DAA Practical 1

Fibonacci numbers are a series of numbers in which each number is the sum of the two preceding ones, starting from 0 and 1. So, the sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,……

There are two main ways to calculate Fibonacci numbers: recursively and non-recursively.

Recursive Fibonacci Numbers

A recursive function is a function that calls itself. In the case of Fibonacci numbers, the recursive function is defined as follows:

```
fib(n) = fib(n - 1) + fib(n - 2)
```

This function says that the nth Fibonacci number is equal to the sum of the (n-1)th and (n-2)th Fibonacci numbers. For example, fib(3) = fib(2) + fib(1) = 1 + 1 = 2.

This recursive definition is a very concise way to define the Fibonacci sequence, but it is also not very efficient. This is because the function calls itself repeatedly for every Fibonacci number that is calculated. This can lead to a lot of overhead, especially for large values of n.

Non-Recursive Fibonacci Numbers

A non-recursive function is a function that does not call itself. There are a few different non-recursive formulas for calculating Fibonacci numbers, but one of the most common is:

```
F_n = (φ^n - ψ^n) / √5
```

```

where $\varphi = (1 + \sqrt{5}) / 2$ and $\psi = (1 - \sqrt{5}) / 2$. This formula is based on the Binet formula, which is a mathematical formula for the Fibonacci sequence.

This non-recursive formula is much more efficient than the recursive formula, especially for large values of n. This is because the formula does not require any recursion, so it avoids the overhead of repeated function calls.

**Comparison of Recursive and Non-Recursive Fibonacci Numbers**

Here is a table that compares the recursive and non-recursive methods for calculating Fibonacci numbers:

| Method | Advantages | Disadvantages |
|---|---|---|
| Recursive | Easy to understand | Inefficient for large values of n |
| Non-Recursive | Efficient for large values of n | More complex to understand |

**Time Complexity:**

**Recursive Fibonacci:**

The time complexity of the recursive Fibonacci function is exponential, specifically O(2^n). This is because, for each call to the Fibonacci function, two more calls are made, leading to an exponentially growing number of recursive calls.

**Non-Recursive Fibonacci:**

The time complexity of the non-recursive Fibonacci function is O(n). This is because it iterates through the loop n times, and each iteration involves constant time operations.
```

DAA Practical 2

Greedy Method:

The greedy method is a problem-solving strategy that makes locally optimal choices at each stage with the hope of finding a global optimum. It is called "greedy" because, at each step, it makes the best possible decision without considering the consequences in the future. Greedy algorithms are often used for optimization problems where a sequence of choices must be made, and at each step, the best immediate choice is selected.

One classic example of the greedy method is the Huffman Encoding algorithm, which is used for lossless data compression.

Huffman Encoding:

Huffman Encoding is a technique to compress data without losing information. It is widely used in file compression algorithms like ZIP.

The basic idea is to assign variable-length codes to input characters, with shorter codes assigned to more frequent characters. This way, the more common characters are represented by shorter bit sequences, leading to overall compression.

Huffman Encoding step by step:

1. Character Frequency Calculation: Count the frequency of each character in the input data. This step is essential for determining the priority of characters during the encoding process.

2. Priority Queue (Min-Heap) Creation: Create a priority queue (or min-heap) based on the character frequencies. Each node in the queue represents a character and its frequency.

3. Building the Huffman Tree: While there is more than one node in the priority queue:

Remove the two nodes with the lowest frequencies from the priority queue.

Create a new internal node with a frequency equal to the sum of the frequencies of the two nodes.

Insert the new internal node back into the priority queue.

The final remaining node in the priority queue is the root of the Huffman tree.

4. Assigning Huffman Codes: Traverse the Huffman tree from the root to each leaf.

Assign '0' for a left branch and '1' for a right branch.

The codes for each character are the binary digits along the path from the root to the corresponding leaf.

5. Generating Huffman Codes: After assigning codes to each character, create a table that maps each character to its corresponding Huffman code.

6. Encoding: Replace each character in the original data with its Huffman code.

The encoded data is a binary sequence of '0s' and '1s' that represents the original data in a compressed form.

DAA Practical 3

Greedy Method: A greedy algorithm is an algorithm that makes the best choice at each step, without regard for the overall outcome. This can sometimes lead to suboptimal results, but it can also be a very effective way to solve problems.

For example, consider the problem of finding the shortest path from one city to another. A greedy algorithm would choose the next city to visit that is closest to the current city, without regard for the overall distance of the path. This algorithm would not always find the shortest path, but it would often find a path that is close to the shortest path.

Greedy algorithms are often used for problems that are difficult or impossible to solve exactly. They are also often used for problems where an approximate solution is good enough.

Fractional Knapsack Problem: The fractional knapsack problem is a classic optimization problem that asks for the maximum value that can be fit into a knapsack of a given capacity. The items in the knapsack can be divided into fractions, so that it is possible to fit more items into the knapsack than would be possible if the items had to be whole.

1. Define the problem:

* You have a knapsack with a limited capacity (W).

* You have a set of items, each with a weight (w) and a value (v).

* Your goal is to fit as much value as possible into the knapsack without exceeding its weight limit.

2. Sort the items by their value-to-weight ratio (v/w): This will ensure that you are always adding the items with the highest value per pound to the knapsack first.

3. Initialize a variable to track the total weight of the items in the knapsack (current_weight): Initialize another variable to track the total value of the items in the knapsack (current_value).

4. Iterate through the sorted items:

For each item, check if adding it to the knapsack would exceed the weight limit.

* If it would not exceed the weight limit, add the item to the knapsack and update the current_weight and current_value variables accordingly.

* If it would exceed the weight limit, but there is still some capacity left, add as much of the item as possible to the knapsack without exceeding the weight limit. Update the current_weight and current_value variables accordingly.

5. Once you have iterated through all of the items, the current_value variable will contain the maximum value that can be fit into the knapsack

DAA Practical 4

Dynamic Programming: Dynamic programming is an algorithmic technique for solving problems by breaking them down into smaller subproblems and storing the solutions to these subproblems to avoid recomputing them. This technique is particularly useful for problems that have overlapping subproblems, which are subproblems that are shared by multiple larger problems.

Dynamic programming algorithms typically consist of two phases:

Memoization: This phase involves storing the solutions to subproblems in a table. This table is called a memoization table.

Tabulation: This phase involves using the memoization table to solve the larger problem. The tabulation phase typically proceeds in a bottom-up manner, starting from the smallest subproblems and working up to the larger subproblems.

Dynamic programming is a powerful technique that can be used to solve a wide variety of problems, including optimization problems, scheduling problems, and search problems.

0/1 Knapsack Problem: The 0/1 Knapsack problem is a classic optimization problem that asks for the maximum value that can be put into a knapsack of a given capacity. The items in the knapsack can either be included or not included, so that each item is either in the knapsack or not in the knapsack.

Problem Statement:

You have a knapsack with a maximum weight capacity, and you are given a set of items, each with a specific weight and value. The goal is to fill the knapsack with items in a way that maximizes the total value while staying within the weight capacity.

Step 1: Understand the Input

- You're given a list of items, each with its weight and value.

- You have a knapsack that can carry a maximum weight.

Step 2: Identify Constraints

- You can either include an entire item in the knapsack or exclude it entirely (0/1 choice).

- You cannot take a fraction of an item.

Step 3: Define the Problem Mathematically

- Let n be the number of items.

- Let W be the maximum weight the knapsack can carry.

- Let w_i and v_i represent the weight and value of the i-th item.

Step 4: Create a Table

- Create a table with rows representing items and columns representing the weight capacity (from 0 to W).

Step 5: Fill in the Table

- Start filling in the table from the top-left corner.

- For each cell, consider whether including the current item in the knapsack at its given weight would increase the total value.

Step 6: Make Decisions

- At each cell, you decide whether to include the current item or not based on its value and weight.

- If the item fits in the remaining capacity, compare the value of including it with the value of excluding it.

Step 7: Optimal Solution

- The bottom-right cell of the table represents the maximum value achievable with the given items and knapsack capacity.

Example:

Imagine Packing a Magic Backpack:

1. You have a magical backpack:

- Imagine you have a special backpack that can hold a maximum weight.

2. You're going on an adventure:

- You're going on an exciting adventure, and you want to bring some items with you.

3. Every item has weight and value:

- Each item you want to bring has a weight (like how heavy it is) and a value (how much you like it).

4. Your goal is to maximize fun:

- Your goal is to bring items that make your adventure as fun as possible. But here's the catch - your backpack has a limit on how much it can carry!

5. Decision time at each step:

- As you decide which items to bring, you have to make choices at each step. Do you bring this toy or skip it?

6. Table to help you decide:

- Imagine you have a table to help you decide. On one side of the table, you have all the items, and on the other side, you have different weights your backpack can carry.

7. Fill in the table:

- You start filling in the table. For each item, you think about whether it's better to bring it or leave it behind based on its weight and how much you like it.

8. Making choices:

- You decide item by item. For example, if bringing a toy doesn't make your backpack too heavy, and it adds a lot of fun, you'll bring it. But if it's too heavy, you might leave it behind.

9. Find the best combination:

- By filling in the table and making these choices, you're trying to find the best combination of toys that fit in your magical backpack and give you the most fun for your adventure.

Knapsack Problem: The knapsack problem is a more general problem than the 0/1 knapsack problem.

In the knapsack problem, items can be included in fractions, so that it is possible to fit more items into the knapsack than would be possible if the items had to be whole.

The knapsack problem is typically solved using a greedy algorithm.

0/1 Knapsack Problem: The 0/1 knapsack problem is a special case of the knapsack problem in which items can either be included or not included, so that each item is either in the knapsack or not in the knapsack.

The 0/1 knapsack problem is typically solved using dynamic programming.

DAA Practical 5

Backtracking: Backtracking is a search algorithm that solves problems by trying all possible solutions and backtracking when it reaches a dead end. It is a problem-solving technique that explores all possible paths until a solution is found or all paths have been explored.

The backtracking algorithm is based on the idea of recursively trying different solutions. At each step, the algorithm tries one possible solution. If the solution leads to a dead end, the algorithm backtracks to the previous step and tries a different solution.

Backtracking is a powerful technique that can be used to solve a wide variety of problems, including optimization problems, scheduling problems, and search problems.

N-Queens Problem: The N-Queens problem is a classic optimization problem that asks for the maximum number of queens that can be placed on a chessboard such that no two queens attack each other. A queen attacks another queen if they are in the same row, column, or diagonal.

The N-Queens problem can be solved using backtracking. The backtracking algorithm for the N-Queens problem is as follows:

1. Initialize a chessboard with all squares empty.
2. Place a queen on the first row of the chessboard.
3. Recursively place queens on the remaining rows of the chessboard.
    - For each row, try all possible positions for the queen.

- If a position is not valid, backtrack to the previous row and try a different position.
- If a position is valid, place the queen on the row and move on to the next row.

4. If all queens have been placed on the chessboard, then a solution has been found.

5. If all possible positions have been tried for a row without finding a valid position, then backtrack to the previous row and try a different position.

ML Practical 1

Identifying Outliers: Outliers are data points that significantly differ from the rest of the data. To identify outliers:

1. Visual Inspection: Plot the data using a box plot, scatter plot, or histogram to visually identify points that stand out.

2. Z-Score: Calculate the z-score for each data point. Z-score measures how far a particular data point is from the mean in terms of standard deviations. Points with high absolute z-scores may be considered outliers.

3. IQR (Interquartile Range): Use the IQR to identify outliers. Calculate the IQR (difference between the third quartile and the first quartile) and consider points outside a certain range as outliers.

Correlation: Correlation is a measure of the strength of the linear relationship between two variables. It can range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

1. Correlation Coefficient: Use Pearson's correlation coefficient for linear relationships. It ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no correlation.

2. Scatter Plot: Plot a scatter plot to visually assess the relationship between two variables. If points form a clear pattern, there's likely a correlation.

Linear Regression: Linear regression is a statistical method to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship and is represented by the equation \(y = mx + b\), where \(y\) is the dependent variable, \(x\) is the independent variable, \(m\) is the slope, and \(b\) is the intercept.

To perform linear regression:

1. Fit a Line: Find the line that best fits the data. This is done by minimizing the sum of the squared differences between the observed and predicted values.

2. Coefficients: The slope (\(m\)) and intercept (\(b\)) of the line are the coefficients of the regression equation.

Random Forest Regression Model: Random Forest Regression is an ensemble learning method that uses multiple decision trees to make predictions. Here's how it works:

1. Build Multiple Trees: Construct a multitude of decision trees during training.

2. Predictions: When making predictions, each tree gives an output, and the final prediction is the average (for regression) or mode (for classification) of the individual tree predictions.

3. Reduce Overfitting: Random Forest helps reduce overfitting compared to a single decision tree by introducing randomness in the tree-building process.

4. Feature Importance: Random Forest can provide a measure of feature importance, indicating which features contribute more to the model's predictive performance.

Random Forests are versatile, robust, and effective for a variety of tasks, making them popular in machine learning applications. They are particularly useful for regression tasks when you want accurate and robust predictions.

ML Practical 2

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a supervised learning algorithm that is used for classification and regression. It is a non-parametric algorithm, which means that it does not make any assumptions about the distribution of the data.

The KNN algorithm works by classifying new data points based on the similarity of the new data points to the existing data points. The similarity between two data points is typically measured using a distance metric, such as the Euclidean distance.

The KNN algorithm has a number of advantages, including:

* It is easy to understand and implement.

* It is non-parametric, so it does not make any assumptions about the distribution of the data.

* It can be used for both classification and regression.

* It can be used for high-dimensional data.

The KNN algorithm also has a number of disadvantages, including:

* It can be slow for large datasets.

* It is sensitive to outliers.

* It can be difficult to choose the appropriate value of k.

Support Vector Machines (SVM)

Support Vector Machines (SVM) are a supervised learning algorithm that is used for classification. They are a type of large margin classifier, which means that they find a hyperplane that maximizes the margin between the two classes.

The SVM algorithm works by finding the hyperplane that maximizes the margin between the two classes. The margin is the distance between the hyperplane and the closest data points from each class.

The SVM algorithm has a number of advantages, including:

* They are very effective for high-dimensional data.
* They are robust to outliers.
* They are not sensitive to the choice of features.

The SVM algorithm also has a number of disadvantages, including:

* They can be slow to train.
* They are not as easy to understand and implement as other algorithms.
* They can be sensitive to noise in the data.

ML Practical 3

AI (Artificial Intelligence): Artificial intelligence (AI) is a branch of computer science that deals with the creation of intelligent agents, which are systems that can reason, learn, and act autonomously. AI research has been highly successful in developing effective techniques for solving a wide range of problems, from game playing to medical diagnosis.

Keras: Keras is a high-level neural networks library written in Python. It is part of the TensorFlow suite of tools and libraries for machine learning and deep learning. Keras is known for its ease of use and modularity, making it a popular choice for developing deep learning models.

TensorFlow: TensorFlow is an open-source software library for machine learning and deep learning. It was originally developed by Google and is now maintained by a large community of contributors. TensorFlow is a powerful and versatile tool for developing deep learning models, and it is widely used in both research and industry.

Normalization: Normalization is the process of transforming data into a standard form. This can be useful for a variety of reasons, such as making it easier to compare data from different sources or to improve the performance of machine learning algorithms. There are a variety of different normalization techniques, but some of the most common include:

- Min-max normalization: This technique scales the data to a specific range, such as 0 to 1 or -1 to 1.
- Z-score normalization: This technique scales the data to have a mean of 0 and a standard deviation of 1.
- Decimal scaling: This technique scales the data by dividing each value by a power of 10.

Confusion Matrix: A confusion matrix is a table that is used to summarize the performance of a classification algorithm. It shows the number of correct and incorrect classifications that the algorithm made for each class. The confusion matrix is typically divided into four quadrants:

- True positives: The number of cases where the algorithm correctly classified an instance as positive.
- False positives: The number of cases where the algorithm incorrectly classified an instance as positive.
- True negatives: The number of cases where the algorithm correctly classified an instance as negative.
- False negatives: The number of cases where the algorithm incorrectly classified an instance as negative.

Neural Network-based Classifier: A neural network-based classifier is a type of machine learning classifier that uses a neural network to learn how to classify data. Neural networks are inspired by the human brain, and they are able to learn complex patterns from data. Neural network-based classifiers are often very effective, and they are used in a variety of applications, such as image recognition, natural language processing, and fraud detection.

ML Practical 4

K-Nearest Neighbors (KNN) Algorithm

The K-Nearest Neighbors (KNN) algorithm is a simple yet effective supervised machine learning algorithm used for both classification and regression tasks. It works by classifying new data points based on the similarity of the new data points to the existing data points in the training set.

Step-by-Step Explanation of KNN Algorithm:

1. Choose the number of neighbors (k): The value of k determines the number of nearest neighbors considered for classification or regression.
2. Calculate the distance between the new data point and each training data point: Various distance metrics can be used, such as Euclidean distance, Manhattan distance, or Minkowski distance.
3. Sort the training data points based on their calculated distances: Arrange the training data points in ascending order of their distances from the new data point.
4. Select the k nearest neighbors: Identify the k training data points with the smallest distances to the new data point. These k nearest neighbors represent the most similar data points to the new data point.
5. Classify or predict the new data point:

a. Classification: For classification tasks, determine the most common class among the k nearest neighbors. Assign the new data point to the class that appears most frequently among its k nearest neighbors.
b. Regression: For regression tasks, calculate the average value of the target variable among the k nearest neighbors. Assign the new data point the predicted value based on the average of its k nearest neighbors' target values.

Random Forest Classification Model: Random Forest Classification is an ensemble learning method that combines multiple decision trees to improve the overall classification accuracy. It utilizes the concept of bagging and random subspace sampling to build a collection of diverse decision trees.

Key Steps in Random Forest Classification:

1. Bootstrap sampling: Create multiple subsets of the training data by randomly sampling with replacement. These subsets serve as training sets for individual decision trees.
2. Random subspace sampling: At each tree node, randomly select a subset of features to consider for splitting. This reduces the correlation between trees and prevents overfitting.
3. Build decision trees: Train individual decision trees using their respective bootstrap samples and random subspace features.
4. Classification: For a new data point, pass it through each of the decision trees in the forest. Obtain the classification predictions from each tree.
5. Aggregate predictions: Combine the predictions from the individual trees using majority voting or other ensemble methods. The final classification is determined by the most frequent class among the trees' predictions.

Performance Metrics for Classification and Regression

1. **Confusion Matrix:** A confusion matrix is a table that summarizes the performance of a classification algorithm. It compares the predicted labels with the actual labels, providing a clear overview of the classification results.
2. **Accuracy Score:** Accuracy score is the most common metric for evaluating classification performance. It measures the proportion of correctly classified data points.

3. **Mean Squared Error (MSE):** MSE is a common metric for evaluating regression performance. It measures the average squared difference between the predicted values and the actual values.
4. **R-squared Score (R2):** R2 is another metric for evaluating regression performance. It measures the proportion of the variance in the predicted values explained by the model.
5. **ROC AUC Score (Area Under the Receiver Operating Characteristic Curve):** ROC AUC score is a metric for evaluating binary classification models. It measures the ability of a model to distinguish between positive and negative classes across all possible thresholds.
6. **ROC Curve (Receiver Operating Characteristic Curve):** ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. It provides a visual representation of the model's performance across different classification thresholds.

ML Practical 5

K-means clustering

K-means clustering is an unsupervised machine learning algorithm that partitions a set of data points into a predefined number of clusters (k). The algorithm aims to find the cluster centers that minimize the within-cluster variance, which is the average squared distance between each data point in a cluster and its cluster center.

Unsupervised and supervised learning

Unsupervised learning and supervised learning are two main approaches to machine learning. In unsupervised learning, the algorithm is given a set of data points without any labels or target variables. The goal of the algorithm is to discover patterns or structure in the data. In supervised learning, the algorithm is given a set of data points that have been labeled with their corresponding target variables. The goal of the algorithm is to learn a mapping from the input data points to the output target variables.

Clustering
Clustering is a data analysis technique that groups data points together based on their similarity. The goal of clustering is to identify groups of data points that are similar to each other and different from data points in other groups. Clustering is often used in unsupervised learning tasks, such as customer segmentation, image recognition, and social network analysis.

Elbow method
The elbow method is a graphical method for determining the optimal number of clusters in k-means clustering. It works by plotting the within-cluster sum of squares (WCSS) as a function of the number of clusters k. The elbow point is the point where the WCSS starts to decrease rapidly, indicating that adding more clusters does not significantly improve the clustering.

Gradient descent algorithm
The gradient descent algorithm is an optimization algorithm used to find the minimum of a function. It works by iteratively moving in the direction of the steepest descent, which is the direction in which the function decreases the fastest. The gradient descent algorithm is often used to train neural networks, as it is a powerful and efficient method for finding the optimal weights of the network.