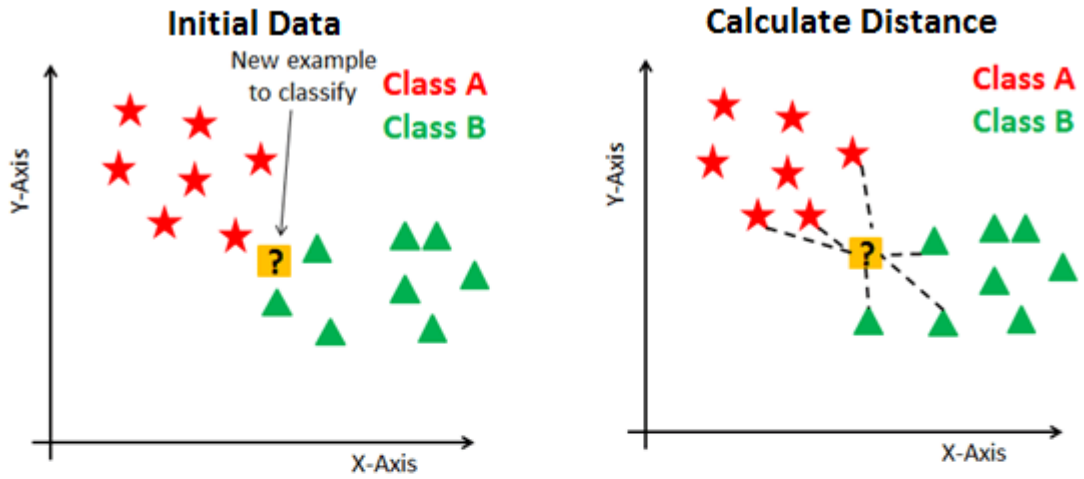
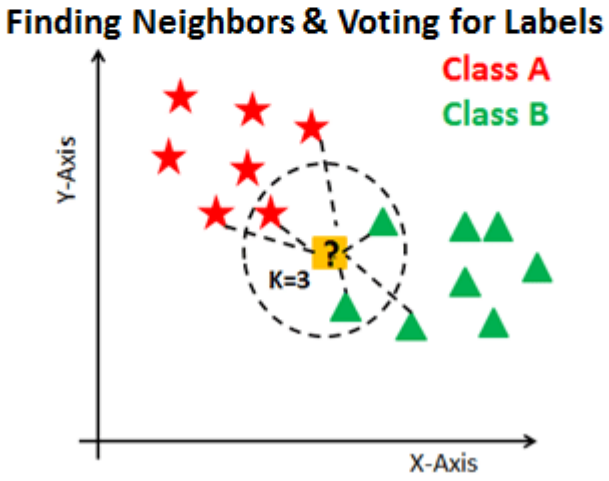
UNIT-II

Nearest Neighbor–Based Models Introduction

Nearest Neighbor–Based Models are a class of instance-based learning algorithms in machine learning. These models make predictions by comparing a new data point with already stored training examples. Unlike other machine learning models, nearest neighbor models do not create an explicit model during training. Instead, they store the entire training dataset and perform computation only at the time of prediction.

These models are also called lazy learners, because learning is delayed until a query or test instance is given. The core idea behind nearest neighbor models is that similar data points lie close to each other in the feature space.





Introduction to Proximity Measures

Proximity measures are used to quantify how close or similar two data objects are. In nearest neighbor–**based** learning, proximity measures play a central role because predictions depend entirely on how closeness is measured between data points.

There are two main types of proximity measures:

- Similarity measures: Higher value indicates more similarity
- Distance measures: Lower value indicates more similarity

Proximity measures are required in:

- Classification
- Regression
- Clustering
- Pattern recognition

Without proper proximity measures, nearest neighbor algorithms cannot correctly identify neighbors, leading to poor predictions.

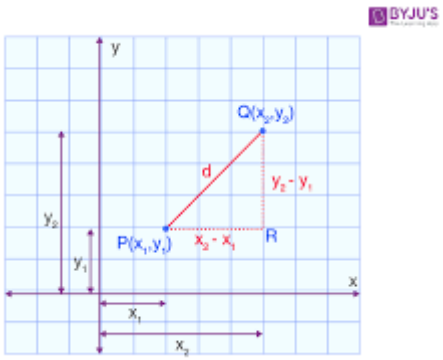
---

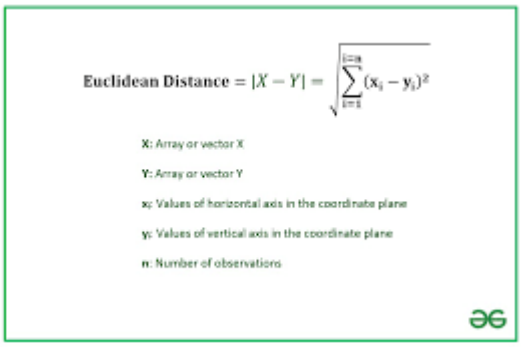
2. **Distance Measures**

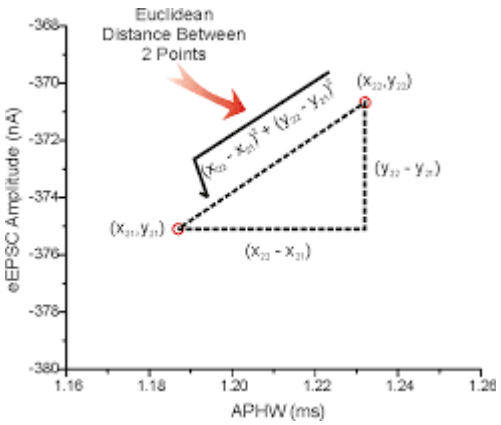
Distance measures calculate the numerical distance between two feature vectors. The choice of distance measure greatly affects the performance of nearest neighbor models.

a) Euclidean Distance

Euclidean distance is the most commonly used distance measure. It represents the straight-line distance between two points in multidimensional space.







Used when

- Continuous features
- Features are on similar scales

Algorithms

- k-means
- k-NN
- hierarchical clustering

Example:
Used in image classification, spatial data analysis.

Advantages:

- Simple and intuitive
- Works well for continuous data

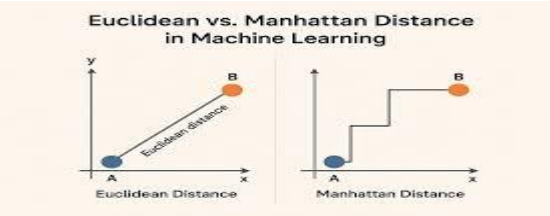
Disadvantages:

- Sensitive to scale
- Affected by outliers

---

b) Manhattan Distance

Manhattan distance measures distance as the sum of absolute differences along each dimension.



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

d(x,y)=∑|xi−yi|
d(x,y) = sum |x_i - y_i|
d(x,y)=∑|xi−yi|
For two points P=(x1,y1),Q=(x2,y2)$P=(x1,y1),Q=(x2,y2)$, the Manhattan distance is:

- For 2-dimensions: |x1−x2|+|y1−y2||$x1−x2|+|y1−y2|$
- For 3-dimensions: points |x1−x2|+|y1−y2|+|z1−z2||$x1−x2|+|y1−y2|+|z1−z2|$
- For n-dimensions: ∑i=1n|xi−yi|=|x1−y1|+|x2−y2|+⋯+|xn−yn|$\sum i=1n|xi−yi|=|x1−y1|+|x2−y2|+⋯+|xn−yn|$
- *Points* P=(1,2),Q=(4,0)$P=(1,2),Q=(4,0)$
- D(P,Q)=|x1−x2|+|y1−y2|=|1−4|+|2−0|=3+2=5$D(P,Q)=|x_1−x_2|+|y_1−y_2|=|1−4|+|2−0|=3+2=5$
-

Example:
Used in grid-based path problems and city block navigation.

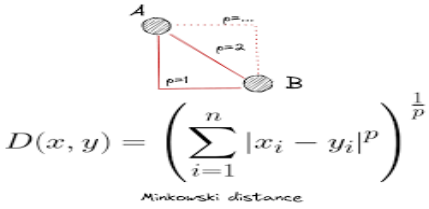
---

c) Minkowski Distance

Minkowski distance is a generalized distance measure that includes Euclidean and Manhattan distances as special cases.

Minkowski Distance is a generalized metric that unifies various distance measures used in mathematics and machine learning. It provides a flexible way to compute distances between points in an n-dimensional space.

It is a powerful distance function that encompasses several well-known distance metrics, such as Manhattan Distance and Euclidean Distance, as special cases, with different values of p leading to well-known distances such as Manhattan Distance (p=1), Euclidean Distance (p=2), and Chebyshev Distance (p→∞).The Minkowski Distance between two points A = (A_1, A_2, …, A_n) and B = (B_1, B_2, …, B_n) in an n-dimensional space is given by:D(A,B)=(∑i=1n|Ai−Bi|p)1pD(A,B)=(∑i=1n|Ai−Bi|p)p1



Distance measures must be selected carefully based on data type and application.

Non-Metric Similarity Functions

Non-metric similarity functions are used when similarity cannot be expressed as a true distance. These functions do not strictly satisfy mathematical properties like triangle inequality or symmetry.

Non-Metric Similarity Functions in Nearest Neighbor-Based Models

▪ Non-metric similarity functions are measures used to evaluate the similarity between data

points without strictly adhering to the properties of a metric space (e.g., triangle inequality).

▪ These functions are often used in scenarios where relationships between data points cannot

be easily captured by traditional metric distance measures, such as Euclidean or Manhattan distance.

Characteristics of Non-Metric Similarity Functions

▪ They focus on measuring similarity rather than distance.

▪ They may not satisfy properties like symmetry, non-negativity, or the triangle inequality.

▪ They are particularly useful in domains where data relationships are non-linear or symbolic

They are commonly used when:

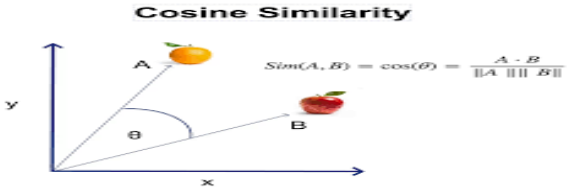
- Data is categorical or symbolic
- Feature vectors represent text or documents

Cosine similarity :– measures angle between vectors

Jaccard similarity: – measures similarity between sets

Cosine similarity:

Measures the angle between vectors (not their magnitude).



Common Non-Metric Similarity Functions:

- **Cosine Similarity:** Measures the cosine of the angle between two vectors. Commonly used in text analysis.

  Formula:

$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

  - $\vec{A} \cdot \vec{B}$: Dot product of vectors A and B.
  - $\|\vec{A}\|$: Magnitude of vector A.
  - $\|\vec{B}\|$: Magnitude of vector B.

  Example:

  Let's compare two text documents represented as term-frequency vectors:

  - $\vec{A} = [1, 1, 0, 1]$ (e.g., frequency of terms in document 1)
  - $\vec{B} = [0, 1, 1, 1]$ (e.g., frequency of terms in document 2)

Step 1: Compute Dot Product ($\vec{A} \cdot \vec{B}$):

$$\vec{A} \cdot \vec{B} = (1 \cdot 0) + (1 \cdot 1) + (0 \cdot 1) + (1 \cdot 1) = 0 + 1 + 0 + 1 = 2$$

Step 2: Compute Magnitudes:

$$\|\vec{A}\| = \sqrt{(1^2 + 1^2 + 0^2 + 1^2)} = \sqrt{1 + 1 + 0 + 1} = \sqrt{3}$$

$$\|\vec{B}\| = \sqrt{(0^2 + 1^2 + 1^2 + 1^2)} = \sqrt{0 + 1 + 1 + 1} = \sqrt{3}$$

Step 3: Compute Cosine Similarity:

$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} = \frac{2}{\sqrt{3} \cdot \sqrt{3}} = \frac{2}{3} \approx 0.667$$

Result: The Cosine Similarity is approximately **0.667**.

- **Application:**
  - Used in text mining, image recognition, and kernel-based machine learning models (e.g., Support Vector Machines).

Jaccard Similarity:

Definition: Measures the similarity between two sets as the size of the intersection divided by the size of the union.

Formula:

$$\text{Similarity}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Range: [0,1], where 1 indicates identical sets and 0 indicates completely disjoint sets.

Use Case: Used for comparing categorical data, such as text document similarity or set overlap.

Example: For sets A={1,2,3}and B={2,3,4}:

$$\text{Similarity} = \frac{|\{2, 3\}|}{|\{1, 2, 3, 4\}|} = \frac{2}{4} = 0.5$$

Hamming Similarity:

Hamming Similarity is a measure used to compare two equal-length binary or string sequences by counting the number of matching positions.

Formula:

$$\text{Hamming Similarity} = \frac{\text{Number of matching positions}}{\text{Total length of the strings}}$$

It is closely related to **Hamming Distance**, which counts the number of differing positions.

$$\text{Hamming Distance} = \sum_{i=1}^{n} (x_i \neq y_i)$$

$$\text{Hamming Similarity} = 1 - \frac{\text{Hamming Distance}}{n}$$

Applications:

- Text mining
- Information retrieval
- Recommendation systems

Non-metric similarity functions are especially important in high-dimensional data.

Proximity Between Binary Patterns:

Binary patterns consist of features that take only 0 or 1 values, representing absence or presence of a property.

To measure proximity between binary patterns, specialized measures are used.

What are binary patterns?

Binary patterns are sequences of 0s and 1s, where each position in the sequence represents a binary attribute. These attributes can indicate the presence or absence of a feature, a yes/no answer, or any other binary outcome.

Proximity measures for binary patterns in machine learning quantify the similarity or dissimilarity between binary data objects, which are often represented as vectors of 0s and 1s.

These measures are crucial for tasks like clustering, classification, and nearest neighbor search.

Standard measures include Jaccard index, hamming distance, and cosine similarity, each offering a different perspective on the relationship between binary patterns.

Why are proximity measures necessary for binary patterns?

• **Clustering:** Grouping similar binary patterns based on their proximity.

• **Classification:** Determining the class of a new data point based on its proximity to labelled data points.

• **Nearest Neighbor Search:** Finding the most similar data points to a query point in a dataset.

Common Proximity Measures:

Jaccard Index:

Measures the similarity between two sets (or binary vectors) by dividing the size of their intersection by the size of their union. It is particularly useful when the number of shared '1's is important, but the number of shared '0's is not.

Hamming Distance:

Calculates the number of differing bits between two binary strings. It is a dissimilarity measure, with a lower distance indicating higher similarity.

Cosine Similarity:

Measures the cosine of the angle between two vectors. For binary data, it can be interpreted as the proportion of shared '1's, but it also considers the overall '1's in each vector.

Simple Matching Coefficient (SMC):

Calculates the proportion of attributes that are the same (both 0 or both 1) between two binary vectors.

Common measures:

- Hamming Distance – counts number of mismatched bits
- Jaccard Coefficient – ignores matching zeros
- Simple Matching Coefficient – considers both matches

Example:
Medical diagnosis where symptoms are represented as binary values.

Binary proximity measures are widely used in:

- Pattern recognition
- Bioinformatics
- Document classification

Distance-Based Classification Algorithms:

DIFFERENT CLASSIFICATION ALGORITHMS BASED ON THE DISTANCE MEASURES

Several machine learning classification algorithms rely on distance measures to categorize data points.

These algorithms use various distance metrics to determine the similarity between data points and assign them to appropriate classes.

Some prominent distance-based classification algorithms include K-Nearest Neighbor (KNN), which classifies based on the majority class of its nearest neighbors, and other algorithms like K-Means, which can be adapted for classification by using distance to cluster data points.

Standard distance metrics include Euclidean distance, Manhattan distance, Minkowski distance, and Hamming distance.

Distance-Based Classification Algorithms:

K-Nearest Neighbors (KNN):

This algorithm classifies a new data point based on the majority class of its k-nearest neighbors in the feature space, where distance is calculated using a chosen distance metric.

K-Means Clustering (for classification):

While primarily a clustering algorithm, K-Means can be adapted for classification by using the distance between data points and cluster centroids. New data points are assigned to the cluster whose centroid is closest.

Support Vector Machine:

Support Vector Machines (SVMs) can also incorporate distance-based calculations, particularly in kernel functions, to map data into higher-dimensional spaces for better separation.

K-Nearest Neighbor (KNN) Algorithm

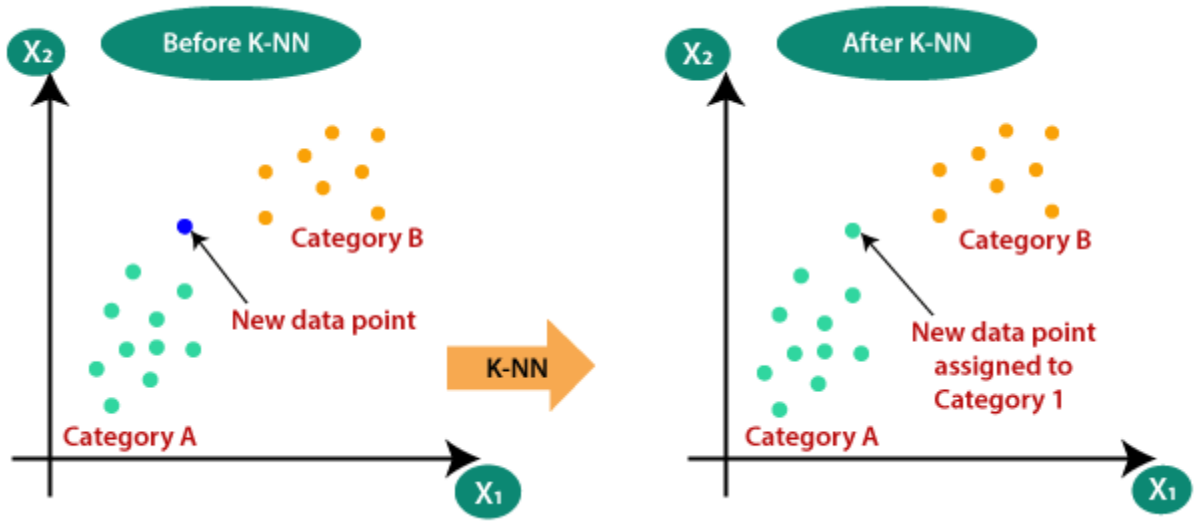
K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification, but can also be used for regression tasks.

It works by finding the "k" closest data points (neighbors) to a given input and makes a prediction based on the majority class (for classification) or the average value (for regression).

Since KNN makes no assumptions about the underlying data distribution, it is a non-parametric and instance-based learning method.

K-Nearest Neighbor(KNN) Algorithm:

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification but can also be used for regression tasks. It works by finding the "k" closest data points (neighbors) to a given input and makes a predictions based on the majority class (for classification) or the average value (for regression). Since KNN makes no assumptions about the underlying data distribution it makes it a non-parametric and instance-based learning method.



Working of KNN algorithm

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

- To measure the similarity between target and training data points Euclidean distance is widely used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- When you want to classify a data point into a category like spam or not spam, the KNN algorithm looks at the K closest points in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called majority voting.
- In regression, the algorithm still looks for the K closest points. But instead of voting for a class in classification, it takes the average of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.

It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e. 5 in this case and assigns test point the majority class label that is grey label class here.

Applications of KNN

- **Recommendation Systems**: Suggests items like movies or products by finding users with similar preferences.
- **Spam Detection**: Identifies spam emails by comparing new emails to known spam and non-spam examples.
- **Customer Segmentation**: Groups customers by comparing their shopping behavior to others.
- **Speech Recognition**: Matches spoken words to known patterns to convert them into text.

Advantages of KNN

- **Simple to use**: Easy to understand and implement.
- **No training step**: No need to train as it just stores the data and uses it during prediction.
- **Few parameters**: Only needs to set the number of neighbors (k) and a distance method.
- **Versatile**: Works for both classification and regression problems.

Disadvantages of KNN

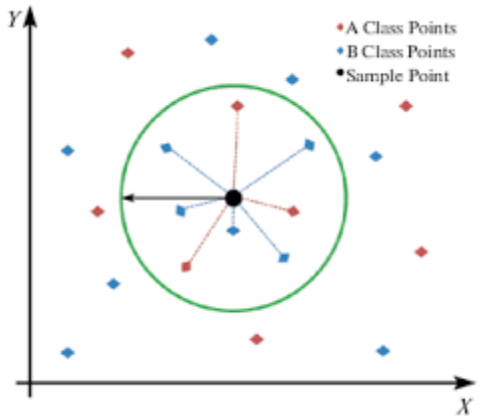
- **Slow with large data**: Needs to compare every point during prediction.
- **Struggles with many features**: Accuracy drops when data has too many features.
- **Can Overfit**: It can overfit especially when the data is high-dimensional or not clean.

Radius Distance Nearest Neighbor Algorithm

The r-Nearest Neighbors algorithm, also known as the Radius Nearest Neighbor algorithm, is a distance-based learning method used for classification and regression. Unlike the K-Nearest Neighbor (KNN) algorithm, which selects a fixed number of nearest neighbors, the r-Nearest Neighbor algorithm selects all training data points that lie within a specified radius (r) from the test instance.

In this approach, a distance threshold r is defined in advance. When a new data point is given, the algorithm finds all training points whose distance from the test point is less than or equal to r. These points are considered the neighbors. The final output is decided based on the labels or values of these neighbors.

The main idea behind r-Nearest Neighbors is that data points closer to each other are more similar, and all points within a certain distance should influence the prediction. This method is especially useful when data is unevenly distributed, because the number of neighbors automatically adjusts according to local data density.



Algorithm for r-Nearest Neighbors

The step-by-step algorithm for the r-Nearest Neighbor method is explained below:

Step 1: Choose the Radius (r)

Select an appropriate radius value r, which defines how far the algorithm should search for neighbors. The value of r plays a crucial role in model performance.

Step 2: Store the Training Dataset

All training data points along with their class labels (for classification) or output values (for regression) are stored. There is no explicit training phase.

Step 3: Compute Distance

For a given test instance, compute the distance between the test point and each training data point using a suitable distance measure such as Euclidean or Manhattan distance.

Step 4: Select Neighbors Within Radius r

Identify all training points whose distance from the test point is less than or equal to r. These points form the neighbor set.

Step 5: Make Prediction

- For classification:
  Assign the class that occurs most frequently among the neighbors within radius r.
- For regression:
  Compute the average or weighted average of the output values of neighbors.

Step 6: Handle Special Cases

If no data points fall within the radius, the algorithm may:

- Increase the radius value, or
- Return no prediction, or
- Fall back to KNN method

Example of r-Nearest Neighbors

Classification Example

Consider a dataset containing two classes:

- Class A (●)
- Class B (▲)

A new test point T is given, and the radius r = 2 units.

1. Distances are calculated between T and all training points.
2. Only the points lying inside the radius circle are selected.
3. Suppose inside the radius:
   - 3 points belong to Class A
   - 1 point belongs to Class B
4. Since Class A has the majority, the test point T is classified as Class A.

This example shows that the number of neighbors is not fixed and depends on how many points fall within the radius.

Advantages of r-Nearest Neighbors

- Automatically adjusts the number of neighbors based on data density
- Performs well in non-uniform datasets
- Simple and intuitive to understand
- Useful for both classification and regression

Disadvantages of r-Nearest Neighbors

- Choosing an optimal radius r is difficult
- Performance is sensitive to data scaling
- If r is too small, no neighbors may be found
- If r is too large, irrelevant points may affect prediction
- Computationally expensive for large datasets

KNN Regression

Introduction to KNN Regression

KNN Regression is a distance-based, non-parametric machine learning algorithm used to predict continuous numerical values. It is an extension of the K-Nearest Neighbor (KNN) algorithm, where instead of predicting a class label, the algorithm predicts a real-valued output such as price, temperature, or salary.

In KNN Regression, the prediction for a new data point is made by identifying the K nearest data points from the training dataset and computing the average (or weighted average) of their output values. The fundamental assumption of this method is that similar input values have similar output values.

KNN Regression is an instance-based and lazy learning algorithm. It does not build an explicit model during training. Instead, all computations are performed at the time of prediction.

Working Principle of KNN Regression

The working principle of KNN Regression is based on the idea of local approximation. When a new input is provided, the algorithm looks for nearby data points in the feature space and uses their known output values to estimate the output of the new input.

Unlike parametric regression methods (such as linear regression), KNN Regression does not assume any fixed relationship between input and output. This makes it flexible and capable of modeling complex patterns.

Algorithm for KNN Regression

The step-by-step algorithm of KNN Regression is explained below:

Step 1: Choose the Value of K

Select a suitable value for K, which represents the number of nearest neighbors to be considered. The choice of K affects prediction accuracy.

Step 2: Store the Training Dataset

Store all training data points along with their output values. There is no separate training phase.

Step 3: Compute Distance

For a given test instance, calculate the distance between the test point and every training data point using a distance measure such as:

- Euclidean distance
- Manhattan distance

Step 4: Identify K Nearest Neighbors

Sort the distances and select the K data points with the smallest distances.

Step 5: Predict Output Value

- Simple KNN Regression:
  Compute the average of the output values of the K nearest neighbors.

- Weighted KNN Regression:
  Assign higher weights to closer neighbors and compute a weighted average.

Step 6: Return the Predicted Value

The computed value is returned as the final prediction.

Example of KNN Regression

House Price Prediction Example

Consider a dataset where house prices depend on house size (in square feet).

House Size (sq.ft)	Price (₹ Lakhs)
800	30
1000	40
1200	50
1500	65
1800	80

Test Input:
House size = 1300 sq.ft
Choose K = 3

Steps:

1. Compute distances between 1300 and all training points.
2. Select the 3 nearest sizes:
   1200, 1500, 1000
3. Corresponding prices:
   50, 65, 40
4. Predicted price:

Predicted Price=50+65+403=51.67 Lakhs\text{Predicted Price} = \frac{50 + 65 + 40}{3} = 51.67 \text{ Lakhs}Predicted Price=350+65+40=51.67 Lakhs

Thus, the predicted house price is ₹51.67 Lakhs.

Advantages of KNN Regression

- Simple and easy to understand
- No assumption about data distribution
- Flexible and adaptive to data
- Effective for nonlinear relationships
- No explicit training phase

Disadvantages of KNN Regression

- Computationally expensive during prediction
- Requires large memory to store data
- Sensitive to noise and outliers
- Performance depends heavily on distance measure
- Not suitable for very large datasets

Performance of Classifiers:

The performance of a classifier refers to how accurately a classification model predicts class labels for unseen data. Evaluating classifier performance is very important because a high accuracy on training data does not always mean the model will perform well on new data. Performance evaluation helps in comparing different classifiers and selecting the best one for a given problem.

Classifier performance is usually evaluated using a confusion matrix and several derived metrics.

Confusion Matrix

A confusion matrix is a table that shows the actual class labels versus predicted class labels.

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Where:

- TP (True Positive): Correctly predicted positive
- TN (True Negative): Correctly predicted negative
- FP (False Positive): Incorrectly predicted positive
- FN (False Negative): Incorrectly predicted negative

The confusion matrix is the foundation for most performance metrics.

Performance Metrics for Classifiers

1. Accuracy

Accuracy measures the overall correctness of the classifier.

Accuracy=TP+TN/TP+TN+FP+FN

Explanation:
It represents the percentage of correct predictions made by the classifier.

Limitation:
Accuracy can be misleading when data is imbalanced.

---

2. Precision

Precision measures how many of the predicted positive cases are actually positive.

Precision=TP/TP+FP

Explanation:
High precision means fewer false positives.

Example:
Important in spam detection, where false positives should be minimized.

---

3. Recall (Sensitivity)

Recall measures how many actual positive cases are correctly predicted.

Recall=TP/TP+FN

Explanation:
High recall means fewer false negatives.

Example:
Important in medical diagnosis, where missing a disease is dangerous.

---

4. F1-Score

F1-score is the harmonic mean of precision and recall.

F1=2×(Precision×Recall)/(Precision+Recall)

Explanation:
Balances both precision and recall, especially useful for imbalanced datasets.

---

5. Specificity

Specificity measures how well the classifier identifies negative cases.

Specificity=TN/TN+FP

Type 1 and Type 2 error

Type 1 and Type 2 error are:

- Type 1 error: It occurs when the model incorrectly predicts a positive instance but the actual instance is negative. This is also known as a false positive. Type 1 Errors affect the precision of a model which measures the accuracy of positive predictions.

Type 1 Error=FP/FP+TN

- Type 2 error: This occurs when the model fails to predict a positive instance even though it is actually positive. This is also known as a false negative. Type 2 Errors impact the recall of a model which measures how well the model identifies all actual positive cases.

Type 2 Error=FN/TP+FN

- ype 1 Error (False Positive): This occurs when the test predicts a patient has the disease (positive result) but the patient is actually healthy (negative case).

- Type 2 Error (False Negative): This occurs when the test predicts the patient is healthy (negative result) but the patient actually has the disease (positive case).

Importance of Classifier Performance Evaluation

- Helps compare multiple classifiers
- Identifies overfitting and underfitting
- Ensures reliability on unseen data
- Improves model selection

  The benefits of a confusion matrix

  As seen in the basic structure of a confusion matrix, the predictions are broken down into four categories: True Positive, True Negative, False Positive, and False Negative.

  This detailed breakdown offers valuable insight and solutions to improve a model's performance:

- Solving Imbalanced Data: As explored, using accuracy as a standalone metric has limitations when it comes to imbalanced data. Using other metrics, such as precision and recall, allows a more balanced view and accurate representation
- Error Type Differentiator: Understanding the different types of errors produced by the machine learning model provides knowledge of its limitations and areas of improvement.
- Trade-Offs: The trade-off between using different metrics in a Confusion Matrix is essential as they impact one another.

  Example:

At this point, we have defined the outcome and collected the data; the next step is to classify the outcomes into the four categories:

- True Positive: 60
- True Negative: 100
- False Positive: 20
- False Negative: 20

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = \frac{60+100}{60+100+20+20} = \frac{160}{200} = 0.8$$

$$Recall = \frac{TP}{TP+FN} = \frac{60}{60+20} = \frac{60}{80} = 0.75$$

$$Precision = \frac{TP}{TP+FP} = \frac{60}{60+20} = \frac{60}{80} = 0.75$$

$$Specificity = \frac{TN}{TN+FP} = \frac{100}{100+20} = \frac{100}{120} \approx 0.833$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} = 2 \times \frac{0.75 \times 0.75}{0.75+0.75} = 0.75$$

-

Performance of Regression Algorithms:

The performance of regression algorithms refers to how accurately a regression model predicts continuous numerical values. Unlike classification, regression performance is evaluated using error-based metrics that measure the difference between actual and predicted values.

Performance evaluation ensures that the regression model generalizes well and provides reliable predictions.

Common Performance Metrics for Regression Algorithms

Types of Regression Metrics

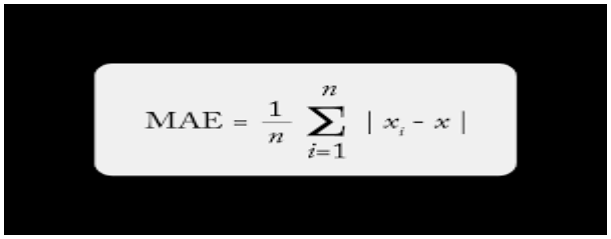
Some common regression metrics in scikit-learn with examples

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- R-squared (R^2) Score
- Root Mean Squared Error (RMSE)

Mean Absolute Error (MAE):

In the fields of statistics and machine learning, the Mean Absolute Error (MAE) is a frequently employed metric. It's a measurement of the typical absolute discrepancies between a dataset's actual values and projected values.

Mathematical Formula



The formula to calculate MAE for a data with "n" data points is:

Where:

- x_i represents the actual or observed values for the i-th data point.
- y_i represents the predicted value for the i-th data point.

Mean Squared Error (MSE):
A popular metric in statistics and machine learning is the Mean Squared Error (MSE). It measures the square root of the average discrepancies between a dataset's actual values and projected values. MSE is frequently utilized in regression issues and is used to assess how well predictive models work.
Mathematical Formula:

$$\mathbf{MSE} = \frac{1}{n}\sum(Y_i - \hat{Y}_i)^2$$

For a dataset containing 'n' data points, the MSE calculation formula is:

R-squared (R^2) Score

A statistical metric frequently used to assess the goodness of fit of a regression model is the R-squared (R^2) score, also referred to as the coefficient of determination. It quantifies the percentage of the dependent variable's variation that the model's independent variables contribute to. R^2 is a useful statistic for evaluating the overall effectiveness and explanatory power of a regression model.

Mathematical Formula

$$R^2 = 1 - \frac{\Sigma_i^N (y_i - \hat{y}_i)^2}{\Sigma_i^N (y_i - \bar{y})^2}$$

The formula to calculate the R-squared score is as follows:

Where:

- R^2 is the R-Squared.
- SSR represents the sum of squared residuals between the predicted values and actual values.
- SST represents the total sum of squares, which measures the total variance in the dependent variable.

Root Mean Squared Error (RMSE)

RMSE stands for Root Mean Squared Error. It is a usually used metric in regression analysis and machine learning to measure the accuracy or goodness of fit of a predictive model, especially when the predictions are continuous numerical values.

The RMSE quantifies how well the predicted values from a model align with the actual observed values in the dataset. Here's how it works:

1. Calculate the Squared Differences: For each data point, subtract the predicted value from the actual (observed) value, square the result, and sum up these squared differences.
2. Compute the Mean: Divide the sum of squared differences by the number of data points to get the mean squared error (MSE).
3. Take the Square Root: To obtain the RMSE, simply take the square root of the MSE.

Mathematical Formula

$$RMSE = \sqrt{\frac{\Sigma (y_i - \hat{y}_i)^2}{N - P}}$$

The formula for RMSE for a data with 'n' data points is as follows:

Where:

- RMSE is the Root Mean Squared Error.
- x_i represents the actual or observed value for the i-th data point.
- y_i represents the predicted value for the i-th data point.

Importance of Regression Performance Evaluation

- Measures prediction accuracy
- Helps compare regression models
- Detects overfitting and underfitting
- Improves model tuning