K-Nearest Neighbor Algorithm (KNN)

The K-Nearest Neighbor (KNN) algorithm is a simple yet powerful supervised machine learning method used for classification and regression tasks. It predicts the label or value of a new data point based on the labels or values of its k nearest neighbors in the training dataset. Below, we delve into the requested aspects of KNN:

---

i. Maths Behind KNN

The mathematical foundation of KNN revolves around the concept of proximity and majority voting for classification or averaging for regression.

1. **Representation of Data**:
    - Each data point is represented as a vector in an n-dimensional feature space.
    - For example, if a dataset has two features (e.g., height and weight), each data point is represented as (x_1, x_2).
2. **Distance Calculation**:
    - To find the nearest neighbors, KNN computes the distance between the query point and all other points in the training dataset using a distance metric (e.g., Euclidean distance).
    - The formula for Euclidean distance between two points $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ is:

    $$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$
3. **Classification**:
    - After identifying the k nearest neighbors, KNN assigns the class label based on majority voting among these neighbors.
    - For example, if $k = 3$ and two of the three nearest neighbors belong to Class A while one belongs to Class B, the query point is classified as Class A.
4. **Regression**:
    - In regression tasks, KNN predicts the value of a query point by averaging the values of its k nearest neighbors:

    $$\hat{y} = \frac{1}{k}\sum_{i=1}^{k} y_i$$

    where y_i are the target values of the k nearest neighbors.
5. **Weighted KNN**:
    - To improve accuracy, weights can be assigned to neighbors based on their distance from the query point. Closer neighbors are given more

weight:

$$w_i = \frac{1}{d(p, q_i)}$$

The prediction can then be computed using weighted averages or weighted votes.

---

ii. Distance Calculations in KNN

The choice of distance metric significantly impacts KNN's performance. Common metrics include:

1. **Euclidean Distance**:
   - Measures straight-line distance between two points.
   - Formula:

     $$d_{\text{Euclidean}}(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

   - Example: In a 2D space, the distance between points $(3, 4)$ and $(7, 1)$ is:

     $$d = \sqrt{(7-3)^2 + (1-4)^2} = 5$$

iii. Elbow Method

The Elbow Method is typically used to determine an optimal value for k, ensuring a balance between underfitting and overfitting.

Steps in Elbow Method:

1. **Error Rate Calculation**:
   - For different values of k, calculate the error rate (e.g., misclassification rate for classification tasks or mean squared error for regression tasks).
2. **Plotting**:
   - Plot error rates against different values of k.
3. **Identifying the Elbow Point**:
   - Look for an "elbow" in the graph where the error rate sharply decreases initially but levels off afterward.
   - The elbow point represents an optimal trade-off between model complexity and accuracy.

Example: Suppose we test k values from 1 to 10 and calculate error rates as follows:

k	Error Rate
1	0.25
2	0.20
3	0.15
4	0.12
5	0.11
6	0.10

When plotted, we observe that after $k = 4$, further increases in k yield diminishing returns in reducing error rates. Hence, $k = 4$ is chosen as optimal.

Application in KNN: The Elbow Method helps avoid overfitting (low k) or underfitting (high k) by selecting an appropriate number of neighbors that balances bias and variance effectively.

Decision Tree: Detailed Explanations

A decision tree is a supervised learning algorithm used for classification and regression tasks. It works by splitting the dataset into subsets based on specific features, forming a tree-like structure where decisions are made at nodes, and outcomes are represented at leaf nodes. Below is a detailed explanation of the requested topics:

---

i. Geometric Intuition of Decision Tree

A decision tree can be geometrically understood as a series of axis-aligned splits in the feature space:

- **Root Node**: Represents the entire dataset.
- **Internal Nodes**: Represent decisions based on feature values (e.g., "Is Age > 30?").
- **Leaf Nodes**: Represent the final outcomes or predictions.

Each split divides the feature space into regions. For example, in a 2D space with features F_1 (e.g., income) and F_2 (e.g., age), the splits create rectangular partitions. These partitions correspond to decision boundaries that classify data points into different categories. In higher dimensions, these boundaries become hyperplanes, dividing the space into hypercuboids.

Example: If a decision tree classifies whether someone buys a product based on income and age, the first split might divide people by income level (e.g., Income $> \$50,000$), creating two regions. Subsequent splits further refine these regions based on age or other features.

---

ii. Sample Decision Tree

Consider a simple classification problem: predicting whether a person will purchase a product based on their age and income.

Dataset:

Age	Income	Purchase
25	Low	No
35	High	Yes
45	Medium	Yes
55	High	Yes
23	Medium	No

Decision Tree:

1. **Root Node**: Split on "Income".
   - If Income = High → Predict "Yes".
   - If Income = Low → Predict "No".
   - If Income = Medium → Split further on "Age".
2. **Internal Node (Medium Income)**: Split on "Age".
   - If Age > 30 → Predict "Yes".
   - If Age   30 → Predict "No".

This tree can be visualized as:

```
          [Income?]
         /    |    \
   Low      Medium  High
  No       [Age?]    Yes
          /     \
     30        >30
    No           Yes
```

---

iii. Building a Decision Tree: Entropy

Entropy measures the impurity or disorder in a dataset. It helps determine how well a feature separates data into classes.

Formula:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where: - S is the dataset. - c is the number of classes. - p_i is the proportion of instances belonging to class i.

Intuition:

- If all data points belong to one class (pure node), entropy is 0.
- If data points are evenly distributed across classes, entropy is maximum (e.g., for binary classification, entropy = 1 when $p_+ = p_- = 0.5$).

Example Calculation: For a dataset with 8 samples (5 “Yes”, 3 “No”):

$$H(S) = -\left(\frac{5}{8}\log_2\frac{5}{8} + \frac{3}{8}\log_2\frac{3}{8}\right) = 0.954$$

Entropy helps identify which feature minimizes disorder after splitting.

---

iv. Building a Decision Tree: Information Gain

Information Gain (IG) measures the reduction in entropy after splitting a dataset on an attribute.

Formula:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Where: - $H(S)$: Entropy of the parent node. - S_v: Subset of data where attribute A takes value v. - $H(S_v)$: Entropy of subset S_v.

Intuition: A feature with higher IG provides better separation of classes and is chosen for splitting.

Example Calculation: Using the dataset from above, if we split on “Income”: 1. Calculate entropy for each subset (Low, Medium, High). 2. Compute weighted average entropy. 3. Subtract from parent node entropy to get IG.

If IG for “Income” is higher than other features, it becomes the root node.

---

v. Building a Decision Tree: Gini Impurity

Gini Impurity measures how often a randomly chosen element would be incorrectly classified if it were labeled randomly according to class distribution.

Formula:

$$G(t) = 1 - \sum_{i=1}^{c} p_i^2$$

Where: - t: Node. - c: Number of classes. - p_i: Proportion of instances in class i.

Intuition:

- Gini Impurity ranges from 0 (pure node) to 0.5 (maximum impurity for binary classification).
- Lower Gini Impurity indicates better splits.

Example Calculation: For a node with two classes ("Yes" and "No"): - Proportions: $p_{Yes} = 0.6, p_{No} = 0.4$.

$$G(t) = 1 - (0.6^2 + 0.4^2) = 0.48$$

The attribute with the lowest weighted Gini Impurity after splitting is selected.

Comparison Between Entropy and Gini Impurity

Metric	Formula	Range	Characteristics
Entropy	$H(S) = -\sum p_i \log_2(p_i)$	[0,1]	Computationally intensive but interpretable.
Gini Impurity	$G(t) = 1 - \sum p_i^2$	[0,0.5]	Faster to compute; slightly less balanced trees.

Both metrics are widely used in decision trees; Gini Impurity is preferred for computational efficiency.