[Clustering Algorithms] (CheatSheet)

Data Preparation

- **Load Dataset**: `data = pd.read_csv('data.csv')`
- **Standardize Features**: `scaled_data = StandardScaler().fit_transform(data)`
- **Principal Component Analysis (PCA) for Dimensionality Reduction**: `pca_data = PCA(n_components=2).fit_transform(scaled_data)`
- **Visualize Data Distribution**: `plt.scatter(data[:, 0], data[:, 1])`
- **Compute Pairwise Distance Matrix for Clustering**: `dist_matrix = pairwise_distances(data, metric='euclidean')`
- **Handle Missing Values**: `data_filled = SimpleImputer(strategy='mean').fit_transform(data)`
- **Create Dendrogram for Hierarchical Clustering**: `dendrogram = sch.dendrogram(sch.linkage(data, method='ward'))`
- **Identify Optimal Number of Clusters via Elbow Method**: `inertia = [KMeans(n_clusters=i).fit(data).inertia_ for i in range(1, 11)]; plt.plot(range(1, 11), inertia)`
- **Normalize Features for Clustering**: `normalized_data = normalize(data)`
- **Use TSNE for Visualization of High-dimensional Data**: `tsne_data = TSNE(n_components=2).fit_transform(data)`

K-means Clustering

- **Perform K-means Clustering**: `kmeans = KMeans(n_clusters=3).fit(data)`
- **Predict Cluster Labels**: `labels = kmeans.predict(data)`
- **Visualize Clusters**: `plt.scatter(data[:, 0], data[:, 1], c=labels)`
- **Determine Centroids**: `centroids = kmeans.cluster_centers_`
- **Silhouette Score for Model Evaluation**: `silhouette_score(data, labels)`
- **Initializing K-means with Smart Start (k-means++)**: `kmeans = KMeans(n_clusters=3, init='k-means++').fit(data)`
- **Mini-Batch K-means for Large Datasets**: `minibatch_kmeans = MiniBatchKMeans(n_clusters=3).fit(data)`
- **Find Optimal K Using the Silhouette Method**: `silhouette_optimal_k(data)`
- **Elbow Method Visualization for Optimal K**: `plot_elbow_method(data)`
- **Assign New Data Points to Existing Clusters**:`new_labels = kmeans.predict(new_data)`
- **Iterative Training to Refine Centroids**: `kmeans.fit(data); kmeans.partial_fit(more_data)`

- **Visualize Cluster Centers on 2D Plot**: `plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')`
- **Calculate Within-cluster Sum of Squares (WSS)**: `wss = kmeans.inertia_`
- **K-means Clustering with Specific Random State for Reproducibility**: `kmeans = KMeans(n_clusters=3, random_state=42).fit(data)`
- **Use K-means for Color Quantization in Images**: `quantized_img = quantize_colors(image_data, n_colors=8)`

Hierarchical Clustering

- **Perform Agglomerative Hierarchical Clustering**: `model = AgglomerativeClustering(n_clusters=3).fit(data)`
- **Extract Cluster Labels**: `labels = model.labels_`
- **Plot Dendrogram Function**: `plot_dendrogram(model, truncate_mode='level', p=3)`
- **Cophenetic Correlation Coefficient**: `c, coph_dists = cophenet(sch.linkage(data, 'ward'), pdist(data))`
- **Agglomerative Hierarchical Clustering with Different Linkage Criteria**: `model = AgglomerativeClustering(n_clusters=3, linkage='average').fit(data)`
- **Scikit-learn's Convenience Function for Ward's Method**: `ward = AgglomerativeClustering(n_clusters=3, linkage='ward').fit(data)`
- **Generate Dendrogram from Linkage Matrix**: `linkage_matrix = ward(children=model.children_); dendrogram(linkage_matrix)`
- **Cutting the Dendrogram to Form Clusters**: `labels = fcluster(linkage_matrix, t=3, criterion='maxclust')`
- **Creating Custom Distance Matrix for Agglomerative Clustering**: `model = AgglomerativeClustering(n_clusters=3, affinity='precomputed', linkage='complete').fit(custom_distance_matrix)`
- **Interactive Dendrogram Plotting with Plotly**: `plot_dendrogram_plotly(linkage_matrix)`
- **Evaluate Model Using Davies-Bouldin Index**: `db_index = davies_bouldin_score(data, model.labels_)`
- **Use SciPy for More Detailed Dendrogram Customization**: `dendrogram(sch.linkage(data, method='ward'), color_threshold=1)`
- **Dynamic Thresholding for Cluster Formation in Hierarchical Clustering**: `dynamic_labels = dynamic_threshold_clustering(linkage_matrix)`

DBSCAN

- **Apply DBSCAN Clustering**: `dbscan = DBSCAN(eps=0.3, min_samples=10).fit(data)`

- **Cluster Labels from DBSCAN**: `labels = dbscan.labels_`
- **Identify Core Samples**: `core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool); core_samples_mask[dbscan.core_sample_indices_] = True`
- **Visualize DBSCAN Clusters**: `plt.scatter(data[:, 0], data[:, 1], c=labels)`
- **Adjusting `eps` and `min_samples` for Density Variation**: `dbscan = DBSCAN(eps=0.5, min_samples=15).fit(data)`
- **Handling Noise Points Identified by DBSCAN**: `noise = data[dbscan.labels_ == -1]`
- **Cluster Core Samples Extraction and Visualization**: `core_samples = data[dbscan.core_sample_indices_]`
- **Evaluate Clustering with Silhouette Score**: `silhouette_score(data, dbscan.labels_)`
- **Visualize DBSCAN Clusters with Matplotlib**: `plot_dbscan_clusters(data, dbscan.labels_, dbscan.core_sample_indices_)`
- **Optimize DBSCAN Parameters with Grid Search**: `optimal_params = optimize_dbscan(data)`
- **Use DBSCAN for Anomaly Detection**: `anomalies = data[dbscan.labels_ == -1]`
- **Scaling Features for Improved DBSCAN Performance**: `scaled_data = StandardScaler().fit_transform(data); dbscan.fit(scaled_data)`
- **Calculate Number of Clusters in DBSCAN**: `n_clusters_ = len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0)`
- **Visualize Spatial Clusters with Folium for Geospatial Data**: `plot_folium_map_geospatial(data, dbscan.labels_)`

Advanced Clustering Algorithms

- **Spectral Clustering**: `spectral_labels = SpectralClustering(n_clusters=3).fit_predict(data)`
- **Mean Shift Clustering**: `meanShift = MeanShift().fit(data); labels = meanShift.labels_`
- **Affinity Propagation**: `affinity = AffinityPropagation().fit(data); cluster_centers_indices = affinity.cluster_centers_indices_`
- **Hierarchical Density-Based Spatial Clustering (HDBSCAN)**: `hdbscan_model = hdbscan.HDBSCAN(min_cluster_size=10).fit(data); labels = hdbscan_model.labels_`
- **Gaussian Mixture Models for Soft Clustering**: `gmm = GaussianMixture(n_components=3).fit(data); labels = gmm.predict(data)`
- **OPTICS Clustering**: `optics = OPTICS(min_samples=10, xi=.05, min_cluster_size=.05).fit(data); labels = optics.labels_`
- **Evaluate Clustering with Adjusted Rand Index**: `adjusted_rand_score(true_labels, predicted_labels)`

- **Clustering Based on Graph Connectivity (Agglomerative)**: `connectivity = kneighbors_graph(data, n_neighbors=10, include_self=False); ward = AgglomerativeClustering(connectivity=connectivity); labels = ward.fit_predict(data)`
- **BIRCH for Large Datasets**: `birch = BIRCH(n_clusters=3).fit(data); labels = birch.predict(data)`
- **CURE Clustering Implementation with PyCaret**: `from pycaret.clustering import *; cure_model = create_model('cure', num_clusters=3, data=data)`
- **Using Silhouette Plots to Evaluate Clustering Quality**: `plot_silhouette(data, labels)`
- **Cluster Validation Using the Davies-Bouldin Index**: `davies_bouldin_score(data, labels)`

Optimization Strategies

- **Grid Search for Optimal Parameters in K-means**: `param_grid = {'n_clusters': range(1, 11)}; grid_search = GridSearchCV(KMeans(), param_grid); grid_search.fit(data)`
- **Using the Gap Statistic to Determine the Number of Clusters**: `gap_statistic, opt_k = optimalK(data, nrefs=3, maxClusters=10)`
- **Auto-Scaling Features Based on Clustering Tendency**: `scaler = autoscale_based_on_clustering_tendency(data)`
- **Parallel Coordinate Plot for Cluster Visualization**: `pd.plotting.parallel_coordinates(data.assign(cluster=labels), 'cluster')`
- **Cluster Stability Evaluation via Bootstrapping**: `stability = bootstrap_stability(data, KMeans(n_clusters=3), n_bootstraps=10)`
- **Elbow Method with Inertia and Silhouette Analysis Combined**: `evaluate_clustering_elbow_silhouette(data, max_clusters=10)`

Specialized Clustering Applications

- **Temporal or Sequential Data Clustering (e.g., Time Series)**: `ts_cluster_labels = TimeSeriesKMeans(n_clusters=3).fit_predict(time_series_data)`
- **Clustering Geospatial Data**: `geo_cluster_labels = DBSCAN(eps=0.1, min_samples=5).fit_predict(geo_data[['latitude', 'longitude']])`
- **Image Segmentation Using Clustering**: `segmented_image = KMeans(n_clusters=3).fit_predict(image_pixels)`
- **Text Clustering for Document Categorization**: `text_cluster_labels = MiniBatchKMeans(n_clusters=5).fit_predict(tfidf_matrix)`

- **Clustering for Anomaly Detection**: `anomaly_labels = IsolationForest().fit_predict(data)`
- **Clustering in Bioinformatics (e.g., Gene Expression Data)**: `gene_cluster_labels = AgglomerativeClustering().fit_predict(gene_expression_data)`

Integrative Approaches and Advanced Techniques

- **Consensus Clustering for Stability and Robustness**: `consensus_labels = consensus_cluster(data, KMeans(), n_clusters_range=[2,10], bootstrap_samples=100)`
- **Feature Learning with Clustering (e.g., Autoencoders)**: `encoded_features = Autoencoder().fit_transform(data); ae_cluster_labels = KMeans(n_clusters=3).fit_predict(encoded_features)`
- **Cluster Ensembles for Improved Performance**: `ensemble_labels = ClusterEnsembles(hyperparameters, data)`
- **Use of Clustering for Dimension Reduction**: `reduced_data = clustering_based_dimension_reduction(data, n_clusters=10)`
- **Integrating Clustering with Classification for Semi-supervised Learning**: `semi_labels = semi_supervised_learning_with_clustering(data, partial_labels)`
- **Leveraging Graph-based Clustering for Complex Networks**: `graph_cluster_labels = SpectralClustering(n_clusters=3).fit_predict(adjacency_matrix)`
- **Multi-view Clustering for Integrating Different Types of Data**: `multi_view_labels = MultiViewClustering().fit_predict([view1_data, view2_data])`
- **Interactive Clustering for User-guided Analysis**: `interactive_clusters = interactive_clustering(data, initial_guesses)`
- **Utilizing Clustering for Data Cleaning and Preprocessing**: `clean_data = data_cleaning_with_clustering(data)`
- **Hierarchical Clustering for Large Datasets via BIRCH**: `large_scale_cluster_labels = BIRCH(threshold=0.5, n_clusters=None).fit_predict(large_data)`
- **Spatial Clustering for Location Data Optimization**: `location_clusters = OPTICS(min_samples=50, xi=0.05, min_cluster_size=0.1).fit_predict(location_data)`
- **Integrating Clustering with Reinforcement Learning for Dynamic Environments**: `dynamic_cluster_labels = reinforcement_learning_with_clustering(state_data)`

- **Clustering for Recommender Systems (User or Item Clustering)**: `recommender_clusters = KMeans(n_clusters=10).fit_predict(user_feature_matrix)`
- **Deep Clustering for Unsupervised Feature Learning**: `deep_cluster_labels = DeepClustering().fit_predict(data)`
- **Clustering Validation in Multi-dimensional Datasets**: `validation_scores = multidimensional_clustering_validation(data, cluster_labels)`