K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, instance-based, non-parametric machine learning algorithm used for both classification and regression tasks. The basic idea is to predict the label or value of a new data point based on the labels or values of the closest data points (neighbors) in the feature space.

How KNN Works

KNN works by following these steps:

1. **Select the number of neighbors (K):** Choose a value of K, which is the number of nearest neighbors to consider for making a prediction.
2. **Compute distances:** Calculate the distance between the new data point and all other points in the dataset using a distance metric (typically Euclidean distance, though other metrics such as Manhattan, Minkowski, etc., can be used).
3. **Identify neighbors:** Find the K points that are closest to the new data point based on the computed distances.
4. **Make a prediction:**
   - **Classification:** Assign the most common class (mode) among the K nearest neighbors to the new data point.
   - **Regression:** Assign the average (mean) value of the K nearest neighbors to the new data point.

KNN for Classification: Example

Let's consider an example where we classify whether a flower is a **setosa** or **versicolor** based on its petal width and length.

- **Dataset:**
  - **Features:** Petal width and length.
  - **Labels:** `Setosa` and `Versicolor`.
- **New Data Point:** Suppose we have a new flower with petal width = 1.5 and length = 4.5, and we want to classify whether it is `Setosa` or `Versicolor`.

Steps:

1. **Choose K:** Let's select (K = 3).
2. **Calculate Distances:** Calculate the Euclidean distance between the new flower and all the other flowers in the dataset:
   `d(x, x_i) = sqrt(sum((x_j - x_{i,j})^2 for j in range(n)))`
3. **Find the 3 Nearest Neighbors:** Based on the calculated distances, identify the 3 nearest flowers to the new data point.
4. **Classify:** Take a majority vote among the 3 nearest neighbors. If 2 out of the 3 neighbors are classified as `Setosa`, and 1 is classified as `Versicolor`, the

new flower would be classified as `Setosa`.

KNN for Regression: Example

Now, let's consider a regression problem where we want to predict the house price based on features like the number of rooms, lot size, and proximity to a city center.

- **Dataset:**
  - **Features:** Number of rooms, lot size, distance from city center.
  - **Labels:** House prices.
- **New Data Point:** Suppose we have a new house with 3 rooms, a lot size of 500 sq meters, and it is 10 km from the city center. We want to predict its price.

Steps:

1. **Choose K:** Let's select (K = 3).
2. **Calculate Distances:** Compute the Euclidean distance between the new house and all the other houses in the dataset based on their features.
3. **Find the 3 Nearest Neighbors:** Identify the 3 houses that are closest to the new house.
4. **Predict the Price:** Take the average of the prices of the 3 nearest houses. Suppose the prices of the nearest houses are $200,000$,220,000, and $210,000. Then the predicted price would be:
   ```
   Predicted Price = (200000 + 220000 + 210000) / 3 = 210000
   ```

Key Points to Consider

1. **Distance Metric:** The choice of distance metric impacts KNN performance. Euclidean distance is the most common, but for categorical data, Hamming distance might be used, and for Manhattan distance, L1 norm can be used.

2. **Value of K:**
   - Small values of K (e.g., (K = 1)) make the model sensitive to noise in the dataset but provide more flexible decision boundaries.
   - Large values of K smooth out predictions but may blur the distinctions between different classes.
3. **Scaling/Normalization:** Since KNN relies on distance calculations, it's important to scale or normalize the feature values to avoid one feature dominating the distance calculation. For instance, features with larger ranges can distort the distances unless the data is scaled.

4. **Lazy Learner:** KNN is a **lazy learning algorithm**, meaning it doesn't learn a model during the training phase. All computation is deferred until prediction, which can lead to high computational cost when the dataset is large.

Advantages of KNN

- **Simplicity:** Easy to implement and understand.
- **No Training Phase:** No explicit model training is required, making KNN suitable for dynamic datasets where data frequently changes.
- **Versatility:** Can be used for both classification and regression tasks.

Disadvantages of KNN

- **Computationally Expensive:** During prediction, KNN requires calculating the distance between the query point and all other points in the dataset, which can be slow for large datasets.
- **Memory Intensive:** KNN stores all training data, which can consume a lot of memory.
- **Sensitive to Irrelevant Features:** If irrelevant features are included, they can distort the distance metric and lead to poor performance. Therefore, feature selection is crucial.
- **Imbalanced Datasets:** KNN struggles with imbalanced datasets where one class dominates, as the majority class will be more likely to be chosen.

Conclusion

KNN is a versatile algorithm that can be used for both classification and regression. Its performance heavily depends on the choice of K, distance metric, and scaling of data. It is best suited for smaller datasets, as it requires a lot of computational resources for large datasets. However, its simplicity and effectiveness make it a popular choice for many applications.

Distance Metrics in KNN

The distance metric is a crucial aspect of KNN, as it determines how "closeness" between data points is calculated.

1. Euclidean Distance

This is the most commonly used distance metric in KNN, which calculates the straight-line distance between two points in a multidimensional space.

- **Formula:**
  `d(p, q) = sqrt(sum((p_i - q_i)^2 for i in range(n)))`
  where (p) and (q) are the points (or vectors) with (n) features.

2. Manhattan Distance

Manhattan distance, also called L1 distance or city block distance, calculates the distance by summing the absolute differences of their coordinates.

- **Formula:**
  `d(p, q) = sum(|p_i - q_i| for i in range(n))`

3. Hamming Distance

Hamming distance is used for categorical variables and counts the number of positions where the corresponding elements differ.

- **Formula:**
  `d(p, q) = sum(1 if p_i != q_i else 0 for i in range(n))`

4. Minkowski Distance

Minkowski distance is a generalization of both Euclidean and Manhattan distances.

- **Formula:**
  `d(p, q) = (sum(|p_i - q_i|^r for i in range(n)))^(1/r)`

For Euclidean distance, (r = 2), and for Manhattan distance, (r = 1).

Hyperparameters in KNN

1. K (Number of Neighbors)

- **Role:** Determines how many neighbors are considered when making predictions.
- **Choosing K:** The optimal value of (K) can be determined using cross-validation. Often, a good starting point is (K = \sqrt{n}), where (n) is the number of data points.

2. Distance Metric

- **Role:** Defines how the distance between points is measured. Common choices include Euclidean, Manhattan, and Hamming distances.

3. Weights

- **Role:** Assigns weights to the neighbors. You can use uniform weights, where each neighbor is weighted equally, or distance-based weights, where closer neighbors have more influence.

Choosing the Value of K

1. **Cross-Validation:** Cross-validation is often used to select the best value of (K).
2. **Empirical Rule:** A commonly used rule of thumb is to set (K) to the square root of the number of samples:

   ```
   K = sqrt(n)
   ```

3. **Error Rates:** Larger values of (K) lead to smoother decision boundaries but increase bias (underfitting), while smaller values decrease bias but increase variance (overfitting).

Conclusion

KNN's performance is influenced by the choice of distance metric, the number of neighbors (K), and how the neighbors are weighted. Cross-validation is a robust way to select the optimal value for these hyperparameters, ensuring that the model performs well on unseen data.

KNN: Pros and Cons

Pros of KNN

1. **No Strong Assumptions:**
    - **Explanation:** Unlike other algorithms like Linear Regression or SVM, KNN doesn't make strong assumptions about the data. It is a non-parametric algorithm, meaning it doesn't assume an underlying probability distribution or model structure.
    - **Benefit:** This makes KNN highly flexible and able to work well with data that doesn't fit into predefined assumptions, such as nonlinear data or complex decision boundaries.
2. **Simple to Implement and Understand:**
    - **Explanation:** KNN is conceptually simple and easy to understand. The idea of finding the closest points to make a decision based on the majority is intuitive.
    - **Benefit:** Easy implementation in applications, even with minimal data preparation or preprocessing.
3. **Adaptable to Classification and Regression:**
    - **Explanation:** KNN can be used for both classification and regression tasks, making it versatile.
    - **Benefit:** Useful in various domains and applications like pattern recognition and data mining.
4. **Low Training Cost:**
    - **Explanation:** As a lazy learner, KNN doesn’t involve any model training phase. All the computation happens at prediction time.
    - **Benefit:** No expensive computations are required to build a model beforehand. You only need to store the data points.

Cons of KNN

1. **Lazy Learner:**
    - **Explanation:** KNN is considered a lazy learner because it doesn't learn a model during the training phase. It simply memorizes the training data and

makes predictions during the testing phase by calculating distances to the training data.
- **Drawback:** The prediction phase can be very slow, especially with large datasets, because KNN has to compute distances between a test point and every single training point. This leads to high computational cost during prediction, which makes it inefficient for real-time applications.

2. **Choice of K (Optimal Value):**
   - **Explanation:** The accuracy of KNN heavily depends on the choice of (K) (number of neighbors).
   - **Drawback:** Too small a (K) leads to high variance (overfitting), while too large a (K) leads to high bias (underfitting). It is often necessary to use techniques like cross-validation to find the optimal value of (K).
3. **High Memory Requirement:**
   - **Explanation:** Since KNN needs to store all the training data, it requires a significant amount of memory, especially with large datasets.
   - **Drawback:** This can make KNN impractical for memory-constrained systems or for handling very large datasets.
4. **Sensitive to Outliers and Noisy Data:**
   - **Explanation:** KNN is sensitive to the presence of noisy data points or outliers because they can skew the classification or regression results.
   - **Drawback:** Preprocessing steps like outlier detection, noise removal, and scaling become crucial to improving KNN's performance.
5. **Scaling and Normalization Required:**
   - **Explanation:** KNN relies on distance measures, which means features with larger magnitudes will dominate the results. Therefore, data needs to be scaled or normalized before applying KNN.
   - **Drawback:** Preprocessing can become an additional step in the workflow.

Applications of KNN

1. **Recommendation Systems:**
   - KNN is used in recommendation systems to find users or items that are similar to a given user or item. By calculating the distance between a user and others based on preferences, KNN can recommend items that similar users liked.
2. **Image Recognition:**
   - KNN is applied in image recognition tasks to classify images based on features extracted from the images. For example, handwritten digit recognition using KNN on image pixel values.
3. **Medical Diagnosis:**

- In medical diagnosis, KNN can be used to classify patients based on the similarity of their symptoms or test results to other known cases, aiding in diagnosing diseases.

4. **Anomaly Detection:**
   - KNN is effective for anomaly detection, where the algorithm can detect outliers or unusual data points based on their distance from the majority of the data.
5. **Pattern Recognition:**
   - KNN is commonly used in pattern recognition tasks, such as text recognition and speech recognition, where it helps in categorizing text or audio data into pre-defined categories.

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is a popular technique used to address imbalanced datasets, where one class significantly outnumbers the other(s). In such cases, KNN's performance can suffer because the majority class dominates the decision boundaries. SMOTE helps by synthetically generating new data points for the minority class.

How SMOTE Works:

1. **Identify Minority Class Samples:**
   - SMOTE identifies data points from the minority class that are under-represented.
2. **KNN on Minority Class:**
   - For each minority class data point, SMOTE identifies its K-nearest neighbors from the same minority class using the KNN algorithm.
3. **Create Synthetic Data:**
   - SMOTE creates synthetic data points by interpolating between the original minority class data point and its neighbors. This is done by selecting a random point along the line connecting the two points.
   - **Formula for Generating Synthetic Data:**

     ```
     new_sample = original_sample + (neighbor_sample - original_sample) * random(0, 1)
     ```
4. **Add Synthetic Samples to Dataset:**
   - The newly generated synthetic data points are then added to the dataset, increasing the representation of the minority class.

Advantages of SMOTE:

- Reduces bias in favor of the majority class by creating a balanced dataset.
- Helps improve the performance of classifiers on imbalanced data.

Disadvantages of SMOTE:

- It can introduce noise by generating synthetic data points that might not follow the true distribution of the minority class.
- If applied without careful tuning, it might lead to overfitting, as synthetic samples might be very similar to the original ones.

Summary

- **KNN Pros:** Works without strong assumptions, simple implementation, adaptable to both classification and regression, and has a low training cost.
- **KNN Cons:** Slow during prediction, sensitive to the choice of (K), requires high memory, sensitive to noise, and requires data scaling.
- **Applications of KNN:** Widely used in recommendation systems, image recognition, medical diagnosis, anomaly detection, and pattern recognition.
- **SMOTE:** Balances imbalanced datasets by generating synthetic minority class samples through interpolation between minority class neighbors.