Cheat Sheet: Building Unsupervised Learning Models

Unsupervised learning models

Model Name	Brief Description	Code Syntax
UMAP	UMAP (Uniform Manifold Approximation and Projection) is used for dimensionality reduction. **Pros:** High performance, preserves global structure. **Cons:** Sensitive to parameters. **Applications:** Data visualization, feature extraction. **Key hyperparameters:** • **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).	`from umap.umap_ import UMAP` `umap = UMAP(n_neighbors=15, min_dist=0.1, n_components=2)`
t-SNE	t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique. **Pros:** Good for visualizing high-dimensional data. **Cons:** Computationally expensive, prone to overfitting. **Applications:** Data visualization, anomaly detection. **Key hyperparameters:** • **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).	`from sklearn.manifold import TSNE` `tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)`
PCA	PCA (principal component analysis) is used for linear dimensionality reduction. **Pros:** Easy to interpret, reduces noise. **Cons:** Linear, may lose information in nonlinear data. **Applications:** Feature extraction, compression. **Key hyperparameters:** • **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').	`from sklearn.decomposition import PCA` `pca = PCA(n_components=2)`
DBSCAN	DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. **Pros:** Identifies outliers, does not require the number of clusters. **Cons:** Difficult with varying density clusters. **Applications:** Anomaly detection, spatial data clustering. **Key hyperparameters:** • **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).	`from sklearn.cluster import DBSCAN` `dbscan = DBSCAN(eps=0.5, min_samples=5)`
HDBSCAN	HDBSCAN (Hierarchical DBSCAN) improves on DBSCAN by handling varying density clusters. **Pros:** Better handling of varying densities. **Cons:** Can be slower than DBSCAN. **Applications:** Large datasets, complex clustering problems. **Key hyperparameters:** • **min_cluster_size:** The minimum size of clusters (default = 5). • **min_samples:** Minimum number of samples to form a cluster (default = 10).	`import hdbscan` `clusterer = hdbscan.HDBSCAN(min_cluster_size=5)`

<table>
<tr><th>Model Name</th><th>Brief Description</th><th>Code Syntax</th></tr>
<tr><td>K-Means clustering</td><td>K-Means is a centroid-based clustering algorithm that groups data into k clusters.
Pros: Efficient, simple to implement.
Cons: Sensitive to initial cluster centroids.
Applications: Customer segmentation, pattern recognition.
Key hyperparameters:
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).</td><td><pre>from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)</pre></td></tr>
</table>

Associated fuctions used

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

Method	Brief Description	Code Syntax

pca.explained_variance_ratio_	Returns the proportion of variance explained by each principal component.	`from sklearn.decomposition import PCA`
		`pca = PCA(n_components=2)`
		`pca.fit(X)`
		`variance_ratio = pca.explained_variance_ratio_`

Author

[Jeff Grossman]
[Abhishek Gagneja]