Q: How does the KNN algorithm work?

KNN is a non-parametric, instance-based learning algorithm. It stores the training data and, for a new data point, it calculates the distance to all training points. It then selects the k nearest neighbors and classifies the new point by majority voting (for classification) or by averaging (for regression). It doesn’t build an explicit model but relies on data similarity.

Q: What role does the value of k play in KNN, and how do you choose it?

The value of k controls the smoothness of the decision boundary. Small k gives high variance and sensitivity to noise. Large k produces smoother predictions but may introduce bias. Typically, k is chosen using cross-validation or heuristics like $\sqrt{n}$.

Q: What are the pros and cons of using small vs. large k values?

Small k captures local structure but is prone to noise/outliers. Large k reduces variance but can oversmooth decision boundaries and ignore minority classes.

Q: How does KNN handle classification vs. regression tasks?

For classification, it predicts the majority class among k-neighbors. For regression, it predicts the average (or weighted average) of neighbor values.

Q: Why is feature scaling important in KNN?

KNN depends heavily on distance metrics. Features with larger numerical ranges dominate distance calculations, making the algorithm biased toward them. Scaling ensures all features contribute equally.

Q: What distance metrics can be used in KNN, and when would you prefer Manhattan over Euclidean distance?

Common choices: Euclidean (best for continuous data with isotropic distributions), Manhattan (better for high-dimensional or grid-like data where movement is axis-aligned), Minkowski (generalized), Cosine similarity (text/vector data).

Q: What happens if there’s a tie when classifying with KNN?

Possible solutions: Reduce k to an odd number, use distance-weighted voting (closer neighbors get higher weight), or break ties randomly.

Q: How does KNN behave in high-dimensional spaces (curse of dimensionality)?

In high dimensions, all points tend to be far apart, making distance metrics less meaningful. KNN then struggles because it can’t distinguish between 'close' and 'far.' Dimensionality reduction (PCA, feature selection) is often applied before KNN.

Q: Is KNN a lazy learner or eager learner? Why?

KNN is a lazy learner because it doesn’t build a model during training. It simply stores data and performs heavy computation during prediction (distance calculations).

Q: Can KNN be used for imbalanced datasets? What modifications might help?

Yes, but imbalanced datasets bias KNN toward the majority class. Fixes include weighted KNN, resampling (oversample minority, undersample majority), or adjusting class priors.

Q: What’s the difference between StandardScaler and MinMaxScaler?

StandardScaler rescales features to zero mean and unit variance. MinMaxScaler rescales data to a fixed range (default [0,1]).

Q: When would you prefer StandardScaler over MinMaxScaler, and vice versa?

StandardScaler: when data is approximately normally distributed and algorithms like SVM or logistic regression assume standardized data. MinMaxScaler: when working with bounded features (e.g., images, neural nets with sigmoid activation).

Q: Suppose your data has outliers — which scaler is more robust, and why?

Neither StandardScaler nor MinMaxScaler handles outliers well. But StandardScaler is slightly better, since extreme values don’t completely squash other features. The best option is RobustScaler, which uses median and IQR.

Q: What range does MinMaxScaler transform data into?

By default, [0,1]. But the range can be customized, e.g., [-1,1].

Q: How does StandardScaler transform data?

It subtracts the mean and divides by the standard deviation: $z = (x - \mu) / \sigma$. This produces standardized features with mean = 0 and variance = 1.

Q: Can scaling affect model performance in algorithms like decision trees? Why or why not?

No. Decision trees split based on thresholds of feature values. Scaling doesn’t change the order of values, so splits remain the same.

Q: In KNN, why is scaling mandatory, but in Naive Bayes it’s often unnecessary?

KNN relies on distances; without scaling, one feature may dominate. Naive Bayes calculates class probabilities based on feature likelihoods — scaling doesn’t affect those conditional distributions.

Q: What would happen if you forgot to scale your features before training an SVM?

Features with large ranges dominate the dot products, leading to distorted margins and poor performance. SVM is very sensitive to feature scaling.

Q: Explain the key assumption behind Naive Bayes.

Naive Bayes assumes conditional independence of features given the class label. That is, the presence of one feature doesn’t affect the presence of another, once the class is known.

Q: What are the types of Naive Bayes classifiers available in sklearn?

GaussianNB (continuous data, assumes normal distribution), MultinomialNB (count-based data like text), BernoulliNB (binary features), ComplementNB (better for imbalanced text classification).

Q: Why is it called 'Naive'?

Because of the unrealistic assumption of complete independence among features. Despite this simplification, it often works surprisingly well.

Q: How does Naive Bayes handle continuous vs. categorical features?

Continuous: assumes Gaussian distribution, estimates mean and variance per class. Categorical: uses frequency-based probability estimates.

Q: How would you handle a case where a probability becomes zero due to unseen data?

Apply Laplace smoothing (add-one smoothing), which adds 1 to frequency counts to avoid zero probabilities.

Q: What are the advantages of Naive Bayes over KNN or SVM?

Very fast training and prediction, works well with small datasets, handles high-dimensional data like text effectively, robust to irrelevant features.

Q: In which scenarios would Naive Bayes perform poorly?

When features are highly correlated, violating independence assumption. When data distribution doesn’t match the assumed model (e.g., non-Gaussian continuous data for GaussianNB).

Q: Explain the concept of a margin in SVM.

The margin is the distance between the separating hyperplane and the nearest data points (support vectors). A larger margin implies better generalization. SVM aims to maximize this margin while correctly classifying the data.

Q: What is the difference between a hard margin and a soft margin SVM?

Hard margin assumes perfectly separable data, no misclassifications allowed, risk of overfitting and not robust to noise. Soft margin allows some misclassification (controlled by parameter C), better for real-world noisy data.

Q: What is the role of the kernel trick in SVM?

The kernel trick allows SVM to classify non-linear data by mapping it into a higher-dimensional feature space without explicitly computing the transformation. It does this using kernel functions like RBF or polynomial.

Q: Can you explain the difference between linear, polynomial, and RBF kernels?

Linear: best for linearly separable data. Polynomial: models curved decision boundaries, degree controls complexity. RBF: maps data to infinite-dimensional space, highly flexible, handles complex boundaries well.

Q: What happens if the regularization parameter C is set too high or too low?

High C: SVM tries to classify all points correctly, small margin, risk of overfitting. Low C: allows more misclassifications, wider margin, risk of underfitting.