Machine Learning (ML)

Question 1: Explain the working of Decision Trees for classification tasks. How does a Decision Tree determine the splitting criteria?

Answer:
A Decision Tree classifies data by splitting it into subsets based on feature values. Each node in the tree represents a feature, each branch represents a possible value of that feature, and each leaf node represents a class label. The algorithm uses criteria like **Gini Impurity** or **Entropy** (for Information Gain) to decide the best split at each node, selecting the feature that provides the highest homogeneity within subsets.

Additional Question:

1. **Describe the differences between Gini Index and Entropy in Decision Trees. When would one be preferred over the other?**
   Answer:
   Gini Index and Entropy are impurity measures. **Gini Index** measures the likelihood of incorrect classification and is faster to compute, often preferred in larger datasets. **Entropy** calculates information gain and is useful when we want a more detailed measure of impurity. While both yield similar results, Entropy can provide a deeper understanding of feature influence in smaller datasets.

Question 2: Describe the Naïve Bayes Classifier. Why is it called "naïve," and how does it utilize Bayes' Theorem?

Answer:
Naïve Bayes is a probabilistic classifier that applies Bayes' Theorem with an assumption of independence between predictors. It calculates the posterior probability of each class given feature values, assigning the class with the highest probability. It is "naïve" because it assumes that features are conditionally independent, which is rarely true in reality.

Additional Question:
2. **Explain why Naïve Bayes works well even with the "naive" assumption of feature independence. Provide examples of real-world applications.**

Answer:
Naïve Bayes often performs well because the independence assumption simplifies calculations without heavily impacting accuracy, especially in text classification and spam detection where words' frequencies are significant. Despite dependencies, the algorithm is effective in scenarios with many features and sparse data.

Question 3: What is K-Means clustering, and how does it determine the optimal number of clusters?

Answer:
K-Means clustering is an unsupervised algorithm that groups data points into K clusters based on feature similarity. It initializes K centroids, assigns points to the nearest centroid, and recalculates centroids until convergence. The **Elbow Method** helps determine K by plotting within-cluster variance and identifying the "elbow" where adding clusters does not significantly reduce variance.

Additional Question:
3. **What is the Elbow Method, and how can it be used to determine the number of clusters in K-Means clustering?**

Answer:
The Elbow Method plots the within-cluster sum of squares (WCSS) against the number of clusters. The optimal number of clusters is at the "elbow" point, where adding more clusters leads to minimal improvement in reducing WCSS. This is visually assessed as the point where the plot levels off.

Question 4: Differentiate between K-Means and Hierarchical clustering algorithms. When would you use each?

Answer:
K-Means is a partition-based clustering method that requires specifying K in advance, whereas Hierarchical clustering is a hierarchical approach that does not require a predetermined K. K-Means is efficient for large datasets, while Hierarchical clustering, which forms a tree-like dendrogram, is useful for smaller datasets where understanding cluster hierarchy is beneficial.

Additional Question:
4. **What are agglomerative and divisive hierarchical clustering? Provide examples of scenarios where each is appropriate.**

Answer:
Agglomerative clustering starts with individual points as clusters and merges them, while **Divisive clustering** starts with one cluster and splits it. Agglomerative is common, often used for genomic and social network data, while divisive clustering can be useful in organizational structure analysis where splitting larger clusters is preferred.

Question 5: Describe how the Apriori algorithm can be applied to Market Basket Analysis for generating association rules.

Answer:
The Apriori algorithm identifies frequent itemsets in transactional data, using support, confidence, and lift metrics to generate rules. In Market Basket Analysis, it reveals associations between items often bought together, enabling stores to design promotions or product placements.

Additional Question:
5. **Define and differentiate support, confidence, and lift in association rule mining.**

Answer:

- **Support**: The proportion of transactions containing a specific itemset.
- **Confidence**: The likelihood of item B appearing in transactions that contain item A.
- **Lift**: The ratio of observed confidence to the expected confidence if A and B were independent. Higher lift suggests a stronger association between items.

Question 6: Explain Simple Linear Regression. How do you interpret the regression coefficients and the coefficient of determination?

Answer:
Simple Linear Regression models the relationship between an independent variable XXX and a dependent variable YYY, fitting a line with slope and intercept to minimize errors. The regression coefficient (slope) represents the rate of change in YYY per unit change in XXX. **R^2** (coefficient of determination) indicates the proportion of variance in YYY explained by XXX, with values close to 1 indicating a strong relationship.

Additional Question:
6. **How does R^2 differ from adjusted R^2, and when would you prefer using adjusted R^2?**

Answer:
R^2 measures the proportion of variance explained but can overestimate model fit as variables are added. **Adjusted R^2** accounts for the number of predictors, adjusting for potential overfitting. It is preferred when comparing models with different numbers of predictors.

Question 7: What are the different types of kernel functions in Support Vector Machines? How do these kernels influence model performance?

Answer:
SVM kernels transform data into higher dimensions to handle non-linear relationships. Types include:

- **Linear Kernel**: Suitable for linearly separable data.
- **Polynomial Kernel**: Maps data into polynomial dimensions.
- **RBF (Gaussian) Kernel**: Handles complex, non-linear relationships by mapping to infinite dimensions.

Additional Question:
7. **Explain the advantages and drawbacks of using RBF over linear kernels in SVM.**

Answer:
RBF kernels capture complex patterns but are computationally intensive and can overfit on small datasets. **Linear kernels** are faster and effective for high-dimensional data where relationships are more straightforward, such as text classification.

Design and Analysis of Algorithms (DAA)

Question 1: Explain the Greedy approach with an example. How does it apply to the Fractional Knapsack problem?

Answer:
Greedy algorithms make locally optimal choices at each step, aiming for a globally optimal solution. In the **Fractional Knapsack** problem, items are sorted by value-to-weight ratio, and fractions of items are added until the weight limit is reached, maximizing total value.

Additional Question:

1. **Provide examples where the Greedy algorithm does not yield an optimal solution.**
   Answer:
   In the **0/1 Knapsack problem**, a Greedy approach may not yield the optimal solution since it cannot take fractional items, potentially overlooking combinations with higher values. Similarly, in the **Shortest Path problem** with negative weights, Greedy can fail, as it does not consider future costs.

Question 2: Describe the Dynamic Programming approach for solving the 0/1 Knapsack problem. How does it differ from the Greedy approach?

Answer:
Dynamic Programming (DP) divides the problem into subproblems, storing intermediate results.

In the **0/1 Knapsack problem**, a DP table calculates maximum values for each weight and item subset. Unlike Greedy, DP considers all combinations, guaranteeing an optimal solution.

Additional Question:
2. **Explain the time and space complexity of the 0/1 Knapsack problem using Dynamic Programming.**

Answer:
The 0/1 Knapsack DP approach has a **time complexity of O(n * W)** and **space complexity of O(n * W)**, where nnn is the number of items and WWW is the weight limit. Each item is considered at each weight level, filling the DP table iteratively.

Question 3: Explain the Bellman-Ford algorithm for finding the shortest path. How does it handle negative weights?

Answer:
The Bellman-Ford algorithm iteratively relaxes edges up to ∣V∣−1|V| - 1∣V∣−1 times, where VVV is the number of vertices. It can detect negative weight cycles; if a shorter path is found after ∣V∣−1|V| - 1∣V∣−1 iterations, a cycle exists. This makes it suitable for graphs with negative weights.

Additional Question:
3. **How does the Bellman-Ford algorithm compare to Dijkstra's algorithm in terms of complexity and applicability?**

Answer:
Bellman-Ford has **O(V * E)** complexity, slower than Dijkstra's **O((V + E) log V)** but handles negative weights, making it suitable for networks with potential losses or decreases in costs, unlike Dijkstra's which assumes non-negative edges.

Question 4: What is Backtracking? Describe its use in solving the N-Queens problem.

Answer:
Backtracking incrementally builds solutions, abandoning them if they prove infeasible. In **N-Queens**, it places queens row by row, backtracking when a queen placement causes conflict, ultimately finding all valid arrangements or a single solution as needed.

Additional Question:
4. **Describe the role of constraint checking in optimizing backtracking algorithms.**

Answer:
Constraint checking reduces search space by validating moves before full execution. In N-Queens, checking row and diagonal conflicts before placing a queen prevents unnecessary recursive calls, thus optimizing performance.

Question 5: How does the Branch and Bound approach solve the Travelling Salesman Problem?

Answer:
Branch and Bound explores solution branches, calculating lower bounds for partial paths. If a path's cost exceeds the current best, it is pruned. This minimizes the number of explored paths, enabling a more efficient search for the minimum-cost route.

Additional Question:
5. **Compare Branch and Bound with Dynamic Programming for solving TSP.**

Answer:
Branch and Bound is generally faster in practice, using bounds to prune solutions, while Dynamic Programming (e.g., Held-Karp) guarantees optimality but can be more computationally intensive with **$O(n^2 * 2^n)$** time complexity.

Question 6: Explain the concept of time complexity and how it can be verified for algorithms like the Bellman-Ford or the 0/1 Knapsack.

Answer:
Time complexity describes an algorithm's growth rate as input size increases, commonly in Big-O notation. **Bellman-Ford** is **O(V * E)** due to repeated edge relaxations. **0/1 Knapsack** is **O(n * W)** because of DP table filling. Complexity is verified through theoretical analysis and empirical testing on varying input sizes.

Additional Question:
6. **Describe the difference between Big-O, Big-Theta, and Big-Omega notations with examples.**

Answer:

- **Big-O (O)**: Describes the upper bound of complexity, e.g., sorting algorithms like Bubble Sort are **$O(n^2)$**.
- **Big-Theta (Θ)**: Describes exact complexity, e.g., Merge Sort has **Θ(n log n)** as it always operates in that time.

- **Big-Omega (Ω)**: Describes the lower bound, e.g., searching in an unsorted array has **Ω(n)** as the minimum complexity.

Pandas: For data manipulation and analysis. It handles data loading, cleaning, and organizing with DataFrames.
Scikit-Learn (sklearn): For machine learning. It provides tools for model training, evaluation, and preprocessing.
NumPy: For numerical computing. It offers fast array operations and serves as the foundation for many data-related libraries.
Seaborn: For statistical data visualization. It creates aesthetically pleasing and complex statistical plots.
Matplotlib: For general plotting. It's a versatile tool for creating 2D charts, serving as the base for many visualization libraries.