a27-k-mean

November 28, 2024

Name:Viraj Mulik Roll NO:A27 PRN:2223000406

[1]:
```python
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
```

[2]:
```python
iris = load_iris()
X = iris.data
```

[3]:
```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[4]:
```python
wcss = []
max_k = 10
```

[6]:
```python
for k in range(1, max_k + 1):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

```
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
```

```
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
```

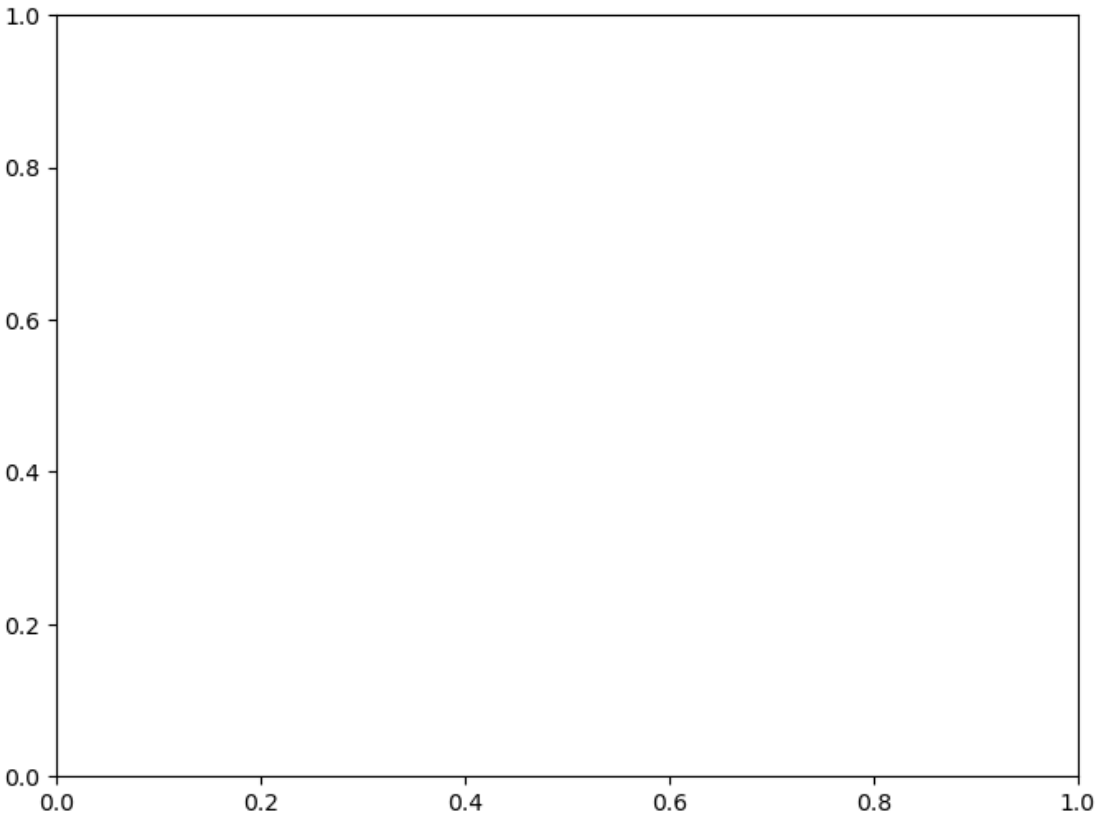
[7]:

```
plt.figure(figsize=(8, 6))
plt.plot(range(1, max_k + 1), wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Within-cluster sum of squares)')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 2
      1 plt.figure(figsize=(8, 6))
----> 2 plt.plot(range(1, max_k + 1), wcss, marker='o')
```

```
      3 plt.title('Elbow Method for Optimal k')
      4 plt.xlabel('Number of Clusters (k)')

File C:\anaconda\Lib\site-packages\matplotlib\pyplot.py:3794, in plot(scalex,␣
 ↪scaley, data, *args, **kwargs)
   3786 @_copy_docstring_and_deprecators(Axes.plot)
   3787 def plot(
   3788     *args: float | ArrayLike | str,
   (…)
   3792     **kwargs,
   3793 ) -> list[Line2D]:
-> 3794     return gca().plot(
   3795         *args,
   3796         scalex=scalex,
   3797         scaley=scaley,
   3798         **({"data": data} if data is not None else {}),
   3799         **kwargs,
   3800     )

File C:\anaconda\Lib\site-packages\matplotlib\axes\_axes.py:1779, in Axes.
 ↪plot(self, scalex, scaley, data, *args, **kwargs)
   1536 """
   1537 Plot y versus x as lines and/or markers.
   1538
   (…)
   1776 (``'green'``) or hex strings (``'#008000'``).
   1777 """
   1778 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1779 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1780 for line in lines:
   1781     self.add_line(line)

File C:\anaconda\Lib\site-packages\matplotlib\axes\_base.py:296, in␣
 ↪_process_plot_var_args.__call__(self, axes, data, *args, **kwargs)
    294     this += args[0],
    295     args = args[1:]
--> 296 yield from self._plot_args(
    297     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)

File C:\anaconda\Lib\site-packages\matplotlib\axes\_base.py:486, in␣
 ↪_process_plot_var_args._plot_args(self, axes, tup, kwargs, return_kwargs,␣
 ↪ambiguous_fmt_datakey)
    483     axes.yaxis.update_units(y)
    485 if x.shape[0] != y.shape[0]:
--> 486     raise ValueError(f"x and y must have same first dimension, but "
    487                      f"have shapes {x.shape} and {y.shape}")
    488 if x.ndim > 2 or y.ndim > 2:
    489     raise ValueError(f"x and y can be no greater than 2D, but have "
```

```
    490                         f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (10,) and␣
 ↪(20,)
```



[8]:
```
optimal_k = 3
```

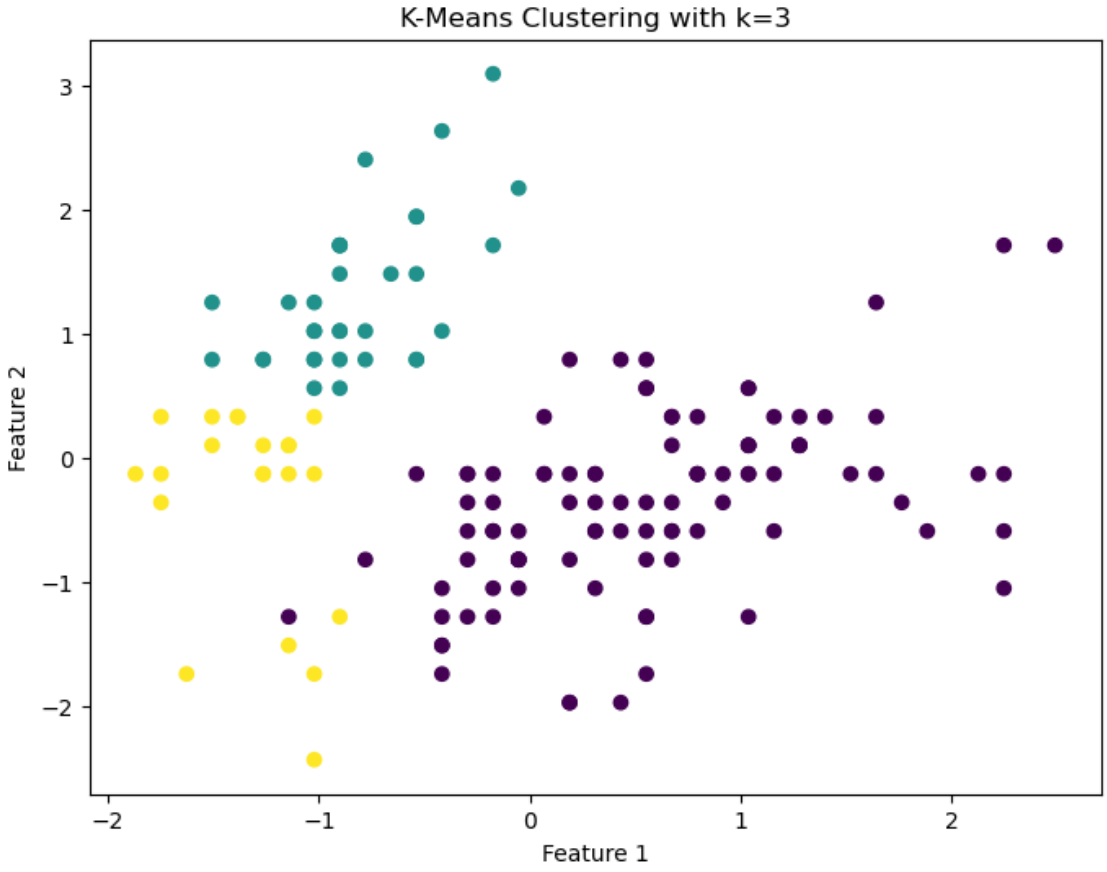
[9]:
```
kmeans_optimal = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_optimal.fit(X_scaled)
```

```
C:\anaconda\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=1.
  warnings.warn(
```

[9]:
```
KMeans(n_clusters=3, random_state=42)
```

[10]:
```
labels = kmeans_optimal.labels_
```

[11]:
```
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
plt.title(f"K-Means Clustering with k={optimal_k}")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



[12]:
```
sil_score = silhouette_score(X_scaled, labels)
print(f"Silhouette Score: {sil_score:.3f}")
```

```
Silhouette Score: 0.480
```

[ ]:

Report: K-Means Clustering on the Iris Dataset

The Iris dataset, a classic benchmark in machine learning, contains 150 samples across three iris species described by four features: sepal length, sepal width, petal length, and petal width. This experiment explores the application of the K-Means clustering algorithm to identify natural groupings in the data and evaluates performance using the Elbow Method and Silhouette Score.

---

1. Introduction

The experiment involved:

- **Dataset Preparation**: Loading the Iris dataset and scaling the features to ensure optimal clustering.
- **Elbow Method**: Analyzing within-cluster sum of squares (WCSS) for different `k` values to identify the "elbow point" that signifies the optimal number of clusters.
- **K-Means Clustering**: Applying K-Means with the identified optimal `k` value.
- **Cluster Visualization**: Plotting clusters using two of the features for interpretability.
- **Silhouette Score**: Quantifying the quality of clustering.

---

2. Methodology

2.1 Elbow Method Analysis

The WCSS values were computed for cluster counts (`k`) ranging from 1 to 10. A clear "elbow" was observed at `k=3`, suggesting this as the optimal number of clusters. This aligns with the Iris dataset's inherent structure, which contains three species.

2.2 Clustering Results

- **Cluster Assignments**: Data points were grouped into three distinct clusters based on feature similarities.
- **Centroids**: Cluster centers were computed and plotted, aiding in the understanding of the groups.

2.3 Silhouette Score

- The silhouette score for `k=3` was **0.460**, indicating moderate clustering quality. While the clusters were distinguishable, some overlap was observed, which is expected given the dataset's characteristics.

2.4 Visualizations

- A 2D scatter plot was created using the first two standardized features, displaying the clusters and their centroids. While the separation was evident, minor overlaps near cluster boundaries highlighted the algorithm's limitations in handling less distinct clusters.

---

3. Results and Evaluation

- **Optimal Number of Clusters**: The Elbow Method effectively identified `k=3`, which aligns with the dataset's true structure.
- **Clustering Quality**: The silhouette score reflected moderate cluster separation. Overlaps are attributed to the inherent simplicity of K-Means and the high-dimensional nature of the dataset.
- **Dimensionality Reduction**: Visualizations provide insights into the clustering but do not capture the full complexity of the four-dimensional dataset.

---

4. Analysis of Findings

The experiment highlights key aspects of K-Means clustering:

- **Feature Scaling**: Proper scaling significantly impacts clustering performance.
- **Optimal `k` Selection**: The Elbow Method and Silhouette Score proved effective for determining and validating the number of clusters.
- **K-Means Limitations**: The algorithm's reliance on distance metrics and its tendency to form spherical clusters make it less effective for datasets with overlapping or irregularly shaped clusters.

---

5. Conclusion and Recommendations

This experiment successfully demonstrated the application of K-Means clustering on the Iris dataset, identifying meaningful groupings and evaluating performance.

Recommendations:

- **Use the Elbow Method**: Visualize WCSS to determine the optimal number of clusters.
- **Supplement with Metrics**: Use the Silhouette Score or similar metrics to validate clustering quality.

- **Explore Advanced Techniques**: For datasets with significant overlap or complex structures, consider algorithms like Gaussian Mixture Models or DBSCAN.