Unsupervised Learning Study Guide

Cluster Analysis

Clustering aims to group a set of data points into clusters such that points in the same cluster are more similar to each other than to those in different clusters. Key clustering methods include hierarchical clustering, k-means clustering, density-based clustering (DBSCAN), and Gaussian Mixture Models (GMM). We also discuss how to evaluate clusters and some mathematical foundations.

Hierarchical Clustering

Overview: Hierarchical clustering builds a hierarchy of clusters. In agglomerative hierarchical clustering (bottom-up approach), each data point starts in its own cluster, and pairs of clusters are merged stepwise until one overall cluster remains en.wikipedia.org. The result is typically visualized using a dendrogram, a tree-like diagram that shows the sequence of merges and the distances at which merges occur en.wikipedia.org. An example dendrogram is shown below, where individual points (a–f) are gradually merged into larger clusters and ultimately one cluster containing all points.

The height of each merge (y-axis) indicates the distance or dissimilarity at which the clusters were joined.

Linkage Criteria: A crucial choice in hierarchical clustering is the linkage method, which determines how the distance between clusters is computed en.wikipedia.org:

- **Single Linkage (Nearest Neighbor):** Distance between two clusters = the minimum distance between any member of one cluster and any member of the other. This method can form "chains" of points (long, thin clusters) en.wikipedia.org. Mathematically, for clusters C_i and C_j,

$$d_{\text{single}}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|.$$

- **Complete Linkage (Farthest Neighbor):** Distance = the maximum distance between any member of one cluster and any member of the other statsandr.com. This tends to produce more compact, spherical clusters.

$$d_{\text{complete}}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|.$$

- **Average Linkage:** Distance = the average pairwise distance between all points in one cluster and all points in the other statsandr.com. This is a compromise between single and complete linkage:

$$d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \|x - y\|.$$

- **Centroid Linkage:** Distance = distance between the centroids (mean vectors) of the two clusters statsandr.com.
- **Ward's Method (Minimum Variance):** At each step, merge the two clusters that result in the smallest increase in total within-cluster variance (i.e. minimize the error sum of squares). This is equivalent to merging clusters with the smallest increase in SSE (sum of squared errors) and often yields clusters of relatively equal size statsandr.com.

Distance Metrics: The underlying distance $|x - y|$ is often Euclidean distance, but other metrics (Manhattan, Cosine, correlation distance, etc.) can be used depending on the data characteristics. For example, clustering gene expression data might use correlation as a distance metric.

Cluster Stopping & Interpretation: The dendrogram can be "cut" at a chosen distance threshold to produce a desired number of clusters. The choice of number of clusters is somewhat subjective; one can look for a large jump in fusion distance in the dendrogram (indicating a natural division) or use metrics like silhouette score. Cluster R-squared (the proportion of variance explained by the clustering) increases as more clusters are formed – it is 0 when each point is its own cluster and 1 when all points are in one cluster. In practice, one may choose the smallest number of clusters that achieves a high R-squared or that is interpretable for the domain.

R Implementation (Hierarchical Clustering): In R, hierarchical clustering can be performed with the `hclust()` function on a distance matrix. For example, using the built-in iris dataset (excluding the species label):

```
# Compute distance matrix (Euclidean distance by default)
dist_mat <- dist(iris[, -5])

# Perform hierarchical clustering (e.g., using average linkage)
hc <- hclust(dist_mat, method = "average")

# Plot the dendrogram
plot(hc, labels = FALSE, hang = -1, main = "Iris Data Hierarchical Clustering (Average Linka
```

After obtaining the `hclust` object, you can cut the dendrogram to get cluster memberships. For example, `cutree(hc, k=3)` will cut the tree into 3 clusters and return a cluster index (1,2,3) for each observation. You can then compare these with the actual species in the iris dataset to see how well clustering performed.

K-Means Clustering

Overview: K-means clustering partitions the data into K clusters. It is an iterative centroid-based clustering algorithm: it seeks to find cluster centers (centroids) such that the total within-cluster variance (or equivalently, sum of squared distances of points to their nearest centroid) is minimized. It works as follows:

- **Initialization:** Choose K initial centroids (often randomly selected data points or random positions).
- **Assignment step:** Assign each data point to the nearest centroid (using Euclidean distance or other distance metric). This forms K clusters.
- **Update step:** Recompute each centroid as the mean of all data points assigned to that cluster.
- Repeat the assignment and update steps until convergence (cluster assignments no longer change or centroids move negligibly). This usually happens in a small number of iterations.

The objective function k-means tries to minimize is the within-cluster Sum of Squared Errors (SSE):

$$\mathrm{SSE} = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - \mu_k\|^2,$$

where μ_k is the centroid of cluster C_k. Equivalently, k-means tries to maximize the between-cluster variance or the cluster separation for a given K.

Properties and Considerations:

- K-means is fast and works well for compact, spherical clusters that are roughly of similar size.
- It requires specifying K in advance. To choose K, methods like the elbow method (plotting SSE or within-cluster dispersion vs. K and looking for an "elbow" where improvement tapers off) or the silhouette score can be used.
- The algorithm can converge to a local minimum of the SSE, and results can depend on initialization. It's good practice to run k-means multiple times with different random starts (nstart in R) and take the solution with lowest SSE scikit-learn.org.
- K-means assumes continuous variables (using means). For categorical data, other clustering methods (or using modes/medoids) are more appropriate.

R Implementation (K-Means): R's built-in `kmeans()` function can be used. For example, clustering the iris dataset into 3 clusters (to compare with 3 species):

```
set.seed(1)
km_res <- kmeans(iris[, -5], centers = 3, nstart = 20)
print(km_res$centers)        # output the centroids
print(km_res$size)           # number of points in each cluster
table(km_res$cluster, iris$Species)  # compare clusters to actual species
```

Here, `centers=3` requests 3 clusters, and `nstart=20` will run 20 random initializations and choose the best. The output includes cluster assignments (`km_res$cluster`), the cluster centroids (`$centers`), and sizes. We can see how the clusters correspond to species via the contingency table. Often k-means does well at separating distinct groups (e.g., it perfectly separates Iris setosa in many runs).

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Overview: DBSCAN is a density-based clustering algorithm that can discover clusters of arbitrary shape and identify outliers (noise points) en.wikipedia.org. It groups points that are closely packed together (high density regions), and points in low-density regions that do not belong to any cluster are marked as noise en.wikipedia.org. The algorithm has two main parameters:

- ϵ (epsilon): The radius of the neighborhood around a point (i.e., the maximum distance for two points to be considered neighbors).
- MinPts: The minimum number of points required in an ϵ-neighborhood for that point to be considered a core point.

Key concepts:

- A point is a core point if it has at least MinPts points (including itself) within distance ϵ en.wikipedia.org.
- A point q is directly reachable from a core point p if q is within ϵ of p en.wikipedia.org.
- A point q is reachable from p if there is a chain of core points $p = p_1 \to p_2 \to \cdots \to p_n = q$ where each is directly reachable from the previous. This means q can lie in the density-connected region of p en.wikipedia.org.
- Any point that is not reachable from any other point (i.e., not in a dense region) is labeled as noise (an outlier) en.wikipedia.org.

Clusters are formed as sets of points that are mutually reachable. In essence, DBSCAN finds connected regions of density exceeding a threshold. An illustrative diagram is shown below: red points are core points (each has at least MinPts=4 points in its radius ϵ, indicated by red circles), yellow points are border points (non-core but within ϵ of a core point), and the blue point is an outlier (insufficient neighbors within ϵ) en.wikipedia.org.

Choosing Parameters: A common approach is to use a k-distance plot (plot the distance to the k-th nearest neighbor for each point, sorted descending) to help choose ϵ. A sharp bend ("elbow") in this plot can suggest a good ϵ value. MinPts is often set to something like the dimensionality of the data plus 1 (e.g. MinPts = 4 for 2D, as in the example above) or higher for larger, noisier datasets.

Strengths: DBSCAN can find arbitrarily shaped clusters (e.g., curved shapes) and can automatically determine the number of clusters. It is also robust to outliers (explicitly labels them).

Limitations: For datasets with varying density, a single ϵ/MinPts may not work well (clusters of different density might get merged or some cluster might be split). Also, performance degrades on very large datasets unless using indexing (DBSCAN is $O(n^2)$ in the worst case, though in practice spatial indexing can improve this).

R Implementation (DBSCAN): R provides DBSCAN in package dbscan or fpc. For example, using dbscan package:

```
library(dbscan)
# Assume data matrix 'X'
db <- dbscan(X, eps = 0.5, minPts = 5)
db$cluster          # cluster assignments (0 means noise)
sum(db$cluster == 0)  # number of noise points
```

After running, `db$cluster` gives a vector of cluster IDs for each point (with 0 for noise). We can adjust `eps` and `minPts` to tune the result. Plotting the result (e.g., if 2D data) can show the clusters and outliers.

Gaussian Mixture Models (GMM)

Overview: A Gaussian Mixture Model is a probabilistic model that assumes the data are generated from a mixture of a finite number of Gaussian distributions (components) with unknown parameters scikit-learn.org. In clustering terms, each Gaussian component corresponds to a cluster. Unlike k-means which yields hard assignments of points to clusters, GMMs provide soft clustering: each point has a probability (or responsibility) for belonging to each cluster.

Model Definition: A GMM with K components in d-dimensional space is defined by parameters $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ where:

- π_k is the mixing proportion for component k (prior probability of cluster k), with $\sum_{k=1}^K \pi_k = 1$.

- μ_k is the d-dimensional mean vector of Gaussian k (cluster centroid).
- Σ_k is the $d \times d$ covariance matrix of Gaussian k (shape/volume of the cluster).

The probability density function of the mixture is:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k),$$

a weighted sum of Gaussian densities. The model can capture clusters that are elliptical (via Σ_k) and of different sizes and orientations, unlike k-means which assumes spherical clusters of equal size scikit-learn.org. In fact, k-means is a special case of GMM when each cluster has an identity covariance (spherical) and clusters are equal-size and well-separated scikit-learn.org.

Learning/Inference: The typical algorithm to fit a GMM to data is Expectation-Maximization (EM):

- **E-step:** Given current parameters, compute the responsibility r_{ik} of component k for each data point x_i, which is essentially the posterior probability that x_i belongs to cluster k:

  $$r_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}.$$

- **M-step:** Update the parameters using these responsibilities:

  $$N_k = \sum_{i=1}^{n} r_{ik}, \quad \pi_k = \frac{N_k}{n}, \quad \mu_k = \frac{1}{N_k} \sum_{i=1}^{n} r_{ik} x_i, \quad \Sigma_k = \frac{1}{N_k} \sum_{i=1}^{n} r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T.$$

  These updates maximize the expected log-likelihood of the data given the assignments. Then return to E-step and repeat until convergence. EM is guaranteed to find a local maximum of the likelihood.

Choosing K: GMMs can use likelihood-based model selection criteria to choose the number of components. Commonly used are BIC (Bayesian Information Criterion) or AIC (Akaike Information Criterion) – they add a penalty for model complexity (number of parameters) to the log-likelihood. The model with the lowest BIC (or AIC) is often selected as having the appropriate number of clusters.

Soft Clustering Output: After training, each data point x_i can be assigned to the cluster with highest responsibility $\arg\max_k r_{ik}$ (hard clustering), or we can retain the probabilities r_{ik} as soft assignments. The mixing proportions π_k indicate the size of clusters (the expected proportion of points in each cluster).

R Implementation (GMM): The mclust package provides convenient functions for Gaussian mixture modeling. For example:

```
library(mclust)
mc <- Mclust(iris[, -5])       # let mclust choose optimal K via BIC
summary(mc)                    # summary of the model found
mc$G                           # selected number of clusters
head(mc$classification)        # hard cluster labels for each observation
```

Mclust will try models with different K and covariance structures and select one by BIC. The summary(`mc`) prints the chosen model and BIC values. We could also specify a fixed number of clusters using `Mclust(data, G=3)` for 3 clusters, etc. The output `mc$classification` gives the cluster assignment, and `mc$z` gives the matrix of probabilities (each row sums to 1, with entries being r_{ik} for each cluster k).

Comparison with k-means: GMM clustering is often more flexible since it can model clusters with covariance (correlations) and different sizes scikit-learn.org. K-means is a special case where each cluster's covariance is $\sigma^2 I$ and identical for all clusters – in such a case, maximizing likelihood under those assumptions leads to assigning points to the nearest centroid (which is k-means). In scenarios where clusters are not spherical or have different point densities, GMM usually performs better by appropriately modeling the cluster shapes.

Cluster Evaluation and Key Concepts

- **Inertia / Within-Cluster SSE:** Sum of squared distances of points to their cluster centroids (used by k-means). Lower values indicate tighter clusters, but always decreases with more clusters.
- **Between-Cluster SSE:** Sum of squared distances of cluster centroids to the global data mean, weighted by cluster size. The ratio of between-SS to total SS is sometimes called R^2 of the clustering (how much variance is explained by the clustering).
- **Silhouette Coefficient:** For each point, $s = (b - a)/\max(a, b)$, where a is the average distance to points in the same cluster, and b is the average distance to points in the nearest other cluster. s ranges from -1 to 1 (higher is better, indicating well-separated clusters). The average silhouette over all points can guide the choice of K.
- **Dunn Index, Davies-Bouldin Index:** Other indices to evaluate clustering quality based on inter-cluster and intra-cluster distances.
- **Elbow Method:** Plotting the clustering score (e.g., SSE or inertia) vs. number of clusters and looking for an "elbow" where improvements diminish.
- **Purity / External criteria:** If ground truth class labels are known (in a benchmark dataset), clustering can be evaluated by how well clusters align with true classes (using measures like adjusted Rand index, mutual information, etc.).

Dimensionality Reduction

Dimensionality reduction techniques aim to reduce the number of random variables under consideration – obtaining a more compact representation of data while preserving as much important structure as possible. We cover Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), and Independent Component Analysis (ICA). Key linear algebra concepts here are eigenvalues and eigenvectors of matrices, which underlie PCA and ICA.

Principal Component Analysis (PCA)

Goal: PCA finds a new set of orthogonal axes (principal components) that maximizes the variance of the projected data. Equivalently, PCA seeks the directions in feature space along which the data have the most spread. By projecting data onto the first few principal components, we reduce dimensionality while retaining most of the variance (information).

Procedure: Given a dataset with p variables:

- Standardize the data (subtract mean of each variable, and often scale to unit variance if variables have different units).
- Compute the covariance matrix of the data (or the correlation matrix if using standardized data).
- Eigen-decomposition: Compute eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$ and eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p$ of the covariance matrix. The eigenvectors (unit length) give the directions of principal components (PCs), and eigenvalues give the variance explained by each PC sebastianraschka.com.
- The ith principal component is the linear combination of original variables: $\mathrm{PC}_i = \mathbf{v}_i^\top X$ (where X is the data vector). The data can be projected onto the first k eigenvectors to obtain a k-dimensional representation with maximal variance retention.

Variance Explained: The proportion of total variance explained by the ith PC is $\lambda_i / \sum_{j=1}^{p} \lambda_j$. Often, one chooses k such that this cumulative variance (sum of first k eigenvalues divided by total sum) exceeds some threshold (like 80

Loadings and Interpretation: The loading of original variable j on principal component i is the element v_{ji} of the eigenvector $\mathbf{v}_i$. The loading indicates how much the original variable contributes to that principal component. For standardized data, loadings are also the correlations between original variables and the principal component graphpad.com. In fact, the relationship between eigenvectors and loadings is:

$$\mathrm{Loading}_{ji} = v_{ji}\sqrt{\lambda_i},$$

so loadings = eigenvectors * sqrt(eigenvalue) graphpad.com. Eigenvectors have unit length, whereas loadings have magnitude equal to the sqrt of eigenvalue; loadings thus directly show correlation with the PC graphpad.com. A variable with a high loading (either positive or negative) on PC1, for example, strongly influences that principal component's meaning. Typically we examine the loading matrix to interpret PCs (e.g., "PC1 represents an overall size factor", etc., depending on which original variables have large loadings).

Example: If we perform PCA on a dataset of students' exam scores in different subjects, PC1 might load positively on all subjects (indicating an overall proficiency), while PC2 might load positively on science and math but negatively on literature and art, indicating a contrast between quantitative vs verbal skills. We interpret the PCs based on such loading patterns.

Mathematical Note: PCA solves the optimization: find an axis $\mathbf{v}_1$ that maximizes $\mathrm{Var}(\mathbf{v}_1^\top X)$ subject to $|\mathbf{v}_1| = 1$. The solution is $\mathbf{v}_1$ = eigenvector corresponding to largest eigenvalue of covariance matrix. Subsequent components $\mathbf{v}_2, \ldots$ maximize variance under the constraint of being orthogonal to prior components (hence the eigenvectors form an orthonormal basis). The data transformation can be written in matrix form: if V is the matrix of top k eigenvectors, then $Z = XV$ gives the new k-dimensional coordinates (principal component scores). PCA can also be viewed through the Singular Value Decomposition (SVD) of the data matrix.

R Implementation (PCA): Use `prcomp()` or `princomp()`. For example:

```
pca_res <- prcomp(iris[, -5], scale. = TRUE)
summary(pca_res)            # shows standard deviation (sqrt(eigenvalue) of PCs and varianc
head(pca_res$x)             # PCA scores (coordinates in PC space)
pca_res$rotation[, 1:2]     # loadings for PC1 and PC2
```

Here we scaled the iris data. The output `pca_res$rotation` matrix contains the loadings (eigenvectors) for each principal component. `pca_res$x` contains the transformed data (scores on PCs). We can plot the first two principal components with species labels to see how well the species separate in PCA space. Usually PC1 and PC2 of iris will show Setosa completely separated and some overlap between Versicolor and Virginica.

Eigenvalues & Eigenvectors: In PCA context – eigenvectors of covariance matrix = directions of PCs; eigenvalues = variances explained sebastianraschka.com. Note that if PCA is done on the correlation matrix (i.e., on standardized variables), the eigenvalues are the variances explained on the standardized scale (and their sum equals p). If done on covariance matrix of raw variables, eigenvalues sum to total variance of the data (which is $\sum \mathrm{Var}(X_j)$). Standard practice is to standardize first if variables are on different scales.

Interpretation Tip: A common heuristic is Kaiser's criterion – keep components with eigenvalue ¿ 1 (for standardized data), and Joliffe's criterion – keep eigenvalue ¿ 0.7. But these are guidelines; one should also consider interpretability and cross-validation.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Goal: t-SNE is a nonlinear dimensionality reduction technique, particularly well-suited for embedding high-dimensional data into 2D or 3D for visualization while preserving local neighborhood structure. It is often used to visualize complex data (like high-dimensional gene expression data, images, etc.) into 2D points such that similar objects are likely to appear close together on the plot.

How it Works: t-SNE does not preserve large pairwise distances or global structure; instead, it focuses on local neighborhoods:

- In high-dimensional space, for each point, t-SNE defines a probability distribution over neighbors (usually a Gaussian centered at the point; the perplexity parameter controls the effective number of neighbors).
- In the low-dimensional embedding (typically 2D), it defines a similar probability distribution for distances between points (using a Student t-distribution kernel).
- It then minimizes the Kullback–Leibler (KL) divergence between the high-dim neighbor distribution and the low-dim distribution en.wikipedia.org. In other words, if points i and j were close in high-D, t-SNE tries to make them close in low-D (high probability in both distributions), and if they were far apart in high-D, they may be anywhere in low-D except that t-SNE's cost function doesn't strongly force far points to be far – this allows t-SNE to distort large distances to accommodate local clustering.

The optimization is typically done via gradient descent. The result is that clusters in the original data often become distinct groups of points in the t-SNE plot, which is great for visualization. However, distances between clusters in the t-SNE plot are not necessarily meaningful (the space is distorted).

Parameters: Perplexity is a key parameter (typically between 5 and 50) that roughly indicates the number of effective nearest neighbors. A smaller perplexity will focus on very local structure, a larger one will include slightly more global structure. Another parameter is the learning rate. Tuning these can change the output. Also, t-SNE is randomized (starts with a random initial 2D layout), so each run can yield a slightly different plot; running with different seeds is advised to ensure stability of findings.

Pros and Cons:

- t-SNE can reveal subtle clusters and subclusters that PCA might not (especially if clusters are non-linear separations).
- It is computationally intensive for large datasets (though various approximations exist).
- It should mainly be used for visualization, not for downstream computations (because the axes in t-SNE plot are abstract and distances are not directly interpretable beyond local neighborhood).

- t-SNE may sometimes produce "cluster artifacts" (it will almost always show some clusters even if data is uniformly random, because it emphasizes any slight differences in density).

R Implementation (t-SNE): The Rtsne package can be used. Example:

```
library(Rtsne)
tsne_out <- Rtsne(iris[, -5], perplexity = 30, theta = 0.5)  # theta for speed-accuracy trac
plot(tsne_out$Y[,1], tsne_out$Y[,2], col=iris$Species, pch=19,
     main="t-SNE of Iris Data")
```

This will produce a 2D embedding (`tsne_out$Y` is an $n \times 2$ matrix of coordinates). Typically, iris t-SNE (with appropriate perplexity) will show three distinct blobs corresponding to species (Setosa well apart, Versicolor and Virginica closer but still separable).

Comparison with PCA: PCA is linear and focuses on maximizing variance, which is a global property. t-SNE is nonlinear and focuses on local neighbor relationships, which can capture complex manifold structure. For instance, PCA on the MNIST digit dataset may require 50 components to get a decent separation, while t-SNE on MNIST can produce a 2D plot where different digit clusters are separated.

Independent Component Analysis (ICA)

Goal: ICA is a technique to separate a multivariate signal into additive subcomponents that are maximally statistically independent. A classic example is the "cocktail party problem": given several mixed audio signals from microphones, ICA can separate the voices of individual speakers. Unlike PCA, which looks for uncorrelated factors (a second-order statistic), ICA looks for components that are not only uncorrelated but also as independent as possible (a higher-order statistic).

Model: ICA assumes the data X is a linear combination of independent source signals S. We can write $X = A, S$ where A is an unknown mixing matrix, and S are latent source components to be recovered. ICA attempts to find an unmixing matrix W such that $S = W, X$ yields components S that are statistically independent (usually measured by looking at non-Gaussianity; by Central Limit Theorem, a mixture of independent signals tends to be more Gaussian than the originals, so ICA algorithms often maximize non-Gaussianity of the outputs).

For two variables, a quick intuition: PCA would find orthogonal directions of variance (which are independent only if the data is Gaussian). ICA might find directions such that the distributions of projections are as non-Gaussian (multimodal or heavy-tailed etc.) as possible, indicating they are likely "pure" independent sources.

ICA Algorithms: Common algorithms for ICA include FastICA which uses an iterative method maximizing negentropy or kurtosis. It often involves:

- Centering and whitening the data first (similar to PCA preprocessing).
- Then iteratively finding weight vectors that maximize a contrast function (like kurtosis or log cosh of projected data) that measures non-Gaussianity.

Ambiguities: ICA can determine independent components up to permutation and scaling. That is, the order of output components is not significant (since independence has no inherent order) and each independent component can only be determined up to a multiplicative constant (and sign), because if S is independent, so is $-S$ or $2S$ with half the mixing weight.

When to use: ICA is often used in signal processing (e.g., EEG signal separation, artifact removal), text topic modeling, or any situation we suspect observed data are mixtures of underlying factors. It is not as commonly used just for dimensionality reduction aimed at visualization (since the independent components might not be ordered by variance explained; PCA is usually used first to reduce dimensionality, then ICA can be applied on a subspace).

R Implementation (ICA): Use the fastICA package. For example, if we have a matrix X:

```
library(fastICA)
ica_res <- fastICA(X, n.comp = 3)  # find 3 independent components
head(ica_res$S)    # estimated independent source signals
ica_res$A          # estimated mixing matrix
```

The result gives S (the matrix of independent components, of dimension $n \times 3$) and A (the estimated mixing matrix such that $X \approx A\%*\%S$). We might inspect the distribution of each column in S to see if they look non-Gaussian (which they ideally should, more so than original X).

Relation to PCA: PCA finds uncorrelated components ranked by variance; those components might still be correlated in higher moments or might mix multiple underlying factors. ICA goes further by using higher-order statistics to attempt to split truly independent factors. In practice, if data truly comes from independent sources linearly mixed, ICA can recover those (where PCA would just give uncorrelated combos).

Example: Suppose we have two sensors recording two signals, each sensor capturing a different linear mix of two original signals (like two sinusoidal tones). PCA on the two sensor recordings will give two uncorrelated components, but those may each still contain parts of both tones. ICA could separate it into the two original pure tones (assuming they are independent).

Note on whitening: PCA is often a preprocessing step for ICA (the data is "whitened" so that it has identity covariance). After whitening, ICA just needs to find an orthonormal rotation that makes data independent (since any further scaling is not needed after whitening).

Eigenvalues & Eigenvectors in Context

- **In PCA:** eigenvectors of the covariance (or correlation) matrix are the principal component directions; eigenvalues are variances along those directions. Sum of eigenvalues = total variance. Large eigenvalue indicates an important component. Small (or zero) eigenvalue indicates redundant or linearly dependent features (e.g., if eigenvalue = 0, data lies in a lower-dimensional hyperplane).

- **In MDS (Multidimensional Scaling) or LLE (Locally Linear Embedding) or other techniques, eigen-decompositions also appear (e.g., classical MDS involves eigen-decomposition of a similarity matrix).**
- For clustering, the notion of eigenvalues appears in spectral clustering (which we didn't cover in detail here, but that involves eigenvectors of a graph Laplacian to cluster points).

Association Rule Learning

Association rule learning is used to discover interesting relationships (association rules) in large datasets, most famously in market basket analysis (e.g., "Customers who bought X also bought Y"). Unlike clustering or PCA, this is not about all variables at once but rather about finding specific patterns of co-occurrence.

Key Definitions

- **Itemset:** A collection of one or more items. E.g., $\{milk, bread, butter\}$ could be an itemset in market basket data.
- **Support:** The support of an itemset is the proportion (or number) of transactions in the dataset that contain that itemset. For rule $X \Rightarrow Y$, support($X \cup Y$) is often used en.wikipedia.org. If 100 out of 1000 transactions contain both milk and bread, the support of $\{milk, bread\}$ is 0.1 (10
- **Confidence:** For a rule $X \Rightarrow Y$, confidence = $\frac{\text{support}(X \cup Y)}{\text{support}(X)} = P(Y|X)$ en.wikipedia.org. It is the proportion of transactions containing X that also contain Y en.wikipedia.org. For example, if 5
- **Lift:** Lift is the ratio of the observed support of X and Y together to what would be expected if X and Y were independent en.wikipedia.org. lift(XY)=P(X and Y)P(X)P(Y)=confidence(XY)support(Y).lift(XY)=P(X)P(Y)P(X and Y)=support(Y)confidence(XY). A lift ¿ 1 indicates X and Y appear together more often than expected by chance (positive association) en.wikipedia.org. Lift = 1 means independence, and ¡1 means co-occurrence is less than chance (negative association) en.wikipedia.org. For instance, if confidence($X \Rightarrow Y$) = 0.2 and support(Y) = 0.1, then lift = 0.2/0.1 = 2: Y is twice as likely to be bought when X is bought, compared to Y overall frequency.

- **Antecedent (LHS) and Consequent (RHS):** In a rule $X \Rightarrow Y$, X is the antecedent itemset, and Y is the consequent itemset. Typically X and Y are disjoint itemsets. The rule suggests when X occurs, Y tends to occur.

Example: In a grocery dataset, a discovered rule might be bread, peanut butter $\Rightarrow$ jam with support 2

Apriori Algorithm

Apriori is a classic algorithm for finding frequent itemsets (the first step to finding association rules) using a bottom-up breadth-first search and the Apriori property (downward closure): if an itemset is frequent, all of its subsets are also frequent. Conversely, if an itemset is infrequent, all supersets are infrequent en.wikipedia.org.

Steps:

- Find frequent itemsets:
  - Start with all 1-itemsets and find those that meet the minimum support threshold.
  - Generate candidate 2-itemsets from the frequent 1-itemsets (typically by self-joining the set of frequent 1-itemsets). Count their supports by scanning the database, and prune those below min support.
  - Then generate candidate 3-itemsets from frequent 2-itemsets (joining, and pruning any whose subsets are not all frequent – this is Apriori property).
  - Continue $k = 1, 2, 3, \ldots$ until no more candidates can be generated or no candidates are frequent.
- Generate rules from frequent itemsets: For each frequent itemset F, consider all non-empty proper subsets $A \subset F$. For each such A, let $B = F \setminus A$. If rule $A \Rightarrow B$ meets minimum confidence, output the rule. Confidence is computed as support(F)/support(A).

Apriori is efficient because it drastically cuts down the search space using the principle that infrequent itemsets cannot be extended. However, generating candidates can still explode combinatorially for low support thresholds on large item universes.

Complexity considerations: The number of possible itemsets is $2^N - 1$ for N items, which is enormous even for moderate N. Apriori mitigates this with pruning. Database scanning and candidate counting are also expensive; many optimizations (transaction reduction, hashing, etc.) exist.

FP-Growth Algorithm

FP-Growth is an alternative to Apriori that avoids explicit candidate generation. It compresses the database into a special data structure called an FP-tree (a compact prefix tree of transactions) and then extracts frequent itemsets via recursive mining of this tree.

FP-Tree: It's built by:

- Scanning the database once to find all frequent 1-itemsets and their counts.
- Use those frequent items (in descending order of frequency) to sort items in each transaction and insert into a tree. Common prefixes of transactions are shared in the tree, and each node in the tree has a count (number of transactions sharing that prefix).
- A header table links all nodes of the same item together (to traverse easily).

Mining: Starting from each frequent item (especially the least frequent, using a bottom-up approach), extract its conditional pattern base (paths in the tree leading to that item) and construct a conditional FP-tree for those paths. Recursively find frequent itemsets in these conditional trees. Essentially, it performs a divide-and-conquer: mine smaller conditional databases.

FP-growth is often much faster than Apriori for large datasets because it uses a compressed representation and doesn't generate many candidates upfront. It can handle lower support thresholds more gracefully.

Association Rule Generation and Evaluation

Once frequent itemsets are found (via Apriori, FP-growth, or other methods like Eclat), generating rules is straightforward as described: for each frequent itemset F and each non-empty $A \subset F$, output $A \Rightarrow (F \setminus A)$ if confidence $\geq$ minConf. Often we then rank rules by lift or confidence or other interest measures.

However, one must be careful of redundant rules (e.g., if $A \Rightarrow C$ and $AB \Rightarrow C$ have the same support and confidence, typically the simpler rule is preferred). There are notions of closed itemsets and maximal itemsets that reduce redundancy in frequent itemset mining:

- A frequent itemset is closed if none of its immediate supersets have the same support. Closed itemsets uniquely determine all association rules with that itemset as antecedent or consequent.
- Maximal frequent itemsets are those with no frequent supersets. They are useful as a compact representation but do not directly give rule confidences.

Example Output Rule: Rule: Diapers Beer

- support = 4

- confidence = 40
- lift = 1.33 (beer is 1.33 times more likely given diapers than overall).

Interpretation: "Customers buying diapers are relatively likely to also buy beer." (This is the famous — possibly apocryphal — diaper-beer association story.)

R Implementation (Association Rules): The arules package is widely used. For example, using the built-in Groceries dataset of arules:

```
library(arules)
data("Groceries")
rules <- apriori(Groceries, parameter=list(supp=0.001, conf=0.5))
summary(rules)
inspect(head(sort(rules, by="lift"), 5))   # inspect top 5 rules by lift
```

This will output rules found with support 0.1

Often, one will find many rules, so post-processing by sorting or filtering by lift or using more stringent thresholds is necessary to find truly interesting patterns. Visualization packages (like arulesViz) can plot rules, and grouping by items etc., to help interpret results.

Confidence vs Lift Pitfall: High confidence does not always mean a useful rule. For example, if an item Y is very common, any rule $X \Rightarrow Y$ can have high confidence even if X has little to do with Y. That's why lift is important – it accounts for the baseline frequency of Y. A rule with confidence 80

Example from Groceries: A common result is a rule like whole milk bread with confidence perhaps 30

Anomaly Detection

Anomaly detection (or outlier detection) is the identification of rare items, events or observations which differ significantly from the majority of the data. We will discuss Isolation Forest, One-Class SVM, and make a note on Extended Isolation Forest. Each method assumes the majority of data points are normal, and anomalies are relatively few and different.

Isolation Forest

Concept: The Isolation Forest is an ensemble method that isolates anomalies instead of profiling normal behavior mathworks.com. It is built on the premise that anomalies are "few and different" – meaning they are easier to isolate than normal points. Instead of using distance or density measures, Isolation Forest uses a collection of random binary trees (isolation trees) to split the data.

How it works:

- For each tree, it recursively partitions the data by randomly choosing a feature and a random split value between the min and max of that feature for the subset. This continues until each point is isolated (or some tree height limit is reached).
- Anomalies, being different, tend to get isolated sooner (i.e., at shallower tree depth) because a random split is more likely to separate an outlier from the rest of the data quickly mathworks.com. Normal points, which are in dense clusters, require more splits to be isolated.
- Many such trees (ensemble) are built on random subsets of data (and possibly using subsampling without replacement, typically subsample size of 256 or so is recommended).

Anomaly Score: For a given point, the path length $h(x)$ is the number of splits needed to isolate it in a tree. The shorter the average path length across the forest, the more anomalous the point. The Isolation Forest computes an anomaly score based on the normalized path length mathworks.com:

$$s(x) = 2^{-\frac{E[h(x)]}{c(n)}},$$

where $E[h(x)]$ is the average path length for point x across the trees, and $c(n)$ is a normalization factor (the average path length of unsuccessful search in a binary tree of n samples) stats.stackexchange.com. The score is calibrated so that it's roughly in [0,1]. Points with score 1 are definitely anomalies (very short paths), score 0.5 are borderline, and score 0 are very normal mathworks.com. In fact:

- If $E[h(x)] \approx c(n)$ (about what you'd expect for random data), then $s(x) \approx 0.5$ (neutral).
- If $E[h(x)] \ll c(n)$, then $s(x) \to 1$ (anomaly likely) mathworks.com.
- If $E[h(x)] \gg c(n)$ (point requires long paths, which is unusual unless point is very deep in cluster), $s(x) < 0.5$ (very unlikely to be anomaly).

Usually a threshold is set (like $s(x) > 0.65$ or something) to flag anomalies.

Advantages: Isolation Forest is efficient on large, high-dimensional datasets and does not require distance or density computations. It has linear time complexity in number of samples. It naturally handles mixed variable types (splits can be done on numeric or categorical features randomly). It also scales well as an ensemble method and can be parallelized.

Extended Isolation Forest: The original Isolation Forest chooses splits only perpendicular to axes (univariate splits). Extended Isolation Forest (EIF) is an improvement where splits can be done on a random linear combination of features (random hyperplane) instead of a single feature. This can isolate anomalies that lie in some rotated position relative to axes more quickly. In effect, EIF can isolate certain anomalies that would require multiple axis-aligned splits in a single such hyperplane split. This makes it more effective in some cases, at the cost of slightly more computation. The fundamental idea (random partitioning and path lengths) remains the same, but EIF uses hyperplanes (defined by randomly picking a subset of features and random coefficients) to cut the space stats.stackexchange.com. This allows detection of anomalies that are not well-aligned with axes (e.g., an anomaly lying along a diagonal distribution).

In practice, EIF might give a bit better accuracy or require fewer trees for complex data, but both Isolation Forest and Extended IF are based on the same principle of isolation.

R Implementation (Isolation Forest): In R, one can use the IsolationForest from the solitude package or the isotree package (which implements Extended Isolation Forest). Example with isotree:

```
# using isotree
library(isotree)
model <- isolation.forest(X, ntrees=100, sample_size=256)
scores <- predict(model, X, type="score")  # anomaly score for each point
# Flag anomalies above a threshold, e.g., 0.65
anomalies <- which(scores > 0.65)
```

The predict with type=”score” gives the anomaly score (where higher = more anomalous). We could also do type=”distance” for a measure of isolation depth. Another package IsolationForest (R6 based) can also be used similarly.

One-Class SVM

Concept: A one-class SVM is an adaptation of the Support Vector Machine approach for unsupervised anomaly detection (novelty detection). Instead of separating two classes, it attempts to learn a decision boundary that encloses the “normal” data points in feature space, and anything outside this boundary is considered an anomaly scikit-learn.org.

It effectively tries to estimate the support of the distribution of the data: find a function $f(x)$ that is positive (or “1”) in a region capturing most data points (the normal region) and negative (or “-1”) outside that region.

How it works: One-Class SVM with RBF kernel (for example) will project data into a high-dimensional space and find the smallest hypersphere or hyperplane (depending on formulation) that contains most of the data, allowing some slack for outliers (controlled by a parameter ν). In the popular ν-SVR formulation for one-class SVM:

- A parameter $\nu \in (0, 1]$ roughly controls the fraction of outliers (anomalies) and the trade-off with boundary flexibility. Specifically, ν is an upper bound on the fraction of training points that can lie outside the learned boundary (become support vectors for the outlier side) and a lower bound on the fraction of support vectors (so it also indirectly sets a target for how tight the boundary is) scikit-learn.org.

- The algorithm introduces a slack variable for each point which allows it to be outside the boundary, and minimizes a combination of (a) the volume of the hypersphere (or distance of hyperplane from origin) and (b) the fraction of points outside.

Intuitively, one-class SVM tries to carve out a region (in input space or kernel space) that captures the training data. If a new point lies outside this region, the model outputs "-1" (novel/anomalous); if inside, "+1" (normal).

Kernel usage: The use of an RBF kernel allows capturing nonlinear boundaries. For example, if normal data lie around a complicated shape, the one-class SVM can model a similarly shaped boundary. In input space, this corresponds to something like a "contour" around the data points.

Comparison to Isolation Forest: One-class SVM is more like a boundary approach (like clustering-based methods or density estimation) whereas Isolation Forest is more like a randomness-based scoring. One-class SVM can be sensitive to the choice of kernel parameters (especially gamma for RBF, and ν). It might also scale less well to large datasets (it involves an $O(n^2)$ kernel matrix unless using partial fits). Isolation Forest tends to be faster on large high-d data. One-class SVM can sometimes be too rigid if data has clusters – it may try to enclose all data in one boundary, possibly leaving out clusters if they are far apart (unless kernel captures multi-modal structure well).

R Implementation (One-class SVM): The e1071 package's svm() function can do one-class by type="one-classification". For example:

```
library(e1071)
model <- svm(X_train, type="one-classification", kernel="radial",
             gamma=0.1, nu=0.05)
pred <- predict(model, X_train)       # predictions: TRUE for inliers, FALSE for outliers
table(pred)                           # how many inliers vs outliers in training
```

Because it's unsupervised, typically you train on data that is believed to be mostly normal (or entirely normal). The predict returns a logical: TRUE means the point is predicted to be an inlier (normal), FALSE means an outlier. Internally, the decision function sign is what matters. If you want the raw decision value (distance from boundary) you can use decision.values=TRUE in training and then attr(predict(...), "decision.values") to get the score. The sign of that score (positive/negative) corresponds to inlier/outlier.

Choosing ν and γ: $\nu`roughlycorrespondstothefractionofoutliersyouexpect(ifyouset\nu =$ 0.05, the boundary will be such that roughly 5

Extended Isolation Forest vs One-Class SVM: Extended IF can capture some non-linear patterns by using hyperplanes, but one-class SVM with an RBF kernel can capture even more complex boundaries if tuned right. However, one-class SVM focuses on a single contiguous region (unless the kernel trick effectively makes disjoint pockets join in feature space). If normal data has multiple clusters with low density in between, one-class SVM might include the low-density area as normal (since it tries to enclose everything unless ν forces some points to be outliers). Methods like Isolation Forest or clustering-based outlier detection might more easily separate those.

Extended Isolation Forest (Additional Note)

As mentioned, Extended Isolation Forest (EIF) modifies the splitting strategy of Isolation Forest. In an EIF, a split is defined by a random hyperplane (pick two random data points and draw a hyperplane between them, or pick a random direction uniformly from a unit sphere and a random cut along that direction). This can isolate points in some tricky distributions faster. For example, imagine points all lie roughly on a line except one point off the line. A standard Isolation Forest with axis-aligned splits might need many splits to isolate the off-line point if the line is not axis-aligned. An EIF could choose a split parallel to that line and isolate the outlier in one cut.

In implementation terms (like the isotree library in R/Python which implements EIF), you often don't have to do anything special – it's the default method in some libraries or an option. The scoring mechanism remains the same (path lengths). So from a user perspective, one just uses it similarly to IsolationForest. The isotree package in R uses extended isolation by default (with ndim parameter controlling how many features to combine for splits; ndim=1 would revert to standard isolation forest, ndim=2 or more gives extended behavior).

Self-Organizing Maps (SOM)

Overview: A Self-Organizing Map (SOM), also known as a Kohonen map, is a type of neural network that performs dimensionality reduction while preserving the topological structure of the data en.wikipedia.org. It maps high-dimensional data into a usually 2-dimensional grid of neurons. Each neuron has a weight vector of the same dimensionality as the input data. The neurons are arranged in a grid (often hexagonal or rectangular topology), and during training, nearby neurons on the grid learn to respond to similar input patterns – hence preserving neighborhood relationships in the input space on the low-dimensional map en.wikipedia.org.

Learning Process (Kohonen's algorithm):

- **Initialization:** Set the weight vectors (codebook vectors) for each neuron, often to small random values or by sampling from the dataset.
- For each training sample $x(t)$ (in random order):
  - **Best Matching Unit (BMU) selection:** Find the neuron with weight vector closest to the input x (usually Euclidean distance) en.wikipedia.org. This neuron is the BMU, denoted u (its index) en.wikipedia.org.
  - **Update weights:** Move the BMU's weight vector and its neighboring neurons' weight vectors closer to x en.wikipedia.org. The update

for a neuron v with weight W_v at iteration s:

$$W_v(s+1) = W_v(s) + \alpha(s) \cdot \theta(u, v, s) \cdot (x - W_v(s)),$$

where $\alpha(s)$ is the learning rate at iteration s, and $\theta(u, v, s)$ is the neighborhood function that gives the influence of BMU on neuron v en.wikipedia.org. Typically $\theta(u, v, s)$ is 1 for neurons in the neighborhood of BMU and decays with distance from BMU (often a Gaussian or flat kernel within a certain radius), and this neighborhood radius shrinks over time en.wikipedia.org.

- **Decay parameters:** The learning rate $\alpha(s)$ decreases over time (starts around 0.1 maybe and goes to 0) en.wikipedia.org. The neighborhood radius also decreases over time (start maybe equal to the grid size and shrink to 0). Early in training, a large neighborhood means a broad smoothing (coarse mapping), and later a small neighborhood fine-tunes local details en.wikipedia.org.
- Repeat until convergence or for a fixed number of epochs through the data.

During the process, neurons organize such that neighboring neurons on the grid respond to similar inputs. The end result is that the SOM forms a kind of topology-preserving map of the input space. Clusters in input space will appear as clusters of neurons in the map, and nearby neurons represent inputs that were similar.

BMU and Neighborhood: The BMU is essentially like the "winner" neuron for the input (like in vector quantization). But unlike k-means where only the winner centroid moves, in SOM the winner and neighbors move, which encourages smoothness on the map en.wikipedia.org. The neighborhood function $\theta(u, v, s)$ can be, for example, $\exp(-\text{distgrid}(u, v)^2/(2\sigma^2(s)))$ where distgrid is distance on the map grid and $\sigma(s)$ decreases over time (initially large, then small).

After Training: We often use the SOM for visualization. Each neuron represents a prototype data vector (its weight). One can color the map by various measures:

- **U-Matrix (Unified distance Matrix):** It's a common visualization where for each neuron, you compute the average distance between that neuron's weight vector and its immediate neighbors' weight vectors. Plotting these distances as color on the map highlights cluster boundaries (large distances between neurons indicate cluster separation). Regions of the map with uniformly low distances (same color) indicate a cluster of similar inputs.
- **Component Planes:** A grid showing the value of each dimension of the weight vectors across the map. This shows how each original feature varies across the map, which can aid interpretation of what the SOM has learned (like which area of the map corresponds to high values of feature1, etc.).

- **Labels:** If known class labels exist for data points, one can see where those points map on the SOM and label the map accordingly (e.g., map regions dominated by one class).

Use cases: SOMs were popular for visualizing high-dimensional data (like in marketing for customer segments, or in engineering for sensor patterns). They perform a clustering (each neuron can be seen as a cluster center), but with the added benefit of an organized structure. They can also be used for vector quantization and compression.

R Implementation (SOM): The kohonen package provides SOM. Example with the iris dataset:

```
library(kohonen)
# Prepare training data (scaled)
trainX <- scale(iris[, -5])
# Set up a 5x5 hexagonal grid SOM
som_grid <- somgrid(xdim=5, ydim=5, topo="hexagonal")
som_model <- som(trainX, grid = som_grid, rlen=100, alpha=c(0.05, 0.01))
plot(som_model, type="codes")         # code vectors (weight vectors) for each neuron
plot(som_model, type="quality")       # quantization error of each neuron (measures quality
plot(som_model, type="mapping", labels=iris$Species)  # map data points and label by specie
```

Here `rlen=100` means 100 iterations through the data. `alpha` can be given as a vector of two values: start and end learning rate. The `som()` function in kohonen performs the training. We can visualize the resulting SOM:

- `type="codes"` will show the weight vectors (e.g., for each neuron, a small pie chart or bar chart of its weight components if the dimension is small enough, or a heatmap if many neurons).

- `type="mapping"` plots each data point onto the map (mapping each data sample to its BMU) and can use different symbols or colors for different classes (like Species in iris) to see how classes distribute on the map (often, same classes cluster together on the SOM if the SOM captured the structure).

- `type="quality"` or `type="dist.neighbours"` can show U-matrix (distance between neighbor code vectors).

Interpretation: If the map is well-trained, points that are similar (in original feature space) should map to either the same neuron or nearby neurons on the grid. This makes it easy to spot clusters: e.g., in iris SOM, perhaps setosa maps to a distinct corner of the map, whereas versicolor and virginica occupy other parts with some boundary between them.

Neighbourhood & Adaptation: In early iterations, the BMU's whole neighborhood (maybe entire map) gets nudged towards the input, effectively roughly aligning the map to the data distribution. Over time the radius shrinks, so in final fine-tuning, only very close neighbors (or just BMU itself) move, fine-adjusting the map.

SOM vs. k-means: SOM is like k-means (it finds prototypes) plus an added constraint that prototypes are arranged in a grid that smoothly changes. K-means lacks the topological ordering – centroids have no relation to each other, whereas SOM centroids (neurons) have a neighborhood structure. This can be helpful for producing a visualization where, for instance, neighboring clusters are more similar than distant clusters (which may be more intuitively pleasing than just an arbitrary list of cluster centers).

Use in Anomaly Detection: SOMs can also be used for anomaly detection by noticing if some data points map to neurons with very high quantization error (no neuron close enough), or to neurons that represent very few data points (outliers may either distort a neuron weight far away or map poorly).

Having covered clustering, dimensionality reduction, association rules, anomaly detection, and SOMs, we now summarize key points and provide some quiz-style QA for review.

Quizzes and Key Concepts

Q1: What is the difference between hierarchical clustering and k-means clustering?

A1: Hierarchical clustering builds a hierarchy (usually via agglomerative merging), doesn't need the number of clusters as input (you can choose any level to cut the dendrogram), and can produce a dendrogram to visualize clustering structure. K-means requires specifying K in advance and partitions the data into exactly K clusters by iteratively refining cluster centroids to minimize within-cluster variance. K-means is flat (one partition) and tends to produce spherical clusters, whereas hierarchical can produce nested clusters and can use various linkage metrics allowing non-spherical cluster shapes (though still tends to split by distance). Hierarchical clustering can be more computationally expensive (O(n^3) naive, or O($n^2 \log n$) with optimizations) vs k-means (O(n) per iteration, times a small number of iterations).

Q2: In PCA, how do we decide how many principal components to keep?

A2: Common approaches: (1) Keep enough components to explain some desired cumulative variance threshold (e.g., 90

Q3: True or False – In association rule mining, a rule with high confidence always has high lift.

A3: False. A rule $X \Rightarrow Y$ can have high confidence simply because Y is very common (support(Y) is high). Lift measures how much more co-occurrence there is beyond chance. You can have confidence = 0.8 but if Y appears in 0.8 of all transactions (80

Q4: What is an anomaly score in Isolation Forest and how is it interpreted?

A4: The anomaly score in an Isolation Forest is typically $s(x) = 2^{-\frac{E[h(x)]}{c(n)}}$, where $E[h(x)]$ is the average path length to isolate point x across all trees, and $c(n)$ is a normalization constant for dataset size stats.stackexchange.com.

Scores close to 1 indicate x is very likely an anomaly (it got isolated very quickly, $E[h(x)]$ much smaller than expected) mathworks.com. Scores around 0.5 indicate x is similar to normal points (average isolation length). Scores below 0.5 would mean extremely normal (in very dense area). In practice, one might choose a threshold like 0.7 or the top few percent of scores as anomalies, depending on desired false alarm rate.

Q5: How does a Self-Organizing Map preserve topological structure of data?

A5: A SOM preserves topology by updating not just the best matching unit (BMU) for a given input, but also its neighboring neurons in the map space to make them more like the input en.wikipedia.org. This means neurons that are near each other on the 2D map will respond to similar inputs (their weight vectors become similar). Over training, the map stretches and folds to fit the cloud of data points in the high-dimensional space, maintaining the original neighborhood relations insofar as possible on a 2D surface. Thus, if two inputs are similar, they're likely to excite neurons that are close together on the map.

Q6: What is support in the context of association rules, and why do we need a minimum support threshold?

A6: Support of an itemset (or rule) is the proportion of transactions in the dataset that contain that itemset en.wikipedia.org. A minimum support threshold is needed to ensure that we only find patterns that occur frequently enough to be considered reliable or interesting. It prunes the search space massively – without it, the algorithms would output an astronomical number of infrequent combinations which are likely just noise. Also, very low support rules might not generalize (could be one-off coincidences). For example, if only 2 transactions out of millions have item A, item B, item C together, that itemset (and any rule from it) is probably not useful from a business standpoint and is pruned by requiring support maybe 0.1

Q7: When might you prefer using t-SNE over PCA, and what is a major caution when interpreting t-SNE plots?

A7: Use t-SNE when you want to visualize high-dimensional data and you suspect complex, nonlinear relationships or clusters that PCA (a linear method) cannot disentangle. t-SNE often excels at creating a 2D/3D visualization where distinct clusters in some manifold sense become visibly separated. For example, for image data or word embeddings, t-SNE can reveal groupings that PCA might leave overlapping. Caution: Distances in a t-SNE plot are not globally meaningful – only the local neighborhood structure is reliable. The space between clusters or the exact geometry can be arbitrary due to the cost function. So you shouldn't interpret, say, cluster A being twice as far from B as from C in the t-SNE plot as a quantitative statement. Also, each run can differ slightly; stability and parameter sensitivity mean it's mainly a qualitative tool.

Q8: In One-Class SVM, what does the ν parameter control?

A8: ν in one-class SVM controls the trade-off between allowing outliers and the tightness of the decision boundary scikit-learn.org. More specifically, ν is an upper bound on the fraction of training points that can be outside the boundary (treated as outliers during training) and a lower bound on the fraction of training points that end up as support vectors. If you set a larger ν, you allow more points to be considered outliers (so the boundary will shrink more, focusing on a smaller core of data). If ν is very small, the algorithm will try to include almost all points inside the boundary (risking overfitting or a very loose boundary that might include regions without data). For example, $\nu = 0.1$ aims to allow 10

Q9: What's the "downward closure" property in frequent itemset mining and how does it prune the search?

A9: The downward closure (or Apriori) property says: if an itemset is frequent, all of its subsets are also frequent en.wikipedia.org. Contrapositive: if an itemset is infrequent, all of its supersets are infrequent en.wikipedia.org. This property prunes the search because: when you generate candidate itemsets of size k, you only consider those whose every $(k-1)$-subset was frequent in the previous round. If any subset wasn't frequent, you skip this candidate because it can't possibly be frequent (no need to even count it). This dramatically cuts down the number of candidates when k grows, as the combinatorial explosion is curtailed by early frequency failures.

Q10: How does a Gaussian Mixture Model perform clustering differently from k-means?

A10: GMM is a soft (probabilistic) clustering – it assumes each cluster is a Gaussian distribution and uses the EM algorithm to estimate the parameters. Each point belongs to each cluster with some probability (the responsibility) users.cs.duke.edu. K-means is a hard clustering – each point is assigned to exactly one cluster, and clusters are defined just by centroids. GMM can capture clusters with different sizes and elliptical shapes (full covariance matrices), whereas k-means (which is like GMM with equal spherical covariances) tends to equal-radius spheres scikit-learn.org. Also, GMM provides likelihoods and can use model selection criteria (BIC/AIC) to choose K. In short: k-means finds a partition minimizing variance; GMM finds a mixture of Gaussians that likely generated the data, yielding both a clustering and a density model.

This concludes the study guide on unsupervised learning topics. By understanding these clustering methods, dimensionality reduction techniques, association rules, anomaly detectors, and neural mapping techniques, you have a broad toolkit for exploring and analyzing data without the need for explicit labels. Each method has its assumptions and specialties, so choosing the right one depends on the problem at hand – and sometimes, using multiple in combination (e.g., using PCA or t-SNE to visualize clusters found by k-means, or using hierarchical clustering to summarize association rule segments) can be very powerful.