Clustering and Optimization Techniques

Cindy Njeri Kiarie

2025-04-21

Question 1: Clustering Methods

1. EM (Expectation-Maximization) Clustering

The EM algorithm is widely used for clustering tasks where data is modeled as coming from a mixture of distributions. It optimizes the likelihood of the observed data by iteratively refining the parameters of these distributions.

Overview

- Assumes the data points are generated from a mixture of probability distributions.
- Commonly uses Gaussian distributions for the clusters.

Steps in EM Clustering

1. **Initialization**
   - Define the number of clusters (k).
   - Initialize the parameters of each distribution, such as means (μ), covariances (Σ), and mixing coefficients (π).
2. **Expectation (E) Step**
   - Compute the likelihood of each data point belonging to a specific cluster. This is done using Bayes' theorem based on the current parameters.
3. **Maximization (M) Step**
   - Update the parameters (μ, Σ, π) to maximize the overall likelihood of the data.
4. Repeat E and M steps until convergence or until the likelihood does not significantly improve.

Applications

- Image segmentation
- Anomaly detection
- Customer segmentation

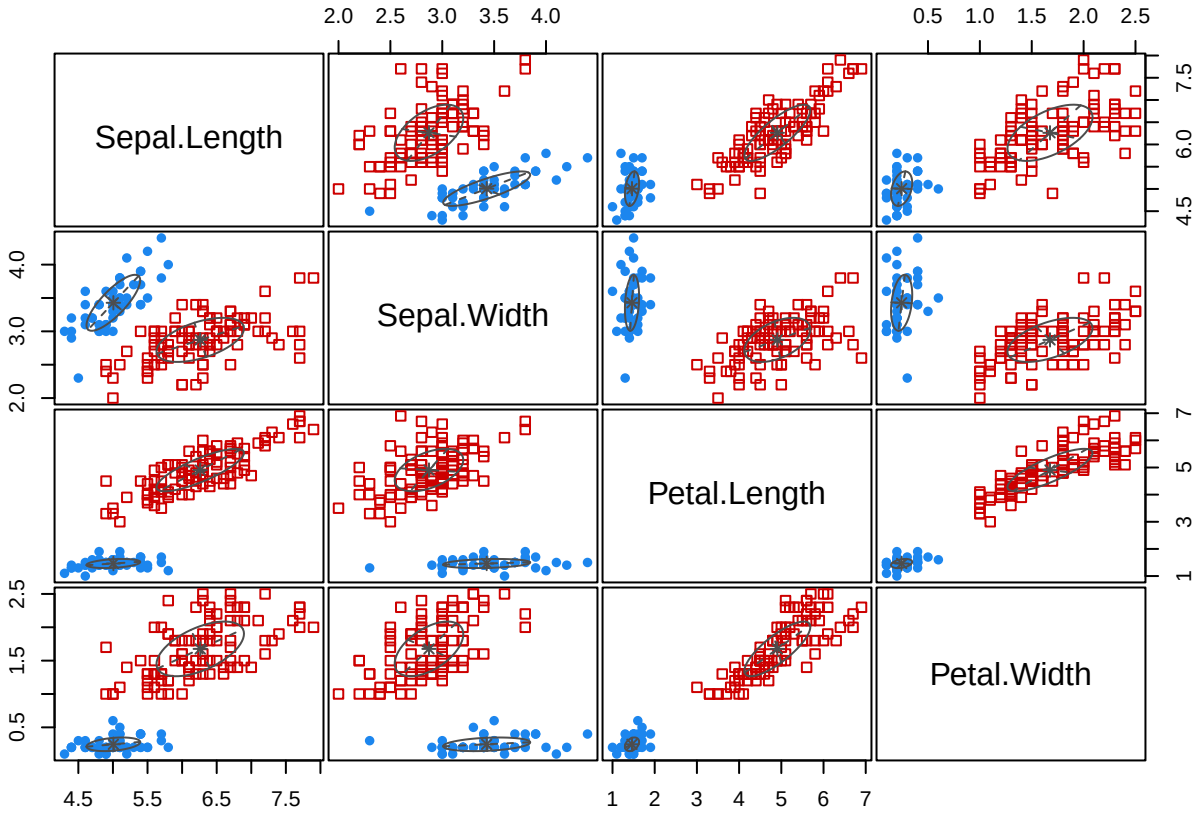
Example Dataset

Use the **Iris dataset** (4 features: sepal length, sepal width, petal length, petal width) for Gaussian clustering.

```r
# Load required library
library(mclust)

# Load the Iris dataset
data(iris)
iris_features <- iris[, 1:4]  # Extract the four numeric features

# Fit a Gaussian Mixture Model using the EM algorithm
gmm_fit <- Mclust(iris_features)

# Display a summary of the fitted model
summary(gmm_fit)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 2 components:
##
##  log-likelihood   n df       BIC       ICL
##        -215.726 150 26 -561.7285 -561.7289
##
## Clustering table:
##   1   2
##  50 100
```

```r
# Add the predicted cluster labels to the original data
iris$Cluster <- as.factor(gmm_fit$classification)

# Plot the classification result
plot(gmm_fit, what = "classification")
```




```
# View the first few rows of the clustered data
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Cluster
## 1          5.1         3.5          1.4         0.2  setosa       1
## 2          4.9         3.0          1.4         0.2  setosa       1
## 3          4.7         3.2          1.3         0.2  setosa       1
## 4          4.6         3.1          1.5         0.2  setosa       1
## 5          5.0         3.6          1.4         0.2  setosa       1
## 6          5.4         3.9          1.7         0.4  setosa       1
```

---

2. K-Means / K-Median Clustering

K-Means and K-Median are partition-based clustering methods focusing on minimizing the distance between data points and their cluster centers.

K-Means Clustering

- **Objective Function**: Minimize the sum of squared Euclidean distances between data points and their cluster centroids.

Steps

1. Initialize k cluster centroids randomly.
2. Assign each data point to the nearest centroid.
3. Update each centroid to the mean of its assigned points.
4. Repeat steps 2 and 3 until centroids stabilize.

K-Median Clustering

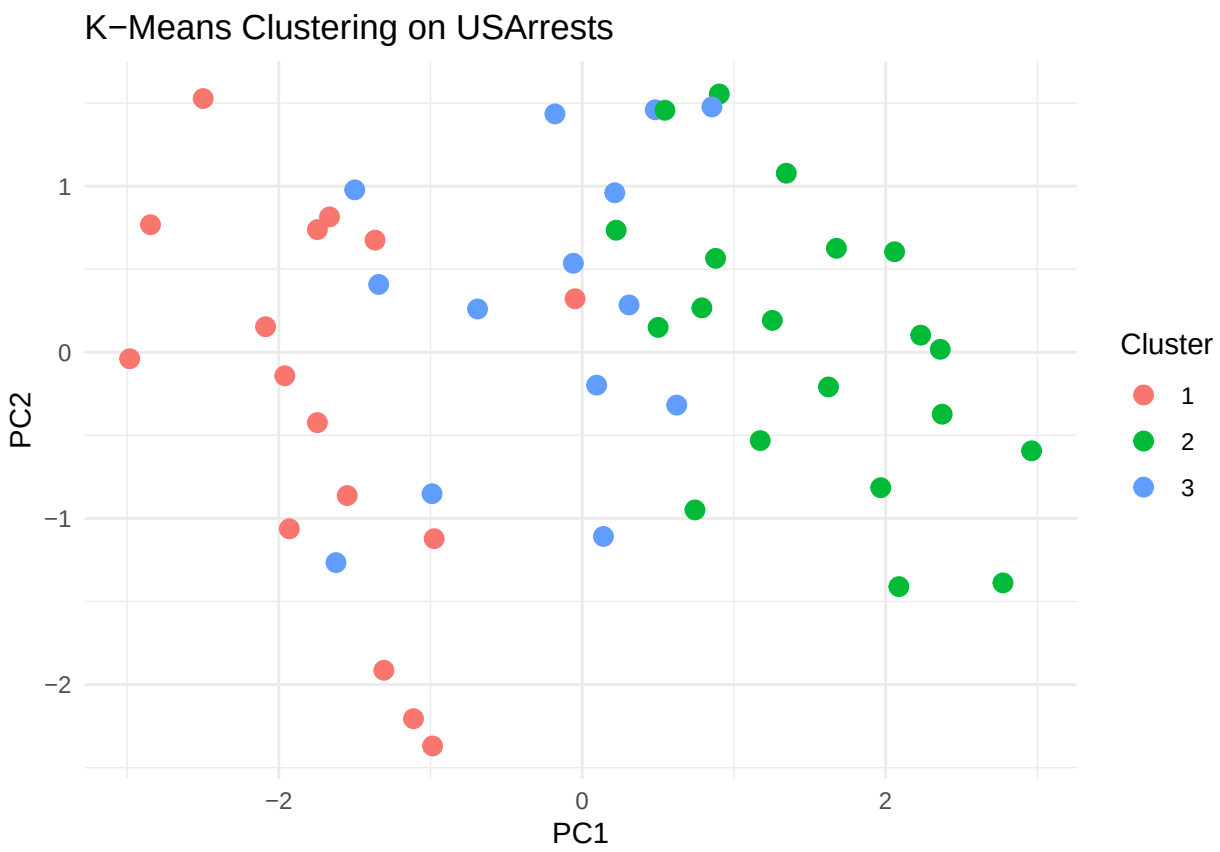
- Similar to K-Means but minimizes the sum of the absolute differences (Manhattan distance).
- Uses **median** instead of **mean** to update cluster centers, making it robust to outliers.

Applications

- Market segmentation
- Document classification
- Recommendation systems

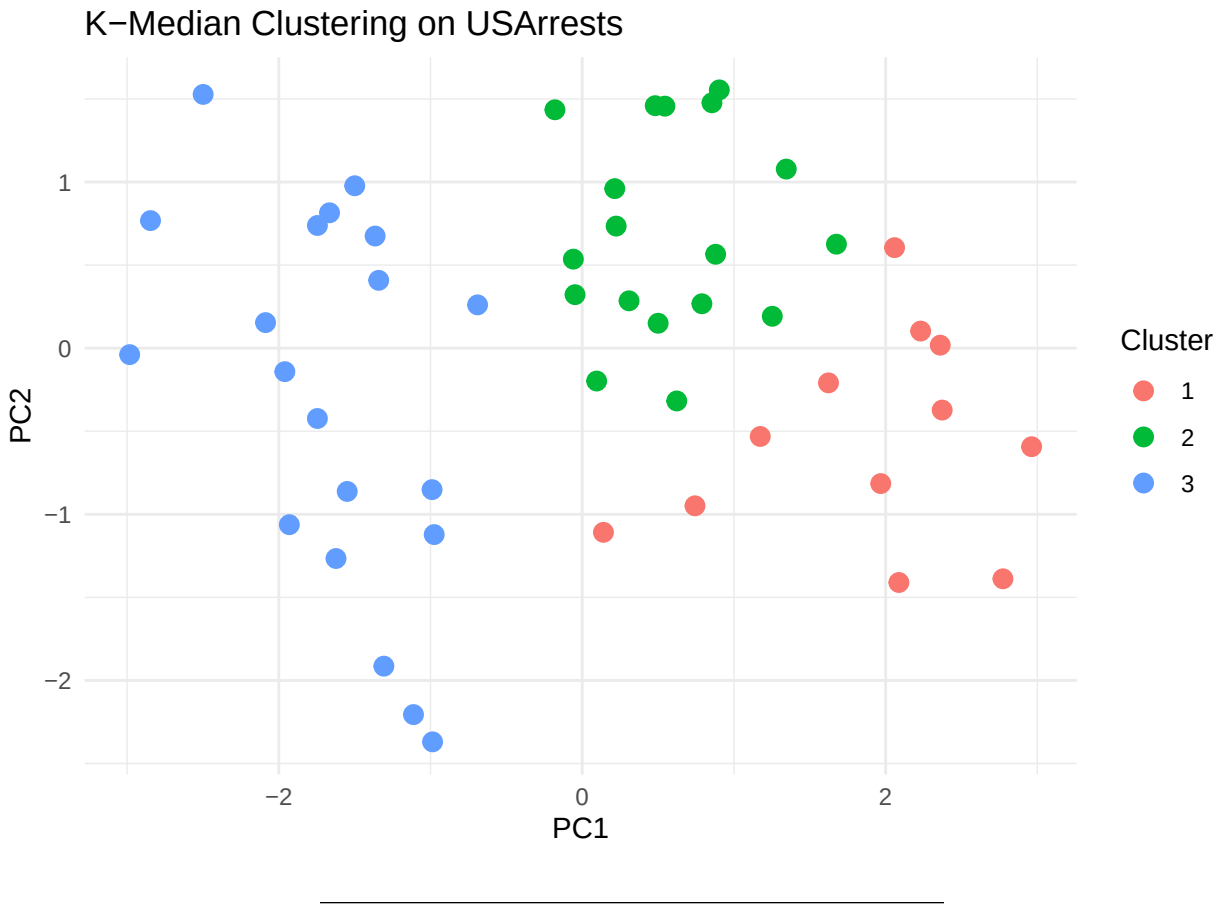
Example Dataset

```r
# Load required libraries
library(tidyverse)
library(ClusterR)  # For K-Median clustering

# Load the USArrests dataset
data("USArrests")
usarrests_data <- na.omit(USArrests)  # Just in case

# --- K-MEANS CLUSTERING ---
set.seed(123)
kmeans_result <- kmeans(usarrests_data, centers = 3)

# Add K-Means cluster labels
usarrests_kmeans <- usarrests_data %>%
  mutate(Cluster = as.factor(kmeans_result$cluster))

# Plot K-Means clusters (first two principal components)
usarrests_pca <- prcomp(usarrests_data, scale. = TRUE)
pca_df <- as.data.frame(usarrests_pca$x)
pca_df$Cluster <- usarrests_kmeans$Cluster

ggplot(pca_df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "K-Means Clustering on USArrests", x = "PC1", y = "PC2") +
  theme_minimal()
```




```r
# --- K-MEDIAN CLUSTERING ---
set.seed(123)
# Create a matrix for initial centroids
initial_centroids <- matrix(data = c(8, 170, 65, 20,
                                     5, 120, 70, 18,
                                     3, 100, 80, 16),
                            nrow = 3, ncol = 4, byrow = TRUE)

# Pass this matrix to the CENTROIDS argument
kmed_result <- KMeans_arma(data = scale(usarrests_data), clusters = 3,
                           n_iter = 100, CENTROIDS = initial_centroids,
                           seed_mode = "keep_existing")

# Calculate Euclidean distances from each point to each centroid
distances <- as.matrix(dist(scale(usarrests_data)))  # Scale the data to match centroids
cluster_assignments <- apply(scale(usarrests_data), 1, function(row) {
  which.min(colSums((t(kmed_result) - row)^2))
})

# Add the cluster labels to the dataset
usarrests_kmed <- usarrests_data %>%
  mutate(Cluster = as.factor(cluster_assignments))

# Add K-Median cluster labels
usarrests_kmed <- usarrests_data %>%
  mutate(Cluster = as.factor(cluster_assignments))
```

```r
# Add the cluster assignments to the PCA data frame
pca_df$Cluster <- as.factor(cluster_assignments)

ggplot(pca_df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "K-Median Clustering on USArrests", x = "PC1", y = "PC2") +
  theme_minimal()
```



---

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based method that identifies clusters based on areas of high data point density, making it effective for non-spherical clusters and noisy data.

Key Parameters

1. ε: Radius defining the neighborhood of a data point.
2. **minPts**: Minimum number of points required to form a dense region (cluster).

Steps

1. For each data point, calculate its neighborhood (points within radius ε).

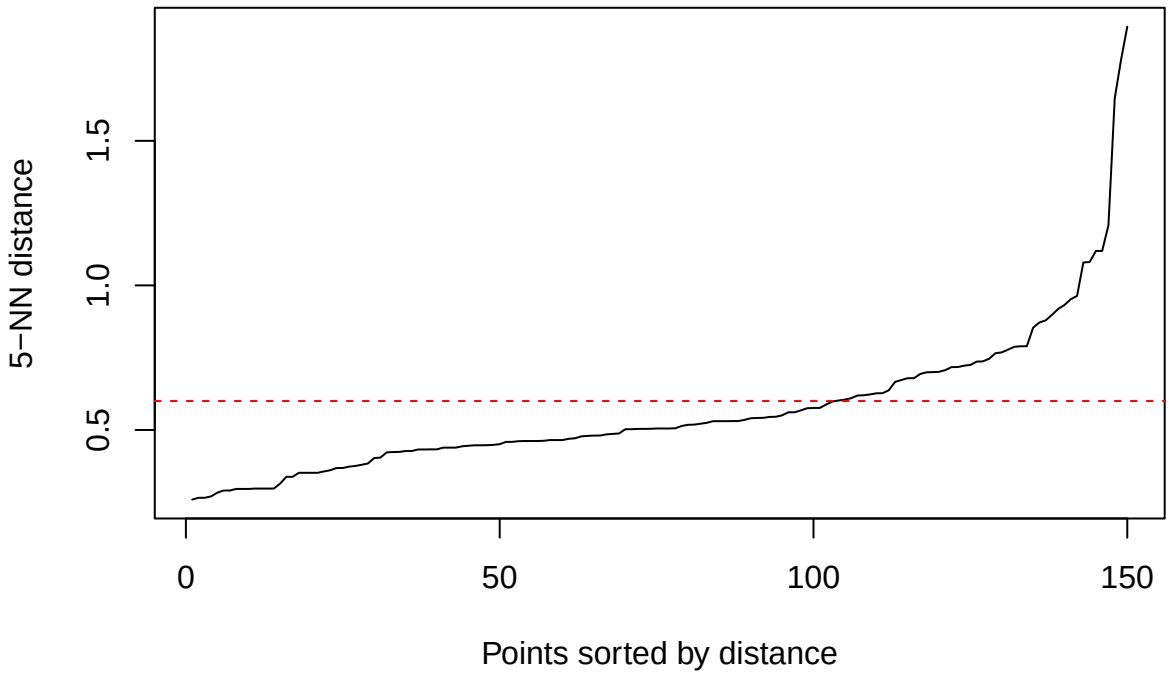
2. Classify data points as:
    - **Core Point**: Has at least minPts neighbors.
    - **Border Point**: Lies in the neighborhood of a core point but has fewer than minPts neighbors.
    - **Noise Point**: Neither core nor border point.
3. Expand clusters from core points, including all reachable core and border points.
4. Label noise points as outliers.

Applications

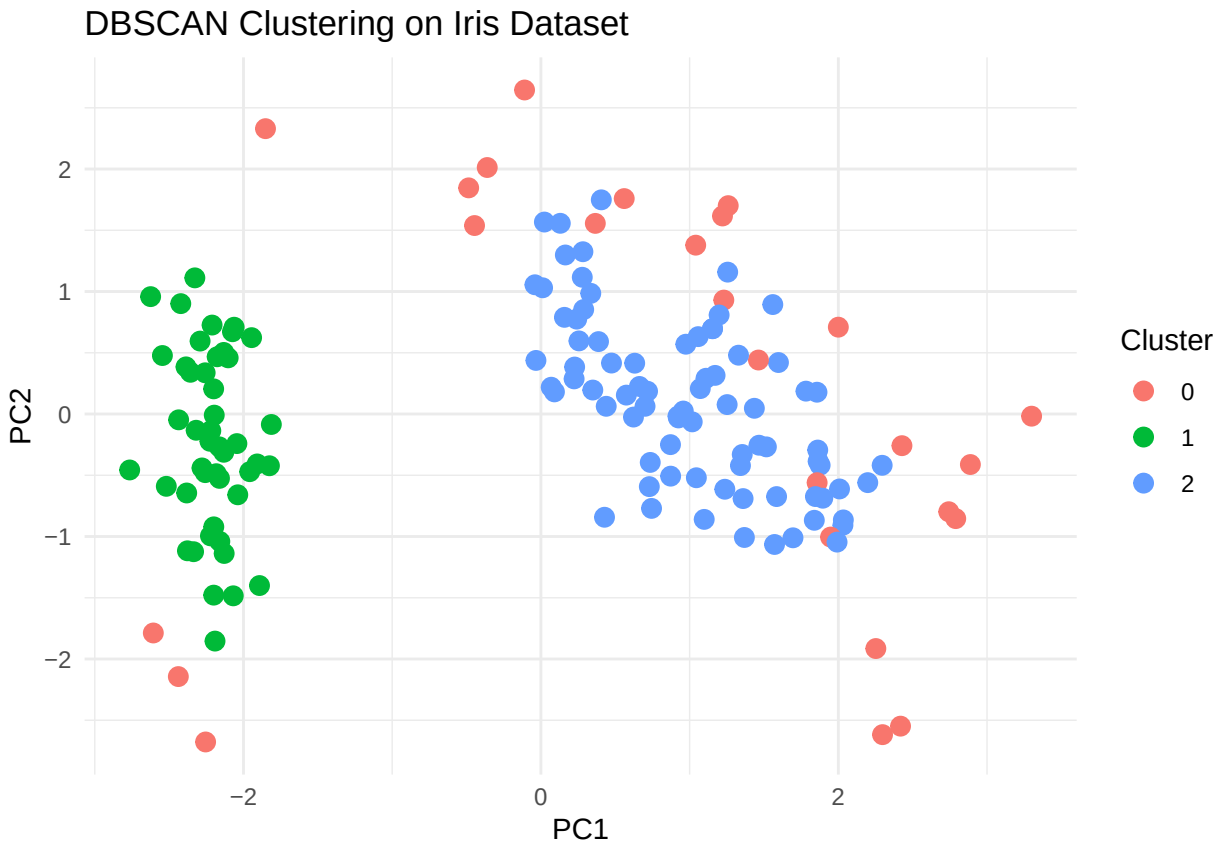
- Geospatial data analysis
- Fraud detection
- Clustering irregular shapes in image data

Example Dataset

```r
# Load required libraries
library(dbscan)
library(tidyverse)
library(factoextra)  # For PCA and visualization

# Load the iris dataset
data("iris")
iris_features <- iris[, 1:4]

# Scale the data
iris_scaled <- scale(iris_features)

# Estimate eps using k-NN distance plot (optional visual aid)
kNNdistplot(iris_scaled, k = 5)
abline(h = 0.6, lty = 2, col = "red")
```




```r
# Apply DBSCAN
dbscan_result <- dbscan(iris_scaled, eps = 0.6, minPts = 5)

# Add cluster labels to the dataset
iris$DBSCAN_Cluster <- as.factor(dbscan_result$cluster)

# Reduce to 2D using PCA for plotting
pca_res <- prcomp(iris_scaled)
pca_df <- as.data.frame(pca_res$x[, 1:2])
pca_df$Cluster <- iris$DBSCAN_Cluster

# Plot DBSCAN clustering result
ggplot(pca_df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "DBSCAN Clustering on Iris Dataset", x = "PC1", y = "PC2") +
  theme_minimal()
```



Question 2: Other Clustering Methods

Agglomerative Clustering

- Builds a hierarchy of clusters by starting with each data point as its own cluster and successively merging the closest pairs of clusters.
- Merging is based on a similarity metric (e.g., single linkage, complete linkage, or average linkage).
- Output can be visualized using a **dendrogram**.

Mean Shift Clustering

- Iteratively shifts data points towards the densest area in the feature space (density peaks).
- The number of clusters is determined automatically based on the data distribution.

Spectral Clustering

- Constructs a similarity graph where nodes represent data points and edges are weighted by a similarity measure.
- Applies **eigenvalue decomposition** to this graph and performs clustering (e.g., K-Means) in the transformed space.

Question 3: Gradient Operator and Gradient Descent

Gradient Operator (∇)

The gradient operator is a vector of partial derivatives that points in the direction of the steepest ascent of a scalar function.

For a function $f(x_1, x_2, \ldots, x_n)$, the gradient is:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Properties

- Gradient magnitude represents the rate of change.
- Direction of ∇f is perpendicular to the level sets of f.

Gradient Descent

Gradient Descent is an iterative optimization algorithm for minimizing a function.

Steps

1. Initialize parameters (θ) randomly.
2. Compute the gradient ($\nabla f(\theta)$).
3. Update parameters:

$$\theta = \theta - \eta \nabla f(\theta)$$

where η is the learning rate.

4. Repeat steps 2 and 3 until convergence.

Variants

1. **Batch Gradient Descent**: Computes gradient using the entire dataset.
2. **Stochastic Gradient Descent (SGD)**: Computes gradient using one sample at a time.
3. **Mini-batch Gradient Descent**: Uses a small subset of the data for computing gradients.

Applications

- Training machine learning models
- Optimizing cost functions