algorithms

MDPI

Article

A Closest Resemblance Classifier with Feature Interval Learning and Outranking Measures for Improved Performance

Nabil Belacel

Digital Technologies Research Center, National Research Council Canada, Ottawa, ON K1A 0R6, Canada; nabil.belacel@nrc-cnrc.gc.ca

Abstract: Classifiers today face numerous challenges, including overfitting, high computational costs, low accuracy, imbalanced datasets, and lack of interpretability. Additionally, traditional methods often struggle with noisy or missing data. To address these issues, we propose novel classification methods based on feature partitioning and outranking measures. Our approach eliminates the need for prior domain knowledge by automatically learning feature intervals directly from the data. These intervals capture key patterns, enhancing adaptability and insight. To improve robustness, we incorporate outranking measures, which reduce the impact of noise and uncertainty through pairwise comparisons of alternatives across features. We evaluate our classifiers on multiple UCI repository datasets and compare them with established methods, including k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), Random Forest (RF), Neural Networks (NNs), Naive Bayes (NB), and Nearest Centroid (NC). The results demonstrate that our methods are robust to imbalanced datasets and irrelevant features, achieving comparable or superior performance in many cases. Furthermore, our classifiers offer enhanced interpretability while maintaining high predictive accuracy.

Keywords: machine learning; classification; supervised learning; feature interval learning; outranking measures

Academic Editors: Mario Rosario Guarracino, Laura Antonelli and Pietro Hiram Guzzi

Received: 12 July 2024
Revised: 13 December 2024
Accepted: 23 December 2024
Published: 31 December 2024

Citation: Belacel, N. A Closest Resemblance Classifier with Feature Interval Learning and Outranking Measures for Improved Performance. *Algorithms* **2025**, *18*, 7. https://doi.org/10.3390/a18010007

1. Introduction

Classification stands as a cornerstone in machine learning, tasked with assigning new data points to predefined classes. Commonly employed concept models such as Decision Trees (DTs), Support Vector Machines (SVMs), and k-Nearest Neighbors (k-NN) have long served this purpose [1]. Many existing classifiers rely on complex learning mechanisms. These mechanisms combine or transform data features, which can lead to overfitting, high computational costs, and a lack of interpretability, often referred to as the 'black box' effect. Additionally, these models struggle to handle noisy or missing data. To address these limitations, this paper proposes a novel classification framework that leverages feature interval learning (FIL) and outranking measures. FIL partitions the feature space into intervals, associating each interval with a specific class [2,3]. This approach offers enhanced robustness to noise and variability in the data. Outranking, rooted in preference learning, provides a mechanism for comparing alternatives based on their feature values, further improving the classifier's ability to handle uncertainty and noise [4,5]. The primary objectives of this study are as follows:

- Develop classification methods that combine the strengths of FIL and outranking measures to address the shortcomings of existing approaches.

- Evaluate the performance of the proposed classifiers on various real-world datasets and compare them to established methods.

By achieving these objectives, we aim to contribute to the advancement of classification techniques, offering a more robust, interpretable, and effective solution for a wide range of applications. FIL algorithms segment the feature space into distinct intervals for each feature, with each interval linked to a specific class. Feature intervals represent features as intervals rather than individual values, providing robustness to noise and variability in the input data, particularly for continuous data types [6,7]. A set of feature intervals embodies a concept in each feature dimension separately. Subsequently, each feature contributes to the classification process by distributing votes among classes, with the predicted class being the one receiving the highest vote [8]. This method's reliance on clear feature ranges offers interpretability, rendering decision rules easy to understand [6]. Furthermore, FIL can be effective with smaller training datasets compared to data-hungry deep learning models [9]. In response to these challenges, we have introduced an approach to supervised classification that harnesses the synergies of feature interval learning and outranking measures, called Closest Resemblance (*CR*). Outranking, entrenched in preference learning [10,11], transcends simple distance measures by autonomously learning feature intervals from data, thereby capturing nuanced patterns and enhancing adaptation to specific datasets. Moreover, outranking adeptly navigates noise and uncertainties in real-world data through pairwise comparisons on each feature. The proposed *CR* method amalgamates FIL's interpretability and efficiency with outranking's data-driven pattern recognition and robustness to noise. This integrated approach aims to surmount the limitations of existing methods, particularly in scenarios with imbalanced data and irrelevant features. To validate the effectiveness of our proposed approach, we conducted extensive empirical evaluations on a variety of datasets from the UCI repository [12]. Empirical evaluations compare the proposed classifier *CR* against established classifiers such as k-Nearest Neighbors (k-NN), Support Vector Machines (SVM), Random Forests (RF), Multi-Layer Perceptron Neural Networks (MLP), Naive Bayes (NB), and Nearest Centroid (NC). The results underscore the efficacy of FIL with outranking, showcasing comparable or superior performance while delivering more interpretable models. This enhanced interpretability fosters trust and understanding in the decision-making process, bridging the gap between predictive power and internal workings.

2. Related Works

In the domain of supervised classification, various methodologies have been proposed to tackle the challenges posed by batch data and to enhance the interpretability and adaptability of classifiers. Among these, feature interval learning (FIL) techniques have emerged as prominent approaches. FIL approaches offer a distinct paradigm for classification, wherein the feature space is partitioned into intervals for each feature [8]. Each interval is associated with a specific class, enabling a more nuanced representation of feature patterns and enhancing robustness to noise and variability in the data [6]. FIL methods encompass a range of techniques, including hypercuboid or hyperrectangle learning and prototype-based classification, which learn decision boundaries or representative prototypes to distinguish between classes [13]. The concept of hypercuboids has been exploited for many years [14,15].

Hypercuboid approaches represent one category of FIL methods, where the feature space is partitioned into hypercuboid regions based on the distribution of training data [14]. These regions serve as decision boundaries, facilitating efficient classification of new instances. Hypercuboid approaches are particularly effective in handling high-dimensional data and are known for their computational efficiency and simplicity. Prototype-based FIL

methods, on the other hand, generate prototypes for each class based on the distribution of feature intervals in the training data [8,16]. These prototypes encapsulate the characteristic features of each class and are used to classify new instances by measuring their resemblance to the prototypes. Prototype-based FIL methods offer interpretability and transparency in decision making, as the classification process is based on clear feature ranges. In [6], a family of FIL algorithms is described. Some examples of FIL algorithms are voting feature intervals [8,13,17], k-nearest neighbors on feature projections [18], and Naive Bayes classifier (NBC) [1,19]. The common idea of FIL algorithms is to learn the concept description in the form of a set of disjoint intervals separately for each feature. At the testing phase, the FIL algorithms predict the class of a new data point as the label with the highest total weighted votes of the individual features. For example, in Figure 1, the horizontal axis is feature f_1 and the vertical axis is feature f_2. The two rectangles represent the two prototypes of classes C^1 and C^2. The prototype of class C^1 is represented by the intervals $I_{1,1}$ and $I_{2,1}$ (intervals determined using the training sets of class C^1 'the blue x sign'). And the prototype of class C^2 is represented by the intervals $I_{1,2}$ and $I_{2,2}$ (intervals calculated using the training sets of class C^2 'the green o sign'). The objective is to assign the unlabeled data (the red star $*$ in Figure 1) to the closest rectangle.

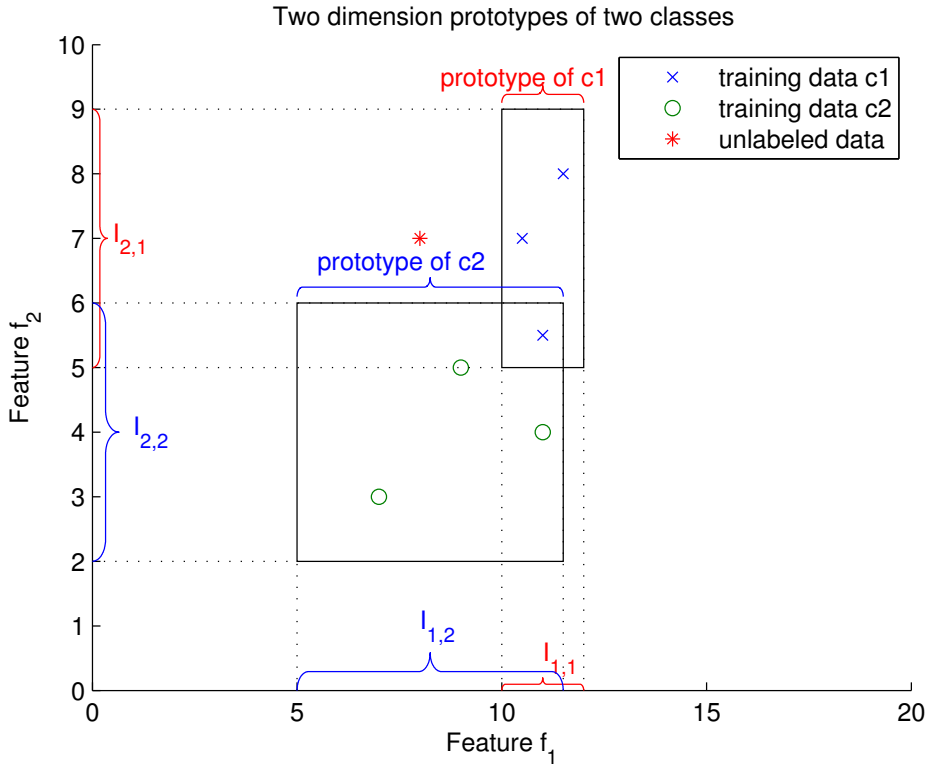

Figure 1. Two-dimensional prototypes of two classes in *CR* algorithm.

Overall, while hypercuboid or hyperrectangle learning algorithms offer computational efficiency and simplicity, FIL methods provide greater flexibility, robustness, and interpretability in classifying new data points, especially in scenarios with complex data distributions and high-dimensional feature spaces [20–22]. However, while predicting class labels based on the highest total weighted votes of individual features is a straightforward approach in FIL algorithms, it may have limitations in handling feature importance, noise, feature relationships, imbalanced datasets, and the trade-off between interpretability and accuracy. Addressing these inconveniences requires careful consideration of feature selection, noise reduction techniques, feature engineering, and robust mechanisms for handling imbalanced data. While FIL provides a robust foundation for classification, out-

ranking measures offer a complementary approach for handling uncertainty and noise in the data [23]. Beyond traditional majority voting approaches, another advancement in FIL is the integration of outranking measures into the classification process [24]. Outranking measures, rooted in preference learning principles, extend the classification paradigm by capturing subtle preferences and trade-offs among alternative solutions [25]. By conducting pairwise comparisons on each feature, outranking measures facilitate more nuanced decision making and enhance adaptability to specific datasets [10,23]. Several FIL classifiers based on outranking measures have been developed through the decades, among them are PROAFTN [16], and K Closest Resemblance (K-CR) [26,27]. These classifiers offer several advantages in the context of classification tasks. By evaluating the data points based on their pairwise performance on each feature, outranking techniques can capture subtle differences in feature importance and contribution to class membership. Instead of blindly aggregating votes from all features, outranking considers the relative performance of alternatives on each feature, effectively filtering out noise and focusing on discriminative features that contribute most to the classification decision. This robustness to noise enhances the reliability and accuracy of classification outcomes. Outranking techniques can capture complex relationships between features by conducting pairwise comparisons on each feature individually. Unlike majority voting, which treats features independently, outranking takes into account the interactions and dependencies between features, allowing for more accurate and nuanced classification decisions [28]. Outranking techniques provide clear and interpretable decision rules by quantifying the resemblance relationship between data points. Instead of relying on a simple count of votes, outranking assigns a degree of resemblance between alternatives based on their pairwise comparisons, making the classification process more transparent and understandable [16]. This enhanced interpretability fosters trust and understanding in the classification model's decision-making process. Outranking techniques can effectively handle imbalanced datasets by considering the relative performance of alternatives on each feature [29]. Instead of biasing the classification decision towards the majority class, outranking evaluates alternatives based on their overall resemblance relationship, ensuring that all classes are appropriately considered in the classification process. This helps mitigate the impact of class imbalance and ensures more balanced and accurate predictions. Outranking techniques offer flexibility and adaptability to different types of data and classification tasks [28]. By capturing subtle differences in feature importance and performance, outranking can accommodate various data distributions and class structures, making it suitable for a wide range of classification scenarios. The FIL classifiers based on outranking measures have been applied to the resolution of many real world practical problems such as medical diagnosis [30,31], image processing and classification [29,32], engineering design [33], recommendation system [27], cybersecurity [34], and inventory management [35]. Even though these classifiers have several advantages, they suffer from computational complexity in their learning phase, specifically when the number of features is very large. The PROAFTN and the K-CR build a set of prototypes in each class and then the inductive is based on the recursive approach, which is very complex. In this paper, we introduce an FIL approach based on outranking measures for classification problems called Closest Resemblance, *CR*. The *CR* algorithm is simple, effective, efficient, and robust to irrelevant features. In the remaining paper, we present the *CR* classifier, where each class is represented by one prototype and the classification rule is expressed as "the data point is assigned to the class if the data point is resembled or roughly equivalent to the prototype of this class". Therefore, our proposed *CR* classifier is a prototype-based classification method that incorporates FIL techniques to build the prototype of each class during the learning phase. The intervals in *CR* are inspired from the reference intervals used in the era of evidence-based medicine [36]. Unlike confidence

intervals, which quantify uncertainty around a parameter estimate, prediction intervals capture the range within which a data point is likely to fall—they estimate the uncertainty associated with individual predictions.

3. The Closest Resemblance Algorithm

The *CR* is a supervised learning algorithm and it allows resemblance measures to be calculated by determining the preference relations. The rule of the *CR* method in assigning data points to a class is as follows: "the data point s is assigned to a class if and only if s resemble or is (roughly) equivalent to the prototype of this class". Therefore, to assign the point s to the appropriate class, *CR* calculates the resemblance relation to measure the closeness of the sample s to the prototype of the class. To calculate the resemblance relation between the data point to assign and the prototype of the class, *CR* is based on the preference relational system [10,11]. It employs a partial comparison between the sample and the prototypes of the classes, for each feature. Then, it applies a global aggregation using the concordance principle on the partial resemblance measure [10,11]. The *CR* classification method is considered as a weighted voting classifier in which each feature or attribute "votes" for the class membership of an unknown sample according to which class' prototype is the closest [16,26]. The partial comparison employed by *CR* to calculate the resemblance between a sample and class prototype eliminates the need for conventional metric or non-metric distances, which aggregate feature scores into a single unit of measurement. This approach addresses challenges associated with datasets expressed in varying units, reducing the dependency on precise pre-processing and normalization techniques while enhancing the model's adaptability to diverse data representations The *CR* algorithm proceeds in two phases.

3.1. Phase 1: Learning Phase

Suppose that a set of L labeled examples $X = \{x_1, x_2, \ldots, x_L\}$ with a set of n features $F = \{f_1, f_2, \ldots, f_n\}$ are given. This example set is known as the training set. Each example x_i is represented by a vector $f(x_i) =< f_1(x_i), f_2(x_i), \ldots, f_n(x_i) >, i = 1, \ldots, L$, where $f(x_i)$ is an n-dimensional vector, where each component $f_j(x_i)$ represents the value of the feature f_j for the training example x_i. The training examples have exactly one of K distinct class labels. In the learning phase, a prototype of each class is determined using the training set X. For each class $C^h, h \in \{1, \ldots, K\}$, *CR* determines a prototype $P(C^h)$ from training set X. The prototypes are considered as good representatives of their class and are described by the score of each of the n features. More precisely, for each feature f_j and for each class C^h, an interval I_j^h is determined. The interval I_j^h is the value range of feature f_j with respect to class C^h. An n-dimensional prototype of each class $C^h, h = 1, \ldots, k$ is built as presented in Equation (1):

$$p(C^h) = \left\{ I(f_1, C^h), I(f_2, C^h), \ldots, I(f_j, C^h), \ldots, I(f_n, C^h) \right\} \quad (1)$$

The value domain of each dimension f_j is the value range or interval $I_j^h = I(f_j, C^h) = \left[I^1(f_j, C^h), I^2(f_j, C^h) \right]$, where $I^1(f_j, C^h)$ is the lower bound and $I^2(f_j, C^h)$ is the upper bound interval of feature f_j for the class C^h.

In the learning phase, we have used two approaches to build the prototype of classes. Both approaches utilize prototypes defined by reference intervals for each feature in a class. In this work, we have chosen simple approaches to determine the interval of each feature for each class (see Figure 2). The first one is based on the mean μ and the standard deviation σ and the second one is based on the percentiles. The reference intervals based

on percentiles and the mean and standard deviation are non-parametric methods as they do not require any assumptions about the underlying distribution of the data.

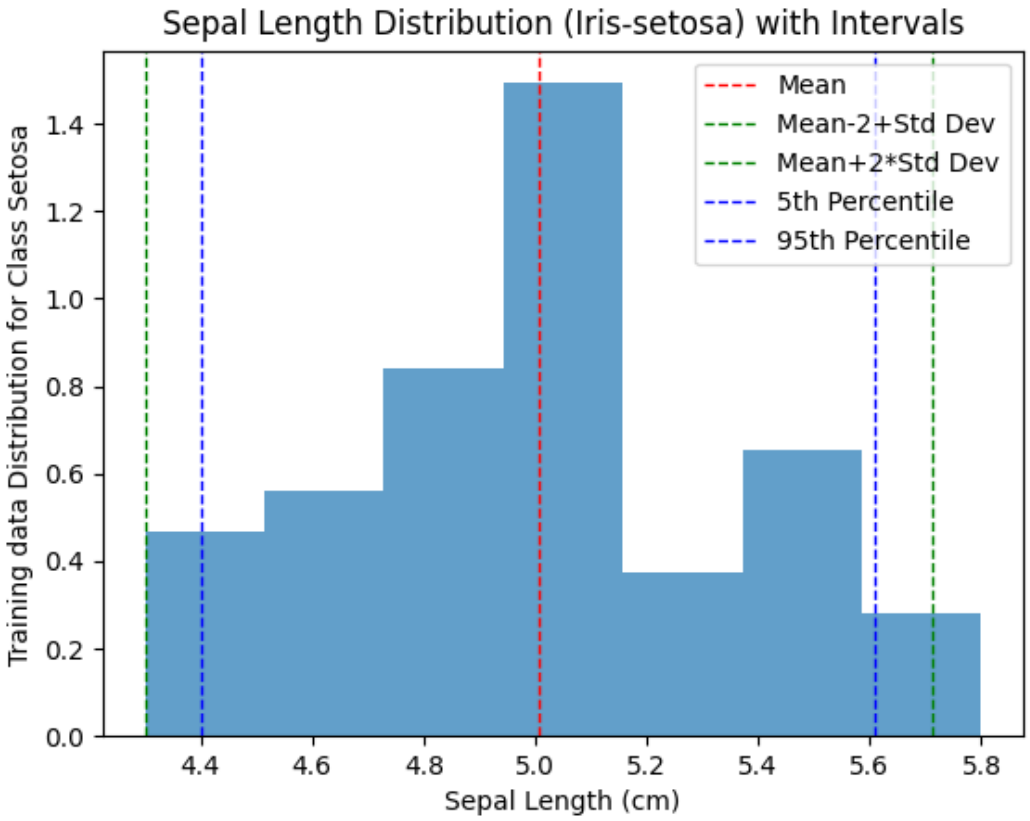

Figure 2. Reference intervals based on percentiles and mean and standard deviation.

3.1.1. Reference Intervals Based on Mean and Standard Deviation

To classify unlabeled data, the *CR* classifier determines the prototype of the classes by determining the reference interval of each feature based on the mean and standard deviation from the training set. The mean (μ) represents the "center" of the data distribution for a specific feature within a class. The standard deviation (σ) captures the spread of the data around the mean. They provide basic summaries of the data's central tendency (μ) and spread (standard deviation σ) without assuming a specific underlying distribution. This technique accounts for the spread of feature values within each class, creating intervals that extend $t > 0$ standard deviations from the mean. It adapts to the variability within each class.

For each class C^h and for each feature f_j, the mean μ and the standard deviation σ are calculated from the training data X, as presented in Equation (2).

$$I_j(f_j, C^h) = [\mu(f_j, C^h) - t * \sigma(f_j, C^h), \mu(f_j, C^h) + t * \sigma(f_j, C^h)] \quad (2)$$

By defining the interval as in Equation (2), we capture a range that encompasses a certain portion, determined by t, of the data points $X = \{x_1, x_2, \ldots, x_L\}$ in the class C^h for the feature f_j. Varying t allows for control over the strictness of the interval. A higher t captures a wider range, a lower t captures a tighter range. This approach utilizes the mean and standard deviation to build the prototype of classes from the training set by leveraging Chebyshev's theorem; we establish boundaries around the mean based on a chosen number t of standard deviations. Data points falling within these boundaries are considered typical, while those exceeding the boundaries are classified as outliers of the corresponding class. This

method is particularly advantageous when the underlying data distribution is unknown, as Chebyshev's theorem makes minimal assumptions about the data.

Chebyshev's theorem: For any data distribution, at least a proportion equal to $(1 - (\frac{1}{t})^2)$ of the data points will fall within t standard deviations of the mean, where t is any positive constant.

Chebyshev's theorem provides a lower bound to the proportion of measurements that are within a certain number of standard deviations from the mean.

3.1.2. Prototype-Based Classifiers with Percentile-Based Reference Intervals

This approach modifies the previous method by using percentiles to define reference intervals for each feature within a class.

For each class C^h and for each feature f_j, calculate the lower quartile L_j^h and upper quartile U_j^h using the training dataset X (see Equation (3)).

$$I_j(f_j, C^h) = [L_j^h(f_j, C^h), U_j^h(f_j, C^h)] \tag{3}$$

Percentiles capture the distribution of data points within a class for each feature. The L_j^h and U_j^h percentiles define an interval that encompasses a specific proportion L% to U% of the data points in that class for that feature. This method is robust to outliers as extreme values have less influence on percentiles compared to the mean and the standard deviation.

This technique captures the central tendency and spread of feature values within each class, offering a robust representation of the distribution by focusing on the lower and upper quartiles. Using percentile intervals for each feature of each class to represent a profile of the class in a classification problem can offer several benefits: By calculating percentile intervals for each feature within each class, create distinct profiles that characterize each class. This can help in understanding the unique characteristics and behaviors associated with each class in classification problem. Percentile intervals highlight the range of values that are most representative of each class. Features with wider intervals or larger differences between classes may indicate greater importance in distinguishing between classes. Class profiles based on percentile intervals are intuitive and easy to interpret. They provide clear boundaries that define the typical range of feature values associated with each class, making it easier to understand the characteristics of each class. Class profiles based on percentile intervals can aid in feature selection by identifying features that are most discriminative between classes. Features with significant differences in percentile intervals across classes may be more informative for classification tasks. When presenting classification results to stakeholders or end-users, class profiles based on percentile intervals provide a transparent explanation of how features contribute to class prediction. This enhances the interpretability and trustworthiness of the classification model. Similar to using percentile intervals for the entire training set, calculating class profiles "prototypes" based on percentile intervals can make the classification model more robust to outliers within each class. Outliers have less influence on percentile intervals compared to other summary statistics like mean and standard deviation. Class profiles based on percentile intervals can be visually represented using box plots or similar visualization techniques. These visual representations can offer a concise summary of the distribution of feature values within each class, facilitating comparison and interpretation. Overall, using percentile intervals to represent class profiles in a classification problem can enhance model interpretability, aid in feature selection, and provide valuable insights into the characteristics of each class, thereby improving the overall performance and understanding of the classification model. These techniques serve as crucial steps in learning the *CR* classification, offering different perspectives on how to define feature intervals. The mean and standard deviation technique presented in Equation (2) adapts to class-specific variations, and the percentile-based approach presented in

Equation (3) provides a robust summary of the feature distributions within each class. Each technique contributes to the overall goal of capturing essential patterns and characteristics for the subsequent phases of the classification process. To learn the *CR* classifier, we follow the different steps of Algorithm 1.

Algorithm 1 Prototype-based classification with reference intervals: Training phase

Input: Training set; $t \geq 0$: Number of standard deviations to consider for interval width; L: Lower percentile and U: Upper percentile;
Output: a set of prototype $p(C^h), h = 1, \ldots, k$
Divide training set into classes and calculate intervals for each feature of each class:
for each class C^h **do**
  for each feature f_j **do**
    if Reference Interval==mean&Standard Deviation **then**
      Calculate the mean μ and standard deviation σ of f_j for training points in class C^h.
      Calculate the low boundary of the interval as $I_j^1(f_j, C^h) = \mu(f_j, C^h) - t \times \sigma(f_j, C^h)$
      Calculate the high boundary of the interval as $I_j^2(f_j, C^h) = \mu(f_j, C^h) + t \times \sigma(f_j, C^h)$
    else if Reference Interval==Percentile **then**
      Calculate the Lth and Uth percentiles of f_j for the training points in class C^h.
      Calculate the low boundary of the interval as $I_j^1(f_j, C^h) = percentileL(f_j, C^h)$
      Calculate the high boundary of the interval as $I_j^2(f_j, C^h) = percentileU(f_j, C^h)$
    end if
    Define the reference interval for feature f_j in class C^h as:
    $I_j(f_j, C^h) = [I_j^1(f_j, C^h), I_j^2(f_j, C^h)]$
    $P(C^h) = P(C^h) \cup I_j(f_j, C^h)$
  end for
  Create prototype profile $p(C_h) = [I(f_1, C_h), I(f_2, C_h), \ldots, I(f_n, C_h)]$
end for
return Set of prototypes $\{P(C^1), P(C^2), \ldots, P(C^h)\}$

3.2. Phase 2: Classification Phase

The classification phase of the *CR* classifier is outlined in Algorithm 2.

Algorithm 2 Prototype-based classification with reference intervals: Classification phase

Input: Test set $S = \{s_1, s_2, \ldots, s_m\}$: where each data point s_i is presented by the profile $f(s_i) = \{f_1(s_i), f_2(s_i), \ldots, f_j(s_i), \ldots, f_n(s_i)\}$ where $f_j(s_i)$ is the value of the feature f_j for the sample s_i; $w_j : j = 1, \ldots, n$ a list of features' weights; A set of prototype: $P = \{P(C^1), P(C^2), \ldots, P(C^h), \ldots, P(C^K)\}$ obtained from Algorithm 1
Output: Predicted class label for each test data point $s_i \in S$
for each sample s_i **do**
  for each class C^h **do**
    for each feature f_j **do**
      Calculate the partial distance: Equation (4)
    end for
  end for
  for each feature f_j **do**
    for each class C^h **do**
      for all Classes $C^t, t \neq h$ **do**
        Calculate partial resemblance: $R_j^{S_i}(p(C^h), p(C^t))$: Equation (5)
      end for
    end for
  end for
  for Each class C^h **do**
    for all Classes $C^t, t \neq h$ **do**
      Calculate weighted resemblance matrix $R^{s_i}(p(C^h), p(C^t))$: Equation (6)
    end for
  end for
  for Each class C^h **do**
    Calculate global score: Equation (9)
  end for
  Predict class label for the sample s_i: Equation (10)
end for
return Class labels for $s_i, i = \{1, 2, \ldots\}$

First, for each testing data point s, CR determines a performance matrix of prototypes, as presented in Table 1, where each component of the matrix corresponds to the absolute distance between the data point s to be classified and the prototype of the class according to the feature f_j. Each class $C^h, h = 1, \ldots, k$ is represented by one prototype $p(C^h)$. The absolute distance between the prototype $p(C^h)$ of the class C^h and the data point s according to the feature f_j is calculated by Equation (4). This function essentially measures the maximum deviation of the data point from the interval bounds. A value of 0 indicates perfect alignment, while higher values indicate greater deviation.

$$d_j^h(s, p(C^h)) = max\left\{0, I_{j,h}^1 - f_j(s)), f_j(s) - I_{j,h}^2\right\} \tag{4}$$

where $f_j(s)$ is the feature f_j value of the testing data point s. $p(C^h)$ is the prototype of the class C^h. The interval $\left[I^1(f_j, C^h), I^2 j(f_j, C^h)\right]$ presents the score of the prototype $p(C^h)$ according to feature f_j, as presented in the learning phase in Section 3.1

Table 1. Performance matrix of prototypes for data point s.

	f_1	.....	f_j	.....	f_n
$p(C^1)$	$d_1^s(p(C^1))$	.....	$d_j^s(p(C^1)$	.....	$d_n^s(p(C^1)$
$p(C^2)$	$d_1^s(p(C^2))$	.....	$d_j^s(p(C^2)$	.....	$d_n^s(p(C^2)$
.....	.....	.....	.....	.....	.....
$p(C^h)$	$d_1^s(p(C^h))$	.....	$d_j^s(p(C^h)$	.....	$d_n^s(p(C^h)$
.....	.....	.....	.....	.....	.....
$p(C^k)$	$d_1^s(p(C^k))$	.....	$d_j^s(p(C^k)$	.....	$d_n^s(p(C^k)$

Based on this performance matrix, the prototype nearest to the sample s is selected by calculating the weighted average mean of the partial preference relationship between the sample s and the prototype of different classes [26]. The preference relation between the different prototypes, also called the outranking relation, can be defined as follows.

Definition 1. *The prototype $p(C^h)$ outranks the prototype $p(c^l)$ "$p(C^h)R^s p(C^l)$" if and only if the resemblance between the sample s and the prototype $p(C^h)$ is stronger than the resemblance between s and the prototype $p(C^l)$ on the whole set of features.*

The outranking relation is based on the introduction of the partial outranking indices R_j^s [10,16]. Each index indicates whether the following statement is true or false: "The resemblance between the data point s and a given prototype is at least as strong as the resemblance between data point s and the another prototype according to feature f_j". The R_j^s outranking index according to feature f_j is given by Equation (5).

$$R_j^s(p(C^h), p(C^l)) = \begin{cases} 1 & if \quad d_j^s(p(C^h)) \leqslant d_j^s(p(C^l)) \\ 0 & Otherwise \end{cases} \tag{5}$$

Let w_j denote the relative importance coefficient of feature f_j, for all $f_j \in F$ (assume, without loss of generality $\sum_{j=1}^n w_j = 1$. This coefficient can be viewed as an intrinsic weight: it can be interpreted as the voting power of each feature. The higher the intrinsic weight, the more important the feature is. Note that the voting power in our classifier CR is independent of the value of the feature on the data points. From these partial feature-based outranking indices, and by taking into account the relative importance of features

w_j, we determine for each pair of prototypes $(p(C^h), p(C^l))$ the global outranking index $R^s(p(C^h), p(C^l))$, which is determined by Equation (6).

$$R^s(p(C^h), p(C^l)) = \sum_{j=1}^{n}(w_j \times R_j^s(p(C^h), p(C^l))), h, l = 1, \ldots, K; j = 1, \ldots, n. \quad (6)$$

$R^s(p(C^h), p(C^l))$ translates the degree to which the resemblance between the data point s and the prototype $p(C^h)$ of class C^h is stronger than the resemblance between the data point s and the prototype $p(C^l)$ of class C^l.

The CR classifier is based on a scoring function from an outranking relation R^s to choose the best prototype in terms of its resemblance to the data point s. Based on the outranking relation R^s between the prototypes of classes, CR selects the best prototype in terms of its resemblance with the data point to be classified. The scoring function is used to select a prototype that more closely resembles data point s. The scoring function is inspired from the scoring function used by the multicriteria decision-analysis method PROMETHEE [37]. It allows for calculation of the degree to which a prototype of a class is preferred to all other prototypes, and inversely, how others are preferred to it. This is defined by calculating the globally positive outranking flow given by Equation (7) and negative outranking flow given by Equation (8).

$$\phi^+(p(C^h)) = \sum_{l=1, l\neq h}^{K} R^s(p(C^h), p(C^l)) \quad (7)$$

$$\phi^-(p(C^h)) = \sum_{l=1, l\neq h}^{K} R^s(p(C^l), p(C^h)) \quad (8)$$

The positive outranking flow ϕ^+ defines the strength of the prototype $p(C^h)$ regarding its resemblance to the data point s. The negative flow ϕ^- defines the weakness of the prototype $p(C^h)$ regrading its resemblance to the data point s. From these two flows, determined in Equations (7) and (8), the net flow ϕ can be calculated by Equation (9).

$$\phi(p(C^h)) = \phi^+(p(C^h)) - \phi^-(p(C^h)) \quad (9)$$

The net flow ϕ is the score of the prototype $p(C^h)$ according to the outranking relation R^s. The highest value of ϕ represents the best prototype regarding its resemblance to data point s, which means that the class C^h is the best candidate for the data point s. Hence, the CR classifies the unlabeled data point s by the following decision rule given by Equation (10).

$$s \in C^h \Leftrightarrow \phi(p(C^h)) = \max_{l\in\{1,\ldots,K\}} \left\{\phi(p(C^l))\right\} \quad (10)$$

The classification procedure used by CR to classify data into the preferred classes is summarized by Algorithm 2.

The general framework of our proposed CR classifier is presented in Figure 3, with the following steps.

- **Step 1—data splitting:** Divide the dataset into 5 equal folds.
- **Step 2—iteration loop**: Iterate through folds. For each iteration, perform the following:
  1. Select one fold as the testing set;
  2. Combine remaining folds as the training set.
     - *Phase 1—training phase*: Combine the 4 folds to form the training set.
       - *Class division*: Divide the training set into classes based on the target variable.

  - *Feature interval calculation*: For each class and feature, calculate intervals based on the mean and standard deviation or on percentiles. Determine low and high boundaries for each interval.
  - *Prototype creation*: Define prototypes for each class using the calculated intervals.
- *Phase 2: Testing phase:* For each test sample in the testing fold perform the following:
  - Partial distance calculation: Calculate the partial distance of the testing sample and the prototype of each feature.
  - Partial resemblance calculation: Calculate the partial resemblance between each feature's test sample and the prototypes of different classes.
  - Weighted resemblance: Calculate the weighted resemblance matrix for each test sample and prototype of a class.
  - Global score calculation: Compute the global score for each class.
  - Prediction: Predict the class label for the test sample based on the highest global score.

- **Step 3—evaluate model performance:** Calculate performance metrics on the testing set.
- **Step 4—repeat for all folds:** Iterate through all 5 folds, using each fold as the testing set once.
- **Step 5—average performance metrics:** Calculate the average performance metrics across all 5 folds to obtain an overall evaluation of the model.

Algorithm 1 summarizes in more detail the general steps of learning phase of our *CR* classifier. The classification phase of *CR* on the testing data is detailed in Algorithm 2.

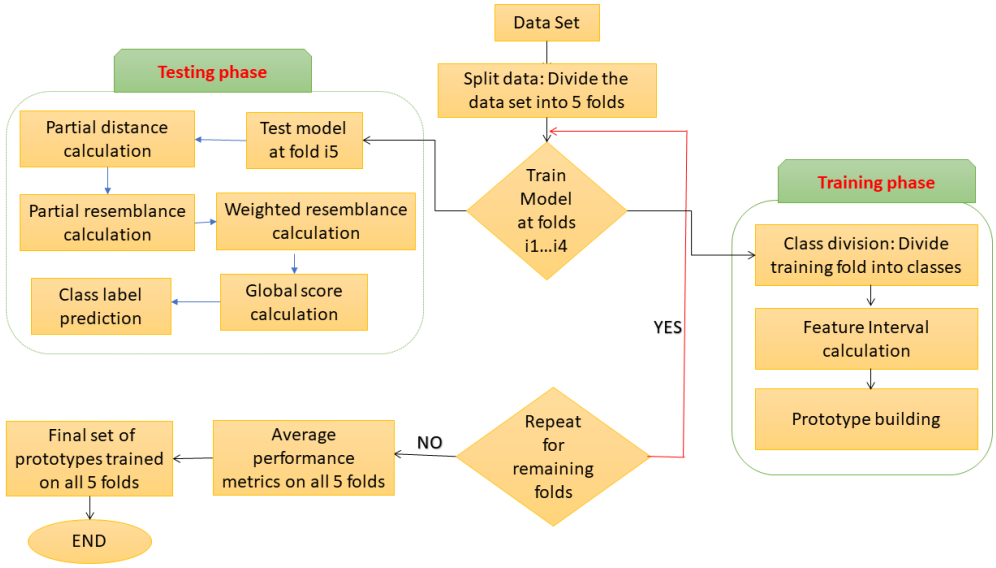

Figure 3. Flowchart of the *CR* classification method.

3.3. Comparison Between CR-Based Mean and Standard Deviation and CR-Based Percentile Intervals

Mean and standard deviation are more sensitive to outliers, while percentiles are more robust. Mean-based intervals might not accurately capture skewed distributions. Percentile intervals are easier to interpret as they directly represent the proportion of data points within the interval. Choosing the appropriate method depends on the data characteristics and the importance of outlier handling. For datasets with outliers or skewed distributions, percentile-based reference intervals offer a more robust alternative to the mean and standard deviation in this prototype-based classification approach.

3.4. Complexity Analysis

This section is dedicated to the complexity analysis of our *CR* classifiers in terms of space and time complexities. The complexity analysis is performed on the training process and the classification of a single new test point. We denote by m, n, and K the number of training sets, number of features, and number of classes.

3.4.1. Training Phase: Algorithm 1

1. Mean and standard deviation calculation (lines 7–9): For each feature f_j in each class C^h, the algorithm calculates the mean (μ) and standard deviation (σ) across all samples in the training set. This process involves iterating over m samples for n features across K classes, resulting in a time complexity of $O(m \times n \times K)$.
2. Percentile calculation (lines 10–12) Alternatively, if percentile-based intervals are used, the L_{th} and U_{th} percentiles are computed. Depending on the percentile algorithm employed, the following may be the case:
   - Selection-based methods have a time complexity of $O(m)$ per feature.
   - For n features and K classes, the time complexity becomes $O(m \times n \times K)$.
3. Prototype profile creation (lines 15–17) Constructing the prototype profile for each class requires iterating through n features for K classes, leading to a complexity of $O(n \times K)$.

Overall training complexity: The training process is dominated by the interval calculations, resulting in a time complexity of $O(m \times n \times K)$. The space complexity for storing the prototype profiles is $O(n \times K)$.

3.4.2. Classification Phase: Algorithm 2

1. Partial distance calculation (lines 5–8): For each feature f_j, the algorithm calculates the partial distance between the test sample and the prototype of each class C^h. This requires $O(n \times K)$ operations per test sample.
2. Partial resemblance matrix calculation (lines 10–13): Calculating the partial resemblance between each class C^h and every other class involves iterating over $K \times (K-1)$ class pairs for n features, yielding a complexity of $O(n \times K^2)$ per test sample.
3. Weighted resemblance matrix calculation (lines 14–17): The weighted resemblance matrix computation also involves $K \times (K-1)$ operations per test sample, resulting in a complexity of $O(n \times K^2)$.
4. Global score calculation (lines 18–22): Computing the global scores for all K classes requires $O(K)$ operations per test sample.

Overall classification complexity: The overall complexity for classifying a single test sample is dominated by the resemblance calculations, resulting in a time complexity of $O(n \times K^2)$. The space complexity remains at $O(n \times K)$ for storing the resemblance matrices and prototype profiles.

4. Experiments

The experiments are conducted on a PC with an Intel(R) processor running at 2.2 GHz and equipped with two processors and 32 GB of RAM. All classifiers and experiments in this research are implemented in Python, utilizing open-source libraries such as Scikit-learn [38] and Pandas [39]. The source code is available on GitHub at https://github.com/nbelacel/Closest-Resemblance, accessed on 20 December 2024.

The real-world datasets used for benchmarking and performance evaluations are sourced from the UCI Machine Learning Repository, well established in machine learning research [12]. A range of datasets are chosen to represent a variety of learning challenges.

They vary in the number of classes (from 2 to 30 classes), number of features (from 5 to 147 features), and the level of separability of the classes. Details regarding the datasets and their dimensionalities are summarized in Table 2.

Table 2. Descriptions of the datasets used in our experiments.

Dataset	Attribute	Class	Instances
Dermatology	34	6	366
HeartDisease	13	5	303
Thyroid	20	2	720
Pageblocks	10	4	548
Breast Cancer	9	2	699
Dermatology	34	4	366
Newthyroid	5	3	215
Geobia	147	9	169
Glass	9	6	214
Leaf	14	30	340
Sonar	60	2	208
SPECTF	44	2	267

Our developed algorithms, termed *CR*, include *CR1*, which employs mean and standard deviation; and *CR2*, utilizing percentile-based approaches during the learning phase (Section 3.1). These classifiers are benchmarked against well-known algorithms: k-Nearest Neighbors (k-NN) with k = 3, Random Forest (RF) with 300 trees, Support Vector Machine (SVM), Multi-Layer Perceptron (MLP) neural network, Naive Bayes (NB), and Nearest Centroid (NC).

To evaluate classification models, various metrics such as accuracy, precision, recall, and the AUC (area under the curve) are commonly used. Unlike accuracy, which solely measures correct predictions without considering the model's ability to rank positive instances higher than negatives, the AUC provides a more comprehensive assessment. The AUC evaluates the model's performance across different thresholds, making it particularly effective for imbalanced datasets or scenarios where prioritizing true positives is crucial [40].

To ensure robust evaluation, we employed 5-fold cross-validation. This technique divides the dataset into five folds, using four for training and one for validation in each iteration. By averaging performance metrics such as AUC across these folds, we obtain a more reliable estimate of each classifier's generalization ability and mitigate the risk of overfitting to specific training data. The hyperparameters (the upper and lower percentiles and the t value of the number of the standard deviation) are tuned using a 5-fold cross-validation strategy. The performance of the classifiers are evaluated using the AUC-ROC metric. By systematically varying the lower and upper percentiles, we found that setting the lower percentile to 0.1 and the upper percentile to 0.9, and setting t to 1.1, yielded the highest average AUC-ROC score across all cross-validation folds. These values were ultimately selected for the final model.

4.1. Results and Discussion

Table 3 presents the classification performance of the *CR1*, *CR2*, k-NN, RF, SVM, MLP, NB, and NC classifiers. The results are based on the average AUC obtained from five repetitions of 5-fold cross-validation. For the *CR1* classifier, which utilizes mean and standard deviation-based reference intervals, we applied $t = 1.1$ standard deviations. For

the *CR2* classifier, which employs percentile-based reference intervals, we used the 90th percentile ($U = 0.9$) as the upper bound and the 10th percentile ($L = 0.1$) as the lower bound. The best classification performance for each dataset is highlighted in boldface.

Based on the experimental study presented in Table 3, *CR1* (percentile-based) achieved the highest average AUC, closely followed by RF and *CR2* (mean and standard deviation-based). As observed in Table 3, both *CR1* and *CR2* consistently outperform other classifiers including NB, SVM, k-NN, MLP, and NC. Notably, *CR1* (percentile-based) achieved the highest average AUC among all classifiers.

Table 4 presents a detailed analysis of the comparative performance of the developed classifiers, *CR1* and *CR2*, alongside other established classifiers, using average normalized AUC scores as the evaluation metric. The ranking process for the normalized scores was carried out in the following systematic steps:

1. Metric calculation:
   - Compute the AUC for each classifier on the test set of each dataset.
   - Calculate the average AUC across all datasets for each classifier.
2. Score normalization: Normalize the AUC scores using the min–max normalization technique, ensuring all scores are on a comparable scale.
3. Average normalized scores: Determine the average normalized score for each classifier by taking the mean of the normalized scores across all datasets.
4. Algorithm ranking: Rank the classifiers based on their average normalized scores, with higher scores indicating better performance.

Furthermore, the classifiers were grouped into high-, medium-, and low-performance categories (Table 5) based on their normalized scores, as shown in Table 4. This grouping concept was inspired by Jenks Natural Breaks Optimization, which seeks to identify natural divisions in data by minimizing intra-group variance and maximizing inter-group variance [41,42]. While Jenks' method provided the conceptual framework, the actual grouping in this study was simplified by employing percentile-based thresholds. The classifiers were divided into thirds for enhanced clarity and interpretability.

Table 3. The performance of all classifiers based on the AUC average using 5-cross validation.

Dataset	*CR1* (0.9–0.1)	*CR2*(t = 1.1)	k-NN	RF	SVM	MLP	NB	NC
Dermatology	0.955	**0.983**	0.934	0.979	0.977	0.981	0.923	0.731
HeartDisease	0.605	**0.670**	0.505	0.580	0.605	0.600	0.618	0.562
Thyroid	0.909	0.763	0.551	**0.919**	0.501	0.505	0.475	0.598
Pageblocks	**0.883**	0.857	0.750	0.779	0.802	0.849	0.768	0.593
BreastCancer	0.969	**0.973**	0.971	0.967	0.960	0.960	0.963	0.956
Shuttle	0.917	0.929	0.885	0.926	0.917	**0.967**	0.872	0.720
Newthyroid	0.946	0.942	0.899	0.943	**0.987**	0.737	0.951	0.865
Geobia	**0.916**	0.895	0.692	0.910	0.752	0.708	0.866	0.658
Glass	0.748	0.719	0.789	**0.855**	0.743	0.604	0.686	0.718
Leaf	0.868	0.862	0.777	**0.876**	0.745	0.753	0.843	0.765
Sonar	**0.840**	0.751	0.815	0.825	0.759	0.806	0.698	0.667
SPECTF	0.696	0.525	0.616	0.627	0.654	0.619	0.761	**0.772**
Average AUC	**0.854**	0.823	0.766	0.849	0.784	0.7489	0.786	0.717

Table 4. Algorithm rankings based on average normalized AUC scores.

Algorithm Ranking	Average Normalized Score
CR1 (Percentile: 0.9–0.1)	**0.631729**
CR2 (t = 1.1)	**0.62406**
RF	**0.622963**
NB	0.608517
SVM	0.553853
k-NN	0.515612
MLP	0.473611
NC	0.363622

Table 5. Classifiers grouped by performance.

Performance Group	Classifiers	Threshold
High Performance	*CR1*, *CR2*, RF	Scores > 0.5427
Medium Performance	NB, SVM, k-NN	Scores between 0.5427 and 0.4544
Low Performance	MLP, NC	Scores < 0.4544

Based on the average rank presented in Table 3, our prototype-based approaches outperformed other prototype-based classifiers, including k-NN, NB, and NC. Notably, *CR1* (percentile-based) achieved the highest average AUC among all classifiers.

4.1.1. Discussion on Complexity Trade-Offs in Training Phase Efficiency

The $O(m \times n \times K)$ training complexity is generally more efficient compared to models like Random Forest, where tree construction can be computationally expensive for large feature spaces.

- Classification phase complexity: The $O(n \times K^2)$ complexity per test sample may be higher than that of models like k-NN ($O(n \times K)$), especially for datasets with a large number of classes K. However, this trade-off is often justified by the higher classification accuracy and robustness provided by the resemblance-based decision mechanism.
- Space complexity consideration: For percentile calculations (Algorithm 1, lines 10–12), selection-based algorithms require $O(m)$ space. However, alternative approaches can reduce this to $O(1)$ additional space. If $m > n \times K$, the space complexity of percentile computation could dominate.

4.1.2. The Implications of *CR* Classifier for Machine Learning Field

The results of our experiments demonstrate that the *CR* classifier offers a promising approach for supervised classification tasks. By combining the strengths of FIL and outranking measures, the *CR* classifier is able to effectively handle noisy and imbalanced data, while also producing highly interpretable models. These findings have several significant implications for the field of machine learning.

1. Advancement of interpretable machine learning: The *CR* classifier's ability to provide clear explanations for its predictions contributes to the growing demand for transparent and accountable AI systems [43]. This is particularly important in domains where understanding the decision-making process is crucial, such as healthcare, finance, and legal applications.
2. Improved performance on challenging datasets: The *CR* classifier's robustness to noise and imbalance makes it well suited for real-world datasets that often contain

imperfections. This could lead to more reliable and accurate predictions in a variety of domains.

3. Potential for new applications: The interpretability and effectiveness of the *CR* classifier could enable its application in areas where traditional machine learning methods have fallen short. For example, it may be useful for analyzing complex medical data or for developing personalized recommendation systems.

Overall, using outranking techniques instead of majority voting provides several advantages, including robustness to noise, consideration of feature relationships, enhanced interpretability, handling of imbalanced datasets, and flexibility in adapting to different types of data. These advantages make outranking techniques a powerful and versatile tool for classification tasks, particularly in complex and challenging real-world scenarios.

4.1.3. Limitations

While the proposed *CR* classifiers demonstrate promising performance, several limitations and avenues for future research are identified. The current approach assumes equal weights for all features, which may not optimally capture feature importance. Future work could explore feature weighting techniques to assign differential weights to features based on their relevance. This could enhance the classifier's performance, especially in scenarios with noisy or irrelevant features. Using the grid search approach for hyperparameter tuning may not guarantee the global optimum. More sophisticated optimization techniques, such as genetic algorithms or Bayesian optimization, could be employed to systematically explore the parameter space and identify optimal hyperparameter settings. Additionally, leveraging feature weight optimization for feature selection purposes could further refine our classifiers, focusing on the most discriminative features for each dataset. The current approach relies on inductive learning, which can be prone to overfitting, especially when dealing with limited data. To mitigate this, incorporating deductive learning techniques could enable the integration of prior knowledge and domain expertise to guide the learning process. By addressing these limitations, future research can further refine the *CR* classifier and extend its applicability to a wider range of classification tasks, such as complex real-world problems involving high-dimensional and noisy data.

5. Conclusions and Future Work

This research introduces a novel classification approach that merges feature interval learning (FIL) with outranking measures to create nested generalized exemplar classifiers. By leveraging feature projections of training samples and outranking measures, our method offers a robust, interpretable, and effective solution for classification tasks. Our empirical evaluation demonstrates the superior performance of these classifiers, particularly in handling imbalanced data and irrelevant features. The inherent interpretability of our approach provides valuable insights into the decision-making process, fostering trust and transparency in machine learning models. While we acknowledge the inherent limitations discussed in Section 4.1.3, our study stands as a catalyst for further exploration in the field of supervised classification and machine learning. Future research could delve into the impact of different outranking methods and feature projection techniques on classifier performance. Additionally, investigating the scalability of the proposed approach to large-scale datasets and exploring the potential of combining FIL with other machine learning techniques are promising avenues for future work. By addressing these areas, we can further enhance the capabilities of FIL-based classifiers and expand their applicability to a wider range of real-world challenges.

Funding: This research was funded by the Digital Technologies Research Center of National Research Council Canada.

Data Availability Statement: The datasets and Python codes of *CR* classifier with the algorithms used in our comparative study are available on GitHub at: https://github.com/nbelacel/Closest-Resemblance accessed 20 December 2024.

Acknowledgments: I would like to express my gratitude to our co-op students Rani Adhaduk and Durga Prasad Rangavajjala for their invaluable assistance in python implementation.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FIL	Feature interval learning
CR	Closest Resemblance classifier
CR1	CR uses reference interval based on mean and standard deviation
CR2	CR uses reference intervals based on percentile
DT	Decision Tree
SVM	Support Vector Machine
RF	Random Forest
NB	Naive Bayes
NC	Nearest Centroid

References

1. Mitchell, T.M. Machine learning and data mining. *Commun. ACM* **1999**, *42*, 30–36. [CrossRef]
2. Agrawal, R.; Ghosh, S.; Imielinski, T.; Iyer, B.; Swami, A.N. An interval classifier for database mining applications. *VLDB* **1992**, *92*, 560–573.
3. Zhang, S.; Liu, H.; Zhang, A.; Hu, Y.; Zhang, C.; Li, Y.; Zhu, T.; He, S.; Ou, W. Learning User Representations with Hypercuboids for Recommender Systems. In Proceedings of the WSDM '21: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, Virtual, 8–12 March 2021; pp. 1–10.
4. Liao, H.; He, Y.; Wu, X.; Wu, Z.; Bausys, R. Reimagining multi-criterion decision making by data-driven methods based on machine learning: A literature review. *Inf. Fusion* **2023**, *100*, 101970. [CrossRef]
5. Belahcène, K.; Mousseau, V.; Ouerdane, W.; Pirlot, M.; Sobrie, O. Multiple criteria sorting models and methods—Part I: Survey of the literature. *4OR* **2023**, *21*, 1–46. [CrossRef]
6. Dayanik, A. Learning feature-projection based classifiers. *Expert Syst. Appl.* **2012**, *39*, 4532–4544. [CrossRef]
7. Dayanik, A. Feature interval learning algorithms for classification. *Knowl.-Based Syst.* **2010**, *23*, 402–417. [CrossRef]
8. Demiröz, G.; Güvenir, H.A. Classification by Voting Feature Intervals. In *Proceedings of the European Conference on Machine Learning: ECML-97*; van Someren, M., Widmer, G., Eds.; Springer: Berlin/Heidelberg, Germany, 1997; pp. 85–92.
9. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Proc . Mach. Learn. Res.* **2010**, *9*, 249–256.
10. Roy, B. *Multicriteria Methodology for Decision Aiding*; Nonconvex Optimization and Its Applications; Springer: New York, NY, USA, 2013.
11. Vincke, P. *Multicriteria Decision-Aid*; John Wiley & Sons: Hoboken, NJ, USA, 1992.
12. Dua, D.; Graff, C. UCI Machine Learning Repository. 2017. Available online: https://archive.ics.uci.edu/ (accessed on 20 December 2024).
13. Shruti Sachdeva, T.B.; Verma, A.K. A novel voting ensemble model for spatial prediction of landslides using GIS. *Int. J. Remote Sens.* **2020**, *41*, 929–952. [CrossRef]
14. Salzberg, S. A nearest hyperrectangle learning method. *Mach. Learn.* **1991**, *6*, 251–276. [CrossRef]
15. Salzberg, S.L. *Learning with Nested Generalized Exemplar*; The Springer International Series in Engineering and Computer Science; Springer: New York, NY, USA, 1990.
16. Belacel, N. Multicriteria assignment method PROAFTN: Methodology and medical application. *Eur. J. Oper. Res.* **2000**, *125*, 175–183. [CrossRef]

17. Pham, B.T.; Tien Bui, D.; Prakash, I.; Nguyen, L.H.; Dholakia, M. A comparative study of sequential minimal optimization-based support vector machines, vote feature intervals, and logistic regression in landslide susceptibility assessment using GIS. *Environ. Earth Sci.* **2017**, *76*, 1–15. [CrossRef]
18. Akkus, A.; Güvenir, H.A. K nearest neighbor classification on feature projections. In Proceedings of the Thirteenth International Conference on International Conference on Machine Learning, ICML'96, San Francisco, CA, USA, 3–6 July 1996; pp. 12–19.
19. Williams, C.K.; Barber, D. Bayesian classification with Gaussian processes. *IEEE Trans. Pattern Anal. Mach. Intell.* **1998**, *20*, 1342–1351. [CrossRef]
20. Güvenir, H.A.; Demiröz, G.; Ilter, N. Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals. *Artif. Intell. Med.* **1998**, *13*, 147–165. [CrossRef] [PubMed]
21. Wei, J.M.; Yang, X.B.; Wang, S.Q.; Gu, L. A Novel Rough Hypercuboid Method for Classifying Cancers Based on Gene Expression Profiles. In Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, Jinan, China, 18–20 October 2008; Volume 2, pp. 262–266. [CrossRef]
22. Wei, J.M.; Wang, S.Q.; Yuan, X.J. Ensemble Rough Hypercuboid Approach for Classifying Cancers. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 381–391. [CrossRef]
23. Cebesoy, M.; Tuncer Şakar, C.; Yet, B. Multicriteria decision support under uncertainty: Combining outranking methods with Bayesian networks. *Ann. Oper. Res.* **2024**, 1–28. [CrossRef]
24. Zopounidis, C.; Doumpos, M. Multicriteria classification and sorting methods: A literature review. *Eur. J. Oper. Res.* **2002**, *138*, 229–246. [CrossRef]
25. Amor, S.B.; Belaid, F.; Benkraiem, R.; Ramdani, B.; Guesmi, K. Multi-criteria classification, sorting, and clustering: A bibliometric review and research agenda. *Ann. Oper. Res.* **2023**, *325*, 771–793. [CrossRef]
26. Belacel, N. The k-closest resemblance approach for multiple criteria classification problems. In *Modelling, Computation and Optimization Information and Management Sciences*; Hoai, L., Tao, P., Eds.; Hermes Sciences Publishing: London, UK, 2004; pp. 525–534.
27. Belacel, N.; Wei, G.; Bouslimani, Y. The k Closest Resemblance Classifier for Amazon Products Recommender System. In Proceedings of the ICAART, Valletta, Malta, 22–24 February 2020; pp. 873–880.
28. Kotsiantis, S.B.; Zaharakis, I.; Pintelas, P. Supervised machine learning: A review of classification techniques. *Emerg. Artif. Intell. Appl. Comput. Eng.* **2007**, *160*, 3–24.
29. Belacel, N.; Duan, C.; Inkpen, D. The K-Closest Resemblance Classifier for Remote Sensing Data. In Proceedings of the Canadian Conference on Artificial Intelligence, Ottawa, ON, Canada, 13–15 May 2020; pp. 49–54.
30. Belacel, N.; Boulassel, M.R. Multicriteria fuzzy assignment method: A useful tool to assist medical diagnosis. *Artif. Intell. Med.* **2001**, *21*, 201–207. [CrossRef]
31. Belacel, N.; Wang, Q.; Richard, R. Web-integration PROAFTN methodology for acute leukemia diagnosis. *Telemed. J. e-Health* **2005**, *11*, 652–659. [CrossRef]
32. Al-Obeidat, F.; Al-Taani, A.T.; Belacel, N.; Feltrin, L.; Banerjee, N. A Fuzzy Decision Tree for Processing Satellite Images and Landsat Data. *Procedia Comput. Sci.* **2015**, *52*, 1192–1197. [CrossRef]
33. Sassi, I.; Belacel, N.; Bouslimani, Y. Photonic-crystal fibre modeling using fuzzy classification approach. *Int. J. Recent Trends Eng. Technol.* **2011**, *6*, 100–104.
34. Al-Obeidat, F.; El-Alfy, E.S. Hybrid multicriteria fuzzy classification of network traffic patterns, anomalies, and protocols. *Pers. Ubiquitous Comput.* **2019**, *23*, 777–791. [CrossRef]
35. Douissa, M.R.; Jabeur, K. A New Model for Multi-criteria ABC Inventory Classification: PROAFTN Method. *Procedia Comput. Sci.* **2016**, *96*, 550–559. [CrossRef]
36. Ceriotti, F.; Hinzmann, R.; Panteghini, M. Reference intervals: The way forward. *Ann. Clin. Biochem.* **2009**, *46*, 8–17. PMID: 19103955. [CrossRef] [PubMed]
37. Brans, J.P.; Mareschal, B. PROMETHEE methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 163–186.
38. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
39. McKinney, W. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; Volume 445, pp. 51–56.
40. Ling, C.X.; Huang, J.; Zhang, H. AUC: A Better Measure than Accuracy in Comparing Learning Algorithms. In *Proceedings of the Advances in Artificial Intelligence*; Xiang, Y., Chaib-Draa, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 329–341.
41. Jenks, G.F. The data model concept in statistical mapping. *Int. Yearb. Cartogr.* **1967**, *7*, 186–190.

42. North, M.A. A Method for Implementing a Statistically Significant Number of Data Classes in the Jenks Algorithm. In Proceedings of the 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Tianjin, China, 14–16 August 2009; Volume 1, pp. 35–38. [CrossRef]
43. Angelov, P.P.; Soares, E.A.; Jiang, R.; Arnold, N.I.; Atkinson, P.M. Explainable artificial intelligence: An analytical review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2021**, *11*, e1424. [CrossRef]