

Cheat Sheet: Building Unsupervised Learning Models

Unsupervised learning models

Model Name	Brief Description	Code Syntax
UMAP	<p>UMAP (Uniform Manifold Approximation and Projection) is used for dimensionality reduction.</p> <p>Pros: High performance, preserves global structure.</p> <p>Cons: Sensitive to parameters.</p> <p>Applications: Data visualization, feature extraction.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">n_neighbors: Controls the local neighborhood size (default = 15).min_dist: Controls the minimum distance between points in the embedded space (default = 0.1).n_components: The dimensionality of the embedding (default = 2).	<pre>from umap.umap_ import UMAP umap = UMAP(n_neighbors=15, min_dist=0.1, n_components=2)</pre>
t-SNE	<p>t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique.</p> <p>Pros: Good for visualizing high-dimensional data.</p> <p>Cons: Computationally expensive, prone to overfitting.</p> <p>Applications: Data visualization, anomaly detection.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">n_components: The number of dimensions for the output (default = 2).perplexity: Balances attention between local and global aspects of the data (default = 30).learning_rate: Controls the step size during optimization (default = 200).	<pre>from sklearn.manifold import TSNE tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)</pre>
PCA	<p>PCA (principal component analysis) is used for linear dimensionality reduction.</p> <p>Pros: Easy to interpret, reduces noise.</p> <p>Cons: Linear, may lose information in nonlinear data.</p> <p>Applications: Feature extraction, compression.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">n_components: Number of principal components to retain (default = 2).whiten: Whether to scale the components (default = False).svd_solver: The algorithm to compute the components (default = 'auto').	<pre>from sklearn.decomposition import PCA pca = PCA(n_components=2)</pre>
DBSCAN	<p>DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm.</p> <p>Pros: Identifies outliers, does not require the number of clusters.</p> <p>Cons: Difficult with varying density clusters.</p> <p>Applications: Anomaly detection, spatial data clustering.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">eps: The maximum distance between two points to be considered neighbors (default = 0.5).min_samples: Minimum number of samples in a neighborhood to form a cluster (default = 5).	<pre>from sklearn.cluster import DBSCAN dbscan = DBSCAN(eps=0.5, min_samples=5)</pre>
HDBSCAN	<p>HDBSCAN (Hierarchical DBSCAN) improves on DBSCAN by handling varying density clusters.</p> <p>Pros: Better handling of varying densities.</p> <p>Cons: Can be slower than DBSCAN.</p> <p>Applications: Large datasets, complex clustering problems.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">min_cluster_size: The minimum size of clusters (default = 5).min_samples: Minimum number of samples to form a cluster (default = 10).	<pre>import hdbscan clusterer = hdbscan.HDBSCAN(min_cluster_size=5)</pre>
K-Means clustering	<p>K-Means is a centroid-based clustering algorithm that groups data into k clusters.</p> <p>Pros: Efficient, simple to implement.</p> <p>Cons: Sensitive to initial cluster centroids.</p> <p>Applications: Customer segmentation, pattern recognition.</p> <p>Key hyperparameters:</p> <ul style="list-style-type: none">n_clusters: Number of clusters (default = 8).init: Method for initializing the centroids ('k-means++' or 'random', default = 'k-means++').n_init: Number of times the algorithm will run with different centroid seeds (default = 10).	<pre>from sklearn.cluster import KMeans kmeans = KMeans(n_clusters=3)</pre>

Associated fuctions used

Method	Brief Description	Code Syntax
make_blobs	Generates isotropic Gaussian blobs for clustering.	<pre>from sklearn.datasets import make_blobs X, y = make_blobs(n_samples=100, centers=2, random_state=42)</pre>
multivariate_normal	Generates samples from a multivariate normal distribution.	<pre>from numpy.random import multivariate_normal samples = multivariate_normal(mean=[0, 0], cov=[[1, 0], [0, 1]], size=100)</pre>
plotly.express.scatter_3d	Creates a 3D scatter plot using Plotly Express.	<pre>import plotly.express as px fig = px.scatter_3d(df, x='x', y='y', z='z') fig.show()</pre>
geopandas.GeoDataFrame	Creates a GeoDataFrame from a Pandas DataFrame.	<pre>import geopandas as gpd gdf = gpd.GeoDataFrame(df, geometry='geometry')</pre>
geopandas.to_crs	Transforms the coordinate reference system of a GeoDataFrame.	<pre>gdf = gdf.to_crs(epsg=3857)</pre>
contextily.add_basemap	Adds a basemap to a GeoDataFrame plot for context.	<pre>import contextily as ctx ax = gdf.plot(figsize=(10, 10)) ctx.add_basemap(ax)</pre>
pca.explained_variance_ratio_	Returns the proportion of variance explained by each principal component.	<pre>from sklearn.decomposition import PCA pca = PCA(n_components=2) pca.fit(X) variance_ratio = pca.explained_variance_ratio_</pre>