

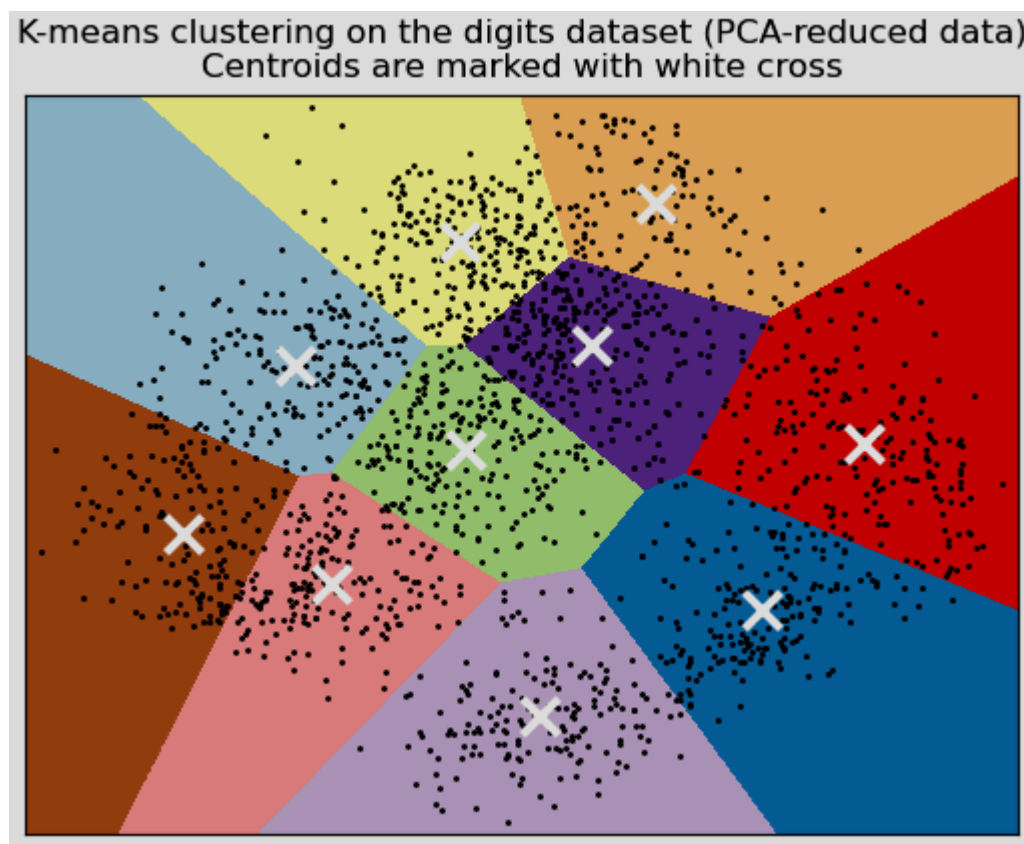
K-Means Algorithm

Developer: Hubert Gąsior

Tester: Kacper Lalik

Algorithm

The K-Means algorithm is one of the most popular clustering methods, used to group data into k disjoint clusters by minimizing the within-cluster sum of squares (inertia). It operates iteratively, starting with a random initialization of centroids, then repeatedly performing two steps: assigning each sample to the nearest centroid, and updating the centroid positions as the mean of all samples assigned to a given cluster. The process continues until the centroid movements fall below a specified threshold, indicating convergence. The algorithm always converges, though it may settle in local minima, and its performance depends on the initial choice of centroids. In high-dimensional spaces, Euclidean distances can lose interpretability, so dimensionality reduction is often applied beforehand. The algorithm can also be understood geometrically through Voronoi diagrams, where each region represents the area of sample assignments for a particular centroid. K-Means is fast, scalable, and widely used, and it additionally supports sample weighting, allowing selected observations to have a greater influence on centroid positions and the inertia value.



Overview

KMeansAlgorithm is a simple, pure-Python implementation of the K-Means clustering algorithm.

It:

- Clusters numeric data into a fixed number of groups (clusters)
- Computes cluster centroids as the mean of assigned samples
- Iteratively refines cluster assignments until convergence or a maximum number of iterations is reached

Project Structure

The core of the project is a single class:

All logic for K-Means clustering is implemented as instance methods of KMeansAlgorithm.

Requirements

- Python 3.x
- No external dependencies

API Reference

Class: KMeansAlgorithm

`__init__(self)`

Constructor, currently does not accept any parameters or perform any setup.

CalculateSquaredDistance(self, x1, x2)

Compute the squared Euclidean distance between two vectors.

Parameters:

- x1: list of floats – first sample
- x2: list of floats – second sample same length as x1

Returns:

- float – sum of squared differences between corresponding components of x_1 and x_2

CalculateCentroids(self, clusters, centroids)

Recalculate centroids as the mean of samples in each cluster.

Parameters:

- clusters: list of clusters, where each cluster is a list of samples

Behavior:

- For each cluster:
 - If the cluster has samples:
 - Compute the mean of each feature across all samples
 - Use that mean as the new centroid
 - If the cluster is empty:
 - Keep the previous centroid

Returns:

- newCentroids: list of recalculated centroids

AssignToClusters(self, samples, centroids)

Assign each sample to the nearest centroid using squared Euclidean distance.

Parameters:

- samples: list of samples, each a list of floats
- centroids: list of current centroids

Behavior:

- For each sample:
 - Calculate squared distance to each centroid using `CalculateSquaredDistance()`

- Find the centroid with the smallest distance
- Append the sample to that centroid's cluster

Returns:

- clusters: list of clusters
Each cluster is a list of samples assigned to that centroid:
clusters corresponds to centroids.

KMeans(self, samples, centoridsNumber, threshold=1e-4, maxIterations=100)

Main method that runs the K-Means clustering algorithm.

Parameters:

- samples: list of samples, each a list of floats
- centoridsNumber: int – number of clusters
- threshold: float, optional. Convergence threshold based on the sum of squared centroid movements. Default: 1e-4.
- maxIterations: int, optional. Maximum number of iterations. Default: 100.

Algorithm Steps:

1. Initialization
2. centroids = random.sample(samples, centoridsNumber)

Randomly selects k samples as initial centroids.

3. Iterative Refinement

For up to maxIterations iterations:

- Assign samples to the nearest centroid:
- Recompute centroids
- Compute total movement of centroids (sum of squared distances between old and new centroids):
- If $\text{diff} < \text{threshold}$, stop (algorithm has converged).
- Otherwise, update centroids and continue:

4. Result

Returns final centroids and clusters.

Returns:

- centroids: list of final centroid positions
- clusters: list of clusters (each a list of samples)

Assumptions and Limitations

- All samples:
 - Are numeric vectors (lists of floats or ints)
 - Have the same length (same number of features)
- The number of centroids (centroidsNumber) must be:
 - A positive integer
 - Less than or equal to the number of samples (due to random.sample)
- No explicit input validation is performed:
 - Invalid inputs (e.g. empty sample lists, inconsistent dimensions) may cause runtime errors.
- Initialization uses random sampling:
 - Different runs may produce different results.
 - There is no K-Means++ or deterministic initialization.
- Convergence criterion:
 - Based on centroid movement (diff), not on cluster assignment stability.

Tests

The project includes a comprehensive pytest-based tests. The tests validate the correctness, stability, and consistency of the custom KMeansAlgorithm implementation. They also verify compatibility with scikit-learn's KMeans on simple datasets, ensuring the algorithm behaves as expected.

1. Distance Calculation

Tests verify that CalculateSquaredDistance correctly computes squared Euclidean distance for:

- Zero vectors
- Positive values
- Negative values

This ensures numerical correctness in the core distance metric used across the algorithm.

2. Centroid Recalculation

The CalculateCentroids method is tested for:

- Correct computation of feature-wise means for clusters
- Handling of empty clusters by preserving the previous centroid
- Multi-dimensional feature vectors

These tests confirm that centroid updates follow expected K-Means behavior.

3. Cluster Assignment

AssignToClusters is tested to confirm:

- Samples are assigned to the nearest centroid
- Proper handling of all samples mapping to one cluster
- Creation of exactly k clusters

This ensures correctness of the core clustering step.

4. Full K-Means Algorithm Behavior

The KMeans method undergoes several behavioral tests:

- Output shape validation: number of centroids and clusters match k
- Cluster completeness: all samples are assigned exactly once
- Convergence: algorithm halts when centroids stabilize
- Performance on structured datasets: algorithm must roughly recover well-separated clusters

5. Comparison with scikit-learn

A dedicated test compares cluster centers produced by this implementation with those produced by:

On a simple two-cluster dataset, the centroids must fall within a small tolerance of scikit-learn's result.

This ensures the implementation behaves similarly to a widely-used reference algorithm.

6. Early Stopping Verification

One test uses monkeypatch to override CalculateCentroids so that it returns unchanged centroids.

The algorithm should detect no movement and stop after one iteration, verifying correct use of the convergence threshold.