



CAP 4630 – KNN

Instructor: Aakash Kumar

University of Central Florida



Introduction



- K-Nearest Neighbors (KNN) is a simple and effective algorithm that can be used for both classification and regression tasks in machine learning.
- As a supervised learning method, KNN is widely applied in pattern recognition, data mining, and intrusion detection due to its ease of implementation and interpretability.



Introduction



- KNN is a **non-parametric** algorithm, meaning it doesn't make any assumptions about the underlying data distribution, making it versatile in many real-world scenarios.
- This is different from algorithms like **Gaussian Mixture Models (GMM)**, which assume a specific distribution.
- The algorithm works by using **training data** to classify new data points based on how similar they are to their **nearest neighbors**, measured by distance.



Introduction



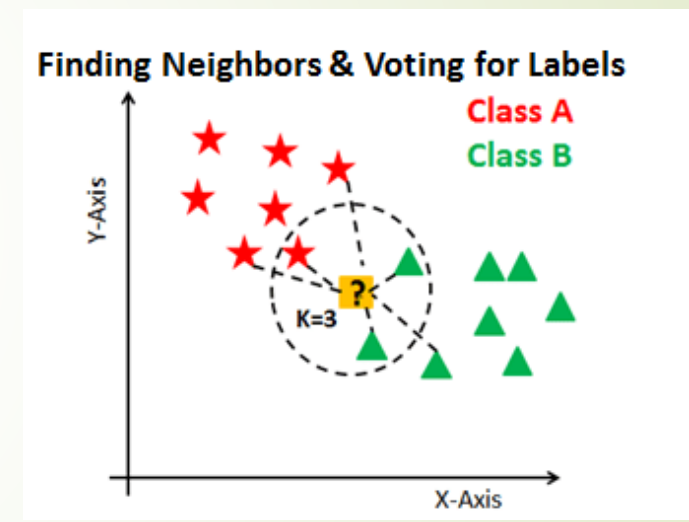
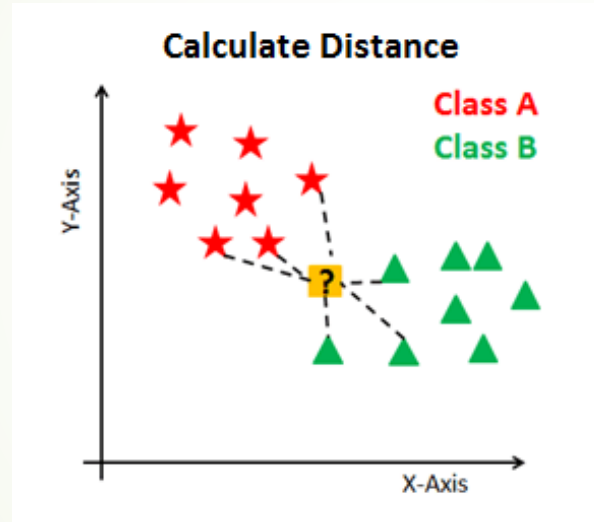
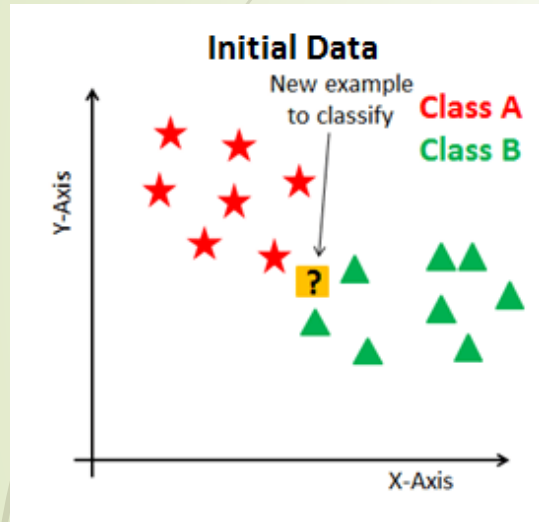
- **K-Nearest Neighbors (KNN)** makes predictions by finding the closest K data points (neighbors) to a new example and using them to classify or predict the outcome.
- **Data pre-processing** is crucial because distance-based algorithms are sensitive to how data is scaled and represented.
- It doesn't create a model using a mathematical equation but relies entirely on the distance from neighbors for its decisions.



Parameter K

- **Choosing K:** The number of neighbors (K) must be carefully selected:
 - **Small K** can lead to a model that's overly sensitive to noise (overfitting).
 - **Large K** can result in a model that's too generalized (underfitting).
- **Efficiency:** KNN is computationally intensive, making it more suitable for smaller datasets, as calculating distances for every data point can be slow on larger datasets.

How it work?



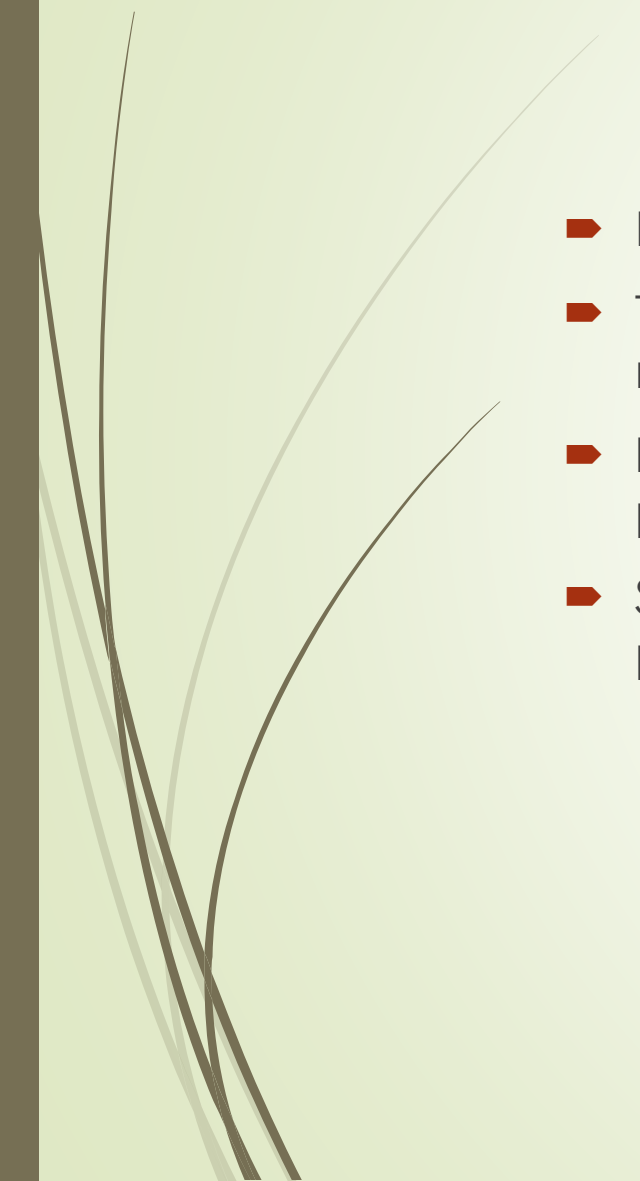


KNN Algorithm

- K-Nearest Neighbors (KNN) is a versatile and widely used machine learning algorithm known for:
 - Simplicity and ease of implementation.
 - Flexibility in handling numerical and categorical data.
- Non-parametric: Does not make assumptions about data distribution.
- Useful for both classification and regression tasks.



KNN Algorithm – Key Assumptions

- KNN assumes that similar data points are located near each other.
 - The algorithm compares new data to the stored dataset and assigns the new point to the class most similar to it.
 - It stores all available data and performs classification when a new data point appears.
 - Sensitive to outliers: Outliers can skew distance calculations and affect predictions.
- 



How KNN Works

- KNN identifies the **K-nearest neighbors** based on a distance metric (e.g., **Euclidean distance**)
- The class or value of the data point is then determined by
 - **Majority vote** (classification).
 - **Average** (regression).
- This method adapts to the **local structure** of data, making it powerful in many scenarios.



Lazy Learning in KNN

- KNN is a **lazy learner**
 - No explicit training phase.
 - It **stores the dataset** and waits until a new data point is introduced to perform computation.
- At the time of classification, KNN finds the closest neighbors and assigns the new data point to the most appropriate class.
- This can make KNN slower on large datasets since it computes distances on the fly.



Distance Metrics



Distance Metrics in KNN

- KNN identifies the nearest points (neighbors) to make predictions based on similarity.
- To determine these closest neighbors, we need a way to measure distance between points.
- For this purpose, we use several common distance metrics:
 - Euclidean Distance (most common for continuous data)
 - Manhattan Distance (useful for grid-like data)
 - Minkowski Distance (generalization of Euclidean and Manhattan)
 - Hamming Distance (for categorical data)

Euclidean Distance in KNN

- Euclidean Distance is the straight-line distance between two points in space (or the shortest path).
- It is the most commonly used distance metric in KNN, especially for continuous data.
- Visualization: Imagine the distance as the length of the line segment connecting two points.
- The formula for Euclidean distance between two points x and X_i in a d -dimensional space is:

$$distance(x, X_i) = \sqrt{\sum_{j=1}^d (x_j - X_{ij})^2}$$

Example:

- ▶ Let's say you're using KNN to predict a label based on two features: height and weight. For the query point x , you have:
 - ▶ (height of query point) = 170 cm
 - ▶ (weight of query point) = 70 kg
- ▶ Now, for one of the data points X_i , you have:
 - ▶ (height of i -th data point) = 165 cm
 - ▶ (weight of i -th data point) = 65 kg

$$distance(x, X_i) = \sqrt{(170 - 165)^2 + (70 - 65)^2} = \sqrt{5^2 + 5^2} = \sqrt{25 + 25} = \sqrt{50} \approx 7.07$$

Manhattan Distance in KNN

- Manhattan Distance (also known as L1 norm or city block distance) is used when we are interested in the total path traveled rather than the straight-line distance.
- It calculates the distance by summing the absolute differences between the coordinates of the points in each dimension.
- The formula for Manhattan Distance between two points x and y is:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- **Key insight:** Imagine walking through a city grid, where you have to move along streets (horizontal or vertical), not diagonally.

Minkowski Distance in KNN

- Minkowski Distance is a generalized distance metric that includes both Euclidean and Manhattan distances as special cases.
- It is controlled by a parameter p , which adjusts how distance is measured:
 - When $p=2$, Minkowski distance becomes Euclidean distance.
 - When $p=1$, Minkowski distance simplifies to Manhattan distance.
- The formula for Minkowski distance between two points x and y is:

$$d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

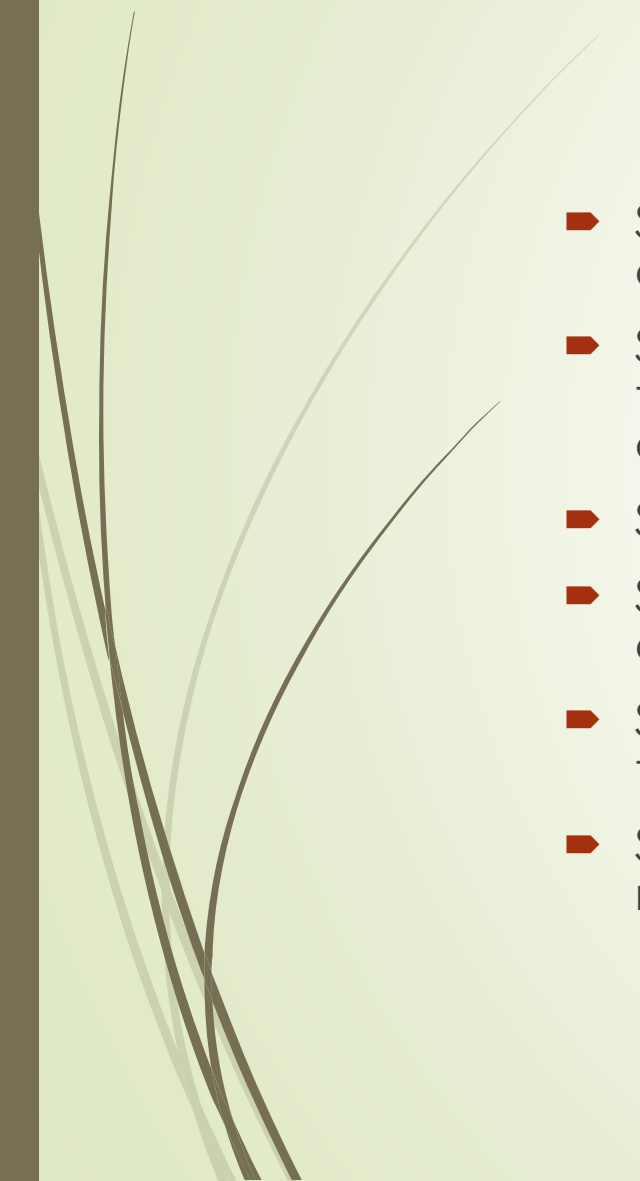


Distance Metrics in KNN – Hamming Distance

- The previously discussed metrics (Euclidean, Manhattan, and Minkowski) are commonly used for numerical data in machine learning.
- However, other metrics like Hamming Distance are useful when working with categorical data.
- Hamming Distance measures the number of positions at which two vectors differ:
 - It's particularly useful for comparing binary strings or Boolean vectors.
 - It can also be applied to strings (e.g., detecting the number of differences between two DNA sequences).

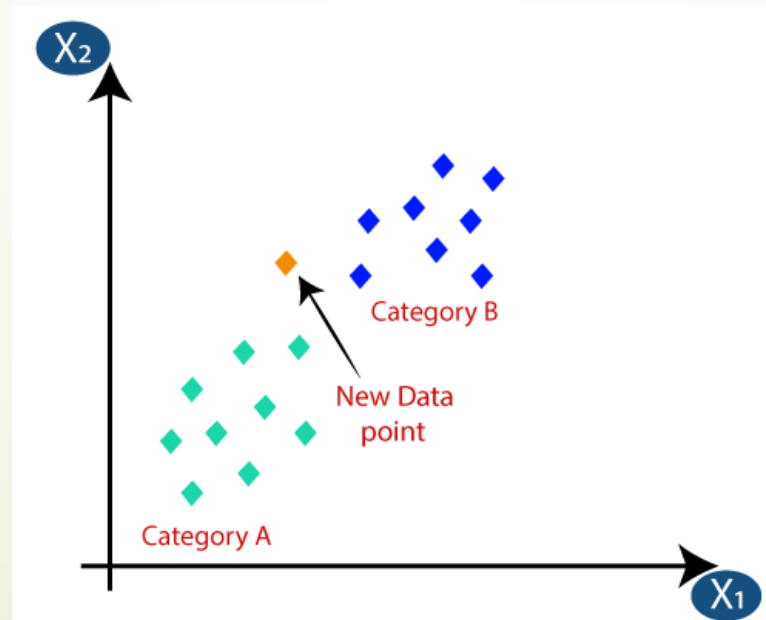


How Does KNN Work?

- Step 1: Select the number of neighbors K (the number of nearest data points to consider).
 - Step 2: Calculate the distance between the query point and all data points in the dataset. (The most common metric is Euclidean distance, but other metrics can be used).
 - Step 3: Identify the K nearest neighbors based on the calculated distances.
 - Step 4: For classification, count the number of neighbors in each class (or category) among the K neighbors.
 - Step 5: Assign the query point to the class with the majority vote (i.e., the class that appears most frequently among the K neighbors).
 - Step 6: The model is now ready to predict new data points based on their nearest neighbors.
- 

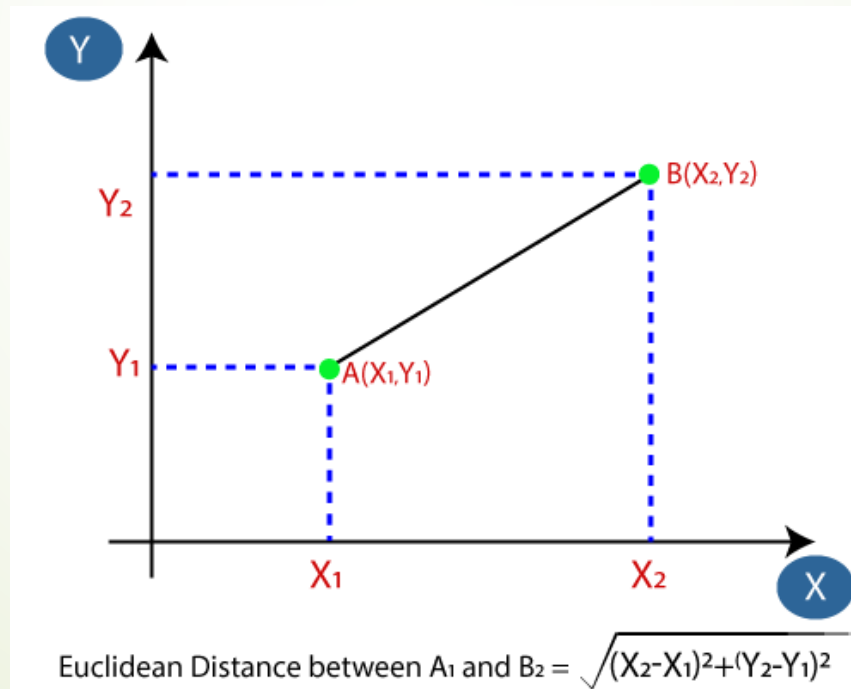
Steps of KNN

- Suppose we have a new data point that we need to classify into one of the categories.
- Step 1: We first select the number of neighbors, K . In this case, we choose $K=5$.



Steps of KNN

- **Step 2:** Calculate the distance between the new data point and all other points in the dataset

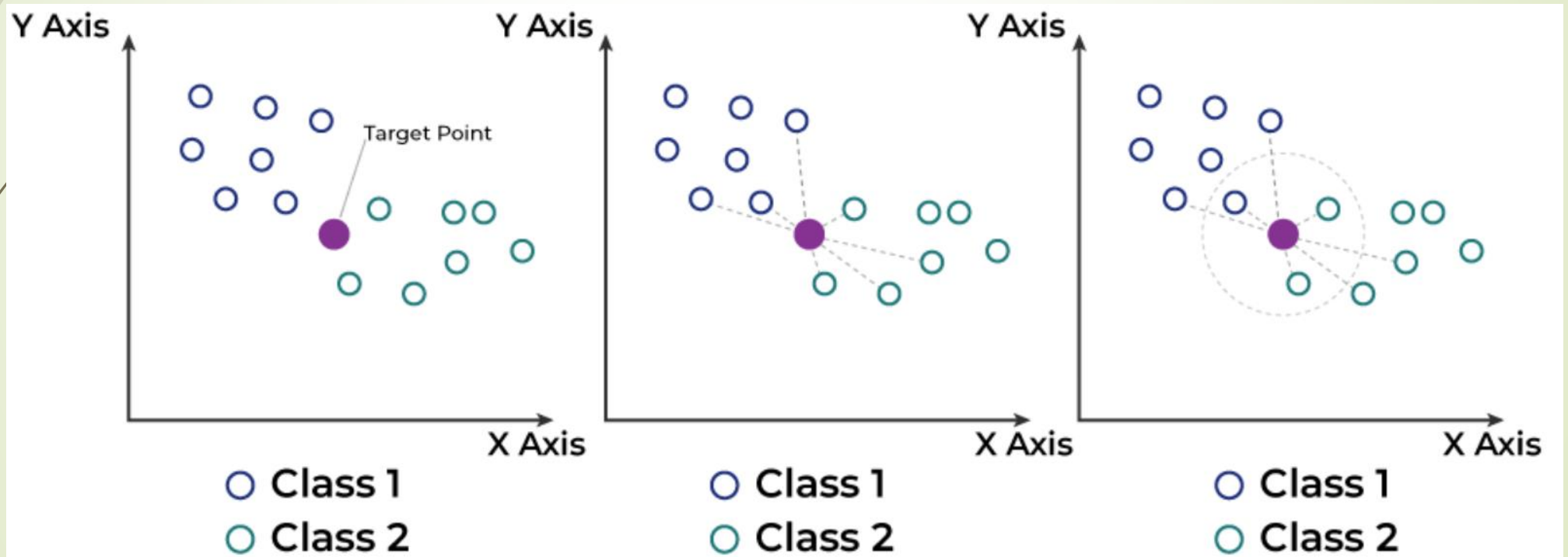


Steps of KNN

- **Step 3:** Identify the 5 closest neighbors based on their distance.
- **Step 4:** The new data point is assigned to the category that most of its 5 nearest neighbors belong to.



Steps of KNN



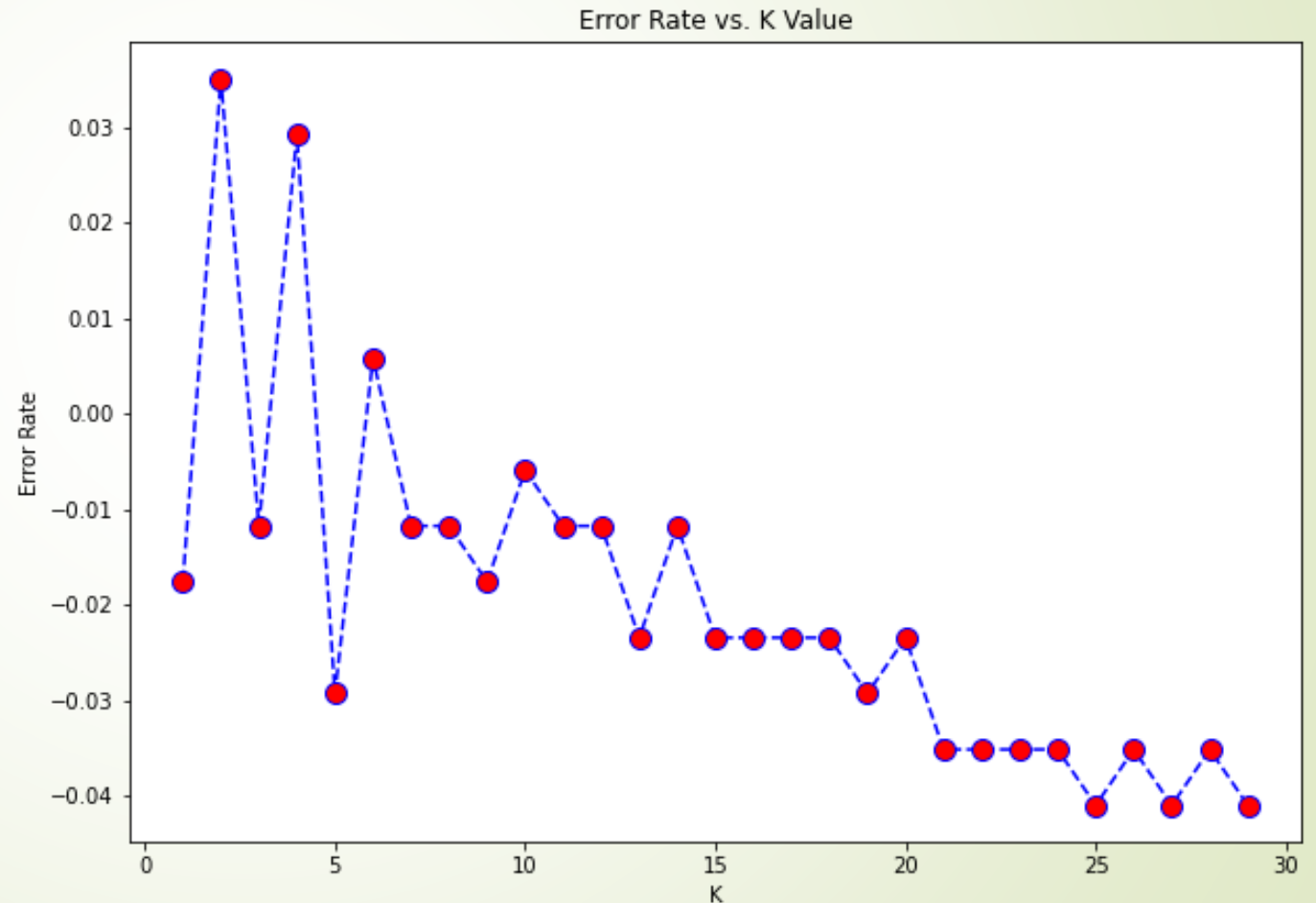


Tips for Selecting Optimal K

- Consider the dataset:
 - If the dataset has outliers or noise, a higher K value is typically more robust.
 - For datasets with clear separation between categories, a smaller K may work effectively.
- Odd vs. Even K:
 - Use odd values of K to avoid ties in classification. If K is even, there's a chance the algorithm will encounter equal votes from different classes.
- Common choices:
 - K values like 5 or 7 are frequently used as they offer a good balance between noise sensitivity and smooth decision boundaries.
- Cross-validation: Use cross-validation techniques to test different values of K and determine which one works best for your dataset.

How to Select the Optimal K

- Cross-validation is a key method to help determine the best K value for a given dataset.
 - To select the optimal K value, we use the error curve:
 - The error rate is plotted against different K values.
 - The goal is to find the K value that minimizes the error while balancing bias and variance.





Data Scaling



- ▶ Why is data scaling important?
 - ▶ In KNN, distances between points are calculated in a multidimensional feature space.
 - ▶ Features with larger ranges may dominate the distance calculation, which can skew results.
- ▶ Solution:
 - ▶ To ensure all features contribute equally, it is important to bring them to the same scale.
 - ▶ This is achieved through:
 - ▶ Normalization: Scaling data to a range between 0 and 1.
 - ▶ Standardization: Scaling data to have a mean of 0 and a standard deviation of 1.



Dimensionality Reduction

- Why reduce dimensions?
 - KNN may struggle with high-dimensional datasets due to the curse of dimensionality—as the number of dimensions increases, the distance between points becomes less meaningful.
- Techniques:
 - Feature Selection: Choose the most important features that contribute to the prediction task.
 - Principal Component Analysis (PCA): A technique that transforms the dataset into a lower-dimensional space while preserving as much variance as possible.



Missing Value Treatment

- Problem:

- KNN requires complete data to compute distances.
- If a feature is missing for a data point, the distance calculation becomes inaccurate.

- Solution:

- Imputation: Replace missing values with estimated values (e.g., the mean, median, or mode of the feature).
- Row Deletion: If the proportion of missing data is small, remove the rows with missing values to maintain data quality.



Advantages of KNN

- Easy to Implement:
 - The algorithm is simple and straightforward to understand and implement.
- Adaptable:
 - KNN is a lazy learner, meaning no explicit training phase is required.
 - The algorithm dynamically adjusts to new data as it's added, making it highly adaptive to changing datasets without needing retraining.
- Few Hyperparameters:
 - KNN requires only two key hyperparameters:
 - The value of K (the number of neighbors).
 - The choice of distance metric (e.g., Euclidean, Manhattan).
 - This simplicity in tuning makes KNN easier to optimize compared to more complex algorithms.



Disadvantages of KNN

- ▶ **Lazy Algorithm:**
 - ▶ KNN is considered a lazy learner because it does not train a model before prediction; instead, it stores all the data and only performs calculations at query time.
- ▶ **High Computational Costs:**
 - ▶ As the dataset grows, the computation required to calculate the distance between the query point and all other points becomes time-consuming.
 - ▶ This makes KNN resource-intensive, especially when dealing with large datasets.
- ▶ **Storage Limitations:**
 - ▶ Since KNN stores all the training data, it requires significant memory, particularly for large datasets.



Curse of Dimensionality

- High Dimensionality Problems:
 - KNN is negatively affected by the curse of dimensionality, meaning that as the number of dimensions increases, the distance between points becomes less distinguishable, making it harder to classify points correctly.
- Distance Becomes Meaningless:
 - In high-dimensional spaces, most points become equidistant from one another, diminishing the usefulness of distance metrics like Euclidean distance.



Prone to Overfitting

- Overfitting:

- With high-dimensional data, KNN can easily overfit, as it becomes sensitive to small variations in the data.

- Solution:

- Use feature selection or dimensionality reduction techniques (such as PCA) to reduce the number of features and make the algorithm more robust.