Lesson Clustering Algorithms

Clustering Defined

Mathematically, clustering looks a bit like classification: we wish to find a mapping from datapoints, , to categories, . However, rather than the categories being predefined labels, the categories in clustering are automatically discovered *partitions* of an unlabeled dataset.

Because clustering does not learn from labeled examples, it is an example of an *unsupervised* learning algorithm. Instead of mimicking the mapping implicit in supervised training pairs$\{x^{(\mathrm{i})}, y^{(\mathrm{i})}\}_i^n = 1,$, clustering assigns datapoints to categories based on how the unlabeled data $\{x^{(\mathrm{i})}\}_{\mathrm{i}=1}^n$ is *distributed* in data space.

- -https://introml.mit.edu/notes/clustering.html

Some Basics

- argparse is a Python module that makes it easy to write programs that accept command-line arguments. You simply use

```
import argparse
parser = argparse.ArgumentParser()
```

- pd.errors.EmptyDataError is an exception raised by pandas when you try to read a CSV (or other file) that contains no data at all.

K Nearest Neighbor

Classification normally requires partitioning samples into training and testing categories. Let **x***i* be a training sample and **x** be a test sample, and let ω be the class of a training sample and ω^ be the predicted class for a test sample (ω,ω^=1,2,…,Ω) . Here, Ω is the total number of classes. This may seem like an overly mathematical representation of the concept, but is actually simpler than it appears. The goal is to use training algorithms to find classes in the data, then match test data to the appropriate classes.

- The K Nearest Neighbors algorithm was first developed long before machine learning. It was created by Evelyn Fix and Joseph Hodges in 1951 for use in statistics. This algorithm can be used for classification or regression.

K Nearest Neighbor

- When used for classification, the concept is relatively simple, to take the input samples and determine what the closest classification is, and classify the input data. The algorithm uses training examples that are vectors in some multidimensional space (recall chapter 1 discussion of vectors). The algorithm then determines which classification is closest to the input data.
- When working with continuous variables, the most commonly used algorithm for determining the closest classification is Euclidean distance. Euclidean distance is the length of a line segment between two points. There are many variations of this formula. The first is for data that has two dimensions. There is a point p and a point q, with coordinates. The distance between p and q is :

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

K Nearest Neighbor

If the data has additional dimensions then that can be extended as shown here:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$$

It is also possible to compute distance using polar coordinates. The polar coordinates of a given point p are (r, θ). This gives the formula shown here:

$$d(p,q) = \sqrt{r_\rho^2 + r_q^2 - 2r_p r_q(\theta_p - \theta q)}$$

K Nearest Neighbor

- KNN depends on dimensionality reduction. There are several algorithms that can be used for this purpose. Principle Component Analysis, is one of the most common. PCA is essentially a statistical technique for reducing the dimensionality of a dataset. The principal components for a set of points in a real coordinate space are a series of p unit vectors where the i-th vector is the direction of a line that best fits the data.
- Another option is linear discriminant analysis. Linear discriminant analysis (LDA) is closely related to the ANOVA test that was discussed in chapter 2. A brief refresher on ANOVA is given here. ANOVA, or analysis of variation, is actually a family of statistical tests. We will focus on just one. The one-way analysis of variance (ANOVA) is used to determine whether there are any statistically significant differences between the means of two or more independent groups. The one-way ANOVA compares the means between the groups and determines if the means are statistically different from each other.
- LDA is a method to find a linear combination of features that either characterizes or separates two or more classes. The combination resulting from the process can be used as a linear classifier or for dimensionality reduction. Consider a machine learning situation where you have a number of classes, each with multiple features. If you use, for example, just one feature to classify data, you will have a great deal of overlap.

K Nearest Neighbor

KNN is implemented in the scikit-learn library, inside:

from sklearn.neighbors import KNeighborsClassifier from sklearn.neighbors import KNeighborsRegressor

from sklearn.neighbors import KNeighborsClassifier knn = KNeighborsClassifier(n_neighbors=5)

knn.predict([[5.6, 2.8]])

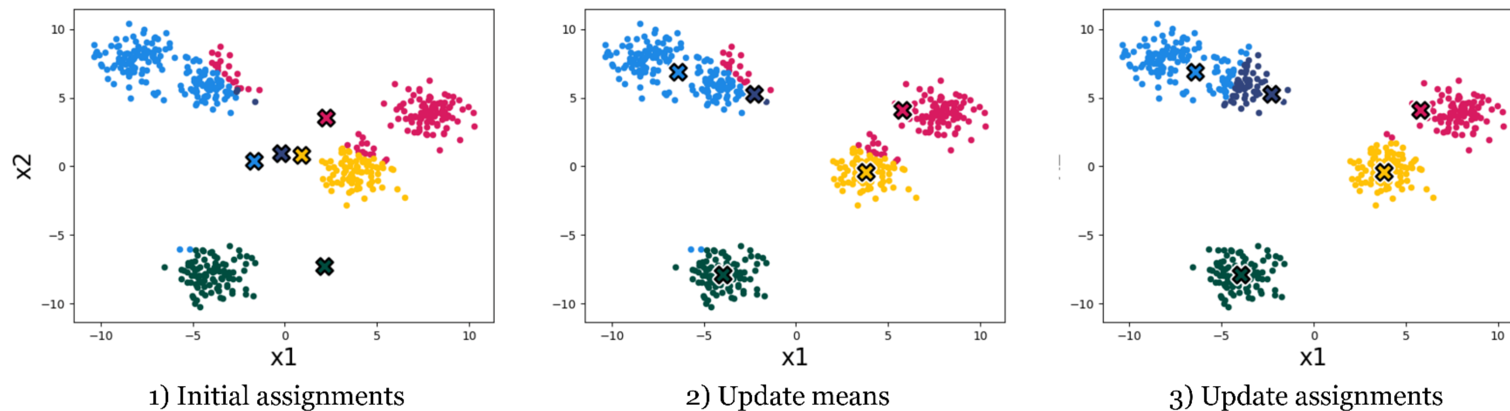
Parameter	Meaning
n_neighbors	Number of neighbors (K)
weights	'uniform' (all neighbors equal) or 'distance' (closer neighbors matter more)
metric	Distance metric (default: 'minkowski', which becomes Euclidean when p=2)
p	Power parameter for Minkowski metric (p=1 = Manhattan, p=2 = Euclidean)
algorithm	Strategy used to speed up neighbor searches ('auto', 'ball_tree', 'kd_tree', 'brute')

K-Means Clustering

K means clustering was originally a method for vector quantization in signal processing. Vector quantization is a method of modeling probability density functions using the distribution of vectors. This technique has also been used for data compression. K means clustering is an algorithm that seeks to find clusters in the dataset. K-means clustering is one of the easiest and unsupervised machine learning algorithms, which also makes it quite popular. The goal is rather simple. Take the input data and group similar data points together. This will aid in discovering patterns in the data. The term k-means was first used in 1967 by James MacQueen. There was a primitive version of the algorithm used by Bell Labs in 1957. The standard algorithm is often called naïve k-means. This algorithm is widely used in data mining.

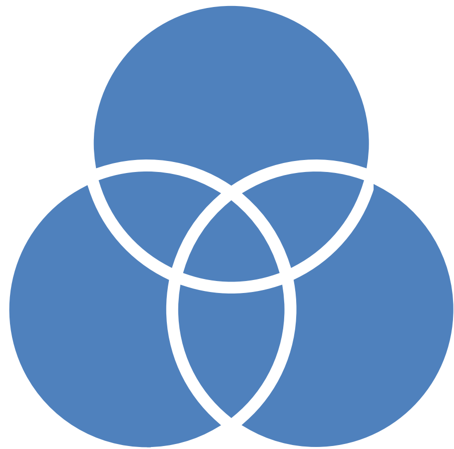
K-Means Clustering

- The k-means algorithm minimizes this loss by alternating between two steps: given some initial cluster assignments: 1) compute the mean of all data in each cluster and assign this as the "cluster mean", and 2) reassign each datapoint to the cluster with nearest cluster mean. Fig. 1.2 shows what happens when we repeat these steps on the dataset from above.



1) Initial assignments 2) Update means 3) Update assignments

- https://introml.mit.edu/notes/clustering.html



Clustering Terms

- Davies-Bouldin score: The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score.
- Calinski Harabasz score: The score is defined as ratio of the sum of between-cluster dispersion and of within-cluster dispersion.
- Silhouette score: The Silhouette Coefficient is calculated using the mean intra-cluster distance and the mean nearest-cluster distance for each sample.

K-Means Clustering

The essence of the algorithm begins by defining the number of centroids you want to find in the data set. This is the target number, k. A centroid is just the center of a cluster. Essentially you are defining how many clusters you want produced from the dataset. At this point, you won't know what data will be in a cluster, nor how it is clusters. You will have simply defined how many clusters you are trying to find. Each data point is then allocated to the centroid that it has the closest match to. The algorithm starts with the first group of randomly selected centroids (you choose the number, but the values are randomly chosen). Then there is an iterative set of calculations to determine the optimum positions (values) for the centroids. Once those have been stabilized, then the algorithm can stop.

K-Means Clustering

- The scikit package is often used for k-means clustering. With that package you have the following parameters:
- **max_iter**: the maximum number of iterations. The default is 300.
- Init: This is the initialization of the centroids.
- **n_init**: The number of times the algorithm will execute using a different seed each time.
- **Algorithm**: There are many variations of k-means clustering you can use. The default is the Lloyd or naïve k-means.
- **tol**: The tolerance regarding the Frobenius norm of the difference in the cluster centers of two consecutive iterations. This is used to determine if convergence has occurred.
- **cluster_centers_**: This is an array of coordinates for cluster centers.
- **labels_**: This is an array of labels for each point.
- **inertia_**: This is a floating point value that has the sum of squared distances of samples with their closest cluster center.
- .

K-Means Clustering

In Python, the KMeans object is a class inside the sklearn.cluster module that performs K-Means clustering, a common unsupervised machine-learning algorithm.

from sklearn.cluster import Kmeans

kmeans = KMeans(n_clusters=3)

he KMeans object has several key attributes after you run .fit().

cluster_centers_

An array of the coordinates of each cluster center.

For example, if K=3, you get 3 centers.

labels_

The cluster assignment for each data point.

Example: [0, 2, 1, 1, 0, ...]

inertia_

A measure of how compact the clusters are (lower is better).

n_iter_

How many iterations the algorithm ran before converging.

K-means vs K-nearest neighbor

Feature	**KNN (K-Nearest Neighbors)**	**K-Means Clustering**
Type of algorithm	Supervised learning	Unsupervised learning
Purpose	Classify or regress new data points	Group unlabeled data into K clusters
Requires labeled data?	✓ Yes	✗ No
When is K used?	At prediction time (how many neighbors to check)	During training (how many clusters to form)
What happens during training?	No "training" — KNN only stores data	K-Means repeatedly adjusts cluster centers
Memory usage	High (stores entire dataset)	Low (stores only cluster centers)
Speed	Slow when predicting (distance to all points)	Slow when training large datasets (many iterations)
Distance metric	User-defined: Euclidean, Manhattan, etc.	Same metrics possible; usually Euclidean
Output	A class or predicted value for a new point	Cluster assignments + center positions
Model shape	Lazy learner (no model built)	Produces explicit structure (cluster centers)
Can handle streaming data?	Poorly	Poorly unless modified
Main idea	“Look at nearby labeled examples”	“Find natural groups in the data”

K-Means++

This algorithm is much like K-means, but focuses on the choice of the initial seed values. The goal of K-means is to find cluster centers that minimize the variance within the class. This is often determined by the sum of squared distances from each data point to its cluster center. K-means++ is directed towards finding the initial cluster centers, thus improving the efficacy of K-Means. The steps of this algorithm are given here:

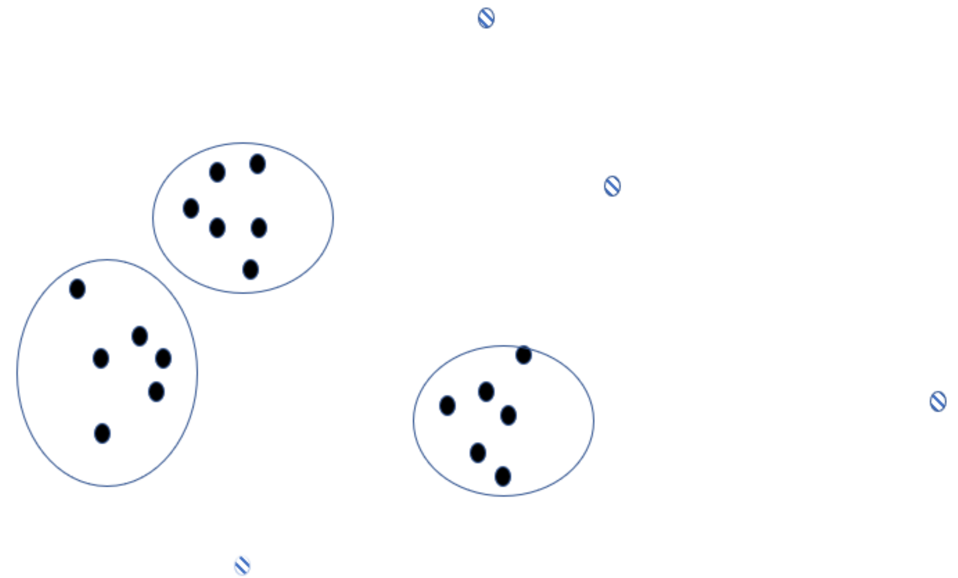
- 1. Select one center uniformly at random among the data points.
- 2. For each data point x not yet selected, calculate the distance between x and the nearest center (Dx) that has already been selected.
- 3. Select one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to D(x)2.
- 4. Repeat Steps 2 and 3 until k centers have been chosen.
- 5. Now that the initial centers have been chosen, proceed using standard k-means clustering.
- As you can see, this process is about selecting better centroids for K-Means clustering. Many libraries allow you to use K-Means++ as the K-Means algorithm. The following code implements a basic K-means++.

K-medoids

Sometimes K-Medians is confused with K-medoids, however they are quite different algorithms. K-medoids is a clustering algorithm that is similar to K-Means. Rather than select centroids then match data points to those centroids, K-medoids selects some of the actual datapoints as centers. This is where the name of this algorithm originates. A medoid is a representative object of a dataset whose sum of dissimilarities to all the other objects in the dataset is minimal. There are variations of K-medoids such as Partitioning Around Medoids (PAM).

DBSCAN

It is a common clustering algorithm, and thus related to K-means clustering. DBSCAN is an acronym for *Density-based spatial clustering of applications with noise*. The term density based requires some explanation. Density-based clustering involves grouping together points that are closely packed together and those that lie in very low-density regions are considered noise. These data points that have others nearby are clustered into groups. These are represented by solid black data points. Those data points that are far from any other data points are treated as noise. Those are represented by striped circles



DBSCAN

The algorithm essentially has the following criteria for clustering:

- A point p is a core point if there are at least some minimum number of other points within distance ε of point p. You decide what the minimum number of points is and the distance.
- A point is q is considered directly reachable from point p, if point q is within distance ε of point p.
- A point q is reachable (note not directly reachable, just reachable) from p if there is some path p1,…,pn and pn = q where each pi+1 is directly reachable from pi. In other words, you can trace a series of points from p to q wherein each point is directly reachable from the previous and subsequent points.
- Any points that are not reachable are considered noise.

Thus, to use DBSCAN you need two parameters. The minimum number of points in an area to consider it dense (minimum points) and the distance ε that constitutes nearby. The scikit-learn library has a DBSCAN sklearn.cluster.DBSCAN, you just set the parameters and you are using DBSCAN.

SciPy

SciPy (pronounced “Sigh Pie”) is an open-source scientific computing library for Python built on top of NumPy. It provides a large collection of algorithms and utilities for:

- Numerical integration
- Optimization
- Linear algebra
- Signal processing
- Statistics and probability
- Sparse matrices
- Differential equation solvers
- Interpolation
- Fourier transforms

SciPy enables Python to function as a powerful computational environment, similar to MATLAB, Mathematica, or R, but entirely open-source and integrated with the broader Python ecosystem. SciPy is built on NumPy.

SciPy

scipy.integrate — Numerical Integration and ODE Solvers

- Provides algorithms for:
- **Definite integrals** (single, double, triple)
- **Ordinary differential equations (ODEs)**
- **Simpson's rule, trapezoidal rule**
- **Quadrature methods**
- Examples of key functions:
- quad() — general-purpose integrator
- dblquad(), tplquad() — multiple integrals
- odeint() — solve ODE systems
- solve_ivp() — modern ODE solver with many methods (RK45, BDF, etc.)

scipy.optimize — Optimization and Root Finding

- Includes algorithms for:
- Function minimization (global & local)
- Least-squares optimization
- Root finding (solve f(x) = 0)
- Linear programming

scipy.linalg — Advanced Linear Algebra

- Extends NumPy’s limited linear algebra with:
- Matrix decompositions:
  - LU, QR, Cholesky
  - Singular Value Decomposition (SVD)
  - Schur decomposition
- Matrix functions:
  - Exponential (expm)
  - Logarithm (logm)
  - Square root (sqrtm)

scipy.stats — Probability and Statistics

- This module is extremely comprehensive and rivals many features of R:
- Hundreds of probability distributions:
  - Continuous: normal, gamma, beta, t, χ^2, etc.
  - Discrete: binomial, Poisson, geometric, etc.
- Statistical tests:
  - t-test, ANOVA, chi-square, KS test
  - Non-parametric tests: Wilcoxon, Mann-Whitney, Kruskal-Wallis
- Descriptive statistics
- Random variable generation
- Kernel density estimation

SciPy

- **scipy.linalg — Advanced Linear Algebra**
- Extends NumPy's limited linear algebra with:
- Matrix decompositions:
  - LU, QR, Cholesky
  - Singular Value Decomposition (SVD)
  - Schur decomposition
- Matrix functions:
  - Exponential (expm)
  - Logarithm (logm)
  - Square root (sqrtm)
- **scipy.stats — Probability and Statistics**
- This module is extremely comprehensive and rivals many features of R:
- Hundreds of probability distributions:
  - Continuous: normal, gamma, beta, t, χ^2, etc.
  - Discrete: binomial, Poisson, geometric, etc.
- Statistical tests:
  - t-test, ANOVA, chi-square, KS test
  - Non-parametric tests: Wilcoxon, Mann-Whitney, Kruskal-Wallis
- Descriptive statistics
- Random variable generation
- Kernel density estimation

SciKit Learn

Let us examine sklearn.neighbors module. Examples include neighbors.kdtree, .neighbors.NearestNeibhors, and neighbors.NearestCentroid. There are other related algorithms build into SciKit Learn. There is neighbors.KernelDensity which is used for kernel density estimation. There is also neighbors.KNeighbors.Regressor used for regression with k-nearest neighbor. Another interesting variation is neighbors.RaduiusNeighborsClassifier. In this algorithm, voting is done by neighbors within a given radius as opposed to using metrics such as Euclidean distance or Mahalanobis distance. There are, as of this writing, three variations built into SciKit Learn: neighbors.RadiusNeighborsClassifier, neighbors.RadiusNeighbors.Regressor, and neighbors.RadiusNeighborsTransformer.

HDBSCAN

- HDBSCAN is a clustering algorithm developed by Campello, Moulavi, and Sander. It extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters. The goal of this notebook is to give you an overview of how the algorithm works and the motivations behind it. In contrast to the HDBSCAN paper I'm going to describe it without reference to DBSCAN. Instead I'm going to explain how I like to think about the algorithm, which aligns more closely with Robust Single Linkage with flat cluster extraction on top of it.

- To find clusters we want to find the islands of higher density amid a sea of sparser noise – and the assumption of noise is important: real data is messy and has outliers, corrupt data, and noise. The core of the clustering algorithm is single linkage clustering, and it can be quite sensitive to noise: a single noise data point in the wrong place can act as a bridge between islands, gluing them together. Obviously we want our algorithm to be robust against noise so we need to find a way to help 'lower the sea level' before running a single linkage algorithm.

- https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

HDBSCAN

- It does not require you to pick the number of clusters.
- It handles clusters with different densities much better than DBSCAN.
- It still marks low-density points as noise/outliers like DBSCAN.
- Points in dense regions become effectively closer to each other.
- Sparse bridges between clusters become effectively longer, making it easier to separate clusters that DBSCAN might connect.

HDBSCAN

Step 1 – Transform to mutual-reachability space
Compute the mutual reachability distance between all pairs of points (or an approximation of that).
Step 2 – Build a Minimum Spanning Tree (MST)
Consider a graph where:
- nodes = data points
- edge weight = mutual reachability distance

Build the minimum spanning tree (MST) of this graph. This MST encodes how points connect as density decreases.
Step 3 – Build a cluster hierarchy (like a dendrogram)
Imagine gradually lowering the density requirement (equivalently, increasing ε). As you move through the MST edges, clusters merge and split based on connectivity.
This creates a hierarchical tree of clusters:
- At high density: many small clusters.
- At lower density: fewer, larger clusters.

Step 4 – “Condense” the hierarchy using min_cluster_size
HDBSCAN uses another parameter: min_cluster_size. Branches in the hierarchy that never have at least min_cluster_size points are pruned away. The result is a condensed tree, where every remaining branch corresponds to groups that are “big enough” to be meaningful clusters.
Step 5 – Compute cluster stability
For each possible cluster (a branch in the condensed tree), HDBSCAN computes a stability score: Roughly: how long does this cluster exist across the hierarchy as density changes, weighted by how many points it has.
Clusters that persist over a wide density range are considered stable, i.e., robust, not just random artifacts.
Step 6 – Select the final flat clustering
Finally, HDBSCAN: Picks a set of clusters that maximizes total stability while avoiding overlaps.
Points that do not belong to any stable cluster are treated as noise.
The output is a flat set of clusters (like a single run of DBSCAN), but chosen from the hierarchy in a principled way.

STING (Statistical Information Grid)

STING (Statistical Information Grid) is a grid-based clustering algorithm that organizes data into hierarchical, rectangular cells and uses statistical summary information rather than raw data to identify clusters efficiently. The data space is first divided into a uniform grid, and each cell stores descriptive statistics such as the number of points, mean, variance, minimum, and maximum values. These cells are then arranged into multiple hierarchical levels—fine grids at the bottom, coarser grids above—similar to a pyramid. When a clustering query is processed, STING moves top-down through the hierarchy, quickly identifying which cells are likely to contain dense regions by evaluating their statistical parameters. Cells that do not meet density or similarity requirements are pruned early, greatly reducing computational effort. Because STING does not rely on iterative optimization and avoids distance-based calculations between every pair of points, it scales extremely well to large datasets, making it suitable for spatial data mining, geographic information systems, and applications involving massive multidimensional datasets. However, because STING's cluster boundaries are constrained by the underlying grid structure, its clusters may appear blocky or less precise compared to density-based methods like DBSCAN. Despite this, STING remains a fast, memory-efficient approach for exploratory clustering in very large spatial datasets.

Markov Clustering (MCL)

Markov Clustering (MCL) is a graph-based clustering algorithm that identifies natural groupings of nodes by simulating random walks on a graph and amplifying the tendency of those walks to remain within densely connected regions. Instead of using distance metrics or feature vectors, MCL operates directly on a graph's adjacency matrix, making it especially powerful for clustering networks such as social graphs, protein–protein interaction networks, citation graphs, and web link structures. The algorithm repeatedly performs two key operations: expansion and inflation. *Expansion* corresponds to taking powers of the adjacency matrix, which simulates random walks of increasing length and spreads flow throughout the graph. *Inflation* rescales each column of the matrix using a power parameter (greater than 1), which strengthens strong connections and suppresses weak ones, effectively sharpening the boundaries between clusters. Over successive iterations, these operations cause the matrix to converge into distinct blocks, each representing a cluster of nodes with strong internal connectivity but weak connections to other clusters. MCL is highly scalable, deterministic, and insensitive to noise, and it does not require specifying the number of clusters in advance. Its inflation parameter allows fine control over cluster granularity, making MCL flexible for both coarse and fine-grained network analysis.

OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm designed to overcome key limitations of DBSCAN, particularly its sensitivity to a single global density threshold. Instead of producing a fixed set of clusters directly, OPTICS generates an ordered list of points together with their reachability distances, which capture how densely each point is connected to its neighbors. This ordering effectively reveals the dataset's intrinsic clustering structure across multiple density levels. Peaks and valleys in the resulting reachability plot correspond to cluster boundaries and internal cluster density variations, allowing analysts to extract clusters at different granularities without rerunning the algorithm. Unlike DBSCAN, OPTICS handles datasets with varying densities much more naturally and reduces the need for carefully tuning parameters like *eps*. It is also capable of identifying complex, arbitrarily shaped clusters and naturally detecting noise points. While OPTICS is more computationally intensive due to maintaining a priority queue and computing reachability distances, its flexibility and ability to visualize density-based structure make it particularly powerful for exploratory data analysis and situations where clusters of differing densities coexist.

OPTICS

From https://www.geeksforgeeks.org/machine-learning/ml-optics-clustering/
A reachability plot is a graph that helps visualize clustering structures. It shows the reachability distance of each point in the dataset. It makes it ordered way based on how OPTICS processes them.
Here clusters appear as valleys in the plot where lower reachability distances indicate dense regions while peaks represent sparse regions or noise. To better understand the concept refer to the below image:

OPTICS

- From https://www.geeksforgeeks.org/machine-learning/ml-optics-clustering/
  - The algorithm selects a starting point and checks if it has at least MinPts neighbors within Eps.
  - If the point meets the density requirement it is marked as a core point and nearby points are then analyzed.
  - Reachability distance is computed for each neighboring point.
  - Points are then processed in order of their reachability distance hence forming a reachability plot.
  - Clusters appear as valleys i.e low reachability distances and noise appears as peaks i.e high reachability distances.

Comparison

Algorithm	Type	Core Idea / Mechanism	Strengths	Weaknesses	Best Use Cases
OPTICS (Ordering Points To Identify the Clustering Structure)	Density-based	Produces an ordering of points representing density-based clustering structure; identifies clusters of varying densities without requiring a global epsilon parameter.	Handles varying densities; identifies cluster hierarchy; robust to noise.	More computationally expensive than DBSCAN; requires interpretation of reachability plots.	Complex datasets with varying density; exploratory cluster structure analysis.
MCL (Markov Clustering)	Graph-based	Simulates random walks on a graph using expansion/inflation operations to discover cluster structure.	Works well for graph data; scalable; easy to implement.	Requires graph representation; performance depends on inflation parameter.	Protein interaction networks, community detection in graphs.
STING (Statistical Information Grid)	Grid-based	Divides space into hierarchical rectangular cells storing statistical summaries; clusters determined by exploring grid hierarchy.	Fast; scalable; incremental updates easy.	Low resolution since grid structure is fixed; cluster boundaries may be inaccurate.	Spatial data mining, large geospatial datasets.
HDBSCAN (Hierarchical DBSCAN)	Density-based	Builds a hierarchy of clusters and extracts stable clusters; no need to choose epsilon; handles varying density better than DBSCAN.	Excellent for complex shapes and varying densities; robust to noise; minimal parameter tuning.	More computationally expensive than DBSCAN; harder to interpret parameters.	NLP embeddings, biological data, anomaly detection, real-world noisy datasets.
DBSCAN	Density-based	Groups points with many neighbors within a radius (ε) and labels sparse points as noise.	Finds arbitrarily shaped clusters; no need to predefine number of clusters; handles noise well.	Struggles with varying density; sensitive to choice of ε and minPts.	Spatial datasets, noise-heavy datasets, anomaly detection.
K-Means++	Centroid-based	Same as K-Means but improves initial centroid selection using a probabilistic method.	Faster convergence; reduces risk of bad initialization; generally better clustering quality.	Still assumes spherical clusters; sensitive to outliers; must specify k.	Large datasets with roughly spherical clusters; general-purpose clustering.
K-Means Clustering	Centroid-based	Iteratively assigns points to nearest centroid and updates centroids to minimize variance.	Simple, fast, scalable.	Poor performance on non-spherical clusters; sensitive to initialization and outliers; must choose k.	Market segmentation, image compression, vector quantization.
K-Nearest Neighbor (KNN)	**NOT a clustering algorithm** (supervised classification)	Predicts label of a data point based on labels of its k nearest neighbors.	Simple, effective; non-parametric.	Slow for large datasets; not used for clustering; sensitive to distance metric.	Classification, regression, recommendation systems.

DENCLUE

- DENCLUE (DENsity-based CLUstEring) is a density-based clustering algorithm that models the overall data distribution using mathematical density functions, typically constructed via kernel density estimation. Instead of relying on distance thresholds alone, DENCLUE identifies clusters by locating density attractors, which are local maxima of the estimated density function, and assigns data points to clusters based on convergence to these attractors through gradient ascent. This approach allows DENCLUE to efficiently discover clusters of arbitrary shape, handle noise naturally by ignoring low-density regions, and scale well to large datasets through the use of compact density representations. DENCLUE is particularly effective in high-dimensional spaces compared to many traditional density-based methods, though its performance depends on appropriate choices of kernel type and bandwidth parameters.

DPC (Density Peaks Clustering)

- Density Peaks Clustering (DPC) is a density-based clustering algorithm that identifies cluster centers as data points characterized by both high local density and a large distance from other points with higher density. The method first computes the local density of each point—typically based on distances to neighboring points—and then measures the minimum distance to any point with greater density. Points that simultaneously exhibit high density and high separation are selected as cluster centers, while remaining points are assigned to the same cluster as their nearest higher-density neighbor. DPC is notable for its conceptual simplicity, ability to detect clusters of arbitrary shape, and minimal assumptions about cluster geometry, though it requires careful selection of density and distance parameters and can be computationally intensive for very large datasets due to pairwise distance calculations.

StreamKM++

- StreamKM++ is a streaming clustering algorithm designed to efficiently approximate the k-means objective for large-scale or continuously arriving data. It extends the k-means++ initialization strategy to the data stream setting by maintaining compact summaries, called coresets, which preserve the essential clustering structure of the observed data while using limited memory. As new data points arrive, StreamKM++ incrementally updates these coresets and periodically reclusters them to produce near-optimal cluster centers without storing the entire dataset. This approach enables StreamKM++ to scale to high-throughput data streams and massive datasets while providing theoretical guarantees on clustering quality, making it well suited for real-time analytics and resource-constrained environments.

SUBCLU

- SUBCLU (Subspace Clustering) is a density-based subspace clustering algorithm designed to discover clusters that exist in lower-dimensional subspaces of high-dimensional data, where traditional clustering methods often fail due to the curse of dimensionality. SUBCLU extends the principles of DBSCAN by exploiting the downward closure property, which states that if a set of points forms a density-based cluster in a k-dimensional subspace, then it must also form clusters in all (k−1)-dimensional projections of that subspace. The algorithm works by first identifying dense clusters in one-dimensional subspaces and then iteratively generating and testing higher-dimensional subspaces by combining those lower-dimensional candidates. In each candidate subspace, DBSCAN is applied to find density-connected point sets, allowing SUBCLU to detect clusters of arbitrary shape while automatically handling noise. Although SUBCLU is effective at uncovering meaningful patterns hidden in subsets of dimensions, it can be computationally expensive due to the large number of possible subspaces, making it best suited for moderate-sized datasets or scenarios where interpretability of subspace-specific clusters is important.

ORCLUS

- ORCLUS (ORiented projected CLUStering) is a subspace clustering algorithm specifically designed for high-dimensional datasets in which clusters may exist in different, cluster-specific subspaces rather than across all dimensions. Unlike density-based subspace methods, ORCLUS is a partitioning-based approach that extends the K-means paradigm by allowing each cluster to be represented in a lower-dimensional, arbitrarily oriented subspace. It begins by initializing a relatively large number of clusters and progressively merges them while simultaneously reducing the dimensionality of each cluster's associated subspace using eigenvalue decomposition of the covariance matrix. This enables ORCLUS to identify clusters that are compact in their own relevant dimensions, even if they overlap in the full feature space.
- A key strength of ORCLUS is its ability to detect clusters that differ not only in location but also in orientation and dimensional relevance, making it well-suited for applications such as gene expression analysis, text data, and other domains with many irrelevant attributes. However, ORCLUS requires the number of clusters and subspace dimensionality to be specified in advance, and its reliance on repeated eigen-decompositions can make it computationally intensive for very large datasets. Despite these limitations, ORCLUS remains an important algorithm for understanding structure in high-dimensional data where traditional clustering techniques are ineffective.

Clustering Algorithms for Temporal Data

- Clustering algorithms for temporal data are designed to group observations that evolve over time, taking into account not only the values of the data points but also their ordering, temporal dependencies, and dynamic behavior. Unlike static clustering, temporal clustering must address challenges such as varying sequence lengths, time shifts, noise, and differing sampling rates. As a result, many temporal clustering approaches rely on specialized distance measures or probabilistic models that capture similarities in shape, trend, or temporal structure rather than simple pointwise proximity.
- One common family of temporal clustering methods is distance-based time-series clustering, where sequences are compared using metrics tailored to time-dependent data. Dynamic Time Warping (DTW)–based clustering is a prominent example, as DTW aligns sequences that may be out of phase in time, allowing similar patterns with temporal distortions to be grouped together. Shape-based approaches such as K-Shape use normalized cross-correlation to cluster time series according to their overall shape while remaining invariant to scaling and shifting. Symbolic methods like SAX (Symbolic Aggregate approXimation) clustering first convert time series into symbolic representations, enabling efficient clustering of large temporal datasets.
- Another important class consists of model-based temporal clustering algorithms, which assume that each cluster is generated by an underlying temporal process. Hidden Markov Models (HMMs) are widely used in this context, with each cluster represented by an HMM capturing state transitions and emission probabilities. Sequences are clustered by evaluating their likelihood under different models, making this approach especially effective for sequential data with latent states, such as speech signals, biological sequences, or user behavior logs. Autoregressive and state-space models are also used to cluster time series based on their dynamic properties.
- Finally, streaming and evolutionary clustering algorithms address scenarios where temporal data arrives continuously and cluster structures may change over time. Methods such as CluStream, DenStream, and D-Stream maintain evolving summaries of the data, allowing clusters to adapt to concept drift and emerging patterns. These approaches are particularly relevant in real-time applications like sensor networks, financial monitoring, and network traffic analysis, where both the temporal ordering of data and its evolution over time are central to meaningful clustering.

Dynamic Time Warping

DTW-based clustering is a time-series clustering approach that groups temporal sequences using Dynamic Time Warping (DTW) as the similarity measure, rather than simple pointwise distances such as Euclidean distance. DTW aligns two time series by non-linearly warping the time axis, allowing similar patterns that occur at different speeds or with temporal shifts to be matched effectively. This makes DTW-based clustering especially well suited for temporal data where sequences may be misaligned in time but share similar underlying shapes, such as speech signals, motion capture data, or sensor readings. In practice, DTW is commonly combined with partitioning algorithms like K-means or hierarchical clustering, although centroid computation requires adaptations such as DTW barycenter averaging (DBA). While DTW-based clustering provides strong robustness to temporal distortions, it can be computationally expensive for large datasets, prompting the use of constraints or approximations to improve scalability.

Deep Embedded Clustering (DEC)

Deep Embedded Clustering (DEC) is an unsupervised deep learning method introduced by Xie, Girshick, and Farhadi for learning a low-dimensional representation of data that is explicitly optimized for clustering rather than just reconstruction or compression. The core idea is that ordinary clustering methods such as k-means often struggle in high-dimensional spaces, while ordinary autoencoders may learn compact features that are good for reconstruction but not necessarily ideal for separating clusters. DEC addresses that gap by first learning an embedding space with a deep autoencoder and then refining that space so that points become increasingly organized around learnable cluster centers. In other words, DEC does not treat feature learning and clustering as separate stages; it couples them so the representation gradually becomes more “cluster-friendly.”

The algorithm begins with representation learning. A stacked autoencoder is trained on the unlabeled input data so that an encoder maps each input x_i into a latent vector z_i, and a decoder reconstructs the original input from that latent code. In the original DEC procedure, this pretraining step is important because it gives the model a meaningful latent space before clustering starts. After pretraining, the decoder is discarded and only the encoder is kept. The latent vectors produced by the encoder are then clustered, typically with k-means, to obtain an initial set of cluster centroids in latent space. Those centroids serve as DEC’s starting point for iterative refinement. This initialization matters because DEC is not trying to discover structure from a random latent space; it starts from an embedding that already captures much of the data’s underlying variation.

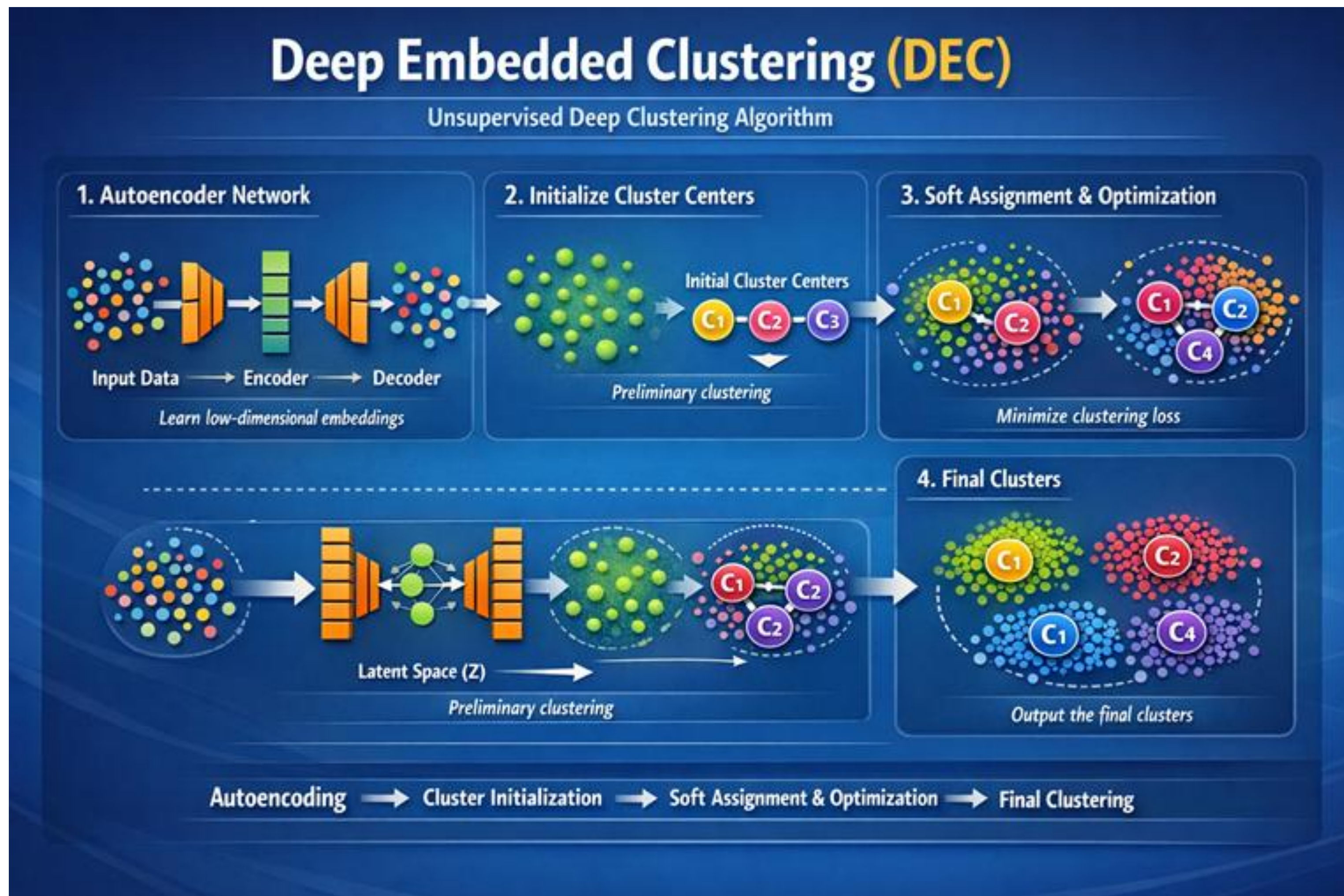
Deep Embedded Clustering (DEC)

- Once the initial centroids are available, DEC defines a soft assignment of each embedded point to each cluster. Instead of assigning each point hard membership immediately, it computes a probability-like quantity using a Student's t-distribution kernel. Intuitively, a point receives a higher assignment score for clusters whose centroids are closer in latent space, and lower scores for distant clusters. This soft formulation is useful because it provides smooth gradients and allows the network to express uncertainty, especially for points near cluster boundaries. The use of the Student's t-kernel also helps handle local structure in a way similar to methods such as t-SNE, making nearby points influential without letting very distant points dominate the assignment.
- DEC's most distinctive feature is its target distribution. Rather than optimizing directly against the current soft assignments, it constructs an auxiliary target distribution that sharpens confident assignments and rebalances cluster influence. In practice, DEC squares the current soft assignment values and normalizes them by cluster frequency, which has two effects: examples already assigned with high confidence become even more emphasized, and large clusters are prevented from dominating purely because they contain more points. This target distribution acts like a self-training signal. The model is effectively telling itself, "I am most confident that these samples belong to these clusters, so I will use those confident assignments to pull the representation and centroids into a clearer configuration."
- The optimization objective is the Kullback–Leibler (KL) divergence between the auxiliary target distribution and the model's current soft assignment distribution. Minimizing this loss updates both the encoder parameters and the cluster centroids. As optimization proceeds, points that are already near a centroid are pulled closer, ambiguous points are gradually nudged toward more coherent groups, and the latent space becomes progressively more separable. This iterative refinement continues until cluster assignments stabilize or another stopping criterion is met. Conceptually, DEC performs a feedback loop: embed the data, compute soft cluster assignments, build a sharpened target, update the embedding and centroids, and repeat.

Deep Embedded Clustering (DEC)

- A useful way to think about DEC is as a form of discriminative self-organization in latent space. The autoencoder pretraining gives the model a compressed representation of the data manifold, and the KL-based clustering phase then reshapes that manifold to make clusters tighter and better separated. Unlike a generative probabilistic clustering model, DEC is not primarily trying to model the full data distribution. Unlike plain k-means, it is not forced to cluster in the original feature space. And unlike a standard autoencoder, it is not satisfied with reconstruction quality alone. Its goal is specifically to learn an embedding in which cluster structure becomes clearer through iterative self-reinforcement.
- DEC has several strengths. It is elegant, relatively simple, and often performs much better than shallow clustering on images, text features, and other high-dimensional data because the learned embedding adapts to the clustering objective. It also does not require labels, which makes it attractive for exploratory analysis and representation learning on large unlabeled datasets. The original paper reported strong results on datasets such as MNIST, STL, and REUTERS compared with several classical baselines.
- At the same time, DEC has limitations that are important to understand. Its performance depends heavily on the quality of the initial autoencoder embedding and on the initial centroid placement. Because the later optimization phase in the original DEC discards the reconstruction objective, the latent space can sometimes drift in a way that sharpens clusters but damages global structure or harms robustness. DEC can also be sensitive to hyperparameters, imbalanced data, and challenging cluster shapes. These issues motivated later extensions such as IDEC, RDEC, and augmentation-based variants that retain reconstruction constraints or introduce regularization to make the clustering process more stable.

Deep Embedded Clustering (DEC)



Additional Resources

- Online textbook chapter on clustering (mathematical) https://www.cs.cmu.edu/afs/cs/academic/class/15456-s13/Handouts/Har-Peled-Chap-4.pdf

- HDBSCAN in scikit https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html

- Boston University 5 clustering algorithms data scientists need to know https://www.cs.bu.edu/fac/snyder/cs4100/Medium.Clustering.Survey.pdf

- Georgia Tech Evolution & Taxonomy of Clustering Algorithms https://sites.gatech.edu/omscs7641/2024/03/10/evolution-taxonomy-of-clustering-algorithms/

- Duke University Clustering Algorithms https://users.cs.duke.edu/~munagala/CPS230PROJ/