ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Understanding Distance Metrics in KNN Imputation: Theoretical Insights and Applications

Vaibhav Tummalapalli

USA

ABSTRACT

K-Nearest Neighbors (KNN) imputation is a widely used technique for handling missing data in machine learning and statistical modeling. The success of KNN imputation heavily depends on the choice of distance metric, as it determines the "closeness" of neighbors. This paper provides a comprehensive overview of the key distance metrics used in KNN imputation, including their theoretical background, mathematical formulations, use cases, and the implications of their selection on imputation outcomes [1-6].

***Corresponding author**

Vaibhav Tummalapalli, Atlanta, USA.

Received: June 26, 2025; **Accepted:** July 04, 2025, **Published:** July 10, 2025

Keywords: Imputation, Machine Learning, K-Nearest Neighbors, Scaling, Distance Metrics

Introduction

Handling missing data is a crucial step in data preprocessing. Among the various imputation methods, KNN imputation is popular for its simplicity and non-parametric nature. The method works by finding the k-nearest neighbors of a data point (with missing values) and imputing the missing values using the neighbors' attributes.

The performance of KNN imputation is influenced by the distance metric employed, which determines the "nearest" neighbors.

This paper explores the most common distance metrics – Euclidean, Manhattan, Minkowski, and Cosine - and their impact on the imputation process.

Distance Metrics

Euclidean Distance

Euclidean distance is derived from the Pythagorean theorem and measures the straight-line distance between two points in multidimensional space. It assumes all dimensions are equally important and uncorrelated. Imagine two points in a plane; the Euclidean distance is the length of the straight line connecting them. It provides an intuitive notion of "closeness" for continuous variables.

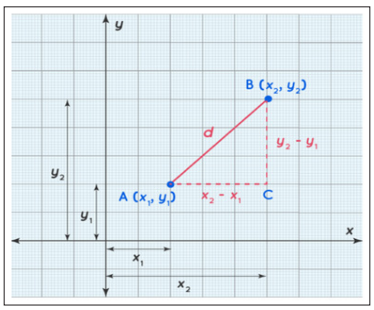

Figure 1: Pythagorean Theorem

$$AB^2 = AC^2 + BC^2$$

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

Taking the square root on both sides,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For two points with 'n' dimensions

$$X = (x_1, x_2, x_3, \ldots x_n) \text{ and } Y = (y_1, y_2, y_3, \ldots y_n)$$

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Suitable for Continuous Data

Euclidean distance is ideal for datasets where the features are continuous and numerical. It measures the "as-the-crow-flies" straight-line distance between two points in multidimensional space, which is meaningful only when the features can be treated as continuous variables. Examples include Physical measurements (e.g., height, weight, temperature) and Geographic coordinates (e.g., latitude and longitude).

Assumes the Features are On the Same Scale or Normalized

Euclidean distance is sensitive to the magnitude of the variables. Features with larger scales can disproportionately influence the distance calculation, even if they are not more important. Therefore, feature normalization or standardization is often necessary to ensure all features contribute equally. Common techniques include:

- **Min-Max Scaling:** scales features to a range of [0, 1]

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- **Z-Score Normalization:** Transforms features to have a mean of 0 and standard deviation of 1

$$x' = \frac{x - \mu}{\sigma}$$

For example, in a dataset with features like income (measured in thousands) and age (measured in years), income would dominate the Euclidean distance unless scaled appropriately.

Ideal When the Variance of all Features is Similar

Euclidean distance assumes that the variance of the features is relatively uniform. Features with significantly different variances can distort the distance metric: Features with high variance contribute more to the distance, even if they are less relevant. Features with low variance may have minimal impact, even if they are important.

To address this:

- **Standardize the Features:** Normalize each feature to have a unit variance, ensuring equal contribution.
- **Assess Feature Importance:** Use domain knowledge or feature selection techniques to eliminate irrelevant features that could skew the distance calculation.

Manhattan Distance

Manhattan distance, also known as taxicab or city block distance, measures the distance between two points by summing the absolute differences of their coordinates. If you navigate a grid-like city, the Manhattan distance is the total blocks walked along streets (no diagonal shortcuts allowed).

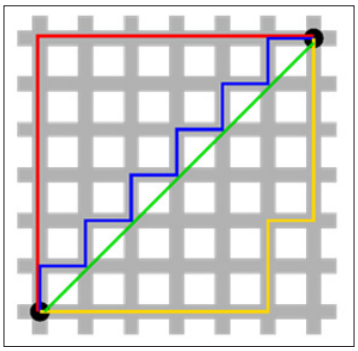

Figure 2: Manhattan Distance

In the above visual the blue line represents the Manhattan distance, and the green line represents the Euclidean distance.

For two points P = (x_1, y_1) and Q = (x_2 , y_2) in a 2D space, the Manhattan Distance is:

$$D_{\text{Manhattan}}(P,Q) = |x_2 - x_1| + |y_2 - y_1|$$

For higher dimensions (n - dimensional space):

$$D_{\text{Manhattan}}(P,Q) = \sum_{i=1}^{n} |q_i - p_i|$$

Appropriate for Continuous or Ordinal Data

Manhattan distance is well-suited for datasets with continuous or ordinal features. It measures the distance between two points by summing the absolute differences across dimensions, effectively calculating the "grid-based" or "city block" distance.

This makes it particularly effective when:

- The data represents quantities or magnitudes that are naturally independent along each dimension.
- The variables are ordinal, such as rankings, where the order matters but the exact differences between values may not be meaningful.

Examples

- Continuous Data: Measurements like temperature, age, or sales figures.
- Ordinal Data: Ratings on a scale (e.g., "low," "medium," "high") converted to numerical values.

Less Sensitive to Outliers Compared to Euclidean Distance

Manhattan distance is less influenced by outliers because it sums absolute differences rather than squaring deviations (as in Euclidean distance).

This property makes it robust in scenarios where:

- Data contains extreme values or irregularities that could skew distance calculations.
- The focus is on capturing overall deviations rather than amplifying the impact of large differences in specific dimensions.
- In a dataset with an extreme value in one feature (e.g., annual income ranging from $30,000 to $1,000,000), Manhattan distance minimizes the disproportionate influence of that feature compared to Euclidean distance. It ensures that neighbors selected for imputing a missing value are not dominated by a single extreme feature, leading to more balanced imputations.

Useful When Dimensions have Different Units or Scales

Manhattan distance does not inherently assume that features are on the same scale. However, its reliance on absolute differences means it is more interpretable when the features have varying units or scales. While scaling may still be necessary for fair comparison, it is often less critical than with Euclidean distance.

- Combining attributes like height (in inches) and weight (in pounds) without heavy preprocessing, as the method measures deviations independently for each feature.
- Comparing customer behaviors where one feature is transaction frequency (counts per year) and another is average transaction value (in dollars)

Manhattan distance is not suitable when the features in the dataset are highly correlated. If features are strongly correlated (e.g., height and weight), Euclidean or other metrics may better capture their relationship. When the relationship between features involves diagonal movement in the feature space, Manhattan distance may fail to capture the geometry.

Minkowski Distance

Minkowski distance generalizes both Euclidean and Manhattan distances by introducing a parameter p. It offers flexibility in measuring distance. By tuning the parameter p, Minkowski distance transitions between different forms

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

When p = 2 Minkowski reduces to Euclidean distance and when p = 1 it reduces to Manhattan.

Data Includes Mixed Distributions

If your dataset contains variables with different distributions (e.g., some Gaussian and some skewed), Minkowski distance allows you to tune 'p' to adapt to the characteristics of the data.

Handling High-Dimensional Data
In high-dimensional spaces, Minkowski distance can reduce overemphasis on outlier dimensions by choosing appropriate P values (e.g., P = 1 or P = 1.5).

Scaling or Normalization is Feasible
Minkowski distance assumes that all dimensions are equally important. If you can scale or normalize the features to ensure comparability, this distance metric becomes effective for imputation.

Need a Generalized Approach
If you're unsure whether Manhattan or Euclidean distance is more suitable for the data, Minkowski distance provides a unified framework to test both (and intermediate metrics) by varying p.

Cosine Metric
Cosine similarity measures the angle between two vectors, focusing on direction rather than magnitude. It is often used for high-dimensional data. Two vectors with similar directions but different magnitudes are considered close. Think of comparing the orientation of arrows regardless of their length [7].

$$\text{Similarity}(X, Y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Distance is computed as 1 - Similarity(X,Y)

High-Dimensional Datasets
In datasets with many features, such as text, documents, or embeddings, the Euclidean or Manhattan distances may not be effective because:

- **Dimensionality increases uniformly:** In high-dimensional spaces, most points tend to be equidistant, making it harder to distinguish "closeness" using metrics like Euclidean distance.
- Sparsity of Data: High-dimensional data is often sparse (e.g., text encoded as bag-of-words or TF-IDF vectors). In such cases, cosine similarity focuses on the overlap of non-zero dimensions, making it robust to sparsity.

Example: Text Embeddings
- Scenario: You have sentence embeddings for a set of documents or queries, where each embedding represents the semantic meaning of a text in a 300-dimensional vector space.
- Why Cosine Similarity Works: The semantic similarity between two documents depends on the alignment (direction) of their vectors, not their magnitudes. For instance:

Document A: [0.1, 0.2, 0.3, 0.4]
Document B: [0.2, 0.4, 0.6, 0.8]
Despite having different magnitudes, they point in the same direction, indicating high similarity.

When Magnitude of Features is Less Important
Cosine similarity disregards the magnitude (length) of the vectors and instead measures the angle between them, which represents their directional alignment. This is crucial when:

- **Feature Magnitudes Differ Significantly:** If features have different scales or units, Euclidean or Manhattan distances might overemphasize high-magnitude features, while cosine similarity remains unaffected.
- **Normalization:** Cosine similarity inherently normalizes vectors (dividing by their magnitudes), ensuring fair comparison regardless of scale.

Example: Text Classification
- **Scenario:** Comparing two news articles represented as word frequency vectors.
  Article A: [5, 0, 3, 8] (mentions "economy" 5 times, "market" 3 times, etc.)
  Article B: [15, 0, 9, 24] (same distribution but with higher counts).
- **Why Cosine Similarity Works:** Both articles focus on the same topics ("economy" and "market") but differ in verbosity. Cosine similarity captures their semantic similarity without being influenced by verbosity.

When Features Represent Semantic or Relational Information
Cosine similarity excels in capturing relational similarities, which are often crucial in applications involving embeddings or sparse data. For instance:

- **Document-Topic Relationships:** TF-IDF vectors or topic distributions often represent the importance of words in a document relative to a corpus. Cosine similarity identifies similar topics or contexts despite differences in word counts.
- **User-Item Preferences:** In recommendation systems, user preferences for items (e.g., ratings) can be represented as sparse vectors. Cosine similarity captures shared preferences without being influenced by absolute ratings.

Example: Recommendation Systems
- **Scenario:** Two users rate movies on a scale of 1–5. Their preferences can be represented as vectors:

  User A: [5, 0, 3, 0, 4] (likes action and drama).
  User B: [10, 0, 6, 0, 8] (same preferences but rates higher).

- **Why Cosine Similarity Works:** By focusing on shared preferences and ignoring magnitude, cosine similarity identifies their preferences as aligned.

Use in Sparse Data
Sparse datasets - where most features are zero - are common in text analytics, recommendation systems, and bioinformatics. Cosine similarity performs well here because:

- It only considers non-zero dimensions, effectively ignoring irrelevant features.
- It measures overlap in the active dimensions of the vectors, emphasizing meaningful relationships.

Example: TF-IDF Vectors
- A corpus of documents represented as TF-IDF vectors typically has thousands of dimensions (one per term), but most terms are absent in any given document. Cosine similarity focuses on terms that co-occur, identifying documents with similar content even if most terms differ.

When Not to Use Cosine Similarity
Cosine similarity is not ideal when:
- Magnitude of Features is Relevant: For datasets where the absolute values matter (e.g., income levels, temperature), metrics like Euclidean or Manhattan distance are better suited.
- Low Dimensionality: In datasets with few features, other metrics like Euclidean distance might be more interpretable and effective

Conclusion

The choice of distance metric significantly impacts the performance of KNN imputation. Understanding the nature of the dataset - whether it's numerical, categorical, high-dimensional, or prone to outliers - guides the selection of the most appropriate metric. Future work could explore hybrid approaches that combine multiple distance metrics to handle diverse data characteristics within the same dataset.

References

1. Geek for Geeks (2025) How to choose the distance metrice https://www.geeksforgeeks.org/how-to-choose-the-right-distance-metric-in-knn/.
2. KD Nuggets (2020) Most Popular distance metrics https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html.
3. Geek for Geeks (2025) K-Nearest Neighbor (KNN) Algorithm https://www.geeksforgeeks.org/k-nearest-neighbours/.
4. USTC Newly K Elissa (2020) K-Nearest Neighbors (KNN) Classification with Different Distance Metrics. https://www.ustcnewly.com/teaching/2020_2_3.pdf?utm_source=chatgpt.com.
5. Jason Brown Lee (2020) KNN Imputation for missing values in Machine learning https://machinelearningmastery.com/knn-imputation-for-missing-values-in-machine-learning/.
6. Scikit Lea, KNN Imputer scikit-learn 1.5.2 documentation https://scikit-learn.org/1.5/modules/generated/sklearn.impute.KNNImputer.html.
7. Arkopal Choudhury, Michael R Kosorok (2020) Missing Data Imputation for Classification Problems. https://arxiv.org/abs/2002.10709.