1. K-Nearest Neighbors (KNN)

Q1: How does the KNN algorithm work?

KNN is a lazy, instance-based supervised algorithm. It stores all training data, and when a new data point arrives, it calculates the distance (e.g., Euclidean) to all training points and selects the K nearest neighbors. The predicted class is the majority vote (for classification) or average (for regression) of those neighbors. It doesn't learn any internal model during training, which makes it simple but computationally expensive for large datasets.

Q2: How do you choose the value of K in KNN?

Choosing K is critical: a low K (e.g., K=1) is sensitive to noise and overfitting, while a high K oversmooths the decision boundary and may underfit. Commonly, cross-validation is used to find the best K. The elbow method is often used, plotting K vs error to find a point beyond which error rate stabilizes. Typically, odd K values are preferred to avoid ties in classification.

Q3: Why is feature scaling important in KNN?

KNN relies on distance calculations. If one feature (e.g., salary) is on a much larger scale than another (e.g., age), it dominates the distance metric, skewing the neighbor selection. Standardization (zero mean, unit variance) or normalization (rescaling 0-1) ensures all features contribute equally to the distance computation. Without scaling, model performance can degrade significantly.

Q4: Is KNN sensitive to outliers?

Yes, KNN is highly sensitive to outliers because it makes predictions based on nearby instances. If outliers are close to a new data point, they may wrongly influence classification or regression results. Using robust distance metrics or cleaning data beforehand can help mitigate this issue. Larger values of K can also reduce the impact of individual outliers.

Q5: Can KNN be used for regression?

Yes, KNN can be used for regression. Instead of a majority vote, the prediction is the average (mean) of the K nearest neighbors' target values. It performs well for non-linear problems with sufficient data. However, it is still sensitive to scaling and outliers, and doesn't provide confidence intervals or coefficients, unlike linear regression

2. KMeans Clustering

Q1: How does KMeans clustering work?
KMeans is an unsupervised learning algorithm that partitions the data into K clusters. It initializes K centroids randomly, assigns data points to the nearest centroid, and recalculates centroids as the mean of points in each cluster. This process repeats until assignments stabilize or max iterations are reached. The goal is to minimize intra-cluster variance (sum of squared distances).

Q2: How do you determine the optimal number of clusters in KMeans?
The elbow method is commonly used. It involves plotting the number of clusters (K) against the inertia (sum of squared distances) and looking for an "elbow" point where the reduction in inertia slows. This indicates a good trade-off between model complexity and fit. Silhouette score and Gap statistics are also used to assess cluster quality.

Q3: What is the objective function of KMeans?
The objective function of KMeans is to minimize the sum of squared Euclidean distances between each point and its assigned cluster centroid. This function is known as the Within-Cluster Sum of Squares (WCSS). The algorithm iteratively adjusts centroids to reduce this value, aiming to create compact and well-separated clusters.

Q4: Is KMeans suitable for all types of data?
No, KMeans works best with numerical, continuous, and spherical data. It performs poorly with categorical data, irregular cluster shapes, or different density clusters. For categorical features, algorithms like K-Modes or Gower distance-based clustering are more appropriate. Scaling is also important to ensure distance calculations are meaningful.

Q5: What are the limitations of KMeans?
KMeans requires the number of clusters K as input, assumes spherical clusters, and is sensitive to outliers and initial centroid placement. It may converge to local minima. It also struggles with clusters of varying density or size. Using KMeans++ for initialization and scaling features can partially mitigate these issues.

---

3. Hierarchical Clustering

Q1: What is hierarchical clustering and how does it work?
Hierarchical clustering builds a hierarchy of clusters either in a bottom-up (agglomerative) or top-down (divisive) manner. Agglomerative clustering starts with each point as a separate cluster and merges the closest pairs iteratively. A dendrogram is used to visualize the cluster merging process and decide the cut-off point for final clusters.

Q2: What is a dendrogram and how is it used?
A dendrogram is a tree-like diagram that shows the sequence of merges in hierarchical clustering. The y-axis represents the distance (or dissimilarity) at which clusters are merged. By cutting the dendrogram at a chosen height, you can determine the number of clusters. It helps visually assess the optimal clustering structure.

Q3: What is the difference between single, complete, and average linkage?
Linkage criteria determine how distances between clusters are calculated:

- Single linkage: shortest distance between points in two clusters
- Complete linkage: farthest distance
- Average linkage: average distance
  Each method produces different cluster shapes. Single can create "chained" clusters, complete gives compact clusters, and average provides a balance.

Q4: What are the pros and cons of hierarchical clustering?
Pros: No need to pre-specify the number of clusters, intuitive, and provides rich insights via dendrograms. Cons: Poor scalability to large datasets, sensitive to noise and outliers, and once merged/split, cannot undo steps. Choosing the right linkage method and distance metric is crucial for good results.

Q5: When should you prefer hierarchical over KMeans?
Prefer hierarchical clustering when the number of clusters is unknown, data size is small, or when interpretability is important. It’s useful for exploratory data analysis. KMeans is better for larger datasets and when speed is critical. Hierarchical clustering also works well when you want a nested clustering structure.

4. Apriori Algorithm

Q1: What is the Apriori algorithm used for?
Apriori is an unsupervised association rule learning algorithm used to find frequent itemsets and generate rules from transaction-like data. It helps discover relationships such as "if item A is bought, item B is likely to be bought." It works by identifying item combinations with support above a minimum threshold and then deriving rules with sufficient confidence and lift. It’s popular in market basket analysis.

Q2: What is support, confidence, and lift in Apriori?

- **Support**: Proportion of transactions containing the itemset.
- **Confidence**: Probability that consequent appears given antecedent.
- **Lift**: How much more likely items co-occur vs. randomly.
  These metrics help measure rule strength and usefulness. Lift > 1 indicates a strong positive association.

Q3: Why must data be in boolean format for Apriori?
Apriori checks the presence or absence of items. Hence, the input must be in boolean format (True/False or 1/0). If the dataset contains numeric quantities, it must be converted into binary (e.g., 1 if quantity > 0, else 0). Non-binary data can distort frequency counting and yield incorrect itemsets.

Q4: How do you choose support and confidence thresholds?
Start with low values to explore patterns, then adjust based on results. If support is too high, rare but interesting rules may be missed. If too low, you may get too many noisy or insignificant rules. Confidence controls the reliability of rules. Lift helps identify truly interesting associations beyond random chance.

Q5: What are the limitations of the Apriori algorithm?
Apriori is computationally expensive due to candidate generation and repeated dataset scans. It performs poorly on large datasets with many features. Also, it assumes independence between itemsets and doesn’t consider time/context. FP-Growth is often preferred as it’s faster and more memory efficient.

---

5. Linear Regression

Q1: What is linear regression and how does it work?
Linear regression is a supervised algorithm used for predicting continuous values. It assumes a linear relationship between the input features and the target variable. The model tries to fit a straight line ($y = mx + b$) that minimizes the error between actual and predicted values using a loss function, typically Mean Squared Error (MSE).

Q2: What assumptions does linear regression make?
Linear regression assumes linearity, independence of errors, homoscedasticity (equal variance of errors), normal distribution of residuals, and no multicollinearity between features. Violating these can lead to misleading or biased results. Diagnostics like residual plots or variance inflation factor (VIF) can help validate these assumptions.

Q3: What is the cost/loss function used in linear regression?
The most common cost function is Mean Squared Error (MSE), defined as the average squared difference between actual and predicted values. The model learns by minimizing this function using optimization techniques like gradient descent. MSE penalizes larger errors more heavily, ensuring smoother predictions.

Q4: How do you evaluate the performance of a linear regression model?
Metrics include R-squared (explains variance captured by model), MSE, MAE (Mean Absolute Error), and RMSE. R-squared ranges from 0 to 1, with higher values indicating better fit. Residual analysis and plots also help evaluate model assumptions and accuracy.

Q5: Can linear regression work with categorical data?
Yes, but you must convert categorical variables into numeric format using one-hot encoding or label encoding. Without transformation, linear regression cannot process non-numeric values. Care must be taken to avoid dummy variable trap by dropping one dummy column to maintain independence.

---

6. Logistic Regression

Q1: What is logistic regression and when is it used?
Logistic regression is used for binary classification problems where the output is 0 or 1. Instead of fitting a straight line like linear regression, it uses a sigmoid function to output probabilities. It predicts the probability that an input belongs to a class and classifies it based on a threshold (usually 0.5).

Q2: What is the loss function used in logistic regression?
Logistic regression uses the **log loss** or **binary cross-entropy** as its cost function. It penalizes predictions based on how far they are from the actual label (0 or 1). Minimizing log loss ensures predicted probabilities are close to true class labels.

Q3: How do you interpret the coefficients in logistic regression?
The coefficients represent the log-odds change in the outcome for a one-unit increase in the feature. A positive coefficient increases the likelihood of class 1, while a negative coefficient decreases it. Coefficients can be exponentiated to interpret them as odds ratios.

Q4: What assumptions does logistic regression make?
Assumptions include: no multicollinearity among features, linearity between independent variables and the log-odds, and independence of observations. It does not require normally distributed features or equal variance.

Q5: Can logistic regression be used for multi-class classification?
Yes, using one-vs-rest (OvR) or multinomial logistic regression. OvR builds one binary classifier for each class. The class with the highest predicted probability is selected. Scikit-learn supports both approaches using the multi_class parameter.

---

7. Decision Tree

Q1: How does a decision tree algorithm work?
A decision tree splits the dataset into branches based on feature values to make predictions. At each node, the best feature is selected to split the data using metrics like Gini Impurity or Information Gain (entropy). The tree grows recursively until a stopping criterion (max depth, min samples) is met.

Q2: What are Gini Impurity and Entropy in decision trees?
Both are measures of node impurity. Gini Impurity measures the probability of misclassification. Entropy measures information disorder. Lower values indicate purer nodes. While both are used for splitting, Gini is faster and often preferred. Entropy tends to produce deeper trees.

Q3: What are the advantages of decision trees?
They are easy to understand, interpret, and visualize. They handle both numerical and categorical data, require little preprocessing, and can model non-linear relationships. However, they can overfit on noisy datasets without pruning or regularization.

Q4: How do you prevent overfitting in decision trees?
Use pruning (post or pre), set maximum tree depth, minimum samples per split, or use ensemble methods like Random Forests. Cross-validation can also help assess generalization performance and tune hyperparameters.

Q5: What are limitations of decision trees?
They are prone to overfitting, especially with small or noisy data. Small changes in data can lead to different trees (high variance). They can also create biased trees if class imbalance exists. Ensemble methods address many of these issues.

---

8. Random Forest

Q1: How does Random Forest work?
Random Forest is an ensemble method that builds multiple decision trees on different bootstrapped subsets of the data. Each tree is trained on a random subset of features. Predictions are made by majority voting (classification) or averaging (regression). This reduces overfitting and improves generalization.

Q2: What are the advantages of Random Forest?
It provides high accuracy, robustness to outliers and noise, and handles both classification and regression. It reduces variance from individual decision trees, supports feature importance ranking, and requires minimal parameter tuning.

Q3: What is Out-of-Bag (OOB) error in Random Forest?
OOB error is an internal validation method. Each tree is trained on a bootstrap sample (~63% of data), and the remaining ~37% is used to test that tree. Aggregating predictions across trees gives an unbiased error estimate, eliminating the need for separate validation sets.

Q4: What are the key hyperparameters in Random Forest?
Important hyperparameters include:

- n_estimators: number of trees
- max_depth: maximum depth of each tree
- min_samples_split: minimum samples to split a node
- max_features: number of features to consider per split
  Tuning these helps improve model performance.

Q5: How is feature importance calculated in Random Forest?
Feature importance is measured by how much each feature decreases impurity across all trees. Features that lead to larger reductions in Gini or entropy are considered more important. This helps in feature selection and model interpretability.

---

9. Support Vector Machine (SVM)

Q1: What is the basic idea behind SVM?
SVM finds the optimal hyperplane that best separates classes by maximizing the margin (distance) between the nearest data points (support vectors). It's effective in high-dimensional spaces and supports linear and non-linear classification using kernel tricks.

Q2: What is a kernel in SVM?
A kernel transforms data into higher-dimensional space to make it linearly separable. Common kernels include:

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid
  Choosing the right kernel is crucial to handle non-linear patterns.

Q3: What is the loss function in SVM?
SVM uses **hinge loss**, which penalizes points inside the margin or on the wrong side of the decision boundary. The optimization goal is to minimize hinge loss while maximizing margin, controlled by the C parameter.

Q4: What is the role of the C parameter in SVM?
C controls the trade-off between margin size and classification error. A small C allows a wider margin with more misclassifications, while a large C forces the model to minimize classification error, possibly leading to overfitting.

Q5: Can SVM be used for regression?
Yes, using Support Vector Regression (SVR). SVR tries to fit the best line within a threshold (ε margin) around actual values. Points outside the margin are penalized. It works well for small- to medium-sized datasets with clear margin boundaries.

---

10. Naive Bayes

Q1: What is the Naive Bayes algorithm based on?
Naive Bayes is a probabilistic classifier based on Bayes' Theorem. It assumes feature independence given the class label. Despite the strong independence assumption, it works well in practice, especially for text classification. It predicts the class with the highest posterior probability.

Q2: What is Bayes' Theorem and how is it used in Naive Bayes?
Bayes' Theorem calculates the probability of a class given the features:
P(Class | Data) = [P(Data | Class) * P(Class)] / P(Data)
Naive Bayes estimates these probabilities using the training data and classifies the instance based on the highest posterior probability.

Q3: What are the types of Naive Bayes classifiers?

- **Gaussian Naive Bayes**: for continuous data, assumes normal distribution
- **Multinomial Naive Bayes**: for count data (e.g., word counts in documents)
- **Bernoulli Naive Bayes**: for binary features
  Each is suited to different types of data.

Q4: What are the advantages of Naive Bayes?
It is simple, fast, and performs well with high-dimensional data. It requires less training data, handles multi-class problems, and is scalable. It's particularly effective in spam detection, document classification, and sentiment analysis.

Q5: What are the limitations of Naive Bayes?
Its main limitation is the assumption of feature independence, which rarely holds true in real-world data. It can also struggle with zero probabilities (handled by Laplace smoothing) and does not model relationships between features, unlike other algorithms.

End of Guide.