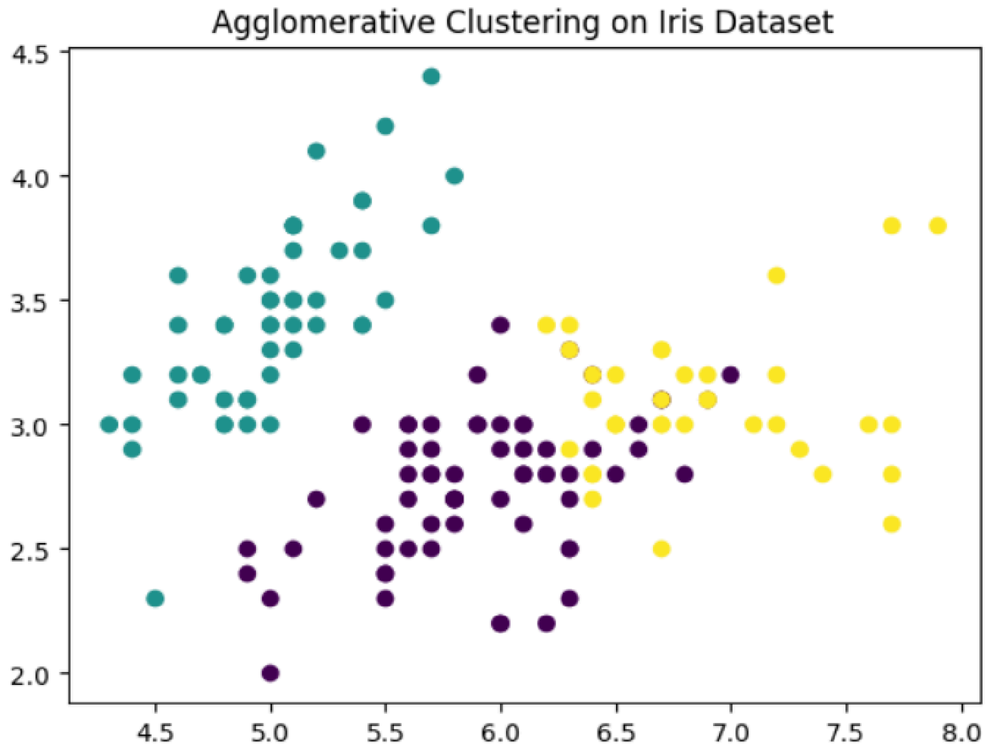
4.5 Agglomerative Clustering Algorithm

Implementation Code

```
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
print("Kishor Lab-4.5")
# Load the Iris dataset
iris = load_iris()
data = iris.data
# Initialize Agglomerative Clustering algorithm with 3 clusters
agg_clustering = AgglomerativeClustering(n_clusters=3)
# Fit the model
labels = agg_clustering.fit_predict(data)
# Plot the clusters
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.title("Agglomerative Clustering on Iris Dataset")
plt.show()
```

Output SnapShot

```
Kishor Lab-4.5
```



4.6 KMedoids Algorithm

Implementation Code

```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
print("Kishor Lab-4.6")
class KMedoids:
    def __init__(self, n_clusters=3, random_state=42):
        self.n_clusters = n_clusters
        self.random_state = random_state
        self.labels_ = None
    def fit(self, X):
        np.random.seed(self.random_state)
        n_samples = X.shape[0]
        # Initialize medoids randomly
        medoid_indices = np.random.choice(n_samples, self.n_clusters, replace=False)
        medoids = X[medoid_indices].copy()
        # Calculate distance matrix
        distances = pairwise_distances(X)
        for _ in range(100):  # max iterations
            # Assign each point to closest medoid
            medoid_distances = pairwise_distances(X, medoids)
            labels = np.argmin(medoid_distances, axis=1)
```

```
            # Update medoids
            new_medoids = []
            for k in range(self.n_clusters):
                cluster_points = np.where(labels == k)[0]
                if len(cluster_points) == 0:
                    new_medoids.append(medoids[k])
                    continue
                # Find point that minimizes total distance to other points in cluster
                min_cost = float('inf')
                best_medoid_idx = cluster_points[0]
                for candidate_idx in cluster_points:
                    cost = np.sum(distances[candidate_idx, cluster_points])
                    if cost < min_cost:
                        min_cost = cost
                        best_medoid_idx = candidate_idx
                new_medoids.append(X[best_medoid_idx])
            new_medoids = np.array(new_medoids)
            # Check for convergence
            if np.allclose(medoids, new_medoids):
                break
            medoids = new_medoids
        # Final assignment
        medoid_distances = pairwise_distances(X, medoids)
        self.labels_ = np.argmin(medoid_distances, axis=1)
        return self
# Load the Iris dataset
iris = load_iris()
data = iris.data
```

```
# Initialize KMedoids algorithm with 3 clusters
kmedoids = KMedoids(n_clusters=3, random_state=42)

# Fit the model
kmedoids.fit(data)

# Plot the clusters
plt.scatter(data[:, 0], data[:, 1], c=kmedoids.labels_, cmap='viridis')
plt.title("KMedoids Clustering on Iris Dataset")
plt.show()
```

Output SnapShot