📘 Machine Learning Interview Questions, Answers & Algorithm Roadmaps

This document provides interview Q&A and **step-by-step roadmaps** for essential machine learning algorithms. Each roadmap explains the algorithm flow, purpose of each step, and basic syntax (no code examples).

🔹 K-Nearest Neighbors (KNN) - Roadmap

Step 1: Import libraries Use from sklearn.neighbors import KNeighborsClassifier. This loads the algorithm into your workspace.

Step 2: Feature Scaling Use StandardScaler. KNN uses distance metrics, so features must be on the same scale.

Step 3: Split data into train/test Use train_test_split() to evaluate generalization.

Step 4: Fit the model model = KNeighborsClassifier(n_neighbors=5) followed by model.fit(X_train, y_train).

Step 5: Predict and evaluate Use model.predict() and accuracy_score() to check performance.

Why these steps? KNN compares data points using distance, so scaling and test splitting are essential. n_neighbors (K) should be chosen carefully.

🔹 KMeans Clustering - Roadmap

Step 1: Preprocessing

- Remove nulls
- Apply StandardScaler()

Step 2: Choose value of K Use Elbow method (plot inertia vs. K) to find optimal cluster count.

Step 3: Initialize and fit Use KMeans(n_clusters=K) and model.fit(data).

Step 4: Get labels and cluster centers Use model.labels_ and model.cluster_centers_.

Step 5: Visualize Plot using matplotlib to understand groupings.

Why these steps? Preprocessing ensures equal weight in clustering. Elbow method helps pick effective number of clusters.

🔹 Hierarchical Clustering - Roadmap

Step 1: Scale features Use StandardScaler() to normalize for fair distance comparison.

Step 2: Compute linkage matrix Use scipy.cluster.hierarchy.linkage() with method='ward' or 'complete'.

Step 3: Plot dendrogram Use dendrogram() to visualize data hierarchy.

Step 4: Choose number of clusters Cut dendrogram at a specific height (threshold) to form clusters.

Step 5: Assign labels Use fcluster() to label each data point.

Why these steps? Hierarchical clustering shows how data clusters step-by-step, and dendrograms help in visual decision-making.

🔹 Apriori Algorithm - Roadmap

Step 1: Transaction Data Preprocessing

- Convert dataset to boolean using df.applymap(lambda x: 1 if x > 0 else 0)

Step 2: Generate frequent itemsets Use apriori() from mlxtend.frequent_patterns with min_support.

Step 3: Generate association rules Use association_rules() with metric='confidence' and min_threshold.

Step 4: Filter strong rules Use lift > 1 and confidence > 0.6 for meaningful patterns.

Why these steps? Apriori finds itemsets frequently occurring together. Data must be in binary form to evaluate presence/absence of items.

🔹 Linear Regression - Roadmap

Step 1: Handle missing values & outliers Linear regression is sensitive to outliers. Clean your dataset.

Step 2: Check linear relationship Use scatter plots or correlation matrix.

Step 3: Split data Use train_test_split().

Step 4: Fit the model Use LinearRegression() and .fit(X_train, y_train).

Step 5: Evaluate Check R^2 score, MSE, or RMSE using metrics module.

Why these steps? Linear regression assumes a linear pattern and is affected by data quality. Evaluation confirms model quality.

◆ Logistic Regression - Roadmap

Step 1: Check class imbalance Use value_counts() to verify if one class dominates.

Step 2: Apply scaling Improves convergence.

Step 3: Fit the model Use LogisticRegression() and .fit().

Step 4: Predict and evaluate Use classification report, confusion matrix, ROC-AUC.

Step 5: Apply regularization Use penalty='l1' or 'l2' to reduce overfitting.

Why these steps? Logistic regression outputs probabilities and works best with balanced, scaled data. Regularization improves generalization.

◆ Decision Trees - Roadmap

Step 1: Preprocess data Trees don't require scaling but handle missing or categorical values properly.

Step 2: Fit model Use DecisionTreeClassifier() with criterion='gini' or 'entropy'.

Step 3: Tune hyperparameters Control depth, min_samples_split, etc., to avoid overfitting.

Step 4: Evaluate model Use accuracy, confusion matrix, and tree visualization.

Why these steps? Tree models learn rules, but without constraints can overfit. Tuning is necessary.

◆ Random Forest - Roadmap

Step 1: Preprocess No scaling required. Handle nulls and categorical variables.

Step 2: Fit model Use RandomForestClassifier() and fit on training data.

Step 3: Evaluate using OOB score Set oob_score=True to get out-of-bag error estimate.

Step 4: Feature importance Use model.feature_importances_ to interpret.

Step 5: Hyperparameter tuning Use GridSearchCV or RandomizedSearchCV.

Why these steps? Random Forest reduces overfitting via ensemble and gives useful insight into feature relevance.

🔹 Support Vector Machine (SVM) - Roadmap

Step 1: Scale the data SVM is very sensitive to feature scale. Use StandardScaler.

Step 2: Fit model Use SVC() with kernel='linear' or 'rbf'. Fit on training data.

Step 3: Tune parameters C (regularization) and gamma (kernel coefficient) using GridSearch.

Step 4: Evaluate performance Use confusion matrix, accuracy, and ROC-AUC.

Step 5: Visualize margin (for 2D) Use support vectors and decision boundary plots.

Why these steps? SVM works best when data is scaled and hyperparameters are tuned.

🔹 Naive Bayes - Roadmap

Step 1: Format categorical/text data Use label encoding for categorical or CountVectorizer/TfidfVectorizer for text.

Step 2: Choose model Use GaussianNB, MultinomialNB, or BernoulliNB based on data type.

Step 3: Fit the model Use .fit(X_train, y_train).

Step 4: Predict and evaluate Use .predict() and classification metrics.

Step 5: Analyze prior and likelihood Use .class_prior_ and .theta_ for understanding model internals.

Why these steps? Naive Bayes is fast and suitable for text but assumes feature independence, which may limit performance on some data.