Unsupervised Learning

Unsupervised learning, also known as unsupervised machine learning, uses machine learning (ML) algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

K-Means clustering

K-Means clustering serves as one of the most widely implemented unsupervised learning algorithms, partitioning data into K distinct, non-overlapping clusters. The algorithm functions by iteratively assigning data points to the nearest cluster center and updating these centers based on the mean position of all points within each cluster.

It is one of the most popular clustering methods used in machine learning. Unlike supervised learning, the training data that this algorithm uses is unlabeled, meaning that data points do not have a defined classification structure.

While various types of clustering algorithms exist, including exclusive, overlapping, hierarchical and probabilistic, the k-means clustering algorithm is an example of an exclusive or “hard” clustering method. This form of grouping stipulates that a data point can exist in just one cluster. This type of cluster analysis is commonly used in data science for market segmentation, document clustering, image segmentation and image compression. The k-means algorithm is a widely used method in cluster analysis because it is efficient, effective and simple.

K-means is an iterative, centroid-based clustering algorithm that partitions a dataset into similar groups based on the distance between their centroids. The centroid, or cluster center, is either the mean or median of all the points within the cluster depending on the characteristics of the data.

How does k-means clustering work?

K-means clustering is an iterative process to minimize the sum of distances between the data points and their cluster centroids.

The k-means clustering algorithm operates by categorizing data points into clusters by using a mathematical distance measure, usually euclidean, from the cluster center. The objective is to minimize the sum of distances between data points and their assigned clusters. Data points that are nearest to a centroid are grouped together within the same category. A higher k value, or the number of clusters, signifies smaller clusters with greater detail, while a lower k value results in larger clusters with less detail.

Steps in K-Means Clustering

1. Initialize K cluster centers randomly
2. Assign each data point to the nearest cluster center
3. Update cluster centers by calculating the mean of all points in each cluster
4. Repeat steps 2-3 until convergence or maximum iterations reached

Applications in Customer Segmentation

Customer segmentation represents a prime application of K-Means clustering, offering businesses valuable insights into their customer base. This technique enables organizations to:

- Identify distinct customer groups based on purchasing behavior
- Develop targeted marketing strategies for different customer segments
- Optimize product recommendations and personalization
- Improve customer retention through segment-specific engagement strategies

For example, an e-commerce platform might cluster customers based on features such as purchase frequency, average order value, and browsing patterns, revealing distinct groups like "high-value regular customers," "occasional big spenders," and "browse-heavy, low-conversion visitors."

```
k_means = KMeans(n_clusters = 2, random_state = 2744)
penguin_data['Cluster'] = k_means.fit_predict(X_penguin_data)
```

```
C:\Users\kalat\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, w
there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

Cluster Centers

```
k_means.cluster_centers_
```

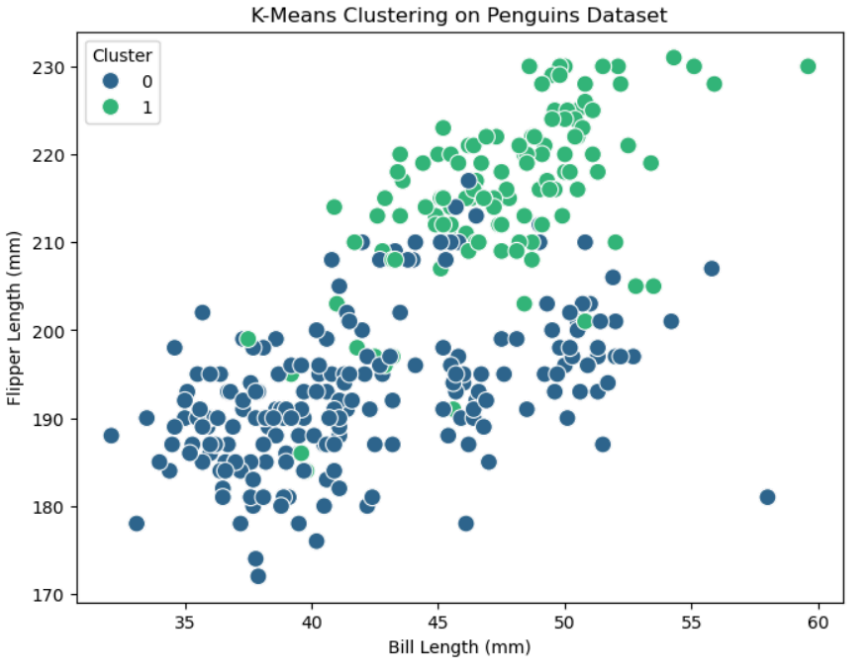
```
array([[  42.02075472,   18.02877358,  192.68396226, 3687.14622642],
       [  47.44793388,   15.65123967,  215.47933884, 5117.97520661]])
```

Visualizing

```
plt.figure(figsize=(8, 6))

sns.scatterplot(x = penguin_data["bill_length_mm"], y = penguin_data["flipper_length_mm"], hue = penguin_data["Cluster"], palette="viridis", s=100)

plt.xlabel("Bill Length (mm)")
plt.ylabel("Flipper Length (mm)")
plt.title("K-Means Clustering on Penguins Dataset")
plt.legend(title="Cluster")
plt.show()
```



Dimensionality Reduction

Dimensionality reduction is a method for representing a given dataset using a lower number of features (that is, dimensions) while still capturing the original data's meaningful properties.1 This amounts to removing irrelevant or redundant features, or simply noisy data, to create a model with a lower number of variables. Dimensionality reduction covers an array of feature selection and data compression methods used during preprocessing. While dimensionality reduction methods differ in operation, they all transform high-dimensional spaces into low-dimensional spaces through variable extraction or combination.

Principal component analysis (PCA)

Principal component analysis (PCA) reduces the number of dimensions in large datasets to principal components that retain most of the original information. It does this by transforming potentially correlated variables into a smaller set of variables, called principal components.

Key Concepts:

1. *Dimensionality Reduction:* PCA transforms the data into a new coordinate system (called principal components) such that:
    a. The first principal component (PC1) captures the largest amount of variance in the data.
    b. The second principal component (PC2) captures the second-largest variance, and so on.
    c. Each new component is orthogonal (uncorrelated) to the others.
2. *Variance:* The idea behind PCA is to preserve the most important information (variance) in the data while reducing the number of dimensions. Variance refers to how spread out the data is along different axes.
3. *Linear Combinations:* PCA creates linear combinations of the original features (variables) to form new dimensions (principal components). These new dimensions are ranked based on the amount of variance they capture from the original data.

Why Use PCA?

- *Data Visualization:* PCA helps to visualize high-dimensional data by projecting it into 2 or 3 dimensions (principal components). This is especially useful when dealing with datasets that have many features (e.g., hundreds or thousands).
- *Noise Reduction:* By reducing the number of features and keeping only the most important ones, PCA can help remove noise from the data.
- *Improved Performance:* For some algorithms (like clustering, classification, or regression), reducing the number of features can improve performance by mitigating issues like overfitting and making the model computationally more efficient.

```
X_pca = country_data.iloc[:, 1:8]
y_pca = country_data['gdpp']
```

Scalling

```
X_pca = scaler.fit_transform(X_pca)
```

Applying PCA

```
pca = PCA(n_components = 2)
after_pca = pca.fit_transform(X_pca)
```

```
pca_df = pd.DataFrame(after_pca, columns = ['PC1', 'PC2'])
```

```
pca_df.head()
```

	PC1	PC2
0	-2.260407	0.165759
1	0.309301	-0.552512
2	-0.196068	-0.387944
3	-2.147040	1.852047
4	1.035825	0.026052

```
sns.scatterplot(pca_df, x = pca_df['PC1'], y = pca_df['PC2'], hue = country_data['gdpp'])
plt.show()
```