Compactness-Weighted KNN Classification Algorithm

Bengting Wan, Zhixiang Sheng*, Wenqiang Zhu, Zhiyi Hu
School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China

***Abstract*—The K-Nearest Neighbor (KNN) algorithm is a widely used classical classification tool, yet enhancing the classification accuracy for multi-feature large datasets remains a challenge. The paper introduces a Compactness-Weighted KNN classification algorithm using a weighted Minkowski distance (CKNN) to address this. Due to the variability in sample distribution, a method for deriving feature weights based on compactness is designed. Subsequently, a formula for calculating the weighted Minkowski distance using compactness weights is proposed, forming the basis for developing the CKNN algorithm. Comparative experimental results on five real-world datasets demonstrate that the CKNN algorithm outperforms eight existing variant KNN algorithms in Accuracy, Precision, Recall, and F1 performance metrics. The test results and sensitivity analysis confirm the CKNN's efficacy in classifying multi-feature datasets.**

Keywords*—K-nearest neighbors; feature weight; Minkowski distance; compactness

I. Introduction

In the domains of data science and machine learning, the KNN (K-Nearest Neighbors) algorithm, widely recognized as one of the top 10 classification algorithms [1], plays a crucial role in revealing the inherent patterns and structures of data, effectively grouping data points into distinct categories, especially in market segmentation, social network analysis, bioinformatics, image processing, and other fields [2, 3]. Since Fix and Hodges [4] introduced the KNN algorithm, KNN has emerged as a classic and efficient classification tool widely applied in data mining, data classification, and other fields [5, 6]. For instance, Uddin et al. [5] have applied the KNN algorithm for disease risk prediction, and Han et al. [6] have used it to estimate the photometric redshift of quasars.

However, the KNN algorithm faces several challenges in practice, such as the choice of k value, selection of nearest neighbors, nearest neighbor search, and determination of classification rules [7]. Consequently, researchers have proposed various improvement strategies to enhance the performance of KNN [8-10]. For example, Zhang and Li [8] bolstered the classification performance of the KNN algorithm and reduced computational costs through sparse learning and group lasso techniques. Meanwhile the weighted KNN [11] approach adjusts the influence of each neighbor on the classification decision by assigning different weights, enhancing the efficiency and accuracy of classification. In this method, weights are usually based on the distance or similarity of neighbors to the query point, giving closer neighbors more significant influence in classification decisions. This strategy effectively improves the algorithm's ability to handle uneven distributions or irregular data structures. Nevertheless, weighted KNN algorithms also face challenges. Firstly, selecting and calculating appropriate weights is a crucial issue, as different weight distribution strategies directly affect the accuracy of classification results. Secondly, the algorithm may encounter efficiency issues when handling large datasets, especially in scenarios requiring real-time or near-real-time processing [12]. In response, researchers have proposed various variants of the weighted KNN algorithm, such as the Improved K-Nearest Neighbor rule combining Prototype Selection and Local Feature Weighting (IKNN_PSLFW) algorithm developed by Zhang et al. [13], which combines prototype selection with local feature weighting, and the Option out-of-bag (Opt_OOB), a KNN ensemble learning method based on feature weighting and model selection proposed by Gul et al. [14] Chen and Hao [15] introduced a KNN prediction model based on a feature weight matrix by modifying the standard Euclidean distance. Chen and Gou [16] proposed a series of weighted distance functions for classifying attributes and applied these functions to develop nearest neighbor classifiers.

However, these weighted KNN algorithms, without considering datasets with non-uniform feature distributions, still have room for improvement in classification efficiency for unevenly distributed datasets and may be limited in handling high-dimensional classification problems. Therefore, this paper will consider the compactness of data distribution and introduce a dynamic weight adjustment mechanism based on feature compactness. By reconstructing the feature weights, a new weighted KNN algorithm is proposed. The main contributions of this paper are:

- Inspired by the uneven distribution of data, a weight calculation method based on compactness is proposed;
- Inspired by the weighted KNN, the CKNN algorithm based on weighted Minkowski distance is proposed;
- In real-world datasets, the CKNN algorithm was employed for classification purposes and was subsequently compared and analyzed against recent weighted KNN algorithms. Additionally, the CKNN algorithm underwent a sensitivity analysis along with Friedman's and Nemenyi's post-hoc tests. These evaluations demonstrated that the CKNN algorithm possesses specific superior performance characteristics.

The rest of this paper is organized as follows: Section II elaborates on the related research work of KNN; Section III details the construction process of the CKNN algorithm; Section IV implements the CKNN algorithm and compares it with other existing variant weighted KNN algorithms; finally, the

advantages and limitations of the CKNN algorithm are analyzed in this section. Finally, the paper is concluded in Section V.

II. Related Work

A. KNN

Based on similarity, the KNN classifier first identifies the nearest k neighbors of an unknown sample [4]. Then, it determines its category based on the most frequently occurring (highest probability) category among the K-Nearest Neighbor. Below is an outline of the basic principles of the KNN algorithm.

Given a set of labeled samples: $D = \{x_1, x_2, \dots, x_n\}$, a training set $D_T = \{(x_i, y_i)\}_{i=1}^{N}$ is constructed, where $x_n \in D$ is a point in n-dimensional space, y_i is the category label corresponding to x_i and N is the number of samples in the training set. For a query point q with an unknown category, KNN first calculates the Euclidean distance between this point and each sample point x_i in the training set, as shown in Eq. (1).

$$D(x_i, x_j) = \sqrt{\sum_{k=1}^{n}(x_{iv} - x_{iv})^2} \quad (1)$$

Within this framework, x_{iv} denotes the coordinate value of the i^{th}sample point along the v^{th}dimension. Subsequently, the computed distances for N points are arranged in ascending order, from which the nearest k neighbors are selected. At this juncture, the distance set comprising the k nearest neighbors to the query point q can be represented as $D_T^* = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{k}$. Subsequently, the category label of the query point q is predicted through the majority vote of its nearest neighbors, resulting in the target equation, as shown in Eq. (2).

$$predicted = argmax_{c \in C} \sum_{i=1}^{k} I(x_i = c) \quad (2)$$

In Eq. (2), C represents the set of all possible categories, k is the number of nearest neighbors, x_i is the category of the ith nearest neighbor, and $I(x_i = c)$ is an indicator function that equals 1 when y_i equals category c, and 0 otherwise. The KNN algorithm is described as shown in Algorithm 1.

Algorithm 1: KNN algorithm

Input: A test sample and some training samples
Output: The test sample's category
Process:
1. **For** number of training samples **do**
2. calculate the similarity between the test sample and a training sample
3. **End for**
4. find the k training samples that are most like the test sample
5. determine test sample's category

However, the classical KNN algorithm has faced several challenges: (1) Noise sensitivity. The KNN algorithm determines a sample's category based on the k nearest neighbors' labels, making it sensitive to noise and outliers. (2) Redundant sample testing. The nearest neighbors are searched for each test sample given, but this is not necessarily optimal for all test cases. (3) Dependence on the hyperparameter k. The performance of the algorithm varies with different k parameters. (4) Poor stability. The algorithm performs well on some datasets and poorly on others.

To address these issues, researchers have proposed various KNN variants. To tackle the challenge of classifying imperfect data in high-dimensional spaces, Gong et al. [17] improved the traditional KNN method by synchronizing neighborhood search and feature weighting, proposing an enhanced KNN algorithm—AEKNN. Bian et al. [18] improved the traditional fuzzy KNN [19] by adaptively selecting the optimal number of nearest neighbors (the k value) for each test sample Gou et al. [20] optimized the classifier by capturing the proximity and geometric characteristics of the K-nearest neighbors and learning the contribution of each neighbor to the classification of the test sample through linear representation methods, thereby reducing the algorithm's sensitivity to k and enhancing its performance.

B. Weighted Distance KNN

In response to issues such as class overlap and the difficulty in choosing the k factor, researchers have embarked on explorations and achieved many research results. For instance, Zhang et al. [13] have developed the IKNN_PSLFW algorithm, which combines prototype selection with local feature weighting. In this method, the prototype selection part divides the training set into multiple pure subsets, where each subset contains instances of only one class label. Following this, the scope of prototype selection and the weights of features are updated through local feature weighting, optimizing the objective function as illustrated in Eq. (3).

$$rw_s = max_{j=1,2,\dots,n} \sqrt{\sum_{j=1}^{d} w_s^j \left(w_s^j - x_i^j\right)^2} \quad (3)$$

In Eq.(3), $w^j = u^j / \sum_{j=1}^{d} u^j$, $u^j = \begin{cases} \frac{1}{v^j}, (v^j \neq 0) \\ 10 \times max\left\{\frac{1}{v^j} \middle| i \neq j, v^j \neq 0\right\}, (v^j = 0) \end{cases}$, and v^j is the variance of the j^{th} feature in the subset. Ultimately, a representative example is chosen from each subset as a prototype, and both the boundary of the subset and the total count of instances it includes are recorded. During the classification phase, depending on the weighted distance between an unknown instance and each prototype, three potential scenarios are identified: the instance falls within the range of a single prototype, within an overlap area, or outside the range of all prototypes, with different rules applied to predict its category accordingly. Throughout the process, there is no need to predetermine the value of k, and the time complexity of the IKNN_PSLFW algorithm is $o(n^2)$. This method effectively reduces the number of instances and overlap areas, thereby enhancing the accuracy and efficiency of classification.

Gul et al. [14] introduced an ensemble learning method for the k nearest neighbor, termed Opt_OOB, predicated on feature weighting and model selection. This method selects the best model by leveraging the out-of-bag prediction error to assemble the ultimate ensemble classification model. This

methodology establishes an objective function utilizing the distance Eq. (4).

$$D_w\left(X'_{1\times p'}, X_{1\times p'}\right) = \left\{\sum_{j=1}^{p'} w\left(x'_j - x_j\right)^2\right\}^{\frac{1}{2}} \quad (4)$$

In Eq. (4), feature weights w are determined by $w = argmax_w[min\{d_H(\psi(x))\}]$, to identify Out-of-Bag (OOB) observations during the bootstrap sampling process. Subsequently, the prediction error for each base model on its corresponding OOB observations is calculated. Models are then ranked according to the magnitude of OOB error, and a certain proportion of the best-performing models are selected to form the final ensemble classification model. This method reduces the dependency on the parameter k and ensures the diversity and accuracy of the ensemble model, although its performance may decrease on tiny datasets.

Chen and Hao [15] have proposed a K-nearest neighbors predictive model based on a feature weighting matrix by modifying the standard Euclidean distance. The core of this algorithm lies in improving prediction accuracy by altering the positional relations of sample points. The feature weighting matrix is shown in Eq. (5).

$$P = \begin{Bmatrix} InfoGain(f_1) & 0 & \dots & 0 \\ 0 & InfoGain(f_2) & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & InfoGain(f_n) \end{Bmatrix} \quad (5)$$

In Eq. (5), $InfoGain(A) = \sqrt{Info(\mathrm{D}) - Info_A(D)}$, $Info(D) = -\sum_{i\in\{-1,\ +1\}} \frac{|C_{\{i,D\}}|}{|D|} log(\frac{|C_{\{i,D\}}|}{|D|})$, and $Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} Info(D_j)$. Here, D represents the dataset, $|D|$ denotes the size of the dataset, and $C_{\{i,D\}}$ represents the subset of the dataset D that belongs to class C_i. This method has certain advantages for large-scale or complex datasets, but the effectiveness of the model depends on the accuracy of the feature weighting matrix, which requires sufficient prior knowledge or data analysis to determine appropriate weights.

In pursuit of an optimal distance metric for precisely quantifying the dissimilarities among classified samples, Chen and Gou [16] introduced a series of weighted distance functions tailored for categorical attributes, which have been applied to advance nearest neighbor classifiers. The Global Gini K-nearest neighbors (GGKNN) incorporate a weighting scheme as depicted in Eq. (6).

$$\omega_d^{(GG)} = e^{-\frac{M}{M-1}\sum_{s_d\in s_d} P(S_d)\times GG(S_d)} \quad (6)$$

In Eq. (6), $GG(s_d) = -\sum_{m=1}^{M} P(m|s_d)\log_2 p(m|s_d)$, $p(s_d) = \frac{1}{N}\sum_{(X,y)\in tr} I(x_d = s_d)$,and $p(m|s_d) = \frac{\sum_{(X,y)\in c_m} I(x_d=s_d)}{\sum_{(X,y)\in tr} I(x_d=s_d)}$. Herein, tr represents the training dataset, M denotes the number of classes contained within the training dataset, $|s_d|$indicate the discrete values of the d^{th} attribute. The weighting for Global Entropy K-nearest neighbors (GEKNN) is shown in Eq. (7).

$$\omega_d^{(GE)} = e^{-\frac{1}{log_2 M}\sum_{s_d\in s_d} P(S_d)\times GE(S_d)} \quad (7)$$

In Eq. (7), $GE(s_d) = 1 - \sum_{m=1}^{M}[P(m|s_d)]^2$. These methods use global statistical approaches to weight attributes, considering the information from all data points to determine the importance of each attribute. Conversely, the weighting for Local Gini K-nearest neighbors (LGKNN) is shown in Eq. (8).

$$\omega_{md}^{(LG)} = e^{-\frac{|s_d|}{|s_d|-1}\times LG(m,d)} \quad (8)$$

In Eq. (8), $LG(m,d) = 1 - \sum_{s_d\in s_d}[P(s_d|m)]^2$,$p(s_d|m) = \frac{1}{|c_m|}\sum_{(X,y)\in c_m} I(x_d = s_d)$,and c_m represents the m^{th} class within the training dataset tr. The weighting for Local Entropy K-nearest neighbors (LEKNN) is shown in Eq. (9).

$$\omega_{md}^{(LE)} = e^{-\frac{1}{log_2|s_d|}\times GE(m,d)} \quad (9)$$

In Eq. (9), $LE(m,d) = -\sum_{s_d\in s_d}(s_d|m)\log_2 p(s_d|m)$. This employs a local method for computing weights, meaning that it adjusts attribute weights based on the local information surrounding each data point. This method achieves soft feature selection for categorical data, thereby improving the quality of classification.

Furthermore, researchers have introduced multiple variations of the KNN algorithm that utilize weighted distance measures for optimization. Açıkkar and Tokgöz [21] have enhanced the conventional KNN algorithm by introducing a new weighted voting mechanism and adaptive k-value selection techniques. These modifications have improved the performance of the KNN algorithm in scenarios with complex or nonlinear decision boundaries, especially in the context of processing datasets with noise or outliers.

III. Compactness-Weighted K-NN Classification Algorithm

This section delineates an improved K-Nearest Neighbors algorithm (CKNN) predicated on compactness and local feature weighting, devised to augment the classification efficacy of the KNN algorithm. The research methodology unfolds in two pivotal steps: Initially, the compactness for each feature is ascertained, forming the groundwork for the recalibration of feature weights. After that, a CKNN algorithm, hinged on compactness, is introduced.

In most pattern recognition tasks, the relevance of different features differs, particularly in classification tasks. Even if all features in the dataset are relevant, their degrees of relevance may vary. To address this, we propose the concept of feature compactness, as illustrated in Fig. 1.

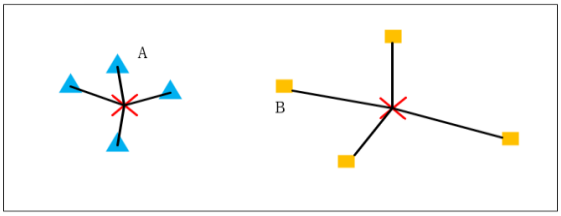


Fig. 1. Feature Compactness, with Feature A having greater compactness than Feature B.

Fi. 1 shows that the sum of distances between the elements in set A and their centroid is significantly less than that in set B, leading to the conclusion that feature A possesses greater compactness than B Various distance metrics are utilized to measure feature compactness, such as Euclidean distance, Manhattan distance, and Minkowski distance. The Minkowski distance [22], in particular, allows for adjusting the parameter p according to different scenarios and is widely employed. Inspired by this, the article adopts the Minkowski distance to measure the distances of features. Within a dataset $X = \{x_1, x_2, \dots, x_n\}$ comprising n samples, each with m features, x_i represents a feature vector of dimension m. Assuming the centroid vector c represents the arithmetic mean of all sample point feature vectors, the compactness c_j for the j^{th}feature, based on the Minkowski distance, is defined as the Minkowski distance between the values of all sample points for that feature and the value of the centroid for that feature. The calculation is as shown in Eq. (11).

$$C_j = \sum_{j=1}^{n} \left| x_{ij} - c_j \right|^p \tag{11}$$

In Eq. (11), n represents the total number of samples in the cluster, C_j denotes the compactness of the j^{th} feature, x_{ij} is the value of the j^{th} feature for the i^{th} sample, C_j is the value of the j^{th} feature of the cluster centroid, and p is the exponent parameter of the Minkowski distance. From Eq. (11), it is inferred that, under the conditions of a given number of samples and a defined centroid, a smaller value of C_j indicates greater compactness, and vice versa.

Specifically, for a given dataset and its corresponding centroid, the initial step involves calculating the difference between each data point and the centroid across all dimensions. Subsequently, these differences are raised to the p^{th} power using the Minkowski formula, where p is a predefined parameter. The steps for solving compactness will be detailed in Algorithm 2.

Algorithm 2: Calculate Compactness (CL, P)

Input: CL: feature vector
P: Minkowski index

Output: A one-dimensional array containing the disper- sion of each feature $S = \{s_1, s_2, \dots, s_k\}$

Process:

1. Set C←The arithmetic mean of the eigen vectors of all points in the cluster
2. **For** each feature in the feature space **do**
3. Add S←Discrete degree calculated by Eq. (11)
4. **End for**

To address the discrepancy that arises from the assumption in the canonical KNN algorithm, where each feature is assigned an equal weight reflecting an assumption of equal contribution to the decision-making process—a scenario often divergent from real-world applications where the importance of features can vary significantly. This study introduces a methodology grounded in compactness to determine the weights of different features. Inspired by the findings in study [23] and assuming a given dataset is presumed to contain K categories, with each category corresponding to a distinct cluster, this paper proposes a novel objective function. This function represents the sum of weighted averages across different classification sets, as calculated in Eq. (12).

$$\mathrm{J} = \sum_{c=1}^{K} \sum_{v=1}^{V} w_{cv}^{\beta} C_{cv} \tag{12}$$

where, K represents the total number of categories, V represents the total number of features, w_{cv} is the weight of the v^{th} feature in the c^{th} category, and β is a weight adjustment parameter. The adjustment parameter β is used to control the extent of the weight influence. When β>1, it indicates a higher emphasis on features with high weights, when β=1, the model degenerates to a traditional equal-weight model. C_{cv} is an indicator measuring the compactness of the v^{th} feature in the c^{th} category, calculated by $\sum_{v=1}^{n} ||x_{cv} - c_v||^p$,, where x_{cv} is the v^{th} feature value of the c^{th} sample, and n represents the total number of samples in the cluster.

By assigning different weights to various classes, the goal is to minimize the weighted average compactness within each class, thereby improving the compactness of classification. To this end, Eq. (12) can be converted into Eq. (13). Within this framework, the degree of classification compactness can be obtained, and the weights of each feature, w_{cv}, can be solved.

$$\sum_{c=1}^{K} \sum_{v=1}^{V} w_{cv}^{\beta} C_{cv} = \sum_{c=1}^{K} \sum_{v=1}^{V} \{ w_{cv}^{\beta} \sum_{v=1}^{n} ||x_{cv} - c_v||^p \} \tag{13}$$

Considering the weights of different features satisfy the constraints: $\sum_{v=1}^{V} w_{cv} = 1$ and $w_{cv} \geq 0$, it is evident that Eq. (13) represents a nonlinear programming equation while also satisfying convex function constraints. To enhance the compactness within classes by optimizing feature weights, the Lagrangian function L is employed to minimize Eq. (14):

$$L = \sum_{v=1}^{V} w_{cv}^{\beta} C_{cv} + \lambda \left(1 - \sum_{v=1}^{V} w_{cv}^{\beta}\right) \tag{14}$$

Taking the partial derivative of w_{cv}^{β} in Eq. (14), and then setting it to zero to find the extremum, as shown in Eq. (15):

$$\frac{\partial L}{\partial w_{cv}} = \beta w_{cv}^{\beta - 1} C_{cv} - \lambda = 0 \tag{15}$$

Solving Eq. (15) yields the weight w_{cv}, as shown in Eq. (16):

$$w_{cv} = \left(\frac{\lambda}{\beta C_{cv}} \right)^{\frac{1}{\beta - 1}} \tag{16}$$

Given the weight constraints $\sum_{v=1}^{V} w_{cv} = 1$ and $w_{cv} \geq 0$, Eq. (16) can be further derived to obtain Eq. (17).

$$\sum_{v=1}^{V} \left(\frac{\lambda}{\beta C_{cv}} \right)^{\frac{1}{\beta - 1}} = 1 \Leftrightarrow \left(\frac{\lambda}{\beta} \right)^{\frac{1}{\beta - 1}} = \frac{1}{\sum_{v=1}^{V} \left(\frac{1}{C_{cv}} \right)^{\frac{1}{\beta - 1}}} \tag{17}$$

Simplifying Eq. (17) gives the formula for solving weight w_{cv}, as shown in Eq. (18). From Eq. (18), it can be seen that under compact classification, the weight of feature v in category C can be obtained by solving the Minkowski distance.

$$w_{cv} = \frac{1}{\sum_{u=1}^{V} \left(\frac{C_{cv}}{C_{cu}} \right)^{\frac{1}{\beta - 1}}} \tag{18}$$

Building on Eq. (18), it can be determined that the weights of various features can be calculated given a classification. However, in the KNN classification process, both the classification and the weights are the objectives to be determined. Inspired by the varying importance of different features and the concept of compactness as discussed in references, a weighted Minkowski distance based on compactness weights is proposed, as shown in Eq. (19).

$$d_w(x_i, x_j) = \sqrt[p]{\sum_{v=1}^{V} w_{cv}(x_{iv} - x_{jv})^p} \quad (19)$$

In Eq. (19), for given data samplesx_i,x_j∈C, where β is a user-defined parameter, $w_{cv}$is the weight of feature weight v. In this case, the weight of each feature no longer depends on a specific cluster but is based on the feature distribution across the entire dataset. Feature weights should be non-negative and satisfy $\sum_{v=1}^{V} w_{cv} = 1$, and $w_{cv} \geq 0$.

Leveraging the concept of compactness-weighted distances within the framework of the KNN algorithm, this section introduces the Compactness-weighted KNN (CKNN) algorithm. The CKNN algorithm begins by calculating the compactness of each feature in the dataset according to Eq. (11). This calculation necessitates using the Minkowski distance measure to ascertain the compactness of each feature relative to its centroid. Drawing on the principle of compactness, the weights for each feature can be determined using Eq. (18). Subsequently, a compactness-weighted Minkowski distance, as delineated in Eq. (19), is constructed to facilitate the computation of distances between samples. Ultimately, the CKNN algorithm replaces the Euclidean distance traditionally employed in KNN with the weighted Minkowski distance, selects the K-nearest neighbors based on this distance, and utilizes a voting mechanism predicated on the category labels of these neighbors to ascertain the category of the target sample. The steps to implement the CKNN algorithm are outlined in Algorithm 3.

Algorithm 3: Compactness-weighted KNN algorithm

Input: D_T: Training data set,
K: The number of nearest neighbors
T: Test Dataset
β: Minkowski index

Output: Y: classification result

Process:

Set $Y \leftarrow \emptyset$, $C \leftarrow$calculateCompactness(CL,P), $w \leftarrow \frac{1}{V}$
For The weight of each feature w **do**
    Update the weight w_{cv} of each feature through Eq. (18)
End For
For Each sample in the test dataset T **do**
    Set list ← $\emptyset$
For Training data set D_T **do**
    Compute weighted Minkowski distance by Eq. (19).
    Add distance to list
    Assign test sample category by majority vote from K nearest neighbors.
    Add Classification Results to Y
End For
Return Y as the classification results for all samples in T

IV. Algorithm Implementation

This section elucidates the datasets employed by the CKNN algorithm, the performance evaluation metrics utilized, and an analysis of the experimental outcomes. The experiments were conducted on a computer with a 12th Gen Intel(R) Core (TM) i7-12700H CPU, clocked at 2.70GHz, and 16.0GB RAM, running the Windows 11 operating system. The Python3.10 programming language executed the implementation.

A. Dataset and Evaluation Metrics

The implementation adopted five datasets from the UCI Machine Learning Repository (Wine, Breast Cancer, Promoters, Mc2, Car) as benchmark datasets (http://archive.ics.uci.edu/). Table I shows the essential characteristics of the five datasets, including the total number of samples, the number of features, and the number of classes. For datasets with some features as strings, traditional label encoding methods will be used. The Car dataset comprises 1728 samples, representing a multi-sample dataset. The Wine, Mc2, Promoters, and Breast Cancer datasets consist of 13, 39, 57, and 30 feature attributes, thus categorizing them as high-dimensional datasets. Wine and Car datasets have three and four categories, respectively.

To evaluate the classification results, this paper uses four evaluation metrics: Accuracy, Recall, Precision, and F1 (F1-measure) to measure the performance of algorithms. Among them, Recall refers to the ratio of correctly predicted positive instances to positive instances; precision refers to the ratio of correctly predicted positive instances to optimistic predictions. Based on the F1 measure, the experiments used macro-F1 (Macro-F1, the average F1 values within classes) for evaluation [22]. All these indicators range from [0, 1], with values closer to 1 indicating better model performance. Accuracy, precision, recall, and F1 are shown in Eq. (20), (21), (22), and (23) respectively. TP, TN, FP, and FN represent the proportions of true positives, true negatives, false positives, and false negatives in the result data.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (20)$$

$$Precision = \frac{TP}{TP+FP} \quad (21)$$

$$Recall = \frac{TP}{TP+FN} \quad (22)$$

$$F1 = \frac{2 \times Recall \times Precision}{Recall+Precision} \quad (23)$$

TABLE I. Basic Characteristics of the Datasets Used in the Experiment

Item	Dataset	Instances	Features	Classes
1	Wine Dataset	178	13	3
2	Mc2 Dataset	161	39	2
3	Car Dataset	1728	6	4
4	Promoters Dataset	106	57	2
5	Breast Cancer Dataset	699	30	2

B. Analysis of Results

The article selects eight existing improved KNN classification algorithms (KNN [4], FWKNN [15], LEKNN [16], LGKNN [16], GEKNN [16], GGKNN [16], IKNN_PSLFW [13], and Opt_OOB [14]) for comparison with CKNN. Among them, KNN represents the classic K-Nearest Neighbor algorithm. FWKNN is a K-Nearest Neighbor prediction model based on a feature weighting matrix. LEKNN calculates feature weights through local entropy, while LGKNN calculates feature weights through local Gini. GEKNN uses global entropy to calculate feature weights, and GGKNN uses global Gini for the same purpose. IKNN_PSLFW is an ensemble learning method based on prototype selection combined with local feature weighting, and Opt_OOB is a K-Nearest Neighbor ensemble learning method based on feature weighting and model selection. The implementation results are shown in Table II (the highest values for each dataset are indicated in bold).

Table II shows that CKNN exhibits superior performance, especially on the Promoters dataset, where its accuracy reached 0.8439, significantly higher than other algorithms. After comparing the performances of different algorithms across multiple datasets, it was observed that the proposed method demonstrates superiority in all evaluation metrics. Specifically, on the Wine dataset, compared to the Opt_OOB algorithm, CKNN showed improvements of 3.71% in Accuracy, 3.69% in the Recall, 3.42% in Precision, and 3.64% in F1 score, the improvement on the MC2 dataset was even more significant, with CKNN surpassing the LEKNN algorithm by 2.04% in Accuracy. Although the improvements in Recall, Precision, and F1 score were closer, they still reflected our algorithm's advantage. On the Car dataset, compared to the FWKNN algorithm, the improvement was particularly notable, with increases of 1.54% in Accuracy, 2.81% in Recall, 11.17% in Precision, and 7.06% in F1 score. On the Promoters dataset, compared to the second-ranked KNN, there were increases of 6.26% in Accuracy, 6.27% in Recall, 6.25% in Precision, and 6.26% in F1 score. Regarding the Breast Cancer dataset, the method also demonstrated its superior performance. Compared to the IKNN_PSLFW algorithm, CKNN improved by 0.58% in Accuracy, 0.47% in Recall, 0.81% in Precision, and 0.62% in F1 score.

To further comprehensively evaluate the performance of the CKNN algorithm, based on the implementation results in Table II, the following will analyze the Sum of Ranking Differences (SRDs) [24], Friedman test [26], Nemenyi test [27], and Bonferroni correction [28].

First, a comparative analysis of the Sum of Ranking Differences (SRDs) was conducted, a multi-criteria decision-making method that achieves evaluation objectives by calculating the sum of absolute differences between each algorithm's actual rankings and reference rankings. Table II presents the values of four evaluation metrics for various algorithms across five datasets; according to the SRDs method, the reference vector contains 20 elements, each of which is the best score among the algorithms. After scaling the SRD values to the [0, 100] interval, their theoretical distribution approximates a normal distribution. Thus, the normal quantiles of each algorithm can serve as the actual SRD values compared to the reference vector, with the implementation results shown in Fig. 2. The scaled Sum of Ranking Differences (SRD) values are plotted on the x-axis and the left y-axis, while the right y-axis displays the relative frequency (black curve). The Gaussian fitting parameters are m=66.72, s=9.87. The SRD values at the 5% probability level (XX1), the median (Med), and 95% (XX19) are also provided.

TABLE II. COMPARISON OF NINE ALGORITHMS ON DIFFERENT DATASETS

Dataset	Methods	Accuracy	Recall	Precision	F1	RANK
Wine	Proposed	**0.963**	**0.963**	**0.961**	**0.963**	**1**
	KNN	0.740	0.726	0.726	0.726	3
	FWKNN	0.740	0.726	0.726	0.726	4
	LEKNN	0.537	0.532	0.538	0.534	8
	LGKNN	0.648	0.644	0.652	0.644	7
	GEKNN	0.444	0.425	0.488	0.402	9
	GGKNN	0.648	0.645	0.680	0.656	6
	IKNN	0.740	0.730	0.723	0.722	5
	Opt_OOB	0.925	0.926	0.926	0.926	2
Mc2	Proposed	**0.714**	0.613	**0.613**	**0.613**	**1**
	KNN	0.612	0.546	0.537	0.534	6
	FWKNN	0.612	0.546	0.537	0.534	7
	LEKNN	0.693	0.600	0.595	0.597	2
	LGKNN	0.653	0.516	0.517	0.517	8
	GEKNN	0.673	0.558	0.558	0.558	4
	GGKNN	0.673	0.558	0.558	0.558	4
	IKNN	0.673	**0.614**	0.596	0.600	3
	Opt_OOB	0.693	0.543	0.554	0.545	5
Car	Proposed	**0.942**	**0.859**	**0.909**	**0.879**	**1**
	KNN	0.859	0.633	0.766	0.681	4
	FWKNN	0.926	0.831	0.797	0.809	2
	LEKNN	0.778	0.419	0.432	0.415	8
	LGKNN	0.724	0.465	0.606	0.493	6
	GEKNN	0.791	0.485	0.528	0.500	5
	GGKNN	0.791	0.485	0.528	0.500	5
	IKNN	0.774	0.390	0.543	0.414	7
	Opt_OOB	0.890	0.667	0.849	0.715	3
Promoters	Proposed	**0.843**	**0.845**	**0.843**	**0.843**	**1**
	KNN	0.781	0.782	0.781	0.781	2
	FWKNN	0.750	0.752	0.752	0.750	3
	LEKNN	0.718	0.727	0.741	0.716	5
	LGKNN	0.718	0.723	0.726	0.718	6
	GEKNN	0.656	0.660	0.662	0.655	8
	GGKNN	0.656	0.660	0.662	0.655	8
	IKNN	0.687	0.690	0.690	0.687	7
	Opt_OOB	0.750	0.749	0.749	0.749	4
Breast Cancer	Proposed	**0.976**	**0.971**	**0.978**	**0.974**	**1**
	KNN	0.941	0.933	0.933	0.936	5
	FWKNN	0.941	0.933	0.933	0.936	4
	LEKNN	0.614	0.562	0.570	0.561	9
	LGKNN	0.713	0.660	0.695	0.666	7
	GEKNN	0.731	0.674	0.722	0.682	6
	GGKNN	0.660	0.619	0.629	0.621	8
	IKNN	0.970	0.966	0.970	0.968	2
	Opt_OOB	0.953	0.946	0.952	0.949	3

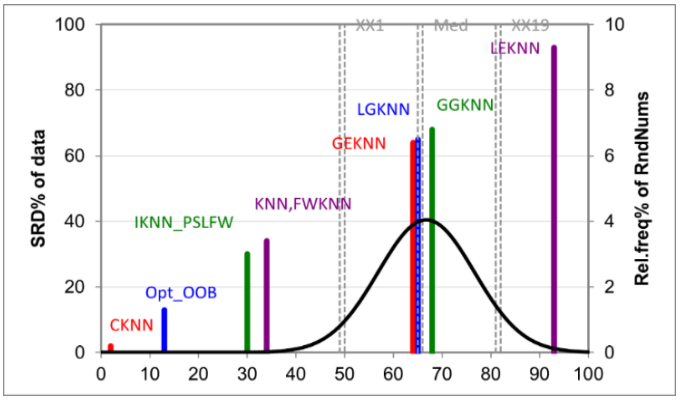


Fig. 2. Evaluation of algorithms using the sum of rank differences.

As can be discerned from Fig. 2, the CKNN algorithm is positioned on the left side of the curve, indicating that CKNN is the algorithm closest to the ideal state. At the same time, CKNN is at a certain distance compared to Opt_OOB and IKNN_PSLFW, signifying a clear advantage of CKNN over Opt_OOB and IKNN_PSLFW. Moreover, aside from GEKNN, LGKNN, GGKNN, and LEKNN, the ranking of the remaining five algorithms shows a significant difference from random ranking (α=0.05).

To highlight the advantages of CKNN, this paper further conducts a Friedman test [26]. Based on the Accuracy, Precision, Recall, and F1 metrics of CKNN, Table III presents the Friedman statistic F_F and the corresponding p-values for KNN, FWKNN, LEKNN, LGKNN, GEKNN, GGKNN, IKNN_PSLFW, and Opt_OOB in terms of accuracy, precision, recall, and F1 metrics. Table III shows that the null hypothesis (i.e., all compared algorithms will have equivalent performances) is significantly rejected at the significance level of α=0.05 for each evaluation metric, meaning there is a significant difference between CKNN and the other algorithms. However, it does not specify which algorithms are superior or inferior.

To further observe the differences among algorithms, this paper uses the Nemenyi test to assess the competitiveness of algorithms. In this test, if the difference in average ranks between two classifiers reaches at least the critical difference $CD=q_\alpha\sqrt{\frac{k(k+1)}{6N}}$, it is considered that there is a significant difference in performance between these two classifiers. At a significance level of α=0.05, q_α is 3.102, and the CD value is 5.369 (where k=9, N=5). Fig. 3 presents the CD diagram of the nine algorithms under Accuracy, Precision, Recall, and F1 metrics. In Fig. 2, any algorithm whose average rank is within a CD interval of CKNN is highlighted with a red line to show its association; otherwise, it indicates a significant performance difference from CKNN. For example, in recall, CKNN's average rank is 1.20, and with the addition of the CD value, the critical value becomes 6.57. At this point, LGKNN and GEKNN, with average ranks of 7.20 and 6.70 respectively, perform poorly. However, for algorithms within the CD interval, it is currently not impossible to determine the performance gap between them and CKNN.

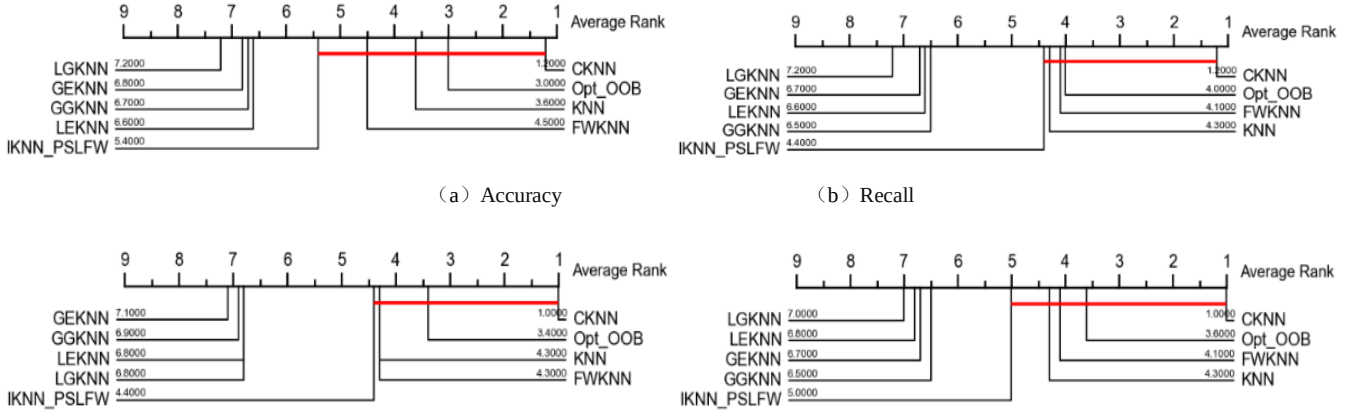
Based on the Nemenyi test, this paper uses the Bonferroni correction [27] to control the type I error (i.e., falsely rejecting a true null hypothesis). Let $\Delta_\xi=\bar{\xi}_{algorithm}-\bar{\xi}_{CKNN}$, when Δ_ξ is more excellent than CD_α, it is marked with "Y", indicating that CKNN outperforms the corresponding algorithm on the respective metric; otherwise, it is not marked. At a significance level of α=0.05, the critical value q_α becomes 2.724.

As shown in Table IV, the Bonferroni assessment results indicate that CKNN's performance exceeds that of LEKNN, LGKNN, GGKNN, and GEKNN algorithms.

TABLE III. SUMMARY OF THE FRIEDMAN STATISTIC F_F (K = 9, N = 5)

Evaluation Criteria	F_F	Critical Value (α=0.05)
Accuracy	22.86	15.51
Recall	19.96	
Precision	23.16	
F1 score	21.30	

Note: k represents the number of algorithms being compared; N represents the number of datasets



（a）Accuracy （b）Recall

（c）Precision （d）F1 score

Fig. 3. Nemenyi test of CKNN (control algorithm) with other variant KNN algorithms.

TABLE IV. COMPARISON OF CKNN WITH OTHER VARIANT KNN ALGORITHMS

	Accuracy	Recall	Precision	F1
KNN	--	--	--	--
Opt_OOB	--	--	--	--
FWKNN	--	--	--	--
IKNN_PSLFW	--	--	--	--
LEKNN	Y	Y	Y	Y
LGKNN	Y	Y	Y	Y
GEKNN	Y	Y	Y	Y
GGKNN	Y	Y	Y	Y

Confidence intervals [25] are employed to assess the degree of performance improvement among different algorithms. This paper utilizes confidence intervals to evaluate the performance of CKNN against eight compared variant KNN algorithms. Confidence intervals for comparisons among the eight algorithms were constructed to quantify these differences, assuming normality for the ranking differences as depicted in Eq. (24).

$$\frac{\Delta\xi}{\sqrt{\frac{k(k+1)}{6N}}} \sim N(0,1) \tag{24}$$

At a 95% confidence level, Fig. 4 shows the confidence intervals for Accuracy, Recall, Precision, and F1 metrics for the nine algorithms. From Fig. 4, it is observed that except for KNN, IKNN_PSLFW, and FWKNN, all intervals for Opt_OOB, LEKNN, LGKNN, GGKNN, and GEKNN appear to be less than 0, indicating significant differences between these algorithms and CKNN. For KNN, IKNN_PSLFW, and FWKNN, although the upper bounds of some evaluation metrics' confidence intervals are more significant than or close to 0, the estimated parameter values within the confidence intervals remain below 0, suggesting that CKNN, on the whole, outperforms KNN, IKNN_PSLFW, and FWKNN, with IKNN_PSLFW showing the closest performance to CKNN.

From the analyses based on the Sum of Ranking Differences (SRDs), Friedman test, Nemenyi test, and Bonferroni correction, it is evident that the CKNN algorithm outperforms the compared algorithms, including KNN, FWKNN, LEKNN, LGKNN, GEKNN, GGKNN, IKNN_PSLFW, and Opt_OOB in terms of performance.

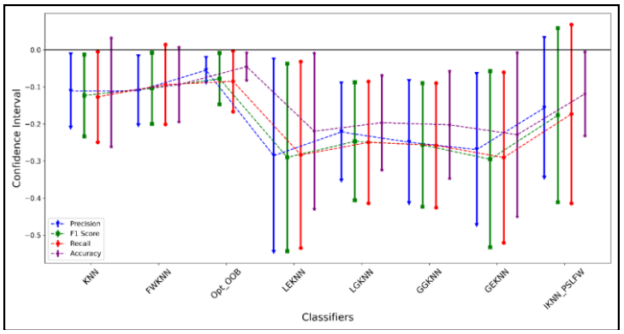
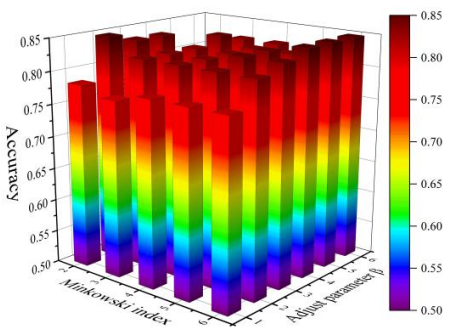


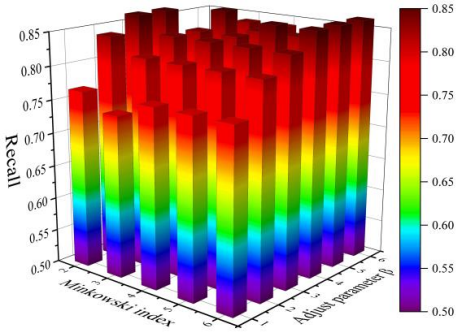
Fig. 4. Confidence intervals for rank differences.

C. Sensitivity Analysis

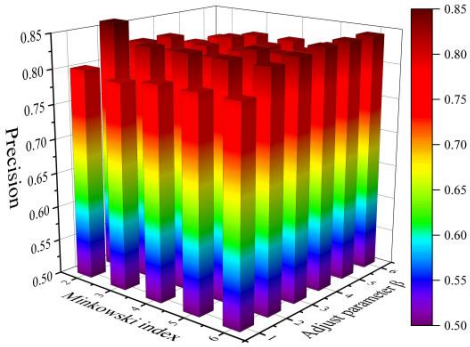
This section, using the Promoters dataset as an example, will analyze the impact of the Minkowski exponent (p-value) and the tuning parameter β on the performance of the proposed CKNN algorithm. The importance of p and β values in affecting the classifier's performance will be demonstrated through specific experimental results, which are displayed in Fig. 5.



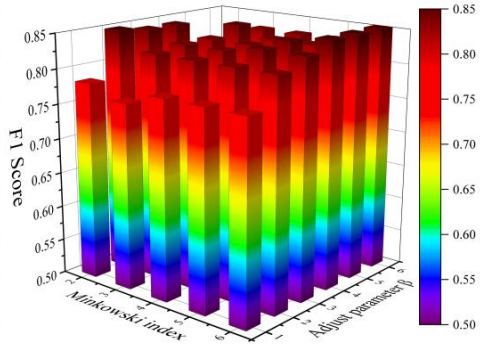
(a) The effect of different p and β on Accuracy.



(b) The effect of different p and β on Recall.



(c) The effect of different p and β on Precision.



(d) The effect of different p and β on F1 score.

Fig. 5. The effect of different Minkowski indices and tuning parameter β on classifier performance.

From Fig. 5, it can be observed that (1) For a specific value of β, the trend of accuracy increasing with an increase in p is quite apparent. The highest accuracy combination occurs at β=6 and p=4, 5, 6, with accuracies all reaching 0.8438. This indicates that a higher combination of β and p values is more likely to produce higher accuracies in this data group. Despite some fluctuations, a general trend can still be seen accuracy tends to increase with an increase in the value of p. The effect of the tuning parameter β seems less direct. However, it can be observed that when the value of the tuning parameter β reaches 6, the accuracy reaches a higher level, especially at higher p values. (2) Recall rates show a certain upward trend with the p increase. Especially at p=3 and subsequent values, recall rates are relatively high, notably at β=2 and β=5, 6, indicating that an increase in p has a positive effect on enhancing recall rates. At β=2, 5, 6 and p=3,4, the recall rates all reached the highest value of 0.8824. Overall, as p increases, there is a trend for an increase in recall rates, although this trend exhibits some fluctuations under different β values. (3) At β =2, precision increases significantly with p, reaching a peak (0.875), then decreasing. For other β values, precision does not vary much across different p values, but overall, when β increases to 6, precision reaches its highest at p=4, 5, 6. This suggests that larger values of β and p might be more beneficial for increasing precision in this specific model. Generally, precision tends to improve with an increase in β, especially at higher p values. (4)F1 score varies under different combinations of β and p. Especially at β=2, the F1 score corresponding to p significantly surpasses other p values, showing the highest score at 0.8485. At β=6 and p=4, 5, 6, the highest F1 scores were observed, each being 0.8571. This finding aligns with previous analyses of precision, suggesting that the model's overall performance may be better with larger values of β and p.

Therefore, CKNN performance metrics (Accuracy, Recall, Precision, and F1 score) generally improve with the increase of the parameter p and perform optimally at larger β values. Reasonable adjustment of the Minkowski index and the tuning parameter β can further optimize the classification performance of the CKNN algorithm.

V. Conclusion

This study proposes an improved K-nearest neighbor (KNN) classification algorithm based on compactness weights, which initially updates feature weights by calculating the compactness of each feature and then employs a compactness-weighted Minkowski distance to calculate the distances between samples, serving as the basis for classification decisions. Experimental results indicate that the CKNN algorithm surpasses traditional KNN and variant KNN algorithms in Accuracy, Recall, Precision, and F1 scores across the selected five datasets, notably showing significant performance improvements on the Promoters dataset.

The analysis of experimental results suggests that when the Minkowski exponent is two and the tuning parameter β is 2, the CKNN algorithm achieves relatively better classification effects. The CKNN algorithm can better balance the local and global information between samples, enhancing classification accuracy. Additionally, the overall results from the SRDs ranking, Friedman test, Nemenyi test, and Bonferroni correction analysis of the CKNN algorithm are superior to those of the compared variant KNN algorithms, confirming the better performance of the CKNN algorithm. Sensitivity analysis results indicate that the performance of the CKNN algorithm is jointly influenced by the Minkowski exponent and the tuning parameter β, and an appropriate selection of these parameters can further enhance the algorithm's performance.

Although the CKNN algorithm proposed in this study enhances the performance of the KNN algorithm, the performance of the KNN algorithm performance remains a key area of research. Therefore, future research will focus on further enhancing the scalability of the K-Nearest Neighbors (KNN) algorithm in large-scale datasets and real-time applications, with an emphasis on exploring parallel processing and distributed computing technologies to improve the efficiency of KNN in big data scenarios. At the same time, by combining the ability of deep learning models to automatically extract features and optimize weights, the KNN algorithm is expected to perform more effectively in handling high-dimensional and unstructured data.

Acknowledgment

This paper is upheld by the Ministry of Education Humanities and Social Sciences Planning Fund Project under Grant 22YJA880051, and in by National Science Foundation of China under Grants 72261016，in part by the Department of Education of Jiangxi Province of China under Grant GJJ2200535, 22YB052.

Data availability

Data will be made available on request.

Conflicts of Interest

These authors state that there have been no competing interests among them.

References

[1] K. Taunk, S. De, S. Verma and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, Madurai, India, 2019, pp. 1255-1260.

[2] Z. Li, H. Wang, S. Zhang, W. Zhang, and R. Lu, "SECKNN: FSS-Based Secure Multi-Party KNN Classification under General Distance Functions," IEEE Transactions on Information Forensics and Security, vol. 19, pp. 1326–1341, Jan. 2024.

[3] M. M. Abualhaj, A. A. Abu-Shareha, Q. Y. Shambour, A. Alsaaidah, S. N. Al-Khatib, and M. Anbar, "Customized K-nearest neighbors' algorithm for malware detection," International Journal of Data and Network Science, vol. 8, no. 1, pp. 431–438, Jan. 2024.

[4] E. Fix and J. L. Hodges, "Discriminatory analysis: Nonparametric discrimination: Consistency properties," PsycEXTRA Dataset. Jan. 01, 1951.

[5] S. Uddin, I. Haque, H. Lu, M. A. Moni, and E. Gide, "Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction," Scientific Reports, vol. 12, no. 1, Apr. 2022.

[6] B. Han, L.-N. Qiao, J.-L. Chen, X.-D. Zhang, Y. Zhang, and Y. Zhao, "GeneticKNN: a weighted KNN approach supported by genetic algorithm for photometric redshift estimation of quasars," Research in Astronomy and Astrophysics/Research in Astronomy and Astrophysics, vol. 21, no. 1, p. 017, Jan. 2021.

[7] S. Zhang, "Challenges in KNN classification," IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 10, pp. 4663–4675, Oct. 2022.

[8] S. Zhang and J. Li, "KNN Classification with One-step Computation," IEEE Transactions on Knowledge and Data Engineering, p. 1, Jan. 2021.

[9] J. Hu, H. Peng, J. Wang, and W. Yu, "kNN-P: A kNN classifier optimized by P systems," Theoretical Computer Science, vol. 817, pp. 55–65, May 2020.

[10] B. Wang and S. Zhang, "A new locally adaptive K-nearest centroid neighbor classification based on the average distance," Connection Science, vol. 34, no. 1, pp. 2084–2107, Jul. 2022.

[11] N. Rastin, M. Z. Jahromi, and M. Taheri, "A generalized weighted distance k-Nearest Neighbor for multi-label problems," Pattern Recognition, vol. 114, p. 107526, Jun. 2021.

[12] A.-J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas, and J. R. Rico-Juan, "Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation," Pattern Recognition, vol. 74, pp. 531–543, Feb. 2018.

[13] X. Zhang, H. Xiao, R. Gao, H. Zhang, and Y. Wang, "K-nearest neighbors rule combining prototype selection and local feature weighting for classification," Knowledge-based Systems, vol. 243, p. 108451, May 2022.

[14] N. Gul, W. K. Mashwani, M. Aamir, S. Aldahmani, and Z. Khan, "Optimal model selection for k-nearest neighbours ensemble via sub-bagging and sub-sampling with feature weighting," Alexandria Engineering Journal /Alexandria Engineering Journal, vol. 72, pp. 157–168, Jun. 2023.

[15] Y. Chen and Y. Hao, "A feature weighted support vector machine and K-nearest neighbor algorithm for stock market indices prediction," Expert Systems With Applications, vol. 80, pp. 340–355, Sep. 2017.

[16] L. Chen and G. Guo, "Nearest neighbor classification of categorical data by attributes weighting," Expert Systems With Applications, vol. 42, no. 6, pp. 3142–3149, Apr. 2015.

[17] C. Gong, Z.-G. Su, X. Zhang, and Y. You, "Adaptive evidential K-NN classification: Integrating neighborhood search and feature weighting," Information Sciences, vol. 648, p. 119620, Nov. 2023.

[18] Z. Bian, C. M. Vong, P. K. Wong, and S. Wang, "Fuzzy KNN method with adaptive nearest neighbors," IEEE Transactions on Cybernetics, vol. 52, no. 6, pp. 5380–5393, Jun. 2022.

[19] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy K-nearest neighbor algorithm," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 4, pp. 580–585, Jul. 1985.

[20] J. Gou et al., "A representation coefficient-based k-nearest centroid neighbor classifier," Expert Systems With Applications, vol. 194, p. 116529, May 2022.

[21] M. Açıkkar and S. Tokgöz, "An improved KNN classifier based on a novel weighted voting function and adaptive k-value selection," Neural Computing & Applications, vol. 36, no. 8, pp. 4027–4045, Dec. 2023.

[22] H. Xu, W. Zeng, X. Zeng, and G. G. Yen, "An evolutionary algorithm based on Minkowski Distance for Many-Objective optimization," IEEE Transactions on Cybernetics, vol. 49, no. 11, pp. 3968–3979, Nov. 2019.

[23] S. Chowdhury, N. Helian, and R. C. De Amorim, "Feature weighting in DBSCAN using reverse nearest neighbours," Pattern Recognition, vol. 137, p. 109314, May 2023.

[24] Á. Ipkovich, K. Héberger, and J. Abonyi, "Comprehensible visualization of multidimensional data: sum of Ranking Differences-Based parallel coordinates," *Mathematics*, vol. 9, no. 24, p. 3203, Dec. 2021.

[25] D. P. Turner, H. Deng, and T. T. Houle, "Understanding and applying confidence intervals," Headache the Journal of Head and Face Pain, vol. 60, no. 10, pp. 2118–2124, Nov. 2020.

[26] H. Lüpsen, "Generalizations of the Tests by Kruskal-Wallis, Friedman and van der Waerden for Split-plot Designs," Austrian Journal of Statistics, vol. 52, no. 5, pp. 101–130, Sep. 2023.

[27] L. Štěpánek, F. Habarta, I. Mala, and L. Marek, "A short note on post-hoc testing using random forests algorithm: Principles, asymptotic time complexity analysis, and beyond," Annals of Computer Science and Information Systems, Sep. 2022.

[28] T. J. VanderWeele and M. B. Mathur, "Some desirable properties of the Bonferroni correction: is the Bonferroni correction really so bad?," American Journal of Epidemiology, vol. 188, no. 3, pp. 617–618, Nov. 2018.