Meta-learning recommendation system for automating the process of selecting the best pool and a dynamic classifier selection algorithm

by

Hesam JALALIAN

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS
M.A.Sc.

MONTREAL, DECEMBER 14, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Rafael M. O. Cruz, Thesis supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Mohammadhadi Shateri, Chair, Board of Examiners
Department of System Engineering, École de technologie supérieure

Mr Robert Sabourin, External Examiner
Department of System Engineering, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON DECEMBER 1, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I extend my heartfelt gratitude to Professor Rafael Menelau Oliveira e Cruz for his invaluable guidance and unwavering support throughout my thesis journey. His profound knowledge, mentorship, and insightful feedback have significantly shaped the trajectory of my research. I am also deeply thankful to Professor Robert Sabourin for his constructive insights and scholarly expertise, which have enriched my work and broadened my understanding of the field.

I am privileged to acknowledge my exceptional colleagues at the LIVIA Lab, whose collaborative spirit and thought-provoking discussions have been instrumental in refining my research ideas. Their dedication to advancing knowledge and their camaraderie have truly made my academic journey a rewarding experience.

My sincerest appreciation goes to my beloved wife, whose unwavering support and understanding during the challenging phases of immigration and academic pursuit have been my cornerstone. Her encouragement and sacrifice have been pivotal in enabling me to focus on my studies and realize my academic aspirations.

Finally, I express my profound gratitude to École de technologie supérieure (ÉTS Montréal) for providing me with an enriching academic environment and resources that have facilitated my research endeavors. The commitment of the faculty, staff, and the university's ethos of fostering innovation and excellence have been pivotal in shaping my academic growth.

As I conclude this chapter of my academic journey, I am humbled by the contributions and support of those who have stood by me. Your collective influence has indelibly shaped my research and personal development, and for that, I am truly grateful.

Système de recommandation de méta-apprentissage pour automatiser le processus de sélection du meilleur pool et un classificateur dynamique ou un algorithme de sélection d'ensemble

Hesam JALALIAN

RÉSUMÉ

Les méthodes d'ensemble consistent en des classifieurs individuels entraînés indépendamment, dont les prédictions sont combinées pour prédire de nouvelles instances. Les recherches antérieures ont montré qu'un ensemble de classifieurs est souvent plus performant qu'un seul classifieur. La sélection dynamique (DS) est reconnue comme une approche efficace dans le domaine des systèmes de classification multiples (MCS). La DS choisit dynamiquement les classifieurs de base pour chaque nouvel échantillon à classer. Au lieu d'utiliser un ensemble fixe de classifieurs pour toutes les instances, la DS s'adapte et sélectionne les classifieurs appropriés en fonction des caractéristiques de chaque échantillon individuel. Cette étude se concentre sur l'évaluation de différents schémas de génération de pools utilisés en tant qu'entrée pour les techniques de DS. L'importance de la sélection d'un pool optimal de classifieurs pour les algorithmes de DS est explorée, en comparant les schémas de génération de pools globaux et locaux.

L'étude vise à répondre à des questions concernant l'efficacité des méthodes de génération de pools locaux pour la sélection dynamique, la dépendance à l'égard de la sélection du meilleur groupe de classifieurs en fonction de la méthode DS utilisée, et la corrélation entre les résultats de l'ensemble statique et les performances de DS. L'analyse révèle que les schémas de génération de pools locaux ne donnent pas systématiquement de meilleurs résultats pour la sélection. De plus, la sélection d'un groupe approprié de classifieurs dépend à la fois de la méthode DS utilisée et des caractéristiques de l'ensemble de données.

Compte tenu des constatations selon lesquelles il n'existe pas de système de génération de pools universellement efficace pour l'algorithme DS sur l'ensemble des ensembles de données, un système d'apprentissage automatisé est proposé. Ce système vise à recommander le schéma de génération de pool optimal pour les méthodes DS en fonction de l'ensemble de données. L'algorithme crée un méta-modèle en extrayant des méta-caractéristiques des ensembles de données et en utilisant une méta-cible représentant les meilleures performances de l'algorithme DS. Ce méta-modèle fournit des recommandations pour les schémas de génération de pools, les méthodes DS et les paires d'un pool et Méthode DS appariées pour des ensembles de données spécifiques.

Une étude expérimentale est menée pour évaluer les performances du système de recommandation d'apprentissage méta. Les résultats montrent que le système de recommandation d'apprentissage méta surpasse la sélection de pool fixe, les méthodes DS fixes ou les paires fixes d'un pool et Méthode DS. En d'autres termes, les résultats expérimentaux montrent que le système d'apprentissage méta recommande systématiquement la meilleure solution avec la plus grande

précision prédictive par rapport aux bases. Cette approche novatrice améliore l'efficacité et l'efficacité de la sélection dynamique de classifieurs dans des tâches de classification complexes.

Mots-clés: sélection dynamique, pool de classificateurs, systèmes de classificateurs multiples, apprentissage local, recommandation de méta-apprentissage, complexité des données

Meta-learning recommendation system for automating the process of selecting the best pool and a dynamic classifier selection algorithm

Hesam JALALIAN

ABSTRACT

Multiple Classifier Systems(MCS) have been widely studied as an alternative to improve pattern recognition applications in recent years. An ensemble of classifiers or an MCS produces a better recognition performance than a single classifier, according to several empirical studies. Ensembles consist of classifiers that are individually trained and whose predictions are combined to predict novel instances. According to previous research, an ensemble of classifiers is often more accurate than a single classifier. Dynamic Selection (DS) is considered one of the most effective approaches in the field of Multiple Classifier Systems (MCS). In DS, the choice of base classifiers is made dynamically for each new sample that needs to be classified. Instead of using a fixed set of classifiers for all instances, DS adapts and selects the appropriate classifiers based on the characteristics of each individual sample being classified. For DS methods, a pool of classifiers is employed for classification. The existing pool of classifiers used in dynamic selection methods lacks stability, leading to a significant difference between minimum and maximum classification results when different subsets of the same problem are used for training. Also, pool generation methods with a global perspective tend to generate redundant classifiers, making the system computationally expensive. Such methods may not effectively cover certain regions of the feature space, limiting the performance of dynamic selection methods in selecting the most competent models for specific instances.

In this study, we present an analysis conducted evaluating multiple pool generation schemes when they are used as input of DS techniques. The significance of selecting an optimal pool of classifiers for dynamic selection (DS) algorithms is explored, and global and local perspective pool generation schemes are compared. Several DS techniques were selected based on their superior performance based on dynamic selection literature and are evaluated in this study to address research questions. The following two research questions are of primary importance: Are local pool generation schemes better than global pools? Does the best pool generation scheme depend on the DS algorithm and dataset in question? The study reveals that local pool generation schemes do not consistently achieve better results for dynamic selection. Furthermore, the selection of an appropriate pool of classifiers depends on both the DS method used and dataset characteristics.

Since the above-mentioned analysis reveals that there is no pool generation scheme that can be the best selection for the DS algorithm on all datasets. Therefore, we present a meta-learning automated system to recommend the best pool generation scheme for DS methods for a given dataset. In this algorithm, a meta-model is created by extracting meta-features from datasets and utilizing a meta-target that represents the best DS algorithm's performance. This meta-model is used to provide three recommendations: predicting a pool generation scheme, a DS method, and a pair recommendation of a pool and DS method simultaneous for a given dataset. Another

experiment study is conducted to evaluate the performance of the meta-learning recommendation system, and the results show that the meta-learning recommendation system performs better than selecting a fixed pool of classifiers, a fixed DS method, or a fixed pair of a pool and DS method. In other words, according to the experimental results, meta-learning recommended the best solution with the highest predictive accuracy over the baselines (majority).

The code and data used in this work are available on this project's GitHub repository: https://github.com/hesamjalalian/Pool_generation_scheme

Keywords: dynamic selection, pool of classifiers, multiple classifier systems, local learning, meta-learning recommendation, data complexity

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

MCS	Multiple Classifier Systems
DSEL	Dynamic Selection Dataset
DS	Dynamic Selection
DES	Dynamic Ensemble Selection
DCS	Dynamic Classifier Selection
FLT	Forest of Local Trees
LIT	Local Independence Training
NS	Neighborhood Selection
KNN	K-Nearest Neighbor
HF-MCDM	Hesitant Fuzzy Multiple Criteria Decision-Making
DES-CS	DES competence based on continuous-valued outputs and weighted class supports
DES-CD	DES system with a dynamic threshold of competence and class-dependent weights in the majority voting procedure
TB	Threshold-Based
OB	Output-Based
PB	Probability-Based
KNOP	K-Nearest Output Profile
KNORA-U	K-Nearest Oracles-Union
DES-P	DES Performance

MCB	Multiple Classifier Behavior
LCA	Local Class Accuracy
LP	Local Pool
RoC	Region of Competence
ASP	Algorithm Selection Problem
ANN	Artificial Neural Networks
SVM	Support Vector Machines
DT	Decision Tree
DWNN	Distance Weighted k-Nearest Neighbor
RF	Random Forests
SVR	Support Vector Regressors
Bagging(DT)	Bagging with Decision Tree as a base estimator
Bagging(P)	Bagging with Perceptron as a base estimator
Boosting(DT)	Boosting with Decision Tree as a base estimator
Boosting(P)	Boosting with Perceptron as a base estimator
META-DES	Dynamic Ensemble Selection framework using Meta-learning
DES-MI	Dynamic ensemble Selection for multi-class imbalanced datasets
DES-P	Dynamic Ensemble Selection performance
OLA	Overall Local Accuracy
MLA	Modified Local Accuracy

KNORA-E	k-Nearest Oracle-Eliminate
EMO	Evolutionary Multi-objective Optimization
SVD	Singular Value Decomposition
CD	Critical Difference

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

c_i	An individual (base) classifier
C	The pool of classifiers includes of M base classifiers, $\{c_1, \ldots, c_M\}$
N	Number of instances in a dataset
Z	Number of datasets used in the meta-learning method
$\mathbf{x}_j$	A test sample and j represents the $j\text{-}th$ sample
$\tilde{\mathbf{x}}_j$	Output profile of the $j\text{-}th$ sample
$\mathbf{y}_j$	Represents the class label of $\mathbf{x}_j$
M	The size of the pool of classifiers
θ_j	Region of competence of the $j\text{-}th$ instance
$\mathbf{x}_k$	An instance belong to the region of competence
μ	Corresponding centroid of a Forest Local Trees (FLT) ensemble
C'	An ensemble containing a subset of base classifiers
C^*	A single classifier
Th	Represent the threshold value that is used as a criterion for selecting classifiers.
T_r	Training dataset
$\mathbb{T}$	A set of datasets
T_e	Test dataset
V_a	Validation dataset
$DSEL$	Dynamic Selection dataset

K	Number of samples in the region of competence
ω	The weight given to training instances by Boosting methods
e_t	The error of the classifier in the AdaBoost method
M_f	Meta-features
M_m	Meta-model
$\mathcal{L}$	Loss function
$\mathcal{A}$	Learning algorithm
T	Dataset
DS	A DS method
f_i	$i-th$ meta-feature set
$\mathbf{x}'$	The meta-feature vector extracted from a dataset $\{f_1, \ldots, f_n\}$
y'	Meta-target
y'_{pool}	Meta-target that represents a pool of classifiers in phase3 in the pair (pool, DS) recommendation
y'_{DS}	Meta-target that represents a DS method in phase3 in the pair (pool, DS) recommendation
MT	Meta-dataset
N_f	The number of features to consider at each split
N_T	The number of trees in Random Forests
Z	The number of datasets used in meta-learning to extract the meta-features
$\mathbb{DS}$	A set of DS methods

$\mathbb{P}$	A set of pool of classifiers
K	The number of pools of classifiers used in the $\mathbb{P}$
W	The number of DS algorithms used in $\mathbb{DS}$

INTRODUCTION

Whenever we have to make an important decision in our lives, we listen to multiple opinions in order to reach an effective decision. In automated decision-making applications, the wide-ranging advantages of consulting "several experts" before making a final decision have been recognized by researchers(Polikar (2006); Ghosh, Shankar, Bruzzone & Meher (2010); Sáez, Galar, Luengo & Herrera (2013)). Both in human decision-making and automated decision-making applications, the idea of merging multiple opinions or predictions has proven to enhance decision effectiveness. In automated decision-making, this is often accomplished using Ensembles or committees, also known as Multiple Classifier Systems (MCS) (Kuncheva (2014b); Polikar (2006)). MCS is a collection of individual component classifiers whose predictions are combined to achieve improved predictions in supervised classification learning. MCS has been shown to be an efficient way of improving predictive accuracy (Wang, Fan, Yu & Han (2003)).

The Multiple Classifier System (MCS) comprises three primary stages (Cruz, Sabourin & Cavalcanti (2018a)): firstly, the generation phase, where classifiers create a pool of classifiers, secondly, the Selection phase, where classifiers are selected either statically or dynamically; and finally, the Aggregation or Fusion phase, where Base experts' outputs are compiled to give the final decision. In the Selection phase of MCS, there are two possible approaches: static selection in which the same set of classifiers is used for each sample, and dynamic selection (DS) in which certain classifiers are selected from the pool according to where the query sample is located. The literature on dynamic selection shows that dynamic selection techniques can be used effectively to solve classification problems that are ill-defined when the training data is small and not enough for modeling a classifier(Cavalin, Sabourin & Suen (2012); Cruz, Sabourin, Cavalcanti & Ren (2015c)). This approach changes the ensemble topology on the fly, selecting only the most competent classifier or ensemble of classifiers for each specific test instance (Cruz *et al.* (2018a)). In this case, the competence of the classifiers is estimated based on a local region of the feature space where the query sample is located.

In the domain of multiple pattern recognition contexts, the usage of dynamic selection (DS) techniques has garnered significant attention and importance (Ko, Sabourin & Britto Jr (2008)).

These techniques extend their reach across diverse classification scenarios, ranging from addressing imbalanced learning challenges (Xiao, Xie, He & Jiang (2012)) to effectively handling noisy data distributions (Cruz, Sabourin & Cavalcanti (2018b)). This underscores the role that DS methods play in enhancing pattern recognition outcomes. Delving deeper, the exploration of DS methods and their associated challenges gains substantial relevance within the research community. As highlighted in various studies, DS techniques are increasingly finding application in One-Class Classification (Krawczyk & Woźniak (2016)), managing concept drift (De Almeida, Oliveira, Britto & Sabourin (2016)), solving One-Versus-One decomposition problems (Mendialdua, Martínez-Otzeta, Rodriguez-Rodriguez, Ruiz-Vazquez & Sierra (2015)), and even tackling complex real-world issues such as signature verification (Batista, Granger & Sabourin (2012)), face recognition (Bashbaghi, Granger, Sabourin & Bilodeau (2017)), music classification (de Almeida *et al.* (2012)), and credit scoring (Xiao, Xiao & Wang (2016)). These versatile applications demonstrate the importance of DS methods in addressing a wide spectrum of classification challenges.

0.1 Problem statement

Dynamic selection techniques aim to enhance classification accuracy by selecting the most competent classifiers from a pool, tailored to each instance at runtime. However, there is a lack of specific guidance in the literature on how to select or train a pool of classifiers optimized for dynamic selection algorithms.

The majority of DS publications use conventional pool generation schemes like Bagging (Breiman (1996)) and Boosting (Freund & Schapire (1997)). There are 2 main limitations of these pool generation schemes which are elaborated upon below.

0.1.1 Instability of Bagging

Bootstrap aggregation, often referred to as bagging, is a method proposed by Breiman that works with various classification and regression techniques. This approach involves drawing

multiple bootstrap samples from the available data, applying a prediction method to each of these samples, and then merging the outcomes. Bagging is based on random subsampling by training different instances of the same base estimator. When multiple pools using bagging are generated, each pool contains a different subset of data due to random sampling. Consequently, the classifiers trained on these different subsets might end up being distinct from each other. Therefore, the Bagging technique can sometimes generate a set of classifiers that yield highly diverse results that are Noticeably varied.

To demonstrate the instability of Bagging, we designed an experimental study. We considered the KNORA-E (Ko *et al.* (2008)) and OLA (Woods, Kegelmeyer & Bowyer (1997)) techniques in this experiment since KNORA-E is one of the superior Dynamic Ensemble selection methods and OLA is one of the superior Dynamic classifier selection techniques based on the article (Cruz *et al.* (2018a)). Decision Tree and Perceptron are selected as base classifiers. For the application of DS methods, each dataset should be divided into three sections. In addition to the conventional training and test sets, another subset, known as Dynamic Selection Dataset (DSEL) is included. DSEL constitutes a subset of the dataset where the estimation of the region of competence is conducted. In this experimental study, the datasets were randomly divided into 50, 25, and 25 respectively for the training set, test set, and the DSEL. For the pool of classifiers sizes 10 and 100 were selected to analyze the results when generating a small and a large pool of classifiers. The experiments were repeated 10 times for each dataset by varying the random seed used to initialize the Bagging algorithm. In order words, all replications considered the same dataset split to remove the randomness associated with different partitions from the result instability analysis.

Table 0.1 displays a summary of three datasets used in the experiments including Adult, Heart, and, Liver collected from the UCI and Statlog machine learning repositories. The table presents various attributes of the datasets, including their names, the number of instances, the dimensionality of the data, the number of classes, and their sources.

Table 0.1 Summary of the 3 datasets used in the experiments

Database	No. of instances	Dimensionality	No. of classes	Source
Adult	48,842	14	2	UCI
Heart	270	13	2	STATLOG
Liver Disorders	345	6	2	UCI

Table 0.2 shows a comparison of mean and variance values for different combinations of dynamic selection (DS) techniques using Bagging as a pool of classifiers with different sizes of the pool. Also, different base estimators, including decision tree and perceptron are used for the pool of classifiers. The values in the table indicate the mean accuracy values followed by the standard deviation values in parentheses. For example, "K(100)(DT)" represents the KNORA-E technique with 100 base classifiers of type Decision Tree, and the mean accuracy for this combination is 85.05% with a standard deviation of 0.75%.

Table 0.2 Comparison of Mean and Variance of KNORA-E (K) and OLA (O) Using Bagging as a pool of classifiers with 10 or 100 Base Classifiers, where the Base Classifiers are Decision Tree (DT) or Perceptron (P)

Dataset	K(100)(DT)	K(10)(DT)	O(100)(DT)	O(10)(DT)	K(100)(P)	K(10)(P)	O(100)(P)	O(10)(P)
Adult	85.05(0.75)	83.35(1.99)	79.59(1.84)	80.45(2.51)	82.53(0.95)	82.30(1.53)	84.85(1.26)	85.19(1.75)
Heart	72.78(1.35)	73.37(2.67)	72.79(2.48)	71.90(5.15)	71.61(2.71)	72.20(3.91)	75.87(3.30)	71.17(2.95)
Liver	75.40(3.68)	64.70(3.63)	62.52(5.19)	62.85(4.52)	71.37(2.26)	71.37(2.64)	73.56(2.95)	66.54(4.78)

Figure 0.1 visually depicts the difference between the maximum and minimum results across three different datasets. The graph highlights the disparity between the highest and lowest outcomes obtained from the dynamic selection techniques employed in the study. Specifically, KNORA-E is denoted as K, while OLA is represented by O in the figure. The base classifiers are indicated by DT for Decision Tree and P for Perceptron. Moreover, the number of base estimators is varied as either 10 or 100 for different configurations. This visualization provides a clear understanding of the difference between maximum and minimum results achieved by employing various dynamic selection methods and base classifiers across the datasets. So as is

evident from the figure, there is a wide gap between the maximum and minimum results when training with different subsets of the same problem. This exploration has shown the importance of understanding and addressing the potential instability associated with Bagging when they are used as a pool of classifiers for DS techniques.

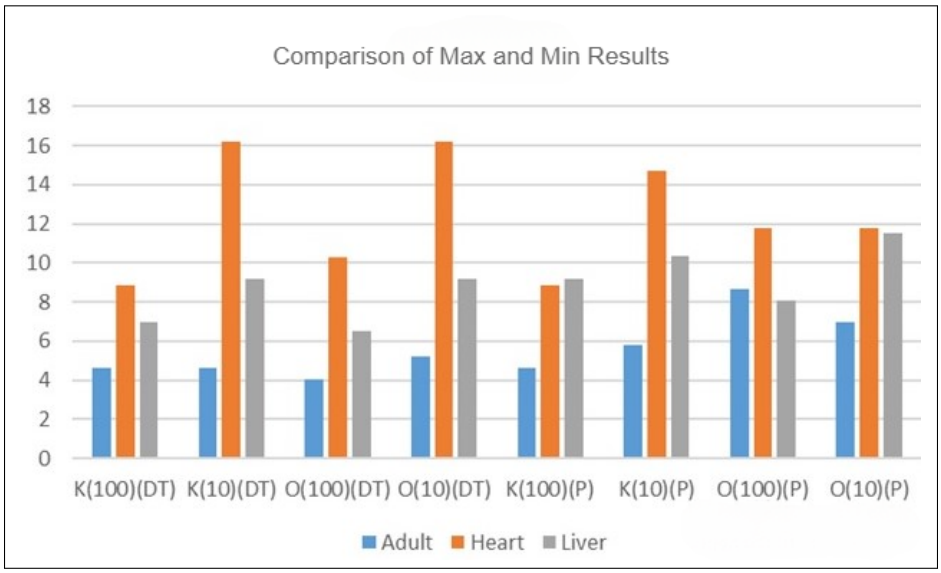


Figure 0.1 The difference between max and min results in 3 datasets. In this figure, KNORA-E is represented by K, O is used for OLA as DS techniques, DT for Decision Tree, P for Perceptron, and the number of the base estimator is 10 or 100

0.1.2 Redundant Classifiers in Bagging

Conventional pool generation schemes like Bagging and Boosting tend to generate redundant classifiers, which make the system computationally expensive (Ruta & Gabrys (2005); Cruz *et al.* (2015c); Woloszynski & Kurzynski (2011)). This redundancy not only consumes additional memory but also increases the computational burden during testing, as more classifiers lead to more calculations for selection and combination at runtime. A recent article (Cruz, Sabourin & Cavalcanti (2015a)) reveals that most of the classifiers generated using the Bagging technique are located within the same region, resulting in redundancy. In the bagging technique, the bootstraps are randomly taken from the training data, and as such, a highly diverse

pool is not guaranteed. To be more precise, while the redundancy may be beneficial for static ensemble models like Bagging since leads to reducing the effect of noise, such properties are not desirable for dynamic ensemble models as in dynamic ensemble we need a proper set of experts that covers the whole feature space.

Figures 0.2 illustrate the pool of classifiers generated by using the Perceptrons as the learning algorithms considering a pool of 5, 10, 25, 50, 75, and 100 base classifiers. This figure illustrates that, even with the addition of more classifiers to the pool, the majority of them closely resemble one classifier that had been generated earlier. Consequently, similar classifiers classify the new sample in a similar manner, and adding more classifiers does not enhance the pool's performance.

Such methods present these behaviors because they have a global perspective on feature space (Cruz *et al.* (2015a)) and are concerned with minimizing the global error of the system instead of focusing on local ones. As such DS methods can be limited by the quality of the generated pool as, if some feature space regions are not well covered, they will suffer in selecting the most competent models for it. Therefore, naively using a pool generation scheme that does not take any local information into consideration and was not proposed with a dynamic ensemble combination in mind may limit DS performance.

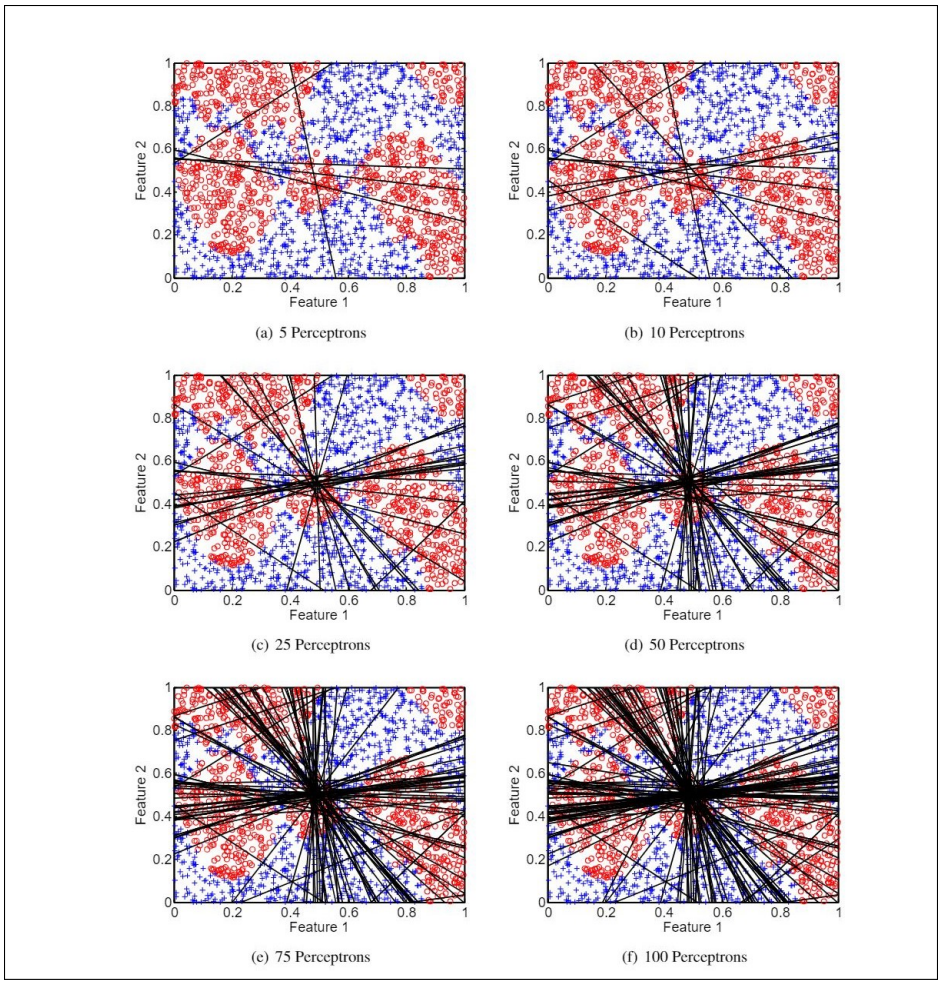


Figure 0.2 The Bagging technique with perceptron as the base classifier is used to generate the pool of classifiers, C, adapted from Rafael M.O. Cruz (2015, pp. 1509.00825)

We hypothesize that the performance of dynamic selection methods can be improved by providing the optimized pool of classifiers as input. Since the selection in DS algorithms is based on local information (i.e., the selection of competent classifiers locally), we hypothesize that pool generation schemes that explicitly consider local information can be a better alternative as it is more likely that it will better cover the feature space producing a pool of classifiers containing several local experts. This study investigated and compared the effects of different pool generation schemes for DS methods.

0.2 Objective

The main objective of this thesis is to develop a framework that optimizes the pool of classifiers to enhance the performance of dynamic selection techniques. The goal is to recommend an optimized pool generation scheme for DS methods by leveraging the meta-learning concept (Krijthe, Ho & Loog (2012)). This involves considering various pool generation schemes, including both local and global perspectives, to identify the most suitable pool for DS algorithms.

Specifically, this project aims to address the limitations and challenges in the current research related to DS techniques. Most existing articles have used conventional pools of classifiers such as Bagging (Breiman (1996)) or Boosting (Freund & Schapire (1997)) without fully exploring their impact on DS method performance. In this project, several pool generation schemes, including both global and local perspectives are compared. The results highlight that the selection of the most appropriate pool depends on the specific characteristics of the problem and the DS model used for selection. Suggesting that this part of a DS framework should not be neglected and a proper scheme for optimizing the pool should be considered.

The proposed approach introduces a novel classifier's pool recommendation system, which leverages meta-learning and dataset characteristics. By extracting the meta-features from each dataset, it becomes feasible to predict the best pool of classifiers. This prediction is based on the performance and the meta-features extracted from previously seen and evaluated datasets during the training stage of the meta-learning process. In other words, the system leverages the performances of models training on past data to predict the best configurations to the new one. As a result, the expectation is that this framework can achieve higher classification accuracy.

0.3 Contribution

The first contribution of this research is the analysis of global and local perspective pool generation schemes for DS algorithms. We aim to understand how these different approaches affect the performance of DS techniques when the generated pool of classifiers is used as input

for these methods. Because the local and global pool generation schemes would be used in the design of a novel method.

Up to this point, the literature has predominantly utilized global perspective pool generation schemes for DS methods (Ko *et al.* (2008); Woods *et al.* (1997); Cruz *et al.* (2015c)), while techniques that explicitly leverage local information for pool generation, such as the Forest of Local Trees (FLT) (Armano & Tamponi (2018)) and Locally independent classifiers (LIT) (Ross, Pan, Celi & Doshi-Velez (2020)), have not been considered in this context yet. Thus, we want to first analyze which is more suitable for the DS method among the global perspective pool generation schemes. Secondly, **we hypothesize that since the base classifiers selection mechanism in DS algorithms is based on local information, local pool generation schemes have the potential to enhance the performance of DS techniques**. It means while global pool generation schemes should not be ignored, local pool generation schemes should not be neglected as well. In addition to global pool generation schemes. So we want to know how much local pool generation scheme would be suitable for DS methods.

In line with this contribution, our research questions are:

- RQ1: Do local pool generation schemes achieve better results when used for dynamic selection algorithms?
- RQ2: Does selecting the best pool of classifiers, C, depend on what dynamic selection is used ?
- RQ3: Does selecting the best DS method depend on what pool of classifiers, C, is used?
- RQ4: Is there a relationship between static ensemble and dynamic selection results?
- RQ5: Does selecting the best pool of classifiers, C, for dynamic selection methods depend on what dataset is used?

The second contribution of this work is the development of a meta-learning recommendation approach for DS methods. With the application of meta-learning, we extract the meta-features from a given dataset and predict using the meta-model built during the training step. This meta-model was built by the meta-features that extracted training datasets and the best performance of

the several pools of classifiers used as input of DS algorithms. The approach consists of three phases. In the first phase, the approach predicts which pool of classifiers is better suited for DS algorithms. This phase considers the data characteristics and selects the pool that would provide the best performance. In the second phase, the approach predicts which DS method is appropriate for a given pool of classifiers based on the characteristics of a given data. This phase ensures that the DS method selected is the best fit for the specific pool of classifiers. In the third and final phase, the approach predicts which combination of pool and DS methods is best suited for a given input data. When classifying a dataset, the recommended pair is used to automatically select both a DS technique and the pool generation scheme for the DS method. Overall, this meta-learning recommendation approach provides a novel solution to optimizing DS methods, taking into consideration the dataset characteristics (i.e., the problem to be solved) and the DS model or criteria that will be used for selecting and combining the classifier later. Thus, it contributes to advancing the field as it is the first research work to specifically model these two properties to optimize the whole system's performance.

0.4 Structure of this thesis

The structure of this thesis consists of six chapters. Chapter 1 is dedicated to explaining the basic concepts related to Multiple Classifier Systems (MCS) and Dynamic Selection (DS) methods. This chapter aims to provide the fundamental concepts required to comprehend the later chapters.

Chapter 2 is a literature review that presents the related works in the field of dynamic selection and Meta-learning. This chapter provides an overview of the research conducted in the field and identifies the gaps the current research aims to fill.

Chapter 3 presents the experimental protocol that is used in chapters 4 and 5. This chapter provides information about the algorithms used, their hyperparameters, the library employed for the algorithms, and details about the collection of datasets used in this research.

Chapter 4 presents information about different pool generation schemes and is divided into two categories: global and local perspectives pool generation schemes. This chapter explains

the differences between these pool generation schemes. This research employs Bagging (B), Boosting (BS), and Random Forests (RF) as global perspective pool generation schemes, and Forest of Local Trees (FLT) and Local Independence Training (LIT) as local perspective pool generation schemes. This chapter aims to provide a comprehensive understanding of their differences and their characteristics. In this chapter, we present a comparison of different pools of classifiers and their performance on dynamic selection methods. This comparison's results show that we can not use just one pool generation scheme to generate a pool for DS techniques for all 288 datasets used. This finding highlights the need for a more sophisticated approach to optimize the pool of classifiers for dynamic selection algorithms. This leads us to the proposed meta-learning recommendation approach, which is presented in Chapter 5.

Chapter 5 presents the meta-learning recommendation approach for DS methods and provides a comparative study of the results. The chapter highlights the performance of the proposed approach and compares it with other state-of-the-art methods. Also, the meta-learning approach demonstrates its ability to recommend a pool generation scheme and a DS method, or both fully automated simultaneously, for a given dataset, leading to improved performance compared to fixed approaches.

Chapter 5.5.1 is the conclusion that summarizes the main contributions of the research and provides an overall evaluation of the research findings. This chapter also highlights the limitations of the research and identifies areas for future research.

CHAPTER 1

BASIC CONCEPTS

In this chapter, we will delve into the fundamental concepts behind Multiple Classifier Systems (MCS) and Dynamic Selection (DS) techniques. These concepts form the bedrock upon which our exploration of more advanced methodologies is built. By thoroughly understanding the principles of MCS and DS, we pave the way for the understanding of this work.

1.1 Multiple classifier System (MCS)

Multiple Classifier Systems (MCS), also called ensemble of classifiers, are based on the assumption that a combination of multiple experts can be an alternative to increasing classification accuracy as it can compensate for the limitations of selecting a single model (Dietterich (2000); Breiman (1996)). Thus, different classifiers are combined to improve the system's performance. Since different classifiers may make different errors on different samples, combining them into an ensemble can lead to more accurate decisions (Ko *et al.* (2008)). For this reason, researchers have extensively investigated MCS as a potential approach for improving accuracy in challenging pattern recognition contexts such as dealing with noisy data (Zhu, Wu & Yang (2004)), imbalance distributions (Souza, Sabourin, Cavalcanti & Cruz (2023)) as well as handling changing environments and concept drift (Almeida, Oliveira, Britto Jr & Sabourin (2018); Jiao, Guo, Gong & Chen (2022)).

According to (Britto Jr, Sabourin & Oliveira (2014)) MCS are composed of three stages, which are visually represented in figure 1.1. These stages are as follows:

1. **Generation**: This phase involves creating a pool of classifiers, C, with the aim of obtaining a set of accurate and diverse base models.
2. **Selection**: In this phase, classifiers can be chosen either statically or dynamically. In static selection, a fixed ensemble model is selected for all instances, whereas in dynamic selection, the most suitable classifiers are chosen at runtime for each instance.

3. **Aggregation or Fusion**: In the aggregation phase, a combination rule is applied to the outputs from the selected classifiers, C' (Cruz *et al.* (2018a)).

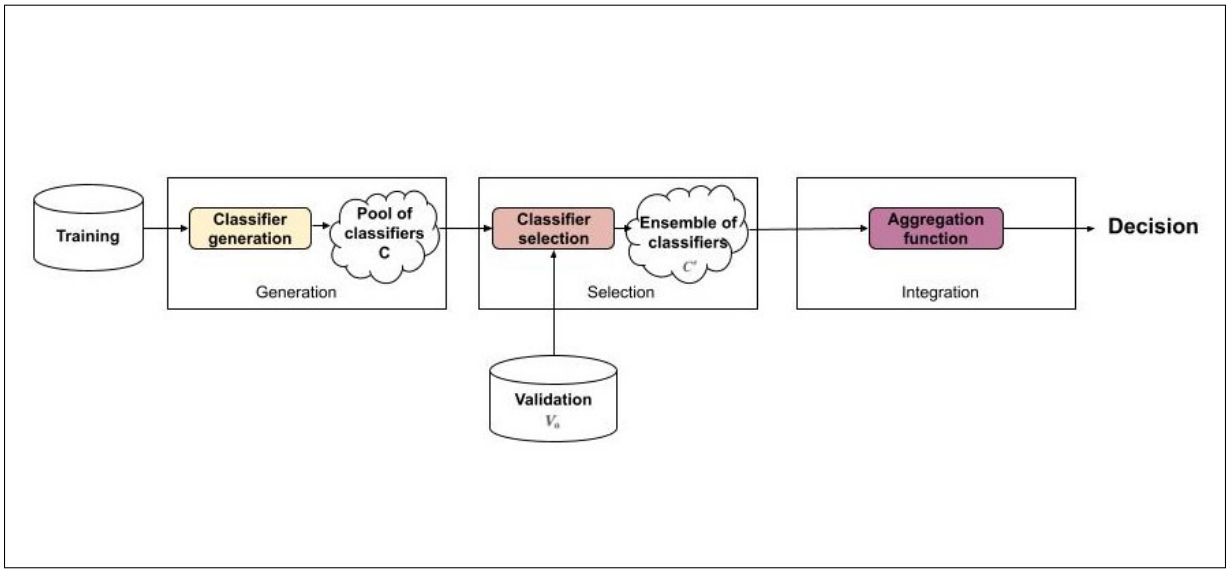


Figure 1.1 The three possible phases of a Multiple Classifier System (MCS)

The selection stage, as depicted in Figure 1.2, can be performed either statically or dynamically. Static selection methods use a criterion estimated using the validation dataset, V_a, to compose an Ensemble of classifiers, C', during the training phase. Then, the same selected ensemble of classifiers, C', predicts all test samples in the generalization phase. Static selection is mainly based on maximizing the accuracy and diversity of the ensemble and has been accomplished through a variety of methods, including greedy search (Cvetkovic & Martinović (2020)), evolutionary algorithms (Gabrys & Ruta (2006)) and heuristic approaches (Cruz, Cavalcanti, Tsang & Sabourin (2013); Cruz, de Sousa & Cavalcanti (2022)).

On the other hand, dynamic selection techniques work by, given a pool of classifiers, C, selecting either a single classifier C^* or an ensemble of classifiers, C', to classify each unknown example. When a single classifier C^* is selected, the system is called Dynamic Classifier Selection (DCS), and as such, only its prediction is taken into account for labeling the query sample. When an ensemble of classifiers $C^{'}$ is selected from the pool, the outputs of all selected classifiers, C',

need to be aggregated to give the final decision. This process is often conducted using classifier combination techniques such as classical Majority Voting (Cruz, Sabourin & Cavalcanti (2015b); Kittler, Hatef, Duin & Matas (1998)) or weighted voting scheme based on the competence level estimations computed during the selection scheme (Cruz *et al.* (2015b)).

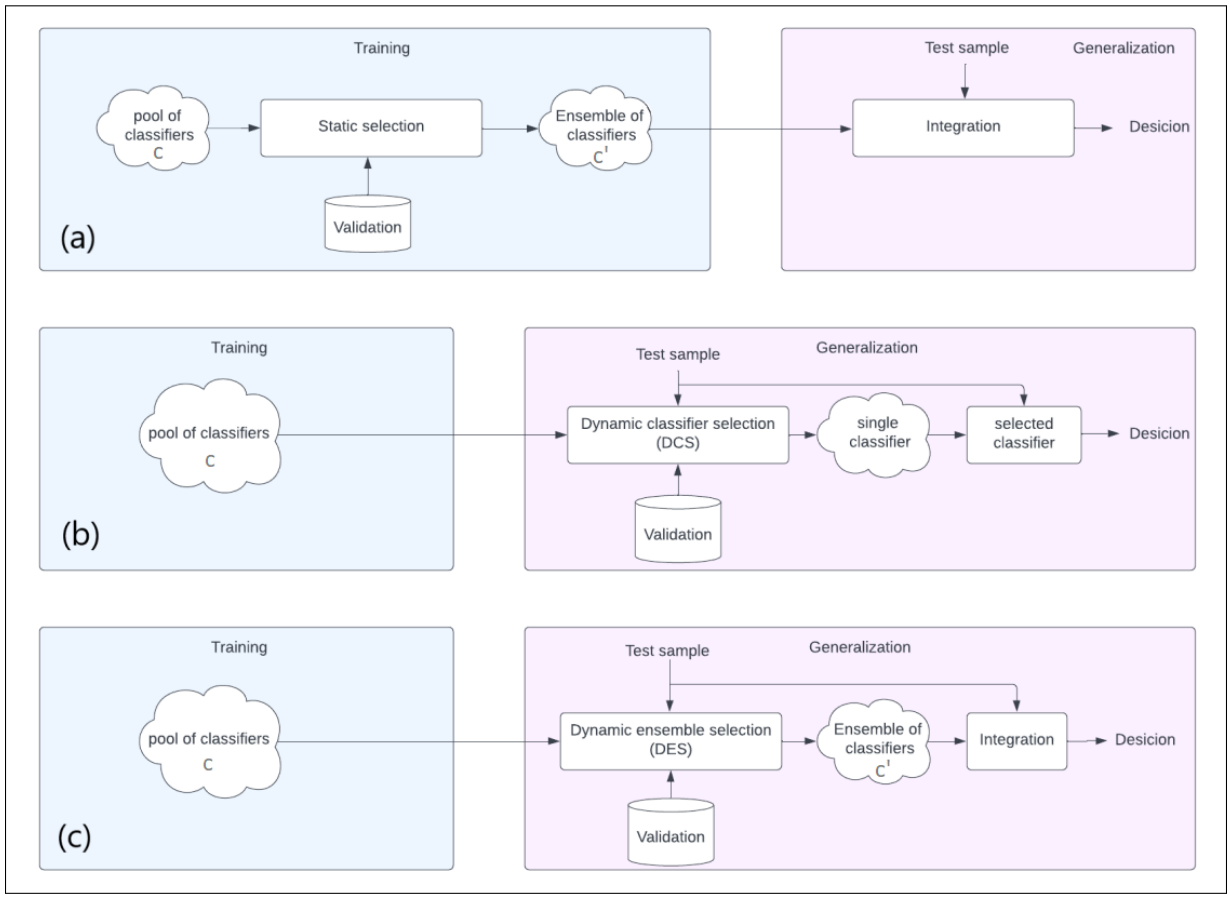


Figure 1.2 The differences between static selection (a), dynamic classifier selection (b), and dynamic ensemble selection (c)

1.2 Pool generation methods

In machine learning, pool generation refers to the process of creating an ensemble or collection of individual classifiers. There are two main approaches to pool generation: a global perspective and a local perspective. Global perspective methods consider the performance of individual algorithms on a global level, which means that the selection process is based on the performance

of each base model across the entire feature space. In other words, trying to model the whole data distribution of a given training set T_r. In contrast, local perspective methods consider the performance of each base model on the region of the feature space located around a query sample, $\mathbf{x}_j$. The goal of pool generation is to select a diverse set of algorithms that complement each other's strengths and weaknesses to produce an accurate and robust ensemble. In chapter 4, we will dive deeper into the details of pool generation schemes to generate a pool of classifiers as input of DS algorithms and compare them.

1.3 Dynamic selection

In order to select the competent classifier for the classification of $\mathbf{x}_j$, dynamic selection (DS) systems operate dynamically on-the-fly (Cruz *et al.* (2018a); Cavalin, Sabourin & Suen (2013)). In DS algorithms, the competence of the classifiers is estimated based on a local region of the feature space where the query sample, $\mathbf{x}_j$, is located. Thus, dynamic selection methods are based on the assumption that each classifier in the pool is a local expert in the particular region of the feature space.

The DS techniques are categorized according to the taxonomy shown in Figure 1.3. It is composed of four properties: Neighborhood-selection techniques, Neighborhood-selection space, Classifier-selection criteria, and Classifier-selection method (Elmi & Eftekhari (2021)).

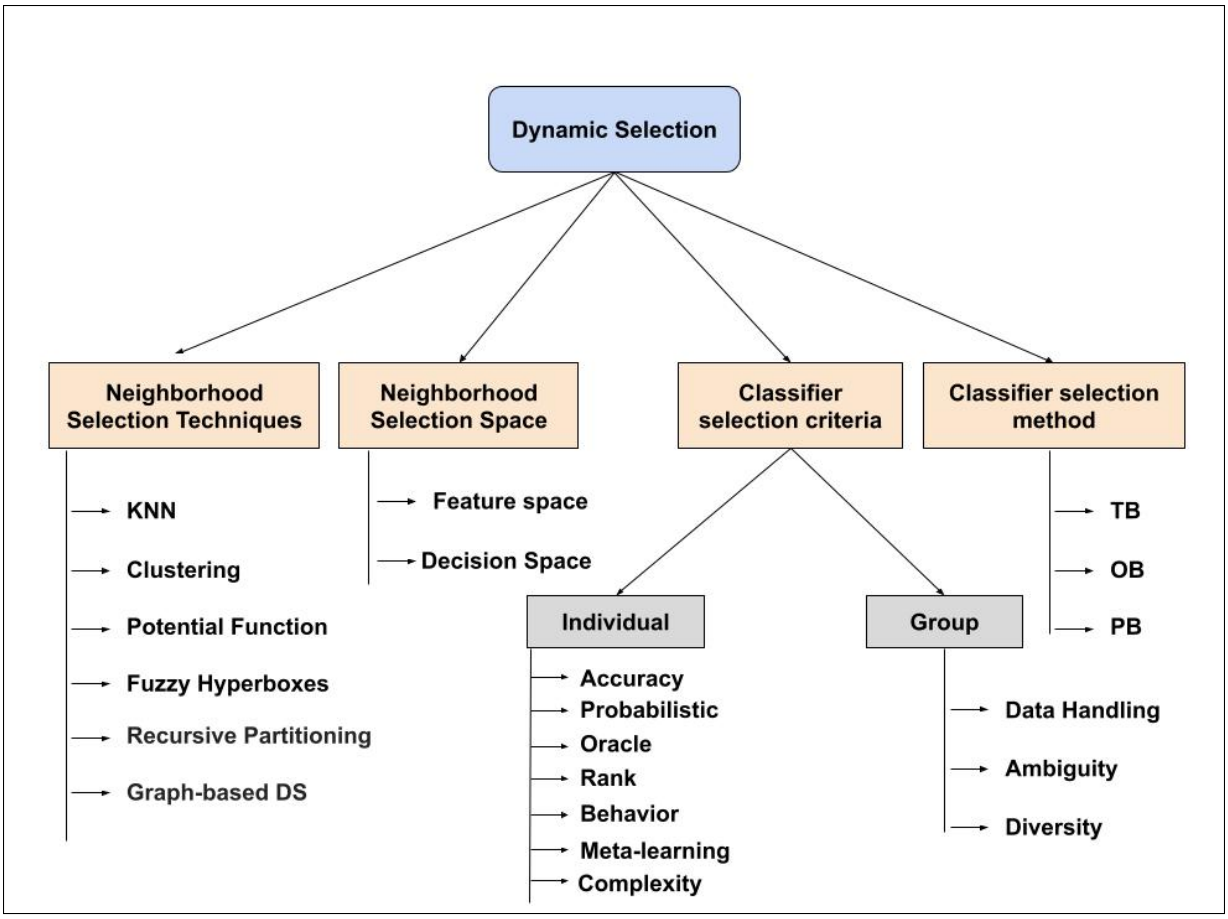


Figure 1.3 Taxonomy of dynamic selection systems adapted of the article. Threshold-Based (TB), Output-Based (OB), and Probability-Based(PB) are three strategies as classifier selection methods

Table 1.1 categorizes dynamic selection methods according to neighborhood selection technique, neighborhood selection space, classifier selection criteria, and classifier selection method. These techniques involve various methods for estimating the competence of individual classifiers and selecting the best one(s) for a given test instance. In order to define a local region around a query instance, neighborhood selection techniques are used, like k-nearest neighbors (KNN), Potential Function, Clustering, Fuzzy Hyperboxes, Recursive Partitioning, and graph-based DS. Also, Neighborhood-selection spaces refer to the specific areas in which the selection of neighboring data instances is carried out. This selection can occur in either the feature space or the Decision space. Classifier-selection criteria are measures used to determine the suitability of individual classifiers or groups of classifiers within a pool. These criteria can be broadly categorized into two groups: individual-based and group-based.

Classifier-selection methods are techniques used in dynamic selection to choose the most suitable classifiers from a pool for a given task. There are three main methods used for classifier selection: Threshold-Based (TB), Output-Based (OB), and Probability-Based (PB). TB method selects classifiers whose competence level is above a certain threshold. OB method selects a fixed number of the best classifiers based on their competitiveness. PB method employs a roulette wheel approach where classifiers are selected based on their competence levels.

Table 1.1 Categorization of dynamic selection methods according to neighborhood selection technique, neighborhood selection space, classifier selection criteria, and classifier selection method. They are ordered by publication year

DS techniques	Neighbor selection technique	Neighbor selection space	classifier selection criteria	classifier selection method	ref	year
LCA	KNN	Feature space	Individual-based	Output-Based	Woods *et al.* (1997)	1997
OLA	KNN	Feature space	Individual-based	Output-Based	Woods *et al.* (1997)	1997
MLA	KNN	Feature space	Individual-based	Output-Based	Smits (2002)	2002
KNORA-U	KNN	Feature space	Group-based	Threshold-Based	Ko *et al.* (2008)	2008
KNORA-E	KNN	Feature space	Individual-based	Threshold-Based	Ko *et al.* (2008)	2008
DES-P	Potential function	Feature space	individual-based	Threshold-Based	Woloszynski, Kurzynski, Podsiadlo & Stachowiak (2012)	2012
DES-KL	Potential function	Feature space	Individual-based	Threshold-Based	Woloszynski *et al.* (2012)	2012
KNOP	KNN	Decision space	Individual-based	Threshold-Based	Cavalin *et al.* (2013)	2013
META-DES	KNN	Meta-features	Individual-based	Threshold-Based	Cruz *et al.* (2015c)	2015
MCB	KNN	Decision space	Individual-based	Output-Based	Sergio, de Lima & Ludermir (2016)	2016
DES-PRC	Potential function	Decision space	Individual-based	Threshold-Based	Kurzynski, Krysmann, Trajdos & Wolczowski (2016)	2016
META-DES.O	KNN	Meta-features	Individual-based	Threshold-Based	Cruz, Sabourin & Cavalcanti (2017)	2017
DISi	KNN	Feature space	Individual-based	Threshold-Based	Pereira, Britto, Oliveira & Sabourin (2018)	2018
DES-MI	KNN	feature space	Individual-based	Threshold-Based	García, Zhang, Altalhi, Alshomrani & Herrera (2018)	2018
FIRE-DES++	KNN	Feature space	Individual-based	Threshold-Based	Cruz, Oliveira, Cavalcanti & Sabourin (2019)	2019
HF-MCDM	KNN	Feature space	Individual-based	Threshold-Based	Elmi & Eftekhari (2020)	2020
DDES	KNN	Feature space	Individual-based	Threshold-Based	Choi & Lim (2021)	2021
$MLSPB_{14}$	KNN	Feature space and Decision space	Individual-based	Probability-Based	Elmi & Eftekhari (2021)	2021
FH-DES	Fuzzy Hyperboxes	Feature space	Group-based	Threshold-Based	Davtalab, Cruz & Sabourin (2022, 2024)	2022

1.3.1 Neighborhood-selection techniques

Neighborhood selection (NS) techniques are utilized to define a local region in relation to a query instance $\mathbf{x}_j$. Figure 1.4 shows a region of competence, θ_j, of the query sample at the center and two samples of different classes. These techniques can be categorized into six distinct types: K-Nearest Neighbor (KNN), clustering, Potential functions (Woloszynski & Kurzynski (2011)), Fuzzy Hyperboxes (Davtalab *et al.* (2022, 2024)), Recursive Partitioning (Souza *et al.* (2023)), and Graph-based DS (Li, Wen, Li & Cai (2019); de Araujo Souza, Sabourin, da Cunha Cavalcanti & e Cruz (2023)). These techniques can identify the neighbors of the input data in both feature space (F-space) and decision space (D-space) (Elmi & Eftekhari (2021)).

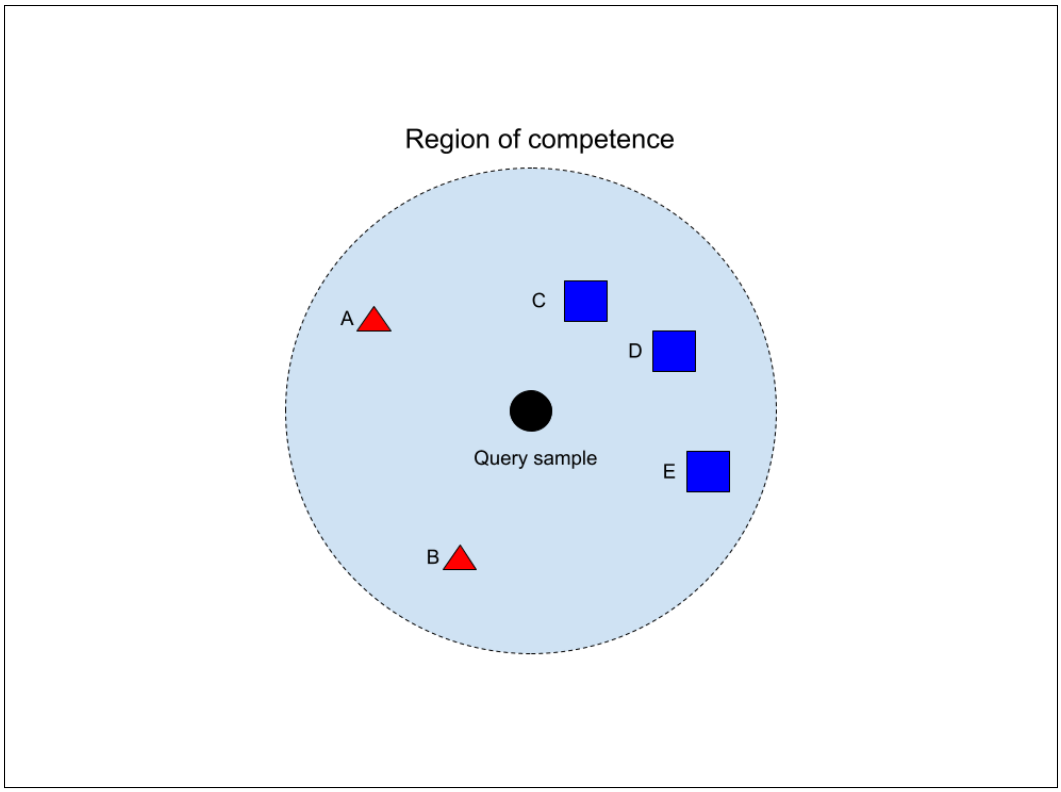


Figure 1.4 The test sample and its region of competence defined by the samples A, B, C, D, and E

The K-nearest neighbors (KNN) algorithm is a popular machine learning algorithm used for classification and regression tasks. Most of the DS techniques use KNN for neighborhood selection including LCA (Woods *et al.* (1997)), OLA (Woods *et al.* (1997)), MLA (Smits (2002)), KNORA-E (Ko *et al.* (2008)), KNORA-U (Ko *et al.* (2008)), KNOP (Cavalin *et al.* (2013)), META-DES (Cruz *et al.* (2015c)), and more. Another example is the article (Elmi & Eftekhari (2020)), the authors use the K-nearest neighbor (KNN) algorithm to define a neighborhood selection space for a given instance. The ensemble members that perform well on the instances in the neighborhood selection space are selected to make the final prediction for the given instance. The KNN approach is used to select a subset of instances in the training set that is closest to the given instance. The outputs of the ensemble members on the instances in the neighborhood selection space are then evaluated using Hesitant Fuzzy Multiple Criteria

Decision-Making (HF-MCDM) to select the most competent ensemble members for the final prediction (Elmi & Eftekhari (2020)).

Clustering is an unsupervised learning task that involves identifying hidden patterns within unlabeled input data through the creation of clusters. In simpler terms, it arranges data into meaningful natural groups based on the similarity between various features, revealing the data's underlying structure (Usama *et al.* (2019)). A clustering algorithm is used in ML, data mining, network analysis, pattern recognition, and computer vision for a large number of applications. In the articles (Kuncheva (2000)) and (Soares, Santana, Canuto & de Souto (2006)) the local region is defined using clustering methods. The ensemble in the article (Soares *et al.* (2006)) is created based on the clusters produced by the k-means clustering algorithm. When a testing pattern is given as input, its distances to the centroids (calculated using Euclidean distance) of the clusters formed by k-means are determined. The pattern is then assigned to the cluster with the nearest centroid. The ensemble is built by selecting the most accurate and diverse classifiers linked to the chosen cluster. The article (Beigy *et al.* (2009)) uses clustering methods to define the local region for the proposed DCS technique. In this approach, when encountering a new test pattern, the system determines which cluster it most closely resembles. After identifying the nearest cluster, the algorithm selects the classifier that was trained on this particular cluster and employs it to classify the test pattern.

Potential functions methods (Woloszynski & Kurzynski (2011)) can be used to determine a local region as a Neighborhood-selection technique. In the potential function method, the whole dynamic selection dataset is used to compute competence rather than just the neighborhood of the test sample. Each data point in the dynamic selection dataset is assigned a weight based on its Euclidean distance to the query instance. This means that data points closer to the query have a greater influence on estimating the competence of the classifiers (Cruz *et al.* (2018a)). Typically, a Gaussian potential function is employed, resulting in data points closer to the query exerting a stronger influence on the competence estimation of the classifiers. The function is presented in equation (1.1).

$$K(x_k, x_j) = \exp\left(-d(x_k, x_j)^2\right) \tag{1.1}$$

Several DS techniques have been proposed utilizing the potential function model. These include Dynamic Ensemble Selection based on Kullback–Leibler divergence (DES-KL) (Woloszynski *et al.* (2012)), the technique based on the randomized reference classifier (RRC) (Woloszynski & Kurzynski (2011)), and the DCS methods based on logarithmic and exponential functions (Woloszynski & Kurzynski (2009)).

The K-Nearest Neighbors (KNN) algorithm is often used in dynamic ensemble selection (DES) methods to assess the competence of classifiers in a small region surrounding the query sample. However, KNN has its limitations. One significant drawback is its sensitivity to the local distribution of data. This means that if the data points in that local region are unevenly distributed or skewed, the performance of KNN can be negatively affected. Furthermore, KNN comes with a high computational cost. It requires the entire dataset to be stored in memory, and during inference, it involves multiple distance calculations to determine the nearest neighbors. This computational complexity can become a limitation, especially when dealing with large-scale datasets. The need to store the entire dataset and perform numerous distance calculations can result in slower performance and increased memory consumption. Due to these limitations, researchers have proposed new methods as alternatives to relying solely on KNN in DES techniques. The paper (Davtalab *et al.* (2022)) presents a new DES framework based on fuzzy hyper-boxes called FH-DES. Each hyper-box can represent a group of samples using only two data points (Min and Max corners). This method is based on calculating the misclassification samples. This method selects the samples inside of each hyper-box based on not being misclassified. Despite the KNN-based approaches, the fuzzy hyper-box is not sensitive to the local data distribution.

The previous works in the field of dynamic selection had limitations concerning the construction of the test samples' region of competence. Despite their successes in selecting classifier subsets and achieving positive outcomes in different domains, these methods did not address how to

accurately define and construct the region of competence for test samples. This aspect, crucial for the effectiveness of dynamic approaches, was largely overlooked. Researchers have proposed new methods to overcome these limitations. Graph-based dynamic ensemble pruning (GDEP) (Li *et al.* (2019)) is a method for neighborhood selection that carefully selects nearby samples, evaluates the performance of individual classifiers, and forms a subset of effective classifiers for making predictions. It aims to overcome the sensitivity of classifier selection to the makeup of the neighborhood and uses graph-based techniques to achieve this. GDEP consists of three main steps: Construct the Neighborhood, Evaluate Classifier Performance, and Construct the Neighborhood. In the first step, GDEP starts by creating a group of neighboring samples around the test sample. These neighbors play a role in the classifier selection process. In the Evaluate classifier performance, the performance of the individual classifiers is assessed using a must-link and cannot-link graph structure. This step helps measure how well each classifier performs in the context of the neighborhood. And in the Form the Selected Classifier Subset step, based on the evaluation of classifier performance, GDEP selects a subset of classifiers that are most effective in recognizing the specific facial expression of the test image (Li *et al.* (2019)).

Conventional ensemble methods encountered limitations in achieving diversity among the classifiers within the ensemble, which is crucial for enhancing classification performance. While various approaches attempted to introduce diversity, they often faced challenges in incorporating local information effectively. Also, the existing approach employed the nearest neighbors rule and Euclidean distance for defining these local regions, which could be problematic in high-dimensional spaces due to the curse of dimensionality. To address these limitations, a novel local ensemble method was proposed in this work. Recursive Partitioning is a technique used in ensemble learning to enhance the performance of a classification system by creating a specialized group of classifiers that focus on distinct areas of feature space (Souza *et al.* (2023)). In Recursive Partitioning, the feature space is divided into smaller sub-regions through a process similar to constructing a decision tree. Each sub-region is associated with a set of local classifiers that are experts in that specific area. These local classifiers are generated based on different node levels of the decision path that a sample takes in the tree. In the context of the proposed method,

(Souza *et al.* (2023)), decision trees are used to partition the feature space into distinct regions. The partitions are defined by recursively splitting the data based on certain features and their values. Each leaf node of a decision tree is associated with a local pool of classifiers. The idea behind recursive partitioning is to break down a complex problem into simpler sub-problems that are easier to model or analyze.

1.3.2 Neighborhood-selection space

Neighborhood selection techniques aim to define a local region around a query instance. This local region contains other instances that are considered "neighbors" of the query instance. Neighborhood-selection techniques can be applied to the feature space (F-space) or the decision space (D-space). In F-space, we group together data that share similar features and consider them as neighbors in a specific area. This means that instances with similar characteristics are considered neighbors. On the other hand, in D-space, we focus on data that have similar output profiles (decisions) and consider them as neighbors in a specific area. The output profile of the instance $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is denoted by $\widetilde{\mathbf{x}} = \{\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_m\}$, where each $\widetilde{\mathbf{x}}_1$ is the decision yielded by the base classifier C_i for the sample $\mathbf{x}_j$ (Cruz *et al.* (2015c)). This process is called transforming the test instance to the output profile. In other words, output profiles refer to the set of classification outputs generated by a pool of classifiers for a given input. It represents the decision space (D-space) where each classifier's output is considered as a dimension.

1.3.3 Classifier-selection criteria

There exist several criteria to measure the competence level of base classifiers for the classification. The classifier-selection criteria are divided into two general groups, individual-based and group-based (Cruz *et al.* (2018a)). The individual-based measures are where the individual performance of the base classifier is used to estimate the level of competence in an independent fashion, i.e., the performance of each base classifier is measured independently of the performance of the other base classifier in the pool. Accuracy (Woods *et al.* (1997)), Probabilistic (Woloszynski & Kurzynski (2011)), Oracle (Ko *et al.* (2008)), Behavior (Cavalin *et al.* (2013)),

Ranking (Woods *et al.* (1997)), Meta-learning (Cruz *et al.* (2015c)), Data Complexity (Brun, Britto, Oliveira, Enembreck & Sabourin (2016)) are examples of individual-based classifier selection criteria.

Group-based measures are composed of criteria that consider the interaction between the classifiers in the pool (i.e., the competence estimation of a single model affects the selection of all others). There are three subgroups of it: Diversity (Soares *et al.* (2006)), Data handling (Xiao *et al.* (2012)), and ambiguity (Dos Santos, Sabourin & Maupin (2008)). These techniques measure the interaction between classifiers, assessing how effectively they collaborate for the selection, rather than treating each one as a completely independent model. The most accurate classifier should be selected, and the system has to check the base classifiers are related to the accurate classifier that was selected in order to add more diversity to the ensemble of classifiers, C'. DES-competence based on continuous-valued outputs and weighted class supports (DES-CS) (Woloszynski & Kurzynski (2011)) and DES system with a dynamic threshold of competence and class-dependent weights in majority voting procedure (DES-CD) (Lysiak, Kurzynski & Woloszynski (2011)) are examples of group-based methods.

1.3.4 Classifier-selection method

There are a few strategies to select a classifier after estimating the classifier's competencies: Threshold-Based (TB), Output-Based (OB), and Probability-Based(PB).

- In the Threshold-Based (TB) the classifiers whose competence level is greater than the threshold, Th, are selected like K-Nearest Output Profile (KNOP) (Cavalin *et al.* (2013)), K-Nearest Oracles-Union (KNORA-U) (Ko *et al.* (2008)), and DES Performance (DES-P)(Woloszynski *et al.* (2012)).
- Output-Based (OB) selects the most competent classifier as output. According to the article (Elmi & Eftekhari (2021)), several DS techniques use the OB method, including Multiple Classifier Behavior (MCB) (Sergio *et al.* (2016)), OLA (Woods *et al.* (1997)), and Local Class Accuracy (LCA) (Woods *et al.* (1997)).

- Probability-Based (PB) methods assign a probability to each classifier in an ensemble based on its competence level and use a roulette wheel approach to select the classifiers. The number of selected classifiers, C', is not fixed and may vary depending on the probabilities assigned. PB methods are a type of dynamic selection technique and can be used in both dynamic classifier selection (DCS) and dynamic ensemble selection (DES)(Elmi & Eftekhari (2021)).

CHAPTER 2

RELATED WORK

In this Section, we provide a brief overview of the selection of pool generation schemes as inputs for DS techniques, as well as the role of meta-learning in algorithm recommendation. We split it into three categories: 2.1. Selection of pool generation schemes. 2.2. Expanding Algorithm Recommendation using Meta-Learning in Dynamic Selection Contexts. This section explores Meta-Learning for Recommending Algorithms in a broader context and its application within the context of DS algorithms. 2.3. Meta-learning for recommending algorithms in the ensemble context.

2.1 Selection of pool generation schemes

A crucial factor in enhancing the performance of DS methods is the selection of an appropriate pool generation scheme to create a pool of classifiers as input for DS algorithms. This is of paramount importance because, in scenarios where there is no single local expert in the pool or the quantity of local experts for a given instance is low, the task of selecting an effective ensemble becomes either impossible or significantly challenging for a DS method. The selection of pool generation schemes for DS algorithms can be categorized into two main perspectives: global and local. It is observed that the majority of DS publications tend to employ global perspective pool generation schemes, while the utilization of local perspective schemes is relatively limited. An online pool generation method (Souza, Cavalcanti, Cruz & Sabourin (2019)) creates a local perspective pool of classifiers specifically for test samples in difficult regions of the feature space, considering the estimated classification difficulty of instances. By using classifiers generated locally, the DCS techniques can more effectively select the appropriate classifier for instances prone to misclassification. For query samples surrounded by easy instances, a simple nearest neighbors rule is employed. The method involves creating a local pool (LP) consisting of specialized classifiers. Each classifier is selected using a DCS technique from a local sub-pool that contains at least one competent classifier for each instance in the class overlap regions of the feature space. If the unknown instance's Region of Competence (RoC) is located in a difficult

region, the LP is dynamically generated using neighboring instances and used to label the query sample. However, if the query instance is far from the class boundaries, no pool is generated, and a simple nearest neighbors rule is used to obtain the output label. This work presents a methodology specifically designed for Dynamic Classifier Selection (DCS) techniques, not Dynamic Ensemble Selection (DES) which are promising in this field. In addition, the authors did not explore the utilization of meta-learning in the article.

Souza et al. (Souza, Cavalcanti, Cruz & Sabourin (2017)) introduced a comprehensive overview of the Oracle model and its relevance in the literature, particularly in relation to DCS techniques. The study highlighted the challenges faced by DCS techniques in selecting the best classifier according to the Oracle model, resulting in noticeable differences in accuracy rates. To address this issue, an ensemble generation method was proposed, ensuring a 100% Oracle accuracy rate in the training set. This incremental method generated binary classifiers by placing hyperplanes in the feature space, guaranteeing at least one competent classifier for each training instance. The method proved to be faster than traditional ensemble approaches and allowed for the automatic determination of the pool size based on the training data. Experimental results revealed a significant disparity between the theoretical limit and DCS techniques when pools were generated without Oracle information. Even with a 100% Oracle accuracy rate, the average accuracy rate achieved by DCS techniques was approximately 85%, indicating their struggle in selecting the most competent classifier. This discrepancy suggests that the Oracle model, despite its use in the literature, may not be the optimal guide for identifying a promising pool for DCS techniques, as these techniques rely on local data rather than global information provided by the Oracle model.

Monteiro et al. (Monteiro, Britto, Barddal, Oliveira & Sabourin (2021)) proposed a method that is a classifier pool generation approach guided by diversity estimated from data complexity and classifier decisions. The process starts by assessing the behavior of complexity measures on different subsamples of the dataset. Complexity measures that exhibit high variability across subsamples are selected for further pool adaptation. An evolutionary algorithm is then employed to optimize diversity in both the complexity space and the decision space. The proposed method focuses on creating a pool of classifiers by leveraging two types of diversity: complexity

diversity and decision diversity. Complexity diversity involves training classifiers on dataset subsets that represent sub-problems with varying levels of complexity. Decision diversity aims to generate classifiers that make different types of errors. The authors hypothesize that both types of diversity can guide the pool generation process as an optimization problem, which is solved using a multi-objective genetic algorithm. The contribution of this work lies in two aspects. Firstly, a new method for pool generation is proposed, which trains classifiers on data subsets with different complexity levels. Secondly, the authors demonstrate the positive impact of this approach when combined with dynamic selection methods. This method has a global perspective on problems.

2.2 Expanding Algorithm recommendation using meta-learning in dynamic selection contexts

In the field of classification, the process of selecting the most suitable classification algorithm for a specific problem is known as the Algorithm Selection Problem (ASP) (Rice (1976); Khan, Zhang, Rehman & Ali (2020)). Meta-learning has emerged as a powerful approach to address ASP, showing significant success in the domain of classification. Conventional approaches for algorithm selection, such as trial and error or expert knowledge, have limitations. Trial error is time-consuming and computationally expensive, while theoretical analysis may not cover all possible algorithms. Relying on domain experts can be costly and biased. To overcome these drawbacks, there is a growing demand for machine learning systems that automate algorithm selection. One approach is meta-learning based algorithm recommendation, which learns from the performance of algorithms on previous tasks. It accumulates knowledge and uses it to recommend suitable algorithms for specific tasks. This approach has been successful in various domains, including classification (Zhu, Yang, Ying & Wang (2018)), clustering (Pimentel & De Carvalho (2019)), regression (Lorena, Maciel, de Miranda, Costa & Prudêncio (2018)), optimization (Muñoz, Sun, Kirley & Halgamuge (2015)), and dynamic ensemble selection for classification improvement (Cruz *et al.* (2015c)). By automating algorithm selection, these systems overcome limitations and enable non-experts to apply machine learning more independently (Khan *et al.* (2020)). Meta-learning is about "learning to learn" (Hutter, Kotthoff & Vanschoren (2019)) and

involves using prior knowledge to improve learning systems. It is a broad field with various dimensions, and one important area is automated algorithm selection. Meta-learning treats algorithm selection as a typical learning problem, where dataset characteristics (meta-features) are the independent variables and the target variable is the estimation of algorithm performance. To handle the large space of problems and algorithms, meta-learning studies select problems of different complexities and diverse algorithms (Smith-Miles (2009)).

In the paper (Gemp, Theocharous & Ghavamzadeh (2017)), the authors explore state-of-the-art meta-learning methodologies and identify the limitations and research challenges in addressing Data preprocessing that is a challenging task in machine learning applications, involving data cleaning, imputation, and feature normalization, dimensionality reduction, and data balancing. Meta-learning encodes datasets with informative statistics called meta-features, assuming that similar datasets in meta-feature space exhibit similar behavior when used with similar models. Also, in the paper (Zagatti *et al.* (2021)), the authors propose a meta-learning-based recommendation system for data preparation. The system suggests five ranked pipelines, catering to users with different levels of experience. The top-ranked pipeline improves the performance of an AutoML system and is comparable to a reinforcement-learning-based algorithm but significantly faster. The method is tested in a real-world application, demonstrating its benefits and limitations. Overall, this work addresses the need for automating data preparation in AutoML platforms using meta-learning, showing promising results. Furthermore, the article (Bilalli, Abelló, Aluja-Banet & Wrembel (2016)), uses meta-learning for automated Data Pre-processing. The extensive evaluation conducted in this paper demonstrated that applying the recommended transformations enhances the final result of the algorithms for a wide range of datasets.

In the meta-learning context, it is important the selection of meta-features extracted from a given dataset. In the past, there have been attempts at meta-feature selection within the meta-learning framework. (Todorovski, Brazdil & Soares (2000)) made the initial effort using the zooming-ranking method. In this study, classical feature selection techniques were employed to identify relevant features. The article (Kalousis & Hilario (2001)) also explored the problem of meta-feature selection. However, their approach was limited to finding relevant

features for pairs of algorithms. This is because their definition of meta-learning focused on detecting the best classification algorithm in the context of pairs of algorithms. The paper (Reif, Shafait, Goldstein, Breuel & Dengel (2014)) conducted an empirical evaluation of various categories of meta-features to determine their suitability for predicting classification accuracies for standard classifiers. They also applied an automatic feature selection method to the entire set of meta-features used. However, the details of this feature selection method were not provided, and the number of datasets used was small. The article (Smith, Mitchell, Giraud-Carrier & Martinez (2014)) discusses the problem of recommending learning algorithms and their associated hyperparameters. The authors propose a meta-learning approach that uses past performance data to recommend the best algorithm and hyperparameters for a given dataset. They compare their approach to other popular methods such as grid search and random search and demonstrate its effectiveness in several experiments. The authors conclude that their method is a promising approach for automating the machine learning process and making it more accessible to non-experts.

The paper (Garcia, Lorena, de Souto & Ho (2018)) explores the use of complexity measures to enhance understanding and differentiate the performance of different techniques, considering factors such as class overlap, data separability, and distribution. The study compares various regression models' effectiveness in predicting classifier accuracies for classification problems. The results show that these models can accurately predict accuracies and identify the best classifier, outperforming randomly chosen or fixed classifiers. Using data complexity measures, accurate meta-models are built to predict expected accuracies and select the best classifier among popular techniques like Artificial Neural Networks (ANN), Support Vector Machines (SVM), Decision Tree (DT), and K-Nearest neighbors (kNN). Comparisons with baseline recommenders highlight the superior performance of meta-regressors Distance Weighted k-Nearest Neighbor (DWNN), Random Forests (RF), and Support Vector Regressors (SVR), with RF utilizing important complexity measures based on neighborhood information and structure.

2.3 Meta-learning for recommending algorithms in the ensemble context

Additionally, meta-learning is employed to recommend algorithms within the context of ensembles. The article (Pinto, Cerqueira, Soares & Mendes-Moreira (2017)) by using meta-learning and rank approach to learn from meta-data proposed an automated bagging system. In this publication, the authors employed a procedure that involved extracting dataset characteristics and leveraging past performance data. They used meta-learning techniques that take into account the characteristics of the data. However, the focus was primarily on using bagging as a global perspective pool generation scheme, without exploring the potential of local perspective pool generation schemes. Furthermore, the authors did not experiment with applying their meta-learning system to DS methods.

The size of the pool is important for controlling computational complexity and performance. Determining the appropriate pool size depends on factors like the choice of base classifiers, the DS method used, and the problem characteristics. Traditionally, researchers set a pre-specified pool size due to the computational expense of evaluating different sizes. However, this may limit the DS method's performance. The article (Roy, Cruz, Sabourin & Cavalcanti (2016)) proposed the prediction of the best size of a pool of classifiers for META-DES based on the classification complexity using the meta-learning method. This method builds a meta-regression model by extracting complexity measures and the best pool size of datasets. By using the meta-regression model, the prediction of the pool size for a given data is accessible. In the paper (Roy *et al.* (2016)), a meta-regression model that predicts a suitable pool size for DS algorithms based on the problem's intrinsic classification complexity is proposed. The prediction process in this article follows a meta-learning framework, involving training and application phases to obtain meta-examples and train a meta-regression learner for pool size prediction. In the application phase, the trained model is used to predict the best pool size for unseen datasets based on their meta-features.

(Feurer *et al.* (2015)) proposed a meta-learning approach for automated machine learning by combining ensemble methods and Bayesian optimization. The meta-learning applied in this

method is performed for warm-starting the Bayesian optimization technique. Their approach involved training meta-models to predict the performance of different learning algorithms on new datasets and using them to construct ensembles that achieved state-of-the-art performance on various classification tasks. In the article "Using meta-learning for multi-target regression" by (Aguiar, Santana, de Carvalho & Junior (2022)), the authors propose a meta-learning method to build ensembles of regression models for multiple targets. They use meta-features to describe the datasets and meta-learning algorithms to select the best models to include in the ensemble. The proposed method is compared to other methods on several datasets, and the results show that it outperforms these methods. The authors also analyze the importance of the meta-features in the performance of the method. The article provides insights into the use of meta-learning for building regression ensembles and highlights the potential of this approach.

The article (Sun, Liu, Chua & Schiele (2019)) presents a meta-transfer learning approach for few-shot learning, which addresses the challenge of learning from a few examples. The authors propose a meta-learning framework that transfers knowledge from pre-trained models to new tasks with few labeled examples. They evaluate their approach on several benchmark datasets and demonstrate its effectiveness in improving few-shot learning performance. The authors also compare their approach to state-of-the-art few-shot learning methods and show that it outperforms them. The results suggest that meta-transfer learning is a promising approach for few-shot learning tasks. Additionally, the authors show that their meta-transfer learning approach can be used to improve the performance of ensembles of few-shot learners. They propose a method for ranking the few-shot learners based on their performance on a validation set and use the ranked learners to construct an ensemble. The results demonstrate that their approach outperforms other ensemble methods on several benchmark datasets. The authors conclude that their meta-transfer learning approach, combined with ensemble learning and ranking, can significantly improve the performance of few-shot learning systems.

2.4 Critical analysis

Previous research works (Souza *et al.* (2019), Souza *et al.* (2017)) have introduced local pool approaches for DCS methods, but these methods lack suggestions for DES methods. Similarly, the article (Monteiro *et al.* (2021)) demonstrated the positive impact of DS methods but did not consider the local perspective. In certain studies, like those discussed in (Zagatti *et al.* (2021)) and (Bilalli *et al.* (2016)), a meta-learning approach was employed for data preparation; however, these works did not delve into the exploration of pool generation scheme recommendations, particularly within the context of ensemble learning.

On the other hand, in the works of (Reif *et al.* (2014)), (Smith *et al.* (2014)), and (Garcia *et al.* (2018)), meta-learning concepts were employed to select the best classifiers for static selection. However, (Roy *et al.* (2016)) and (Roy *et al.* (2016)) only focused on predicting the size of classifier pools, neglecting pool recommendations for DS methods.

While (Feurer *et al.* (2015)) proposed an ensemble for building an ensemble for classification and regression, their approach neither applied to DS methods nor provided recommendations for constructing ensembles. Additionally, (Pinto *et al.* (2017)) presented a meta-learning recommendation, but they solely considered a global perspective on pool generation schemes. As evident from these explanations, there is a notable gap in the literature concerning the use of meta-learning concepts for recommending suitable pool generation schemes as input for DS methods.

To the best of our knowledge, no existing work has directly tackled the specific problem we are addressing in this study, which is the multi-label recommendation for predicting the optimal pool generation scheme for DS techniques and the suitable DS method for a given data based on dataset characteristics. In this study, in addition to traditional global perspective pool generation schemes we employ two local perspective pool generation schemes that have not been explored before for DS algorithms. We then utilize a meta-learning strategy to predict the optimal pool generation scheme for DS techniques. Also, it would be able to predict the suitable DS method for a given data when we are uncertain about the appropriate technique to use. Furthermore,

it would be able to predict both the pool generation scheme and DS methods simultaneously. By combining these novel pool generation schemes and meta-learning, we aim to enhance our decision-making process and address the challenges of selecting the most suitable approach.

CHAPTER 3

EXPERIMENTAL METHODOLOGY

The goal of this study is to find an optimal pool of classifiers for dynamic selection algorithms. To reach this goal, an experimental study was conducted to compare 7 pool generation schemes including Bagging(DT), Bagging(P), Boosting(DT), Boosting(P), Random Forests(RF), Forest of Local Trees(FLT), and Local Independence Training(LIT) as input for 7 dynamic selection algorithms that shown superior performance based on literature including k-Nearest Oracle-Eliminate (KNORA-E) (Ko *et al.* (2008)), k-Nearest Oracle Union (KNORA-U) (Ko *et al.* (2008)), Dynamic Ensemble Selection framework using Meta-learning (META-DES) (Cruz *et al.* (2015c)), Dynamic ensemble Selection for multi-class imbalanced datasets (DES-MI) (García *et al.* (2018)), Dynamic Ensemble Selection performance (DES-P) (Woloszynski *et al.* (2012)), Overall Local Accuracy (OLA) (Woods *et al.* (1997)), and Modified Local Accuracy (MLA) (Smits (2002)). These techniques were selected based on their superior performance in the dynamic selection literature, as indicated by recent surveys and articles (Cruz *et al.* (2018a); Hou, Wang, Zhang, Wang & Li (2020)).

In our study, we prepared datasets by following some preprocessing steps. Initially, we used the StandardScaler also known as Z-score normalization (de Amorim, Cavalcanti & Cruz (2023)) from scikit-learn, to standardize the feature data. Thus, making sure all features have the same scale. To handle any missing data, we employed the SimpleImputer from scikit-learn, which replaces missing values in the standardized data with the mean of other values in the same column. This approach ensures that missing values do not adversely affect the model's performance. Additionally, we utilized the LabelEncoder from scikit-learn to convert categorical labels into a numerical format. This conversion allows the machine learning algorithms to work more effectively with the target variable.

3.1 Datasets

The comparative study uses 288 datasets from the Landscape Contest at ICPR 2010 (Macià, Ho, Orriols-Puig & Bernadó-Mansilla (2010)). This framework enables evaluation of the robustness of supervised classification techniques and identifies their limitations. To provide a thorough understanding of the datasets used in this study and the rationale behind their selection, we begin by introducing the problem and the methodology proposed by Macia (Macià *et al.* (2010)).

The competitiveness of classification techniques has been argued over a small set of repetitive problems over the past two decades. Without control over its characteristics, such as the similarity of data sets, a common test bed can lead to incomplete conclusions about the quality of learning algorithms. Problems that provide adequate coverage of the data complexity space would be necessary to perform studies. For this reason, Macia et al. (Macià *et al.* (2010)) devised a contest that uses a collection of problems selected based on their complexity. Therefore, these datasets offer a well-rounded and comprehensive basis for addressing our research questions.

The idyllic landscape refers to a landscape that covers the characteristic space, all the different types of problems, and their complexity. This landscape dataset involved two sets of data: S1, which was Binary labeled and used for training the algorithms, and S2, which was not labeled and used for testing the algorithms. Also, it consists of artificially generated datasets using an evolutionary multi-objective optimization approach to fill the data complexity space. In other words, having datasets with varying degrees of data complexity according to the different data complexity measures proposed by (Ho & Basu (2002)). By using Evolutionary Multi-objective Optimization (EMO) (Deb (2012)), instances are selected that satisfy the required complexity, i.e., minimize the complexity metrics and maximize the complexity metrics (Macià *et al.* (2010)). It is helpful to understand how well different learning algorithms perform and compare them based on complexity. Several complexity metrics were employed in dataset generation proposed in (Ho & Basu (2000)), including the number of feature dimensions, points, and the maximum Fisher's discriminant ratio. A comprehensive table listing all complexity metrics is presented in table 3.1 for reference.

Table 3.1 Complexity metrics used in dataset generation

	Complexity metric
1	Number of Feature Dimensions
2	Number of Points
3	Maximum Fisher's Discriminant Ratio
4	Volume of the Overlap Region
5	Minimized Error by Linear Programming
6	Percentage of Points on the Boundary
7	The Ratio of Average Intra/Inter-Class NN Distance
8	Error of 1NN Classifier
9	Nonlinearity of 1NN Classifier
10	Error of Linear Classifier by LP
11	Nonlinearity of Linear Classifier by Linear Programming
12	Number of Iterations in SUP K-Means Clustering

In the paper (Macià *et al.* (2010)), 12 complexity measures from (Ho & Basu (2002)), with the omission of the maximum Fisher's discriminant ratio, were used. 80,000 data sets running the EMO approach over five seeds, including Checkerboard, Spiral, Wave Boundary, Yin Yang, and Pima are generated. Checkerboard is a classical non-linear problem with heavily interleaved classes following a checkerboard layout.

Figure 3.1 shows the distribution of Checkerboard, Spiral, Wave boundary, and Tin Yang. Spiral is a problem with a non-linear class boundary following a spiral layout. A Wave Boundary is a linearly separable problem defined by a sinusoidal function. Yin Yang is a linear problem with small disjuncts, and the Pima is Indian Diabetes from the UCI repository.

In this paper, five initial data sets evolved using different objective configurations. Each data set was optimized using three complexity measures, resulting in eight experiments. This selection

of complexity measures led to 8 combinations like 000, 001, ..., and 111. Then Singular Value Decomposition (SVD) was conducted on all the complexity measures. This process allowed the construction of a space based on the first two principal components. To make the contest more manageable, a subset of data sets was chosen. The space was divided into 100 cells, and five data sets were randomly selected from each cell. The authors conducted a deliberate selection process by cherry-picking 300 data sets from a larger pool of 500 datasets. These 300 data sets were utilized in the contest. Figure 3.2 illustrates the distribution of the 300-data set sample from the generated collection.

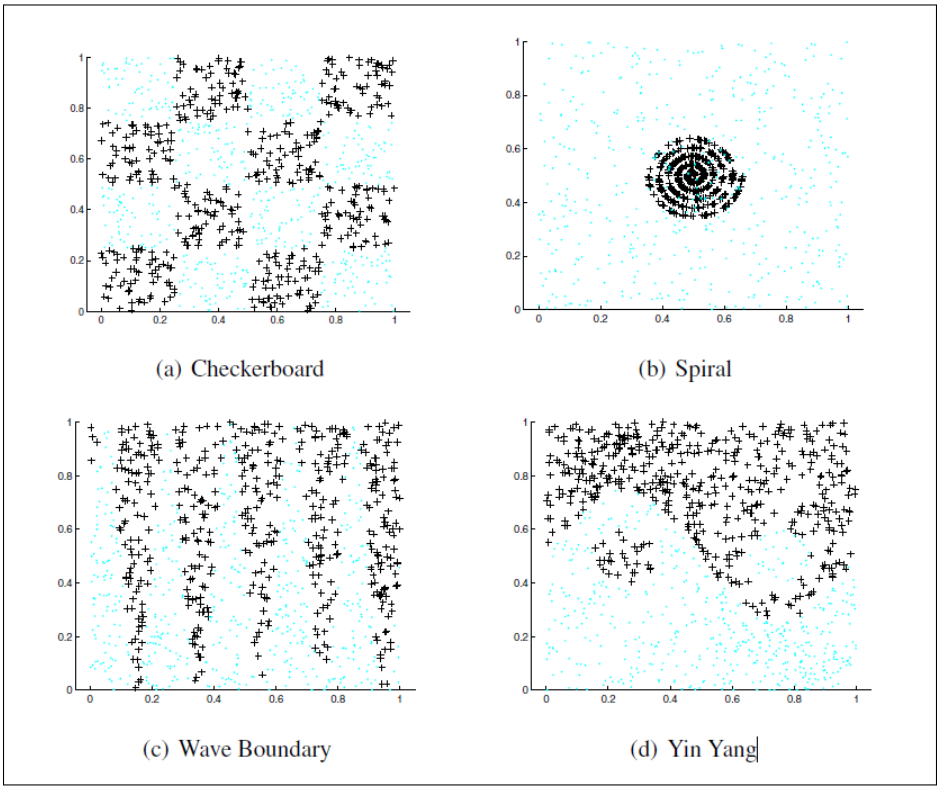
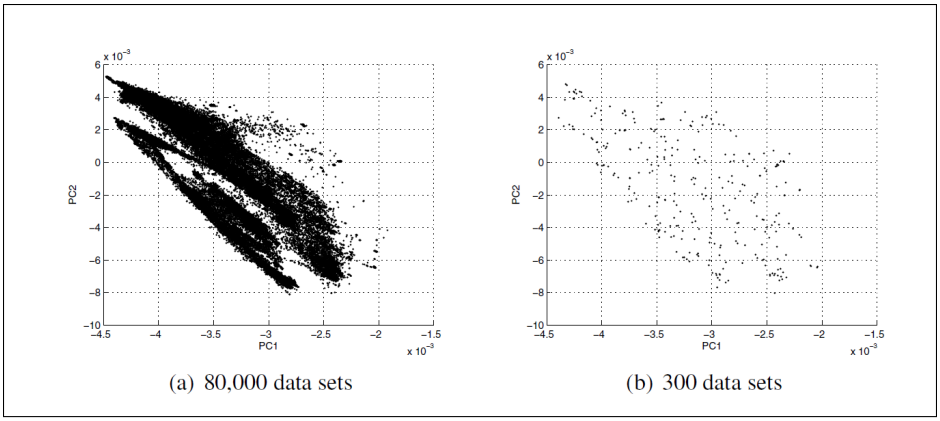


Figure 3.1 Distribution of Checkerboard, Spiral, Wave Boundary, Yin Yang adapted from Núria Macià (2010, pp. 29-45)



(a) 80,000 data sets (b) 300 data sets

Figure 3.2 Based on the complexity measurement, the problems are projected onto the first and second principal components: (a) the entire collection and (b) 300 cherry-picked training data sets adapted from Núria Macià (2010, pp. 29-45)

The characteristics of the data sets also depend on external factors like the number of instances, attributes, or a measure of the balance between the classes. The datasets are partitioned into two subsets distinguished by their attribute counts: one subset containing datasets with 8 attributes and the other containing datasets with 20 attributes. These external factors are considered in this article to generate the datasets by setting certain rules: (1) making sure we have a minimum number of instances, (2) conserving a specific class balance, (3) ensuring that the calculation of the complexity measures is feasible, and (4) avoiding having duplicate instances.

In this study, 13 datasets out of the initial set of 301 had to be removed due to a notable challenge encountered when using Boosting with a perceptron as a base estimator. The AdaBoost algorithm typically requires a diverse collection of classifiers, but it struggled with these 13 datasets. Most of the time, only one classifier was generated, resulting in a failure to meet the requirement of having more than one classifier in the ensemble. This issue arises from the algorithm's difficulty in creating a varied set of base classifiers that meet the boosting criteria, which mandate that each weak model should perform better than random chance. So, the pool of classifiers could not be created due to the model's inability to converge. In other words, the model failed to reach a stable and optimal set of classifiers. This observation highlights the sensitivity of the boosting

perceptron ensemble method to the characteristics of individual datasets, potentially indicating that the data distribution or inherent complexity of these particular 13 datasets may not align optimally with the algorithm's assumptions and requirements. The following datasets were removed from the S1 list: 216, 219, 220, 221, 252, 253, 254, 255, 257, 258, 260, 262, 263.

The datasets share a consistent distribution of instances, with the following characteristics: 58 datasets with 230 to 300 instances, 110 datasets with 301 to 400 instances, 52 datasets with 401 to 500 instances, 67 datasets with 501 to 1000 instances, and one dataset (dataset_301) with 9992 instances. Figure 3.3 shows the distribution of dataset sizes.

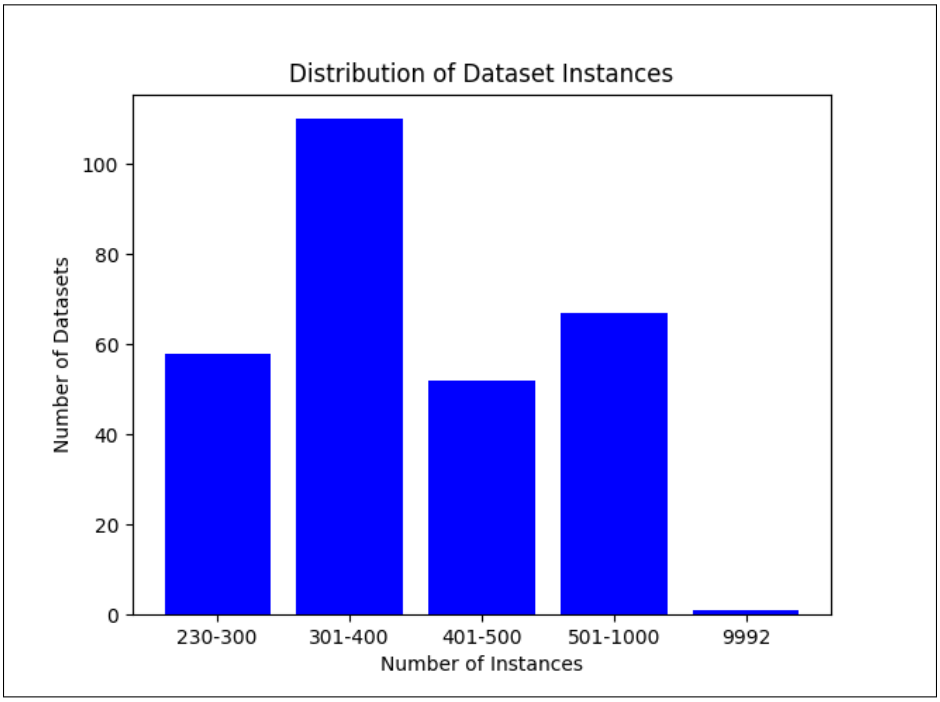


Figure 3.3 The distribution of dataset sizes

3.2 Experimental set up

In the experiment, each dataset is randomly divided into three groups, including 50% of the data used for training, the next 25% for the dynamic selection dataset (DSEL), and the remaining 25% for the data used for training testing. The divisions were performed while maintaining

the prior probabilities of each class. The pool of classifiers, c, comprises 100 base classifiers. In this research, 288 datasets are used to compare the performance of global and local pool generation schemes for dynamic selection methods. The experiments were replicated ten times for each dataset, and then their mean and standard deviation were extracted.

3.3 DS techniques

Seven dynamic selection algorithms were used in this research using DESlib version 0.3.5 as the DS library (Cruz, Hafemann, Sabourin & Cavalcanti (2020)). In order to obtain meaningful results and ensure a diverse set of models, we considered three DCS (Dynamic classifier selection) and four DES (Dynamic ensemble selection) methods. The following is a compilation of the names of the methods: Overall Local Accuracy (OLA) (Woods *et al.* (1997)) and Modified Local Accuracy (MLA) (Smits (2002)) as DCS and KNORA-E (Ko *et al.* (2008)), KNORA-U (Ko *et al.* (2008)), Meta-learning for dynamic ensemble selection (META-DES) (Cruz *et al.* (2015c)), Multiclass Imbalance (DES-MI) (García *et al.* (2018)), and Dynamic Ensemble Selection performance (DES-P) (Woloszynski *et al.* (2012)).

All DS techniques have been built upon the foundation of the K-Nearest Neighbors (KNN) method. We set K, the size of the region of competence, to 7 for the techniques as per previous findings in the article (Cruz *et al.* (2015c)). The number of output profiles used to estimate the competence of the base classifiers is set to 5 for the META-DES algorithm. Also, in the META-DES algorithm, the meta-classifier consists of a Multinomial Naive Bayes. “best” is used to select the base classifier after the competencies are estimated in the OLA and MLA techniques. This selection means that the DCS methods are all executed with their default behavior and the classifier with the highest competence estimate is always selected.

Table 3.2 Hyperparameters used to build the classification models

Category	Name	Hyperparameters	Library
Base estimator	Decision Tree	Criterion = Gini, Splitter = Best	sklearn: v1.0.2
Base estimator	Perceptron	Alpha = 0.0001, Max_iter = 1000, tol = 0.001	sklearn: v1.0.2
Static	Random Forest	n_estimators = 100, criterion = 'gini'	sklearn: v1.0.2
Static	Bagging(DT)	n_estimators = 100, base_estimator = DecisionTree	sklearn: v1.0.2
static	Bagging(P)	n_estimators = 100, base_estimator = Perceptron	sklearn: v1.0.2
Static	AdaBoost(DT)	n_estimators = 100, base_estimator = DecisionTree	sklearn: v1.0.2
Static	AdaBoost(p)	n_estimators = 100, base_estimator = Perceptron	sklearn: v1.0.2
Static	LIT	n_estimators = 100, base_estimator = Neural network(Fully connected 256*256)	sklearn: v1.0.2/TensorFlow v2.12.0
Static	FLT	n_estimators = 100, base_estimator = DecisionTree, maximum number of leaves = 100	sklearn: v1.0.2
DCS	OLA	Pool_classifiers = [Bagging pool], K = 7, Selection_method = Best	deslib: v0.3.5
DCS	MLA	Pool_classifiers = [Bagging pool], K = 7, Selection_method = Best	deslib: v0.3.5
DES	KNORAE	Pool_classifiers = [Bagging pool], K = 7	deslib: v0.3.5
DES	KNORA-U	Pool_classifiers = [Bagging pool], K = 7	deslib: v0.3.5
DES	META-DES	Pool_classifiers = [Multinomial Naive bayes], K = 7	deslib: v0.3.5
DES	DES-MI	Pool_classifiers = [Bagging pool], K = 7	deslib: v0.3.5
DES	DES-P	Pool_classifiers = [Bagging pool], K = 7	deslib: v0.3.5

3.4 Learning algorithms

Table 3.2 shows the entire list of base classifiers, static selection methods, and DS methods, along with their corresponding model hyperparameters. In this research, Decision Tree and Perceptron were utilized as base estimators using scikit-learn version 1.0.2. For the Decision Tree, the hyperparameters used were Gini for the criterion and Best for the splitter. As for the Perceptron, the hyperparameters included an alpha value of 0.0001, a maximum iteration of 1000, and a tolerance of 0.001. The selection of hyperparameters detailed in this table involved

a systematic approach that combined domain knowledge with empirical experimentation. This process included conducting a comprehensive literature review to identify recommended settings for similar models and problems. The initial settings for hyperparameters were based on recommendations from publications (Cruz *et al.* (2015c); Ko *et al.* (2008)), thereby leveraging the collective wisdom of prior research.

Three static selection methods were employed in this research: Bagging, Boosting, and Random Forests (RF). For Bagging and Boosting, the hyperparameters used were the number of estimators, which was set to 100. The base estimator chosen for the first use of Bagging was Decision Tree, while for the second use of Bagging, Perceptron was selected as the base estimator. Furthermore, the hyperparameters used for Random Forest included a base estimator count of 100 and a Gini criterion.

CHAPTER 4

EMPIRICAL ANALYSIS OF POOL GENERATION SCHEMES AND THEIR IMPACT ON DS ALGORITHMS

In this chapter, we empirically demonstrate that the success of a DS algorithm is closely tied to the quality of the input pool of classifiers and the selection of the best pool generation scheme may change according to the DS method used. Therefore, selecting the optimal pool is important, as different pools can result in varying performances. We explore the generation schemes for different pools of classifiers, including global and local perspectives. The Global pool generation scheme employs techniques that take a broad view of the problem and were initially proposed for static selection methods such as Bagging. In contrast, the Local pool of classifiers perspective involves methods with expertise in distinct feature space regions. In this chapter, we compare both local and global perspectives pool generation schemes for DS techniques including the top 4 DES and 3 DCS methods according to an empirical study (Cruz *et al.* (2018a)). This selection enables a meticulous exploration of the influence of pool generation schemes on both DCS and DES methodologies. In addition, a statistical analysis is presented to answer the research questions of this study:

- RQ1: Do local pool generation schemes achieve better results when used for dynamic selection algorithms?
- RQ2: Does selecting the best pool of classifiers, C, depend on what dynamic selection is used ?
- RQ3: Does selecting the best DS method depend on what pool of classifiers, C, is used?
- RQ4: Is there a relationship between static ensemble and dynamic selection results?

4.1 Global pool generation techniques

Pool generation schemes take a global perspective on problems, focusing on generating classifiers that do not rely on specific regions of the feature space and have a global perspective on datasets (i.e., it aims to model the whole data distribution). For instance, we can cite techniques such as Bagging (Breiman (1996)), Boosting (Schapire & Freund (2013)), Random Forests (Breiman

(2001)), Heterogeneous ensemble methods which use different base learning algorithms to directly ensure ensemble diversity (Reid (2007)), Rotation Forest (Kuncheva & Rodríguez (2007)), Random Subspace (Zhang & Pham (2011)). Most DS research publications employ pools of classifiers generated using well-known ensemble generation methods, all of which have a global perspective, as indicated in Table 4.1. This table provides an overview of the different pool generation schemes used in publications. Remarkably, the utilization of local pool generation schemes appears to be neglected by the DS literature as no recent techniques have considered them. In the following sections, global and local perspective pool generation schemes are explained in detail.

Table 4.1 Pool generation schemes used in publications

Publications	Pool of classifiers	Pool perspective	DS methods
Woods *et al.* (1997)	Heterogeneous classifiers	Global	LCA
Woods *et al.* (1997)	Heterogeneous classifiers	Global	OLA
Smits (2002)	Bagging	Global	MLA
Ko *et al.* (2008)	Bagging, Boosting and Random Subspaces	Global	KNORA-U
Ko *et al.* (2008)	Bagging, Boosting and Random Subspaces	Global	KNORA-E
Woloszynski *et al.* (2012)	Bagging and Heterogeneous classifiers	Global	DES-P
Woloszynski *et al.* (2012)	Bagging and Heterogeneous classifiers	Global	DES-KL
Cavalin *et al.* (2013)	Bagging	Global	KNOP
Cruz *et al.* (2015c)	Bagging	Global	META-DES
Kurzynski *et al.* (2016)	Heterogeneous classifiers	Global	DES-PRC
Sergio *et al.* (2016)	Heterogeneous classifiers	Global	MCB
Cruz *et al.* (2017)	Bagging	Global	META-DES.O
Pereira *et al.* (2018)	Bagging	Global	DISi
García *et al.* (2018)	Random Forests	Global	DES-MI
Cruz *et al.* (2019)	Bagging	Global	FIRE-DES++
Islam, Liu, Li, Liu & Kang (2019)	Random Forests	Global	PCC-DES
Elmi & Eftekhari (2020)	Bagging	Global	HF-MCDM
Choi & Lim (2021)	Heterogeneous classifiers and Boosting	Global	DDES
Elmi & Eftekhari (2021)	Bagging	Global	MLSPB_{14}
Choi & Lim (2021)	Boosting	Global	DDES
Davtalab *et al.* (2022, 2024)	Bagging	Global	FH-DES

4.1.1 Bagging

As originally introduced in the paper by (Breiman (1996)), the Bagging technique operates by training base learners independently. It utilizes data transformations to enhance the diversity of the model's predictions, a crucial step in the ensemble learning process. Bagging is a popular ensemble learning method that combines predictions from multiple models to improve the accuracy and stability of the final prediction.

Bagging works by selectively choosing a random subset of the training dataset to train each model within the ensemble (Breiman (1996); Skurichina & Duin (1998); Zhou (2012)). The core concept underlying bagging is to independently train multiple base learners on distinct subsets of the training data. Subsequently, their predictions are combined using an aggregation function. Typically, these base learners are simple models, such as linear Perceptron or Decision Trees, each trained using different bootstrap taken from the original data distribution.

The primary objective here is to heighten diversity among the base learners, which, in turn, effectively reduces the variance in the final prediction. This method ensures equal weight is assigned to each base learner's prediction. As a result, the final prediction is calculated as the mean of all these individual predictions. A graphical representation of the bagging algorithm is shown in Figure 4.1.

Bagging, a pool generation scheme, is formalized in algorithm 4.1. The algorithm takes three inputs, including training datasets, T_r, a Learning algorithm, $\mathcal{A}$, and the number of iterations denoted by M. The output is a pool of classifiers, denoted by C. The first step in the algorithm is to initialize an empty pool of classifiers, C. This pool will eventually contain the base classifiers that are trained during the algorithm. Next, the algorithm enters a loop that will run M times. For each iteration, a bootstrap sample, denoted by, $T_{r(i)}$, is drawn randomly with replacement from the original training set, T_r. This means that some samples will be included multiple times in the bootstrap sample, while others may not be included at all. After the bootstrap sample is obtained, a base classifier, denoted by C_i, is trained on the bootstrap sample using the specified learning algorithm, $\mathcal{A}$. The learning algorithm could be any algorithm used to train a classifier, such as a

Perceptron or Decision Tree. Weak base classifiers are employed in this study because prior research in the DS literature illustrates that utilizing weak models as base classifiers improves their performance (Dos Santos *et al.* (2008); Cruz, Cavalcanti & Ren (2011)). Earlier research shows that it is beneficial for these classifiers to show instability, indicating that even a slight change in the data distribution leads to notable variations in their prediction behavior(Cruz *et al.* (2015c)). Once the base classifier is trained, it is added to the pool of classifiers, C, by appending it to the list of classifiers already in the pool. This process is repeated for each of the M iterations. Finally, the algorithm returns the pool of classifiers, C, which is composed of M base classifiers.

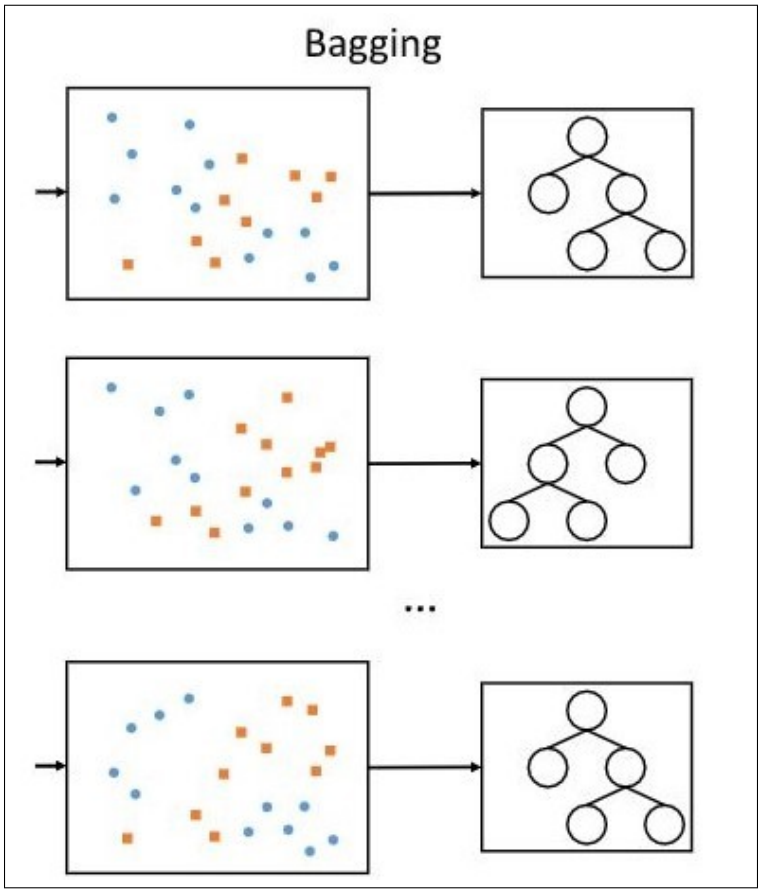


Figure 4.1 Graphical representation of the Bagging algorithm adapted from Sergio González (2020, pp. 205-237), illustrating its iterative process using bootstrap for generating a pool of classifiers

Algorithm 4.1 Bagging assembles a pool of M base classifiers by repeatedly generating bootstrap samples from the training dataset, T_r, and training individual classifiers using a specified learning algorithm $\mathcal{A}$

Input: Training dataset T_r, a learning algorithm $\mathcal{A}$, iterations M

Output: A pool of classifiers, C, composed of M base classifiers

$C = \emptyset$ pool of classifiers is empty

for $i = 1$ *to* M **do**

  $T_{r(i)}$ = bootstrap sample from T_r

  C_i = train a classifier on $T_{r(i)}$ via learning algorithm, [$\mathcal{A}$]

  $C = C \cup C_i$ add C_i to C

end for

Return the trained pool of classifiers C

4.1.2 Boosting

Boosting applies an iterative strategy for turning a weak model into a stronger one to fix its weaknesses (Freund & Schapire (1997)). The idea of boosting is to learn several weak classifiers and combine them using differently weighted versions of the training data, rather than learning one strong classifier (Kuncheva (2014a)). In this ensemble method, in each iteration, the error is calculated for every training instance, and a function of this error is utilized to update the probability that the instance is selected to be part of the next classifier's training set (González, García, Del Ser, Rokach & Herrera (2020)). Every instance in the training data is given a weighting, ω_j, based on the accuracy of the previous classifiers, allowing the algorithm to focus its attention on samples that are still incorrectly classified at the end of each iteration (Freund (2001)). Figure 4.2 illustrates graphically how Boosting algorithms work.

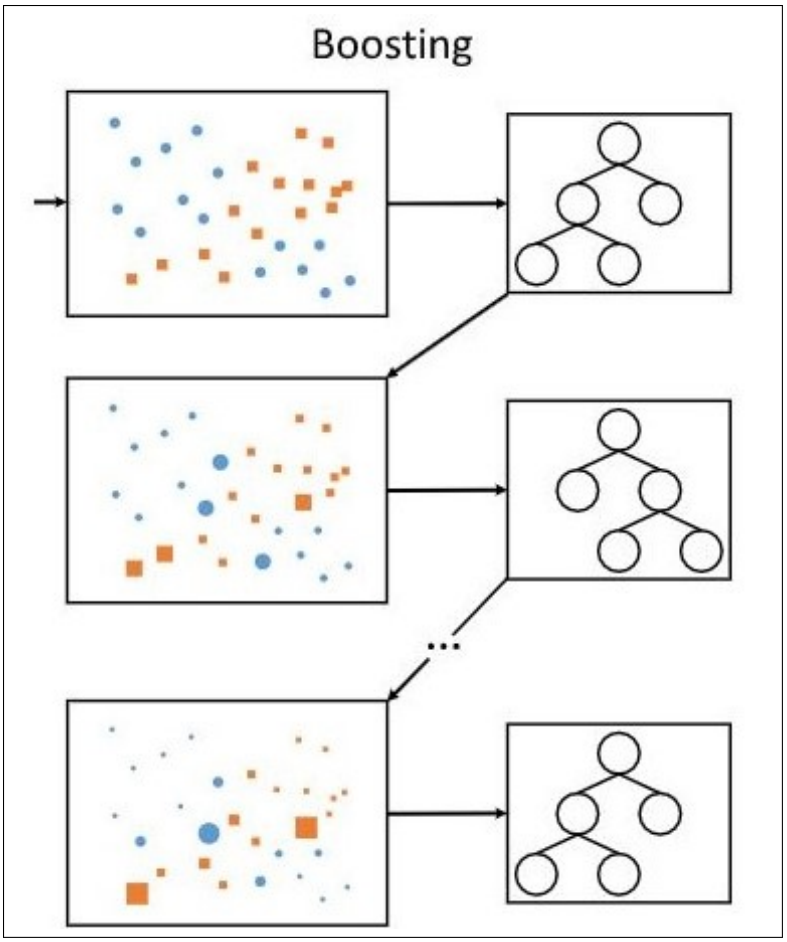


Figure 4.2 Graphical representation of the Boosting algorithm adapted from Sergio González (2020, pp. 205-237), illustrating how boosters learn from previous errors by enhancing the importance of incorrectly predicted training instances in future iterations

AdaBoost, short for Adaptive Boosting, is one of the most recognized boosting algorithms, as introduced by Freund and Schapire in their foundational research (Freund & Schapire (1997)). At its core, AdaBoost focuses on enhancing the performance of a series of weak learners, typically decision trees, by iteratively adjusting the weights of training instances based on previous classification errors. As each weak learner is trained, AdaBoost emphasizes the misclassified instances from the preceding iteration, guiding the subsequent learner to correct these errors. Furthermore, during the training phase, the predictions from each weak learner are aggregated

with weights that reflect their individual accuracy, as further elucidated by Gonzalez (González *et al.* (2020)). Although initially designed with binary classification in mind, its application has been successfully extended to multiclass classification tasks as well (Hastie, Rosset, Zhu & Zou (2009)).

The AdaBoost algorithm, outlined in algorithm 4.2, takes in a training sample, T_r, a learning algorithm, $\mathcal{A}$, and the number of iterations, M, and outputs a pool of classifiers, denoted by C, composed of M base classifiers. The algorithm starts by initializing an empty pool of classifiers, C. This algorithm assigns initial instance weights to each training instance, where is set to 1/N, with N being the total number of instances in the training dataset. AdaBoost then enters a loop that runs for M iterations. In each iteration, a base classifier, denoted by C_i, is trained on the sampled data using the learning algorithm, $\mathcal{A}$. The algorithm then calculates the error rate, e_i, of the base classifier and calculates a weight for it, denoted by α_i. Then, the weights of correctly classified instances are adjusted to reduce their importance in the subsequent iteration, making them smaller. The normalization step occurs after updating the instance weights. This step ensures that the sum of all instance weights remains equal to 1, which helps maintain the overall balance of the weighted dataset. This process ensures that instances that are frequently misclassified receive higher weights, making them more influential in subsequent iterations. Finally, the base classifier is added to the pool of classifiers, C, and the process is repeated for M iterations. The pool of classifiers, C, is then returned as the output of the algorithm.

Algorithm 4.2 AdaBoost leverages a pool of classifiers by iteratively updating base classifiers to improve their performance on a training dataset, T_r

Input: training dataset T_r, Learning algorithm $\mathcal{A}$, iterations M

Output: A pool of classifiers, C, composed of M base classifiers

$C = \emptyset$ pool of classifiers, C, is empty

Initializing M as an array of the size of classifiers and initializing weights, $[\omega]$ for each instance as $1/N$

for $i = 1$ *to* M **do**

  C_i = Fit_classifier(T_r, ω)

  $\epsilon_t = \frac{\sum_{\mathbf{x}_i \in D, T_t(\mathbf{x}_i) \neq \mathbf{y}_i} \omega_i}{\sum_{\mathbf{x}_i} \omega_i}$

  $\alpha_t = \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

  for x_j *in* T_r **do**

    if $T_t(\boldsymbol{x}_i) = \boldsymbol{y}_i$ **then**

      $w_i = w_i \cdot \frac{\varepsilon_t}{1-\varepsilon_t}$

    end if

  end for

  Normalize(ω)

  $C = C \cup C_i$ add C_i to C

end for

Return the trained pool of classifiers, C

AdaBoost is a powerful algorithm for boosting the performance of machine learning models. However, like any other algorithm, it has certain limitations that can impact its performance. One significant concern is its susceptibility to noisy data, which might lead to overfitting and consequently diminish the effectiveness of the models (Zhu & Hovy (2007)). This sensitivity also extends to outliers, which can have a pronounced effect on both the training process and the ultimate results (Bischl, Mersmann, Trautmann & Weihs (2012)). In terms of computational demands, AdaBoost can be resource-intensive. This is primarily due to the requirement to train multiple weak learners, a process that becomes particularly taxing for extensive datasets or

intricate models (Chen & Guestrin (2016)). Additionally, AdaBoost might falter when applied to imbalanced datasets. In cases where the minority class is scarcely represented, there's a risk that the algorithm may skew its focus towards the majority class, neglecting the minority class in the process (Kubat, Matwin *et al.* (1997)).

Gradient Boosting (Friedman (2002)), eXtreme Gradient Boosting (XGBoost) (Chen & Guestrin (2016)), LightGBM (Ke *et al.* (2017)) and CatBoost (Dorogush, Ershov & Gulin (2018)) are other multi-class Boosting methods. Gradient Boosting is a boosting algorithm that combines multiple weak learners to create a strong learner. It uses a gradient descent approach to optimize the loss function, $\mathcal{L}$, and iteratively updates the weights of the training examples (Friedman (2002)). Extreme Gradient Boosting (XGBoost) is a gradient boosting algorithm that uses a more regularized model for better generalization and is optimized for parallel processing. It includes advanced features such as handling missing values, regularization, and cross-validation (Chen & Guestrin (2016)). LightGBM is a gradient-boosting algorithm that uses a histogram-based approach to binning and feature parallelization to achieve faster training and higher efficiency. It includes features such as handling categorical features, outlier detection, and advanced regularization techniques (Ke *et al.* (2017)). CatBoost is a gradient boosting algorithm that handles categorical features and uses a novel algorithm for gradient computation that reduces overfitting. It includes features such as handling missing values, text features, and advanced visualization tools for model interpretation (Dorogush *et al.* (2018)). These techniques have not been considered for this study because all these boosting techniques primarily train regression algorithms as base models rather than classification ones. This specific requirement is driven by the need for a differentiable loss function, which is central to their functioning. The obstacle in classification arises because classification inherently deals with discrete classes. To predict distinct categories, using a differentiable loss function presents challenges. For example, in the case of the classification of images of cats and dogs, there's no smooth way to differentiate between these two distinct categories because they are not continuous values. Consequently, given this fundamental distinction, these boosting techniques are not ideally suited to serve as pool-generation schemes for DS methods in our context. Their inherent focus on regression

limits their applicability to classification tasks, necessitating exploring other methodologies better aligned with our objectives. in other words, as the base model they generate is a regression model, it limits the usage of it as a pool generation scheme for classification. Gradient and XGBoost can be used for classification, but in this scenario, they combine the output of the regressors to give the final decision. As a pool generation scheme, this would not be applicable as we require that base models are classifiers and not regressors.

4.1.3 Random Forests

Random Forests is a type of ensemble learning method proposed by Breiman (2001), which uses a collection of decision trees to make predictions. Each tree in the ensemble is built using a random subset of the input data and a random set of input features. This injection of randomness during the construction of the trees ensures that they are different from each other, which helps to reduce overfitting and improve generalization (Biau (2012)). The final prediction of the Random Forests is obtained by aggregating the predictions of all the individual trees in the ensemble. Since the base constituents of the ensemble are tree-structured predictors, and each tree is grown in accordance with a random parameter, the method is named "Random Forests" (Biau (2012)).

The Random Forests algorithm, outlined in algorithm 4.3, takes as input a training set T_r, the number of trees to create M, and the number of features to consider at each split N_f. The algorithm starts by creating an empty pool of classifiers C. It then loops M times, and for each iteration, a bootstrap sample $T_{r(i)}$ is drawn from the training set T_r. Then, a decision tree C_i is trained on the bootstrap sample using N_f randomly selected features at each split. The trained decision tree C_i is added to the pool of classifiers C. Once all the decision trees have been trained and added to the pool of classifiers, C, the algorithm returns the trained Random Forests model C. During the prediction phase, the model aggregates the predictions of all the decision trees in the pool using the majority voting for prediction.

Algorithm 4.3 Random Forests algorithm trains multiple decision trees. Each tree learns from a bootstrapped sample

Input: Training set T_r, Number of trees M, Number of features to consider at each split N_f

Output: Random Forests model

$C = \emptyset$ pool of classifiers, C, is empty

for $i = 1$ *to* M **do**

  $T_{r(i)}$ = bootstrap sample from T_r

  randomly select features at each split

  C_i = decision tree trained on $T_{r(i)}$

  $C = C \cup C_i$ add C_i to C

end for

Return the trained Random Forests model, C

4.2 Local pool generation techniques

The aim of local pool generation schemes is to develop local expertise because committee members are expected to be more confident in classifying query samples related to their expertise, which is the classification of the query sample in the area close to that. The local region can be defined by different methods, such as using the K-Nearest Neighbors technique and clustering, to find the neighborhood of a query sample. Two algorithms for generating a pool of classifiers, C, using local information are discussed below.

4.2.1 Forest of Local Trees

A novel algorithm for ensemble classifiers named Forest of Local Trees (FLT) was proposed by Armano et al. (Armano & Tamponi (2018)), in which classifiers are trained with a focus on different regions of the feature space. In other words, it aims to create an ensemble of classifiers that leverage specialized local knowledge. Instead of relying on a single expert, FLT draws

inspiration from the concept of a local committee where each member specializes in a subset of the feature space, representing a distinct local region of expertise. This approach ensures that committee members are more confident in answering questions related to their respective areas of expertise (i.e., local region).

In this ensemble, Random Decision Trees (RDT) are employed, similar to the Random Forest (RF) technique. Within the ensemble, each RDT is built using a bootstrap sample, which is essentially a random selection of data points with replacement from the original dataset. At each node during the tree-building process, a random subset of features is considered when making a split decision. This randomness ensures that different trees within the ensemble do not evaluate the same features at every node, leading to a diverse set of decision trees. The criterion used for making these split decisions is the Gini impurity.

Each RDT is associated with a centroid, denoted as, μ. A centroid is the center point or representative point of a cluster in clustering algorithms. It represents the average or central location of data points within the cluster, aiding in cluster identification and characterization (Lee & Antonsson (2000)). The idea behind the centroid mechanism is to distribute the "expertise" of the forest across the entire dataset. To select centroids, μ, initially, each sample in the dataset has an equal chance of becoming a centroid, μ. The formula 4.1 shows the same probability of being picked.

$$P_0^{(\mu)} = \frac{1}{|D|} \tag{4.1}$$

However, once a new centroid, μ, is chosen, the probability of selecting the remaining samples is updated. This adjustment increases the likelihood of selecting samples that are far from the current centroid, μ. The picking distribution is updated after each selection, as illustrated in Figure 4.3 (Armano & Tamponi (2018)). The update for each sample is calculated using the equation 4.2.

$$\mathrm{Q}_{t+1}^{(\mu)}(x) = \mathrm{p}_{t}^{(\mu)}(x) \cdot \log\left(1 + \|x - \mu_t\|\right) \tag{4.2}$$

A normalization step is applied using the formula 4.3 to ensure that the probabilities meet the requirements of being valid probabilities (sum to 1).

$$\mathrm{P}_{t+1}^{(\mu)}(x) = \frac{\mathrm{Q}_{t+1}^{(\mu)}(x)}{\sum_{|i=1|}^{D} \mathrm{Q}_{t+1}^{(\mu)}(x_i)} \tag{4.3}$$

Following that, a random decision tree is generated focusing on a centroid, μ, for training. During the training phase, the importance of a training sample $\mathbf{x}_j$ which belongs to a part of the local region, is established according to its distance from the centroid, μ. The closer the sample, the greater its importance. It is noteworthy that random decision trees see the whole dataset; however, the weights assigned to the samples differ in such a way that samples closer to the centroid have greater importance in the training phase.

To place a particular attribute appropriately in the decision tree, Gini Index is used for attribute selection. The Gini impurity index (g) is defined by the equation 4.4.

$$g(T_c) = \sum_{y \in Y} \hat{P}_c(y) \cdot (1 - \hat{P}_c(y)) \tag{4.4}$$

The expression $\hat{P}_c(y)$ represents the ratio of samples within dataset T_c that are labeled as y to the total number of samples in dataset T_c. In other words, it calculates the proportion of samples in T_c that belong to a particular class y.

In the proposed modification, the Gini index (g) is updated by including sample weights in the calculation. The modified $\hat{P}(w)_c(y)$ is given by the equation 4.5. Here, w_i represents the weight associated with each sample.

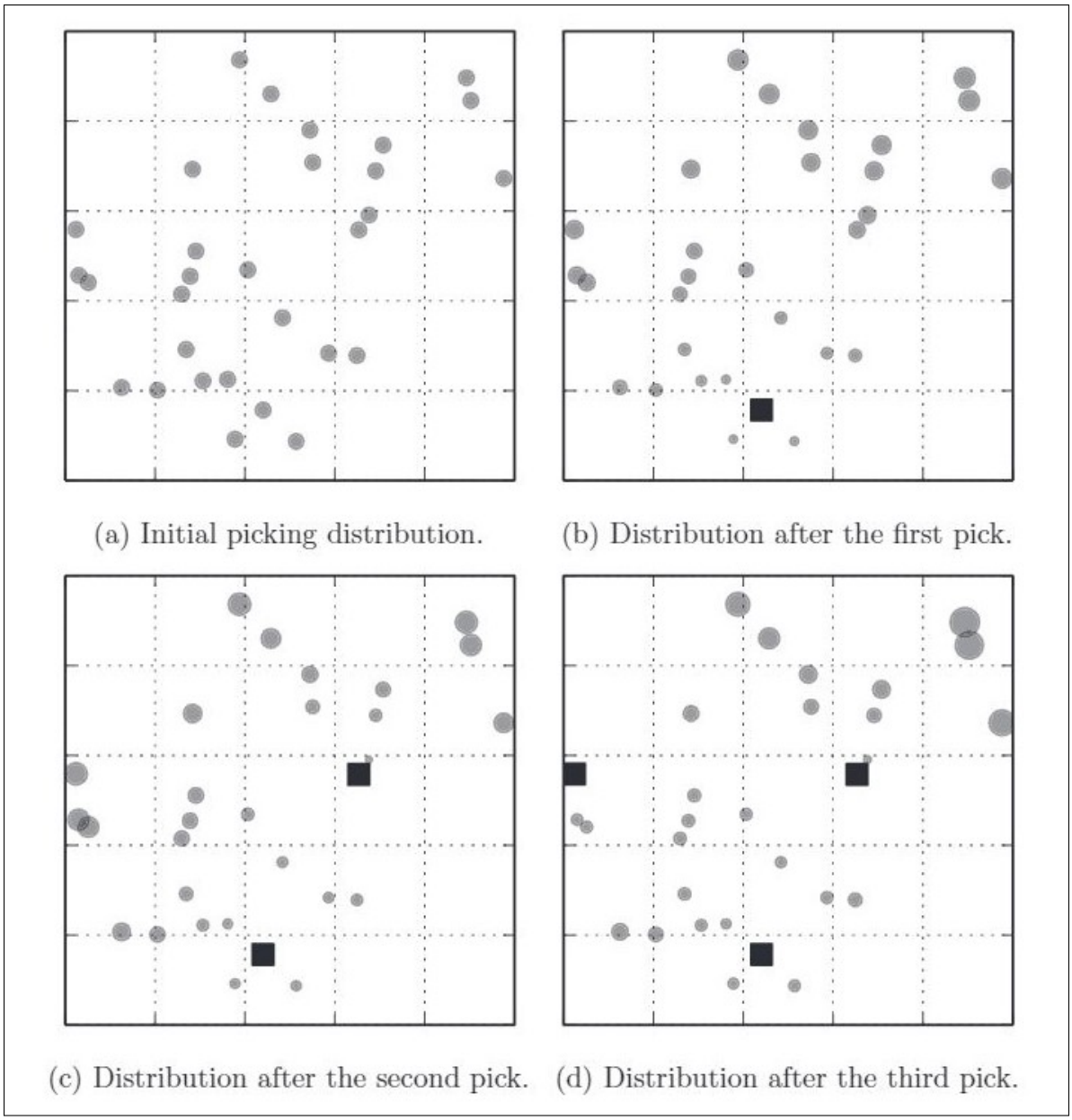


Figure 4.3 Each time a centroid is picked, the picking distribution is updated. The bigger the circle, the higher the probability of a sample being selected. Dataset samples are represented by gray circles, whose radius is proportional to the picking probability, while black boxes show centroids adapted from Giuliano Armano (2018, pp. 380-390)

$$\hat{P}(w)_c(y) = \frac{\sum_{i=1}^{|T^{(w)}|} w_i \cdot I(y_i = y)}{\sum_{i=1}^{|T^{(w)}|} w_i} \tag{4.5}$$

Regarding classification, an RDT's competence over a given instance depends on the same weighting schema, which gives more importance to instances close to its centroid, μ (Armano & Tamponi (2018)). Figure 4.4 illustrates weighting distributions. Each RDT in the FLT receives, in fact, a different weight distribution, which gives samples decreasing importance with their distance from the centroid, μ, over which the RDT has been trained. Dataset samples are represented by gray circles, whose radius is proportional to the corresponding weight, while centroids are shown by black boxes (Armano & Tamponi (2018)). The FLT has been demonstrated to be effective in a wide range of standard classification domains.

The FLT algorithm, outlined in algorithm 4.4, takes as input a training dataset, T_r, a learning algorithm, $\mathcal{A}$, and the number of iterations, M. The algorithm's output is a pool of classifiers, C, composed of M base classifiers. At the beginning of the algorithm, the pool of classifiers, C, is empty. Then, a centroid is randomly chosen from the set of possible centroids for each iteration. The centroid is used to create a random decision tree (RDT). The picking probabilities for the centroids are updated based on their distance from the newly added centroid. This helps ensure that the centroids are well spread across the sample space. The RDT is then trained on the training dataset, and the resulting classifier is added to the pool of classifiers, C. This process is repeated for the desired number of iterations, resulting in a pool of classifiers.

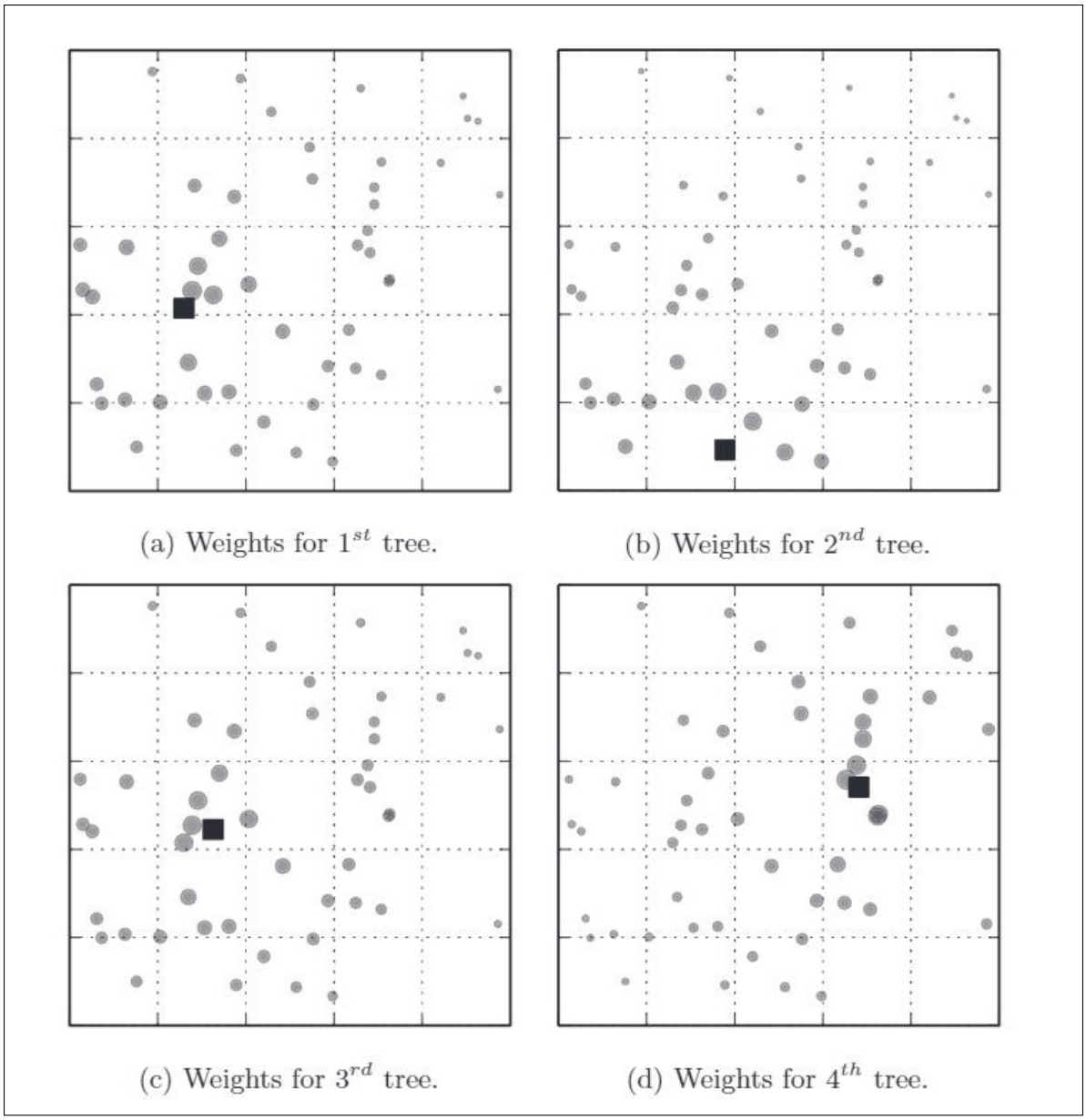


Figure 4.4 These are examples of weighting distributions. The FLT uses different weight distributions for each RDT, resulting in decreasing importance of samples as they move away from the centroid. Black boxes denote centroids, while gray circles denote samples, with the radius proportional to the weight. To ensure good coverage of the sample space, centroids should be chosen as far apart as possible adapted from Giuliano Armano (2018, pp. 380-390)

Algorithm 4.4 Forest of Local Trees (FLT) training scheme

Input: training dataset, T_r, a Learning algorithm $\mathcal{A}$, iterations M
Output: A pool of classifiers, C, composed of, M, base classifiers

$C = \emptyset$ pool of classifiers, C, is empty
for *i = 1 to M (random decision trees)* **do**
    ▷ pick the first centroid, μ, using the same probability of being picked: $P_0^{(\mu)} = \frac{1}{|T|}$
        ▷ update picking probabilities by using: $Q_{t+1}^{(\mu)}(x) = p_t^{(\mu)}(x) \cdot \log(1 + \|x - \mu_t\|)$
            ▷ Normalization of the picking probabilities for centroid selection by using: $P_{t+1}^{(\mu)}(x) = \frac{Q_{t+1}^{(\mu)}(x)}{\sum_{i=1}^{T} Q_{t+1}^{(\mu)}(x_i)}$
                ▷ Using the Gini Index for attribute selection for decision tree; $g(T_c) = \sum_{y \in Y} \hat{P}_c(y) \cdot (1 - \hat{P}_c(y))$
                ▷ Implement a modification to the Gini index by including sample weights; $\hat{P}(w)_c(y) = \frac{\sum_{i=1}^{|T^{(w)}|} w_i \cdot I(y_i = y)}{\sum_{i=1}^{|T^{(w)}|} w_i}$
    $D = \mathrm{RDT}(\mu)$ ▷ Get μ as an input and create a new Random decision Tree
    C = train the RDT
    $C = C \cup C_i$ add C_i to C
end for
Return the trained pool of classifiers C ;

4.2.2 Ensembles of Locally Independent Prediction Models

Ross *et al.* (2020) presented a novel approach to constructing ensembles using explicit constraints to ensure the generated models are diverse locally and make independent predictions. The proposed model is based on the concept of Locally Independent Training (LIT) designed to train an ensemble of models by forcing "locally independent" decision boundaries among the generated classifiers. This local independence refers to the model's ability to make different decisions based on particular small regions of the data manifold. In other words, for any small neighborhood around a data point, the ensemble members are encouraged to have independent error gradients.

This is achieved through the diversity regularization term in the LIT loss function, which penalizes similarity in the gradients of the loss with respect to the inputs across different

Algorithm 4.5 Locally Independent Training (LIT) training scheme

Input: Training dataset T_r, learning rate η, diversity hyperparameter λ, number of classifiers M, number of training iterations I
Output: A pool of classifiers, C, composed of, M, base classifiers

Initialize parameters $\{\theta_m\}_{m=1}^{M}$ for all linear classifiers in the ensemble
for $i = 1$ *to* I **do**
  for *each* $\mathbf{x} \in \mathcal{D}$ **do**
    for $m = 1$ *to* M **do**
      C
    end for
    ompute prediction loss $\mathcal{L}_{pred}(c_m, (\mathbf{x}, y))$
    Compute gradient $\nabla_{\theta_m}\mathcal{L}_{pred}(c_m, (\mathbf{x}, y))$
    for $m = 1$ *to* $M - 1$ **do**
      for $n = m + 1$ *to* M **do**
        Compute gradient cosine similarity:
        $cos_{mn} = \frac{\nabla_x \mathcal{L}_{pred}(c_m,\mathbf{x}) \cdot \nabla_x \mathcal{L}_{pred}(c_n,\mathbf{x})}{\|\nabla_x \mathcal{L}_{pred}(c_m,\mathbf{x})\| \|\nabla_x \mathcal{L}_{pred}(c_n,\mathbf{x})\|}$
        Compute diversity loss $\mathcal{L}_{div}(c_m, c_n, \mathbf{x}) = cos_{mn}^2$
      end for
    end for
    Compute total loss for the ensemble:
    $\mathcal{L}_{LIT} = \sum_{m=1}^{M} \mathcal{L}_{pred}(c_m, (\mathbf{x}, y)) + \lambda \sum_{m=1}^{M-1} \sum_{n=m+1}^{M} \mathcal{L}_{div}(c_m, c_n, \mathbf{x})$
    Update parameters $\{\theta_m\}_{m=1}^{M}$ using gradient descent:
    $\theta_m = \theta_m - \eta \nabla_{\theta_m} \mathcal{L}_{LIT}, \forall m \in \{1, ..., M\}$
  end for
end for
Return C

models. By doing so, each model in the ensemble is pushed to learn different aspects of the data neighborhood rather than all models learning the same patterns. Thus, avoiding generating redundant local classifiers. According to Ross *et al.* (2020), the LIT method's emphasis on local independence is particularly beneficial in complex decision spaces where different regions of the input space may exhibit different characteristics, such as in problems with complex and non-linear decision borders.

Algorithm 4.5 formalizes the main steps of the LIT training. The algorithm begins by initializing the parameters $\{\theta_m\}$ for each classifier in the ensemble. This step sets up the starting point for

the optimization process. The parameters can be initialized randomly or by using any weight pre-training scheme. Then, the training proceeds iteratively over T learning iterations, each consisting of the following steps:

Prediction loss computation. For each data point $(\mathbf{x}, y)$ in the training dataset T_r, the prediction loss $\mathcal{L}_{pred}$ is computed for each classifier c_m. In this work, the traditional cross-entropy loss was considered (Equation 4.6.

$$\mathcal{L}_{pred}(c_m, (\mathbf{x}, y)) = -\log p(y|c_m(\mathbf{x}; \theta_m)) \tag{4.6}$$

where $p(y|c_m(\mathbf{x}; \theta_m))$ is the predicted probability of the true class y given the input $\mathbf{x}$ and the classifier parameters θ_m. Then, the gradient of the prediction loss with respect to the classifier parameters, $\nabla_{\theta_m}\mathcal{L}_{pred}(c_m, (\mathbf{x}, y))$, is computed for each classifier. This gradient is essential for updating the model parameters in the direction that minimizes the loss.

Diversity Loss Computation. The diversity loss $\mathcal{L}_{div}$ is computed between all pairs of classifiers in the ensemble and aims to force independent errors among the ensemble members. According to Ross *et al.* (2020), given two classifiers c_m and c_n, this can be achieve by calculating the cosine similarity of the gradients of the prediction loss with respect to the inputs for each pair of classifiers c_m and c_n (Equation 4.7):

$$cos_{mn} = \frac{\nabla_{\mathbf{x}}\mathcal{L}_{pred}(c_m, \mathbf{x}) \cdot \nabla_{\mathbf{x}}\mathcal{L}_{pred}(c_n, \mathbf{x})}{\|\nabla_{\mathbf{x}}\mathcal{L}_{pred}(c_m, \mathbf{x})\|\|\nabla_{\mathbf{x}}\mathcal{L}_{pred}(c_n, \mathbf{x})\|} \tag{4.7}$$

The diversity loss for each pair is then given by $\mathcal{L}_div = cos^2_{mn}$, which penalizes the similarity in the decision boundaries of the classifiers. Therefore, promoting local independence.

The total loss for the LIT ensemble learning is then computed by summing the prediction loss for each classifier and the weighted diversity loss across all pairs of classifiers, with λ acting as the regularization hyperparameter (Equation 4.8):

$$\mathcal{L}_{LIT} = \sum_{m=1}^{M} \mathcal{L}_{pred}(c_m, (\mathbf{x}, y)) + \lambda \sum_{m=1}^{M-1} \sum_{n=m+1}^{M} \mathcal{L}_{div}(c_m, c_n, \mathbf{x}) \tag{4.8}$$

Finally, the weights of all models in the system are updated with stochastic gradient descent. After T iterations, the trained pool of classifiers C is returned. Following the guidelines by Ross *et al.* (2020), we setup this technique using $\lambda = 10^{-2.3}$ and considered a total of $I = 1000$ training iterations.

The LIT method's emphasis on local performance and independence aligns well with the objectives of dynamic ensemble selection techniques that is, to exploit local information and experts. By providing a set of diverse and locally competent classifiers, LIT is expected to provide pool of candidates from which DES methods can work better, thereby potentially improving the performance of these models.

4.2.3 Summary of pool generation schemes

Bagging is an ensemble learning method that trains multiple base learners independently on different subsets of the training data. This method helps reduce variance in predictions and assigns equal weight to each base learner's output. Boosting, on the other hand, iteratively improves weak models to create stronger ones. It assigns weights to training instances based on the accuracy of previous classifiers, focusing on misclassified samples. Random Forests constructs an ensemble of diverse trees by randomly selecting subsets of data and features for each tree.

As previously highlighted in the introduction, global pool generation strategies exhibit certain limitations, notably concerning instability and the propensity to generate redundant classifiers. These constraints imply that global pool generation schemes may not be the only option as a pool generation scheme for DS methods, particularly across diverse datasets. Consequently, the following sections will explore the explanation of local pool generation schemes.

As mentioned in this chapter, the Forest of Local Trees and the Ensembles of Locally Independent Prediction Models have local perspectives on the problems. Pools of classifiers, with a local perspective have never been explored by the DS literature. We hypothesize that as those methods are based on local information, they are more suitable as pool generation schemes for DS algorithms. Therefore, in the next section, the performance of the aforementioned pool generation schemes is evaluated and compared with the performance of global perspective ensembles for dynamic selection methods.

4.3 Comparative study

In this section, we initiate a comparative study, building upon the datasets and methodology established in chapter 3. In the context of our study, the performance of various algorithms across numerous datasets is evaluated, assessing both local and global pool generation schemes for the DS algorithm, ultimately providing answers to the research questions.

Additionally, we utilize the Friedman test (Friedman (1937)) in this comparative study for robustly evaluating and comparing multiple groups. The use of the Friedman test, well-shown in the literature (Cruz *et al.* (2018a)) for its ability, aligns perfectly with our comparative analysis objectives as it effectively compares multiple algorithms across multiple datasets.

4.3.1 Comparison of a local and global pool generation schemes

The aim of this section of the comparative study is to answer the 4 first questions among 5 research questions. The following section outlines the research questions under investigation:

- RQ1: Do local pool generation schemes achieve better results when used for dynamic selection algorithms?
- RQ2: Does selecting the best pool of classifiers, C, depend on what dynamic selection is used ?
- RQ3: Does selecting the best DS method depend on what pool of classifiers, C, is used?
- RQ4: Is there a correlation between static ensemble and dynamic selection results?

In this section, two distinct analyses are carried out. The first analysis is a comprehensive examination, that encompasses all methods and evaluates the average ranking of each pool within every DS technique. This comprehensive approach provides a holistic perspective, allowing us to compare all conceivable combinations of pools and DS algorithms. In contrast, the second analysis focuses on a fixed DS method and assesses the performance of 7 pools of classifiers, highlighting the ones that attain the lowest rank and are, therefore, better suited for the specific DS technique at hand.

In Friedman rank tests (Friedman (1937)), the first rank is assigned to the best performance, the second best to the next rank, and up to the last rank, respectively. If there was a tie, for example, when two approaches obtain the same classification performance, their average scores are summed up and divided by two. The average rank is then obtained, considering all datasets. The algorithm with the lowest average rank is considered the best-performing one. The Friedman test offers a significance level denoted by α. A significance level of $\alpha = 0.05$ indicates that there is a 5 percent risk of concluding that a difference exists when there is no actual difference. ρ-value being smaller than α means the differences between some of the medians are statistically significant (Hollander, Wolfe & Chicken (2013)).

The test was performed considering these hypotheses:

- H_0: There is no significant difference between the results obtained using different techniques.
- H_1: There is a significant difference between the results obtained using different techniques.

Friedman's test is used here to determine if any pool of classifiers performs significantly better in a fixed DS method. In this research for each DS technique, the Friedman test was applied to determine if there is a pool of classifiers that perform better than others. In KNORA-E, the ρ-value is 0[1] and since this ρ-value is lower than 0.05, there is a significant difference between these 7 pools of classifiers including Bagging(DT), bagging(P), Boosting(DT), Boosting(P), Random Forests, FLT, and LIT. The ρ-value up to 4 decimal places for other DS methods, including KNORA-U, META-DES, DES-MI, DES-P, OLA, and MLA are 0 as well. Bonferonni–Dunn

[1] Value extremely low which returned 0 due to the machine precision.

post-hoc test was used to calculate the Critical Difference (CD) recommended in (Demšar (2006)).

Table II-1 provides the average ranking across 288 datasets for various combinations of 7 pool generation schemes and 7 dynamic selection (DS) methods. Each cell in the table represents the average rank of a specific pool-DS method combination. For instance, FLT achieved average ranks of 26.96 (KNORA-E), 21.40 (META-DES), 21.43 (KNORA-U), 27.54 (DES-MI), 22.89 (DES-P), 31.69 (MLA), and 33.82 (OLA). These statistics for LIT are 28.19(KNORA-E), 21.84(META-DES), 21.73(KNORA-U), 28.76(DES-MI), 22.68(DES-P), 31.84(MLA), 33.35(OLA). The best average ranking belongs to (META-DES, RF) with an average rank of 6.12, while the worst belongs to (DES-MI, BSDT) with an average rank of 47.23.

Table 4.2 Summary of the average rank among 288 datasets between a combination of 7 pools and 7 DS methods

(META-DES, RF)	6.12	(KNORA-E, FLT)	26.96
(KNORA-U, RF)	8.67	(DESP, BSDT)	27.18
(DES-P, RF)	9.50	(MLA, BSDT)	27.43
(META-DES, BDT)	9.56	(DES-MI, FLT)	27.54
(DES-MI, RF)	10.12	(MLA, BP)	27.96
(META-DES, BSDT)	11.53	(KNORA-E, LIT)	28.19
(KNORA-U, BDT)	12.55	(MLA, BSP)	28.32
(DES-P, BDT)	13.07	(DES-MI, LIT)	28.76
(META-DES, BP)	14.64	(KNORA-U, BSP)	30.82
(KNORA-E, RF)	14.92	(OLA, BDT)	30.89
(DES-MI, BDT)	16.08	(MLA, FLT)	31.69
(KNORA-E, BDT)	17.58	(DES-P, BSP)	31.74
(DES-MI, BP)	19.07	(MLA, LIT)	31.84
(KNORA-U, BP)	19.70	(OLA, BSP)	32.02
(DES-P, BP)	19.77	(OLA, BSDT)	32.08
(KNORA-E, BP)	20.32	(MLA, BDT)	32.70
(KNORA-U, BSDT)	20.45	(KNORA-E, BSDT)	33.08
(META-DES, FLT)	21.40	(OLA, LIT)	33.35
(KNORA-U, FLT)	21.43	(OLA, FLT)	33.82
(KNORA-U, LIT)	21.73	(KNORA-E, BSP)	35.57
(META-DES, LIT)	21.84	(OLA, RF)	36.70
(DES-P, LIT)	22.68	(MLA, RF)	38.04
(DESP, FLT)	22.89	(DES-MI, BSP)	46.86
(OLA, BP)	23.61	(DES-MI, BSDT)	47.23
(META-DES, BSP)	26.01		

Figures 4.5 to 4.11 illustrate the CD diagram with the results of the Bonferonni–Dunn post-hoc test. Techniques in which the difference in average ranks is lower than the critical difference are connected by a black bar. This means Techniques are statistically equivalent results according to the ranking analysis. Based on this analysis, it is evident that RF performs better for DES techniques, while BP demonstrates superior performance for DCS techniques. Another point that can be observed for DES methods, including KNORA-E, KNORA-U, META-DES, DES-P, and DES-MI the first 2 ranks belong to RF and BDT, respectively. It is worth noting that in all DS methods, LIT and FLT are not among the first 3 ranks. RF was placed in last place for DCS methods. The LIT and FLT methods both fall within the same range of 4 to 6 as a pool of classifiers for DS methods. in order to answer RQ1, this analysis shows that local pool generation schemes struggle to deliver improved results when applied to dynamic selection algorithms. This statement corresponds to the answer to the initial research question.

Although FLT and LIT were not among the top 3, the benefits of using them cannot be ignored since there are some datasets that these pool generation schemes offer the best performance in the first place. So, using both local and global perspectives pool generation schemes would help us improve DS methods' performance.

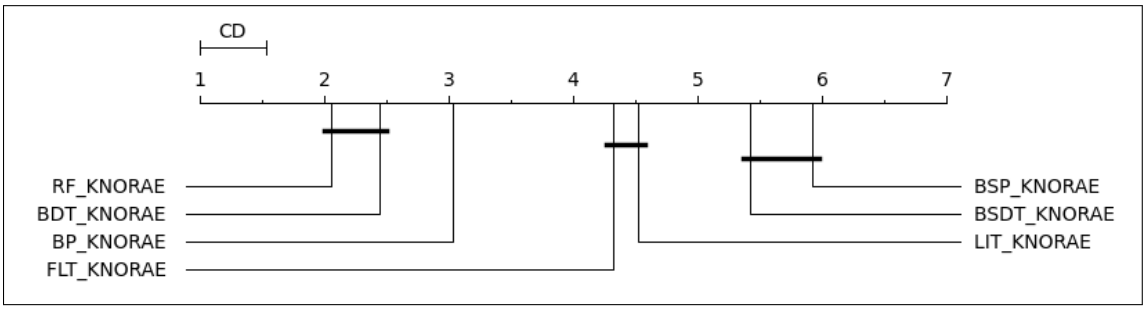


Figure 4.5 The average rank of 7 pools of classifiers for KNORA-E among 288 datasets

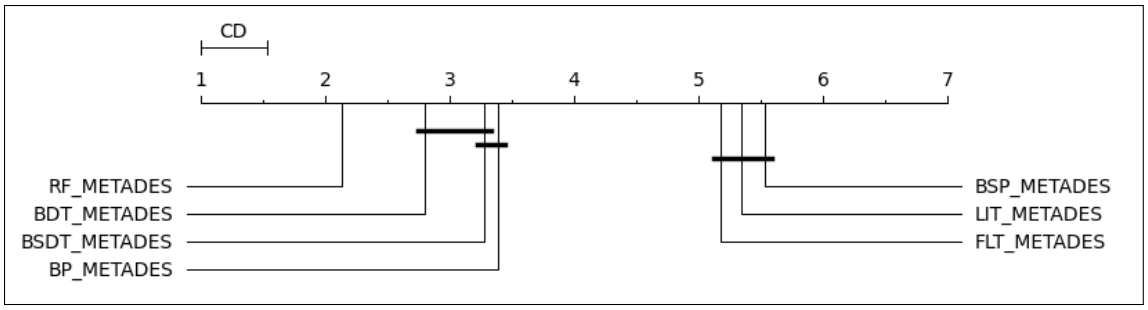


Figure 4.6 The average rank of 7 pools of classifiers for META-DES among 288 datasets

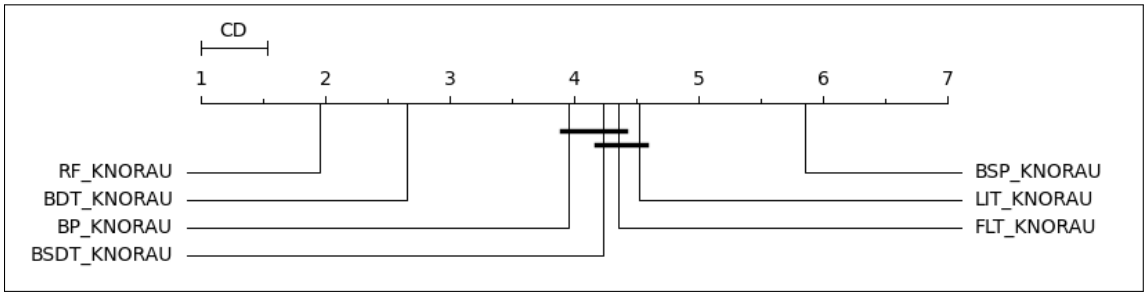


Figure 4.7 The average rank of 7 pools of classifiers for KNORA-U among 288 datasets

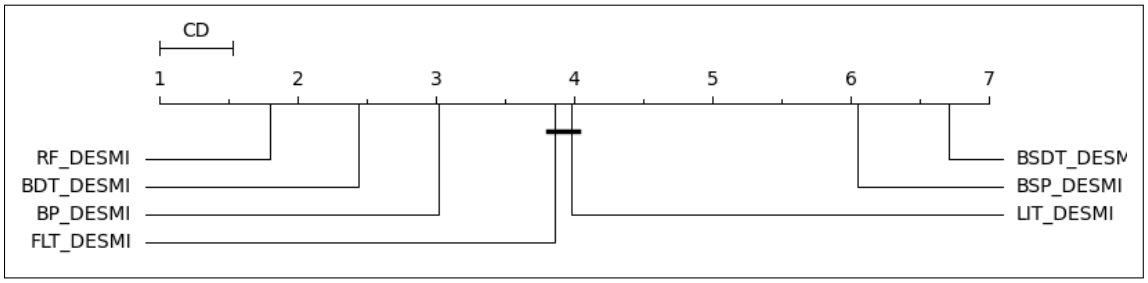


Figure 4.8 The average rank of 7 pools of classifiers for DES-MI among 288 datasets

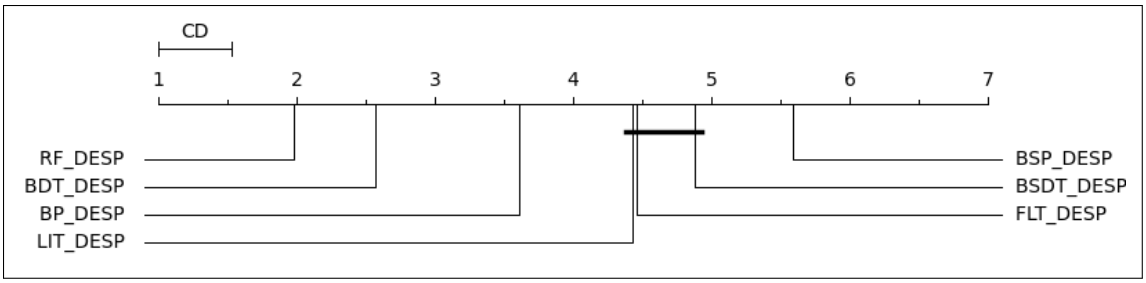


Figure 4.9 The average rank of 7 pools of classifiers for DES-P among 288 datasets

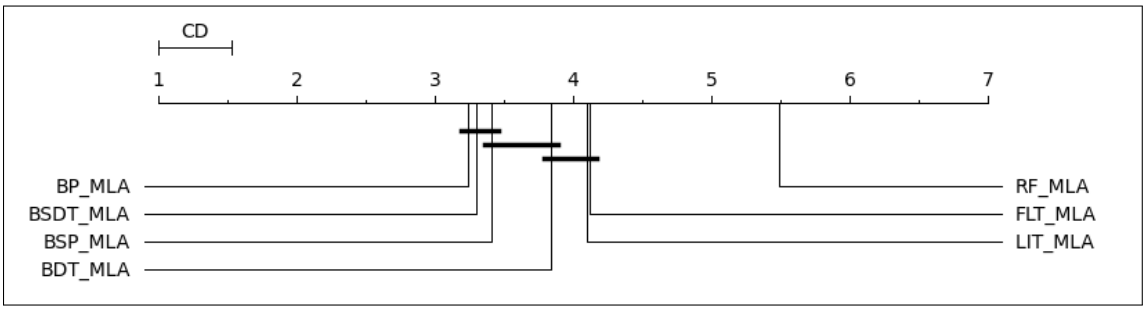


Figure 4.10 The average rank of 7 pools of classifiers for MLA among 288 datasets

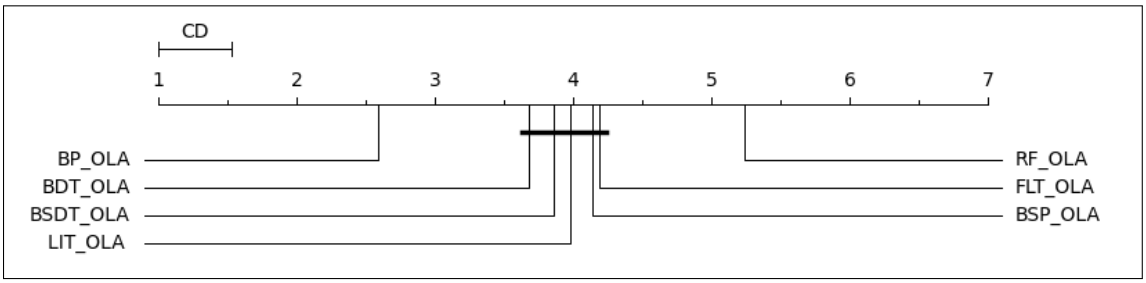


Figure 4.11 The average rank of 7 pools of classifiers for OLA among 288 datasets

4.3.2 Does selecting the best pool of classifiers, C, depend on what dynamic selection is used (RQ2)?

In this section, in order to answer RQ2, we investigate the relationship between the selection of a pool of classifiers when a fixed DS method is available. Figures 4.5 to 4.11 illustrate that all five DES methods, including KNORA-E, META-DES, KNORA-U, DES-MI, and DES-P

performed better when they used first RF and then BDT rather than other pool of classifiers. The two DCS methods, including MLA and OLA performed better when they used BP as a pool of classifiers rather than others. As demonstrated in the figures, to answer RQ2, the selection of the optimal pool of classifiers depends on the specific DS method used. Selecting the best pool of classifiers relies on the specific DS method, which is determined by the best average rank of each pool provided in the figures.

4.3.3 Does selecting the best DS method depend on what pool of classifiers, C, is used (RQ3)?

In this section, we delve into a thorough investigation of the relationship between the selection of a DS method and the choice of a pool generation scheme. This investigation is motivated by a fundamental question: When a specific pool of classifiers is desired for various purposes, such as research or the comparative evaluation of Majority Voting and DS algorithms, how do we determine which DS method is the best option by using a fixed pool of classifiers? An analysis within the scope of this comparative study enables us to address RQ3.

Figures 4.12 to 4.18 demonstrate the CD diagram with the results of the Bonferonni–Dunn post-hoc test. These diagrams compare the performance of different DS algorithms while a pool of classifiers is fixed across 288 datasets. As a result of this classification, META-DES, and KNORA-U are the first and second-best suitable DS methods when we have a fixed pool of classifiers, C, including pools of classifiers that are generated by using local and global pool generation schemes. DES-P placed in third place in 6 of 7 fixed pool generation schemes. DES-P consistently ranked third, among 7 DS methods in all pool generation schemes instead of BP. Instead of Boosting, In the other 5 pool generation schemes, MLA and OLA that ate both DCS placed in the last two places, showing the power of DES algorithms.

Based on this analysis, with the aim of addressing RQ3, it appears that the selection of the best pool of classifiers depends on the choice of the dynamic selection technique. Figures 4.12 to 4.18 provide a ranking for selecting among 7 DS methods when a fixed pool of classifiers is available.

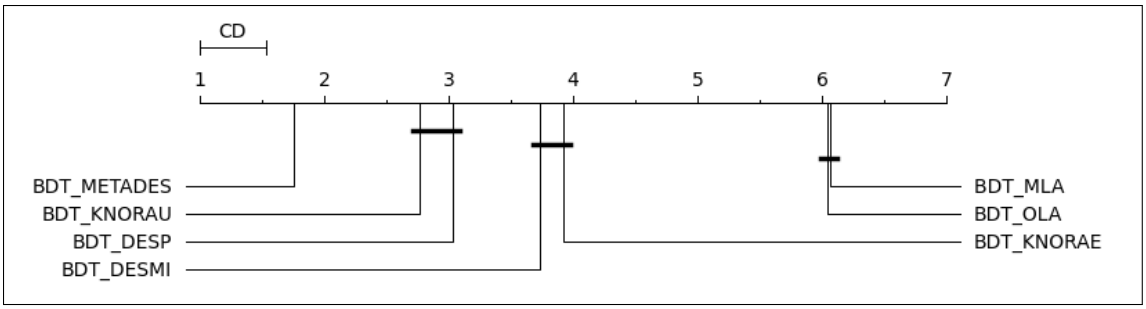


Figure 4.12 The average rank of 7 DS methods using BDT as a pool of classifiers among 288 datasets

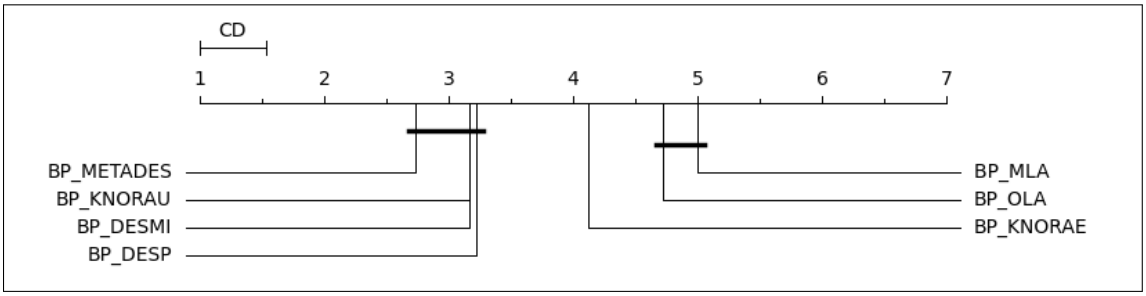


Figure 4.13 The average rank of 7 DS methods using BP as a pool of classifiers among 288 datasets

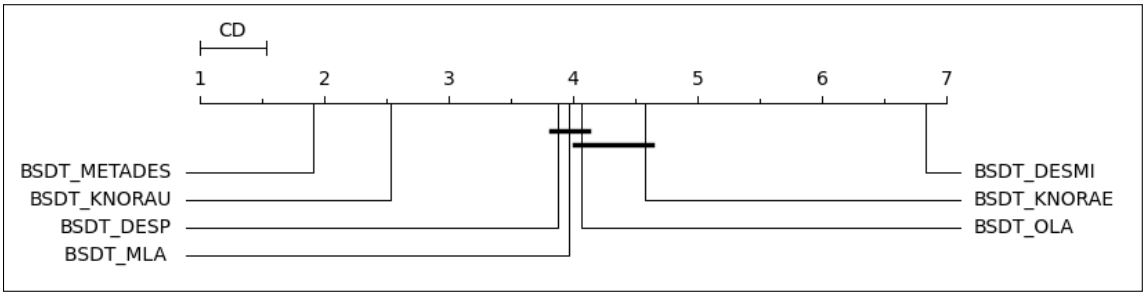


Figure 4.14 The average rank of 7 DS methods using BSDT as a pool of classifiers among 288 datasets

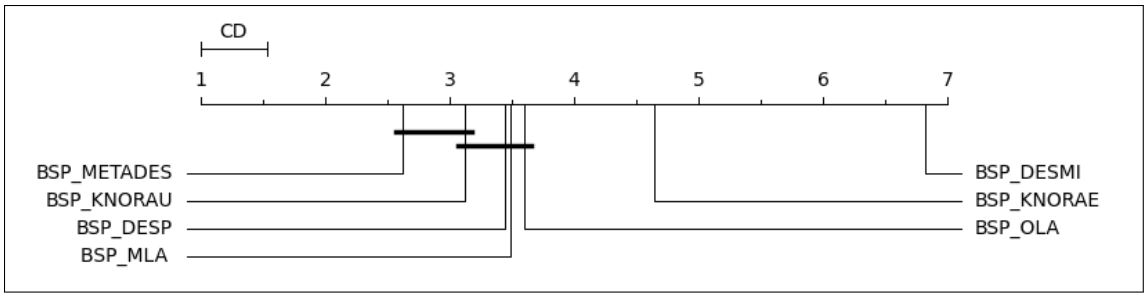


Figure 4.15 The average rank of 7 DS methods using BSP as a pool of classifiers among 288 datasets

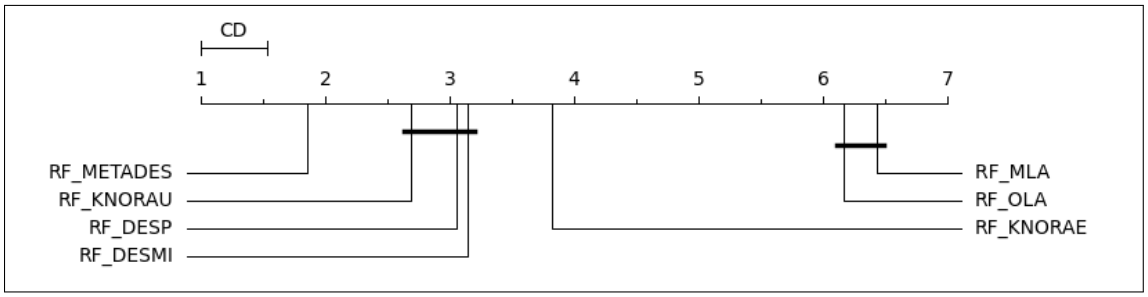


Figure 4.16 The average rank of 7 DS methods using RF as a pool of classifiers among 288 datasets

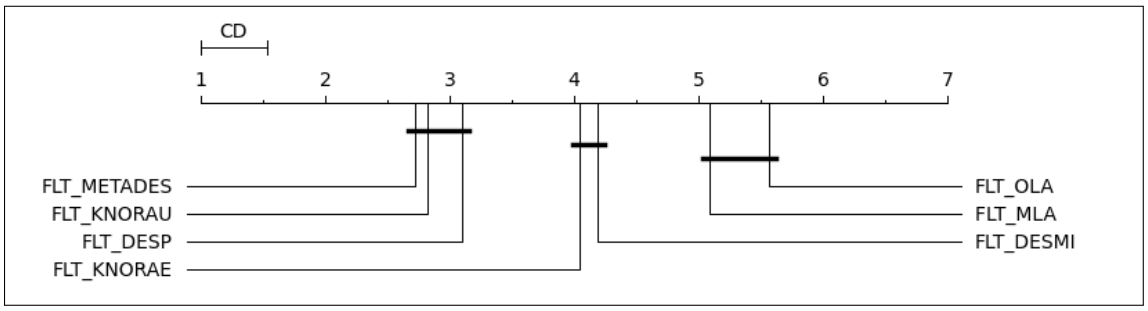


Figure 4.17 The average rank of 7 DS methods using FLT as a pool of classifiers among 288 datasets

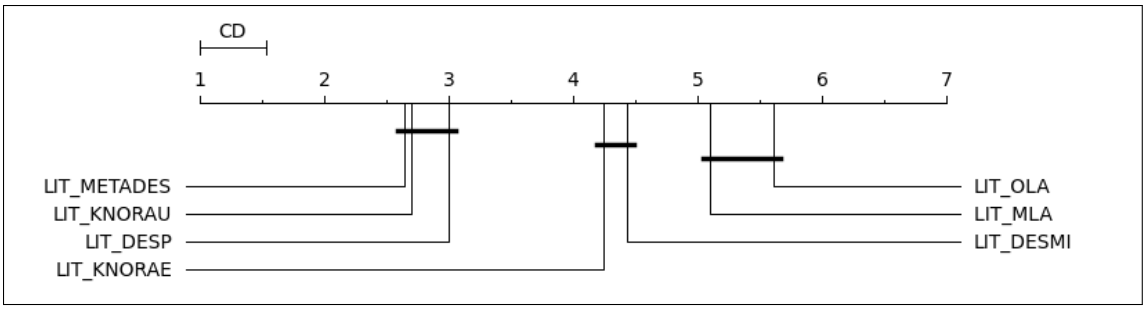


Figure 4.18 The average rank of 7 DS methods using LIT as a pool of classifiers among 288 datasets

4.3.4 Relationship between static ensemble results and dynamic selection results (RQ4)

The aim of this section is to delve into the connection between static selections and DS methods in the pursuit of superior performance outcomes. More specifically, we address the fourth research question (RQ4): If a particular dataset is optimally classified by one of the seven ensemble methods in a static combination, will that same ensemble method excel when its results form the basis for a majority of DS methods? The significance of RQ4 stems from the potential implications for optimization processes. If a strong correlation exists between top-performing pools of classifiers in both static and dynamic settings, the performance of the static combination could serve as a preliminary indicator. This would facilitate the optimization of classifier pools without the need for intensive computational resources typically demanded by DS methods during the optimization phase. The relationship between SS and DS represents to what extent the performance of the DS methods while using a specific pool follows the performance of SS on a given data. It represents the frequency that the best pool generation scheme obtain the best classification performance in both SS and DS settings.

To illustrate how this analysis is conducted, let us consider a practical example. Take Dataset 1, where the BSDT emerges as the top-performing ensemble out of the seven under consideration. The question now becomes: does using BSDT as the classifier pool for the OLA method in Dataset 1 enhance its performance? To answer this, we compared the performance of OLA

across seven different pool generation methods. Interestingly, the results indicated that BSP, when combined with OLA for Dataset 1, surpassed the others. This observation hints that a straightforward relationship between the performance of SS and DS methods might not always hold. Such findings shed light on the intricate relationship between SS and DS techniques and raise the question of whether the best ensemble in a static setting retains its superiority in a dynamic one.

We meticulously evaluated seven diverse ensembles, specifically BDT, BP, BSDT, BSP, RF, FLT, and LIT, to explore this relationship. Our aim was to identify the top-performing ensemble among these seven for each of the 288 datasets. This process allowed us to ascertain the ensemble that has the best performance across all datasets. Following this, we assessed the performance of each DS method under various pool generation schemes. By doing so, we identified the ensemble of classifiers that delivered the best performance when used as a pool of classifiers for each DS method across the datasets. Now, for every dataset, we have two distinct ensembles. The first ensemble showcases the optimal SS performance, while the second ensemble represents the ensemble that yields the best performance when employed as a pool of classifiers for DS methods. The core of our analysis for this section centers on determining the similarity between these two ensembles. If the number of similarities among all datasets is high then it indicates a notable relationship between SS and DS methods. In essence, this relationship signifies that across the 288 datasets, we can express, in percentage terms, how the ensemble that performs better than others in static selection for a given dataset also leads to the best performance of a DS method while it is used as a pool of classifiers.

Table 4.3 provides a concise summary of the analysis conducted between majority voting on the ensemble and various dynamic selection (DS) methods across 288 datasets. Among the DS methods assessed, DES-P, KNORA-U, and META-DES exhibited the strongest relationship with static ensemble results, scoring percentages of 55.90, 54.17, and 48.96, respectively. Conversely, the performance of DCS methods, MLA and OLA, showed comparatively lower agreement with static ensemble results.

Based on the analysis conducted in this section, the results revealed varying degrees of relationship between SS and DS methods, with DES-P, KNORA-U, and META-DES exhibiting the strongest matches. This shows that, for these specific DS methods, when an ensemble is optimal in SS for a given dataset, it tends to perform well based on the percentage provided between 288 datasets in table 4.3 when used as a pool of classifiers in DS applications. However, this relationship is not universal, emphasizing the importance of adopting a case-specific approach when selecting ensemble methods for DS tasks. The analysis highlights that superior performance in SS does not guarantee the same level of performance in DS across all scenarios.

Table 4.3 The correlation between static ensemble results and DS algorithms

DS method	Correlation
KNORA-E	41.67
META-DES	48.96
KNORA-U	54.17
DESMI	47.92
DESP	55.90
MLA	26.74
OLA	22.22

4.3.5 Conclusion

The aim of this chapter was to investigate various pool generation schemes, including both global and local perspectives, as inputs for DS methods. In the course of this empirical study,

we introduced and compared seven distinct pools of classifiers for DS techniques to address the research questions.

In summary, our study yielded a set of key findings. The analysis illustrates that the selection of the optimal pool of classifiers is dependent on the specific DS method used. Different DS methods may have different preferences for pool generation schemes. Furthermore, for a given DS method, we offer an assessment of the performance of seven pools of classifiers, pinpointing the top-performing ones tailored to that particular DS technique.

In addition, this research focused on exploring the relationship between the choice of a dynamic selection (DS) method and the selection of a pool generation scheme. It aimed to answer the question of which DS method works best with a fixed pool of classifiers. Based on the analysis, META-DES and KNORA-U emerge as the top two DS methods that harmonize well with a fixed pool of classifiers, including those generated by both local and global pool generation schemes.

Moreover, another research question addressed is whether the ensemble method that performs best in static selection remains the top choice when utilized as a pool of classifiers for DS methods. While our analysis provides a percentage of correlation for the research question, with the maximum correlation percentage being 55.90%, it emphasizes the need for a specific approach when selecting ensemble methods for DS applications, which led us to use meta-learning for this purpose. It also emphasizes that the pool of classifiers performing well in SS may not always translate to superior DS performance when used as input.

In the upcoming chapter, we propose a meta-learning recommendation system, whose primary aim is to select the optimal pool of classifiers for DS algorithms. Then, we evaluate the quality of our meta-learning recommendation algorithm under several recommendation settings.

CHAPTER 5

META-LEARNING FRAMEWORK FOR RECOMMENDING THE POOL GENERATION SCHEME AND DS ALGORITHM

In Chapter 4, we demonstrated that no pool generation scheme could outperform others for whole datasets when used as input for DS algorithms based on the Friedman test. In the previous experimental study in the chapter 4 it has shown that although there is one or two pool of classifiers that perform better than others like RF, there are still many datasets that DS algorithms outperform using other pools. to improve the performance and reduce the burden of finding the optimal solution through a costly optimization scheme, we propose a meta-learning recommendation system (MLRS) that, based on a dataset characteristic, can work in three scenarios: 1) predict the best pool generation scheme; 2) predict the best DS method; 3) predict the pair (DS, Pool). These dataset characteristics are called meta-features since they describe properties of the data that are predictive for estimating the performance of learning algorithms trained on them ((Brazdil, Carrier, Soares & Vilalta, 2008)). Hence, this chapter also aims to answer RQ5: *Does selecting the best pool of classifiers, C, for dynamic selection methods depend on what dataset is used?*

5.1 Basic definitions

Before delving into the details of the proposed meta-learning recommendation system for DS methods, it is essential to define the basic concepts and mathematical notation used in this study. A dataset is represented by T, where each dataset is a combination of a training set T_r and a test set T_e. A set of datasets is symbolized by $\mathbb{T} = \{T_1, \ldots, T_Z\}$, where Z represents the number of datasets. The meta-feature vector (i.e., dataset characteristics) extracted from a dataset is denoted by $\mathbf{x}' = \{(m_f)_1, \ldots, (m_f)_n\}$ where each m_f represent a meta-features characteristic extracted from a dataset. The meta-target is indicated by y'. Finally, the meta-dataset is defined as $MT = \{(MT)_1, \ldots, (MT)_Z\}$, where each $(MT)_i$ is represented by the i-th tuple $(\mathbf{x}'_\mathbf{i}, y'_i)$ corresponding to the meta-features and meta-target extracted from a dataset T_i.

The meta-learning recommendation system can work in three distinct scenarios:

- In the first scenario, MLRS recommends a pool generation scheme. In this scenario, we assume that the user has already determined a DS method they intend to use, and the meta-learning recommendation system focuses on selecting the most suitable pool generation scheme based on this choice. We refer to this model as MLRS-P.
- In the second scenario, MLRS recommends a DS method. Here, we consider that the user has a predefined pool generation scheme in mind, and the meta-learning recommendation system guides them toward selecting the DS method that complements their chosen pool. We refer to this model as MLRS-DS.
- In the third scenario, MLRS recommends a pair of (DS, Pool). In this case, our system aims for a fully automated approach, where both the pool generation scheme and the DS method are recommended, allowing for a comprehensive solution. We refer to this model as MLRS-PDS.

The meta-target of the third recommendation, represented by (y'_{pool}, y'_{DS}), is a pair of (Pool, DS). The pool generation scheme in the third recommendation is represented by y'_{pool}, and the DS method is indicated by y'_{DS}. The meta-model is symbolized by M_m.

5.2 Meta-training

The meta-learning stage is a crucial step within the meta-learning framework. During this stage, the system undergoes training to acquire the ability to make informed recommendations for new, unseen tasks (i.e., datasets). During the meta-training stage, the objective is to create a meta-model, M_m, by extracting the characteristics of a set of datasets, $\mathbb{T}$, as meta-features, and by utilizing a meta-target, y', that represents the best performance. Figure 5.1 provides an illustrative depiction of the meta-training process. The whole process can be divided into six main steps, as described below:

Step (1) - Meta-features extraction: This process begins with the extraction of meta-features, $\mathbf{x}' = \{(m_f)_1, \ldots, (m_f)_n\}$ from each dataset under consideration. Hence, the output of this step is a meta-feature vector, $\mathbf{x}'$, extracted for each dataset in the set of datasets.

Step (2) - Generating sets of pools and DS methods: In this step, a set of pools of classifiers, represented as $\mathbb{P} = \{P_1, \ldots, P_K\}$, is generated. These pools are then paired with the set of DS techniques, denoted as $\mathbb{DS} = \{DS_1, \ldots, DS_W\}$, creating comprehensive combinations for evaluation. Thus, given that $\mathbb{DS}$ and $\mathbb{P}$ correspond to all DS techniques and pool generation schemes considered in the recommendation system, respectively, a total of $|\mathbb{DS}| \times |\mathbb{P}|$ models are generated during this step.

Step (3) - Assessment of DS Method Performance: In this step, we assess the performance of DS methods when using the generated pools of classifiers as inputs by calculating its classification accuracy for the corresponding dataset. During this step, all combinations of pool generation and methods are evaluated and their corresponding accuracies are obtained to be used in the next stage while defining the meta-target.

Step (4) - Meta-target determination: In this step, the meta-target should be determined. The meta-target is defined as the combination that obtains the highest accuracy. The meta-learning recommendation system allows us to investigate three possible recommendation scenarios:

- **Scenario I, MLRS-P:** recommend an optimal pool generation scheme. In this meta-learning scenario, the meta-target $y' = y'_{pool}$ is the pool generation scheme that obtained the higher classification performance when used as input for the predefined DS (PDS) method specified by the user.
- **Scenario II, MLRS-DS:** involves recommending the best DS method only. Here, the meta-target $y' = y'_{DS}$ is the DS method that exhibited the highest classification performance according to the predetermined pool generation scheme (PP).
- **Scenario III, MLRS-PDS:** consists of recommending an optimal pair (Pool, DS) automatically. In this context, the meta-target represents the pair (Pool, DS) that delivered the best performance in comparison to other pairs across our study. Hence, in this scenario, the meta-target consists of a tuple $y' = (y'_{pool}, y'_{DS})$ and can be considered a multi-label meta-learning problem.

Step (5) - Meta-dataset: At the outset of this step, a meta-dataset, MT, is constructed by combining the meta-feature vector, $\mathbf{x}'$, with their corresponding meta-target, y' extracted of all datasets in $\mathbb{T}$.

Step (6) - Meta-model training: The meta-learning recommendation system trains a meta-model, M_m, using the meta-dataset, MT. Thus, the meta-model M_m is trained to learn the relationship between the meta-features $\mathbf{x}'$ (i.e., dataset characteristics) and the meta-targets y' to propose recommendations while used to suggest recommendations for new datasets. It is important to mention here that different meta-learning formulations (i.e., different scenarios) generate a different meta-classifier M_m and are considered as a distinct meta-problem. These three scenarios are evaluated in the experimental study of this chapter in order to validate their impact on the system's performance.

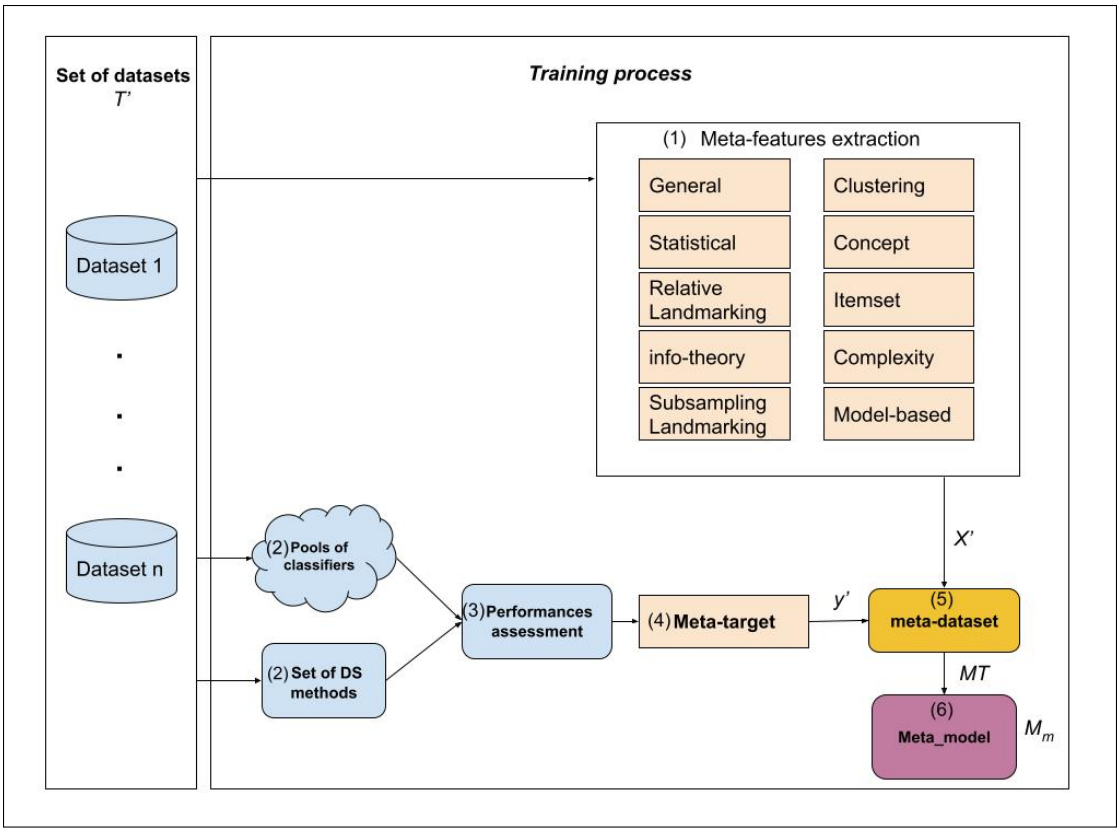


Figure 5.1 Overview of the meta-training process. In the first step, the meta-features, m_f, are extracted from datasets. In step 2, a set of pools and DS methods are generated for assessment in step 3. Then, based on the highest accuracy, the meta-target, y', is defined (step 4). In step 5, the meta-dataset, MT, is constructed, and then it is used to train a meta-model, M_m (Step 6)

The training phase of the meta-learning recommendation system for DS algorithms is outlined in Algorithm 5.1. The algorithm takes inputs: the set of datasets, $\mathbb{T}$, a set of pool generation scheme $\mathbb{P}$, and a set of DS algorithms $\mathbb{DS}$. The output of the algorithm is the meta-model, M_m, that learns the relationship between the dataset characteristics and which models are more likely to obtain higher performance. The algorithm then iterates over each dataset $T_i \in \mathbb{T}$. For each iteration, the algorithm extracts the meta-features vector, $\mathbf{x}'_i$, capturing the characteristics of the training partition of the dataset T_i denoted by $(T_r)_i$. These meta-features are detailed in Section 5.2.1. Then, the combinations of pool and DS methods are assessed using its test

partition denoted by $(T_e)_i$. The configuration that obtains the highest performance is used as the meta-target y'_i.

Then, a meta-dataset, MT, is constructed using the meta-feature vector, $\mathbf{x}'_i$ and the meta-target, y'_i, after iterating over all datasets in the set of dataset $\mathbb{T}$. Subsequently, a meta-model is trained using the meta-dataset, MT. This meta-model, M_m, is used to recommend the best configuration for a given query dataset $\mathbf{Q}$.

Algorithm 5.1 Meta-learning recommendation system for DS algorithms in the training phase

Input: A set of training datasets, $\mathbb{T}$, a set of DS methods, $\mathbb{DS}$, a set of pools, $\mathbb{P}$

Output: Meta-model, M_m

Initialize: $MT = \emptyset$; /*empty list to store the meta-information */.
for *each* T_i *in* $\mathbb{T}$ **do**
    Extract meta-feature vector, $\mathbf{x'}_i$, from the dataset $(T_r)_i$ (Section 5.2.1) ;
    for *each* $DS_j \in \mathbb{DS}$ **do**
        for *each* $P_k \in \mathbb{P}$ **do**
            Assess the performance of DS_j using P_k;
        end for
    end for
    Define the configuration with highest performance as y'_i ;
    $MT = MT \cup (\mathbf{x'}_i, y'_i)$;
end for
Train the meta-model, M_m, on the meta-dataset, MT;
Return M_m;

5.2.1 Meta-features

A crucial step in the meta-training stage involves the extraction of meta-features, denoted as m_f, from a collection of datasets, represented by $\mathbb{T}$. These meta-features, serve as descriptors that

characterize each dataset. Within this work, diverse categories of meta-feature groups, each encompassing different facets, are considered in order to enhance our recommendation capabilities. These categories include Statistical, Information-theoretic, Model-based, Relative Landmarking (Reif *et al.* (2014); Smith-Miles (2009); Rivolli, Garcia, Soares, Vanschoren & de Carvalho (2018)), Subsampling Landmarking (Soares, Petrak & Brazdil (2001)), Clustering-based (Pimentel & De Carvalho (2019)), Concept (Rivolli *et al.* (2018)), itemset (Song, Wang & Wang (2012)), and complexity (Lorena, Garcia, Lehmann, Souto & Ho (2019)). A comprehensive review of these meta-features is made available in (Alcobaça *et al.* (2020)). Below is a brief description of each category:

- **Statistical**: There are standard statistical measures to describe the numerical properties of data distribution (Brazdil *et al.* (2008)). This category for extracting statistical meta-features from datasets covers a range of statistical computations, including computing correlations, covariances, median, minimum, and maximum values, and various other statistical measures for each attribute.
- **Information-theoretic**: It encompasses a diverse set of measures used to analyze datasets' characteristics. They include the assessment of class label and attribute entropy, the quantification of relationships between variables, the evaluation of signal-to-noise ratios, and the estimation of effective attributes (Brazdil *et al.* (2008)).
- **Model-based**: meta-features that are designed to capture characteristics of machine learning models, specifically Decision Tree (DT) models, induced from a dataset. These meta-features encompass specific properties, including depth, shape, and size, which can be indicative of the complexity and structure of the dataset. (Pavel & Soares (2002)).
- **Landmarking**: Performance of simple and efficient learning algorithms such as DEcision trees (DT) and K Nearest neighbors (KNN). The meta-features, m_f, collected in this category include relative and subsampling performances (Reif *et al.* (2014); Smith-Miles (2009); Rivolli *et al.* (2018), soares2001sampling).
- **Clustering-based**: This meta-feature collects information regarding both correlation and dissimilarity. The Correlation Measure quantifies the statistical relationships between pairs of instances in the dataset. The stronger similarity in attribute behaviors among instances

results in a higher correlation. The Dissimilarity Measure is based on the Euclidean distance between instances, representing the distance between data points in the dataset (Pimentel & De Carvalho (2019)).

- **Concept**: Estimate the variability of class labels among examples and the example's density. (Rivolli *et al.* (2018)). Methods for extracting meta-features related to the Concept group from datasets cover computations such as cohesiveness, concept variation, improved concept variation, and weighted distance between examples of the same classes as well as different classes. Hence, they are measures to define whether a feature space is highly irregular (several small groups of samples belonging to the same class) or more uniform in which large regions of samples share the same class labels.
- **Itemset**: Compute the correlation between binary attributes (Song *et al.* (2012)).
- **Complexity**: Estimate the difficulty in separating the data points into their expected classes (Lorena *et al.* (2019)). The extraction of meta-features related to dataset complexity involves various methods, covering computations such as entropy, imbalance ratio, clustering coefficient, network density, Fisher's discriminant ratios, hub scores, and a range of other complexity measures.

Since it is preferable to use a maximum number of relevant meta-features, to grasp the important characteristics of the dataset (Rivolli, Garcia, Soares, Vanschoren & de Carvalho (2022)), we consider the whole set composed of 129 meta-features in this study, including measures coming from all the above-mentioned groups. All these meta-features were computed using the PyMFE library (Alcobaça *et al.* (2020)), version 0.4.2. We believe these are relevant for extracting good dataset descriptions for the meta-learning process. The list containing all individual meta-features used, the group it belongs, and a brief description is presented in the Appendix I.

5.3 Generalization

The generalization stage of our meta-learning recommendation system operates within the framework illustrated in Figure 5.2. This framework enables us to make dataset-specific recommendations, enhancing the performance of DS algorithms by automating the selection of

crucial design steps in deploying a DS solution. To accomplish this, we leverage the meta-model, M_m, developed during the training phase to offer recommendations for a given dataset according to the scenario specified by the user.

The process is initiated by taking a query dataset **Q** under consideration, from which we extract its meta-features representation based on its available training set partition to avoid any bias in our meta-learning prediction. Subsequently, we employ our trained meta-model, M_m, to make predictions and provide the recommendations according to one of the three scenarios in question: provide the optimized pool generation scheme, a DS method, or both as a pair (Pool, DS).

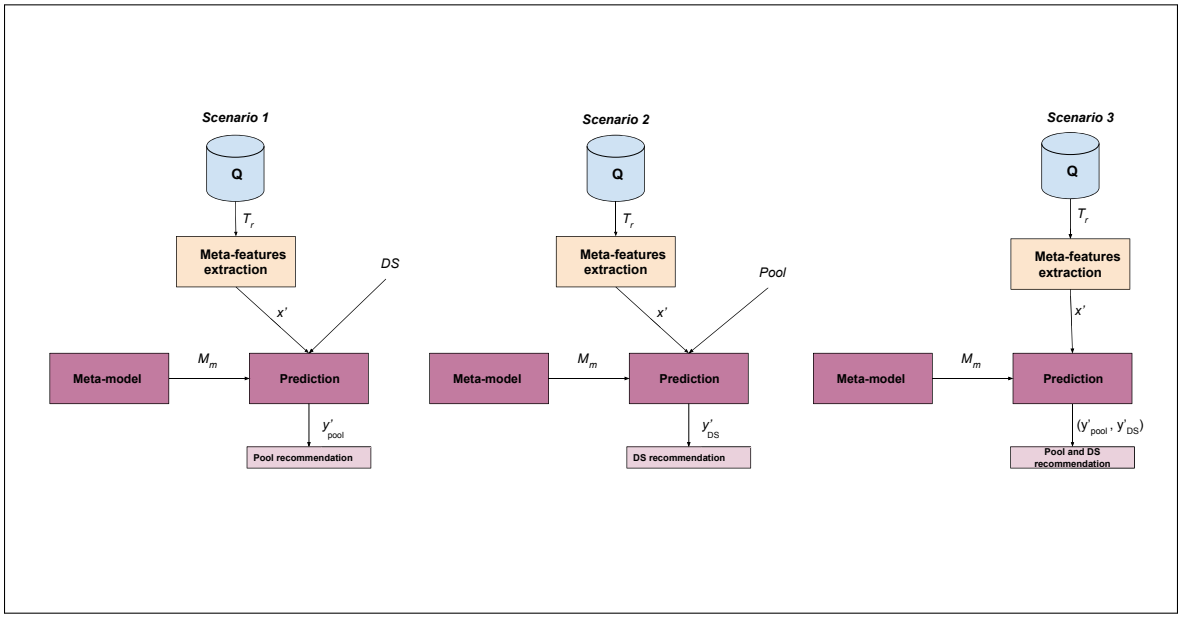


Figure 5.2 The generalization process for the 3 distinct scenarios. Scenario I, a pool generation scheme is recommended. Scenario II, a DS method is recommended. Scenario III, the best pair of (Pool, DS) is recommended

Then, the meta-learning recommendation system can produce three recommendations as its outputs.

- **Scenario I, MLRS-P:** As the first recommendation, the meta-learning recommendation system suggests a pool generation scheme, C, where a DS method is fixed. This is useful when a specific DS method should be used, and an optimized pool generation scheme, C, is needed for a given query dataset.
- **Scenario II, MLRS-DS:** As the second type of recommendation for this study involves suggesting a DS method with a pre-determined pool generation scheme to use. This approach is beneficial when a specific pool generation scheme should be used, and the best DS method is needed for a given query dataset.
- **Scenario III, MLRS-PDS:** The third type of recommendation proposed, consists in defining both the pool and DS model, just taking into account the dataset characteristics (i.e., the meta-features). This consists of a multi-label recommendation problem, to obtain the best pair of (Pool, DS) for a given query dataset **Q**. MLRS-PDS works by. first recommending the pool based on the meta-features. Then, given the selected pool and the meta-features, it recommends the DS model to obtain the (Pool, DS) in a fully automated fashion to optimize the performance according to the query dataset **Q**.

5.4 Experimental study

In this section, the proposed meta-learning recommendation system's performance is evaluated. The datasets and experimental setup align with the details outlined in Chapter 3, consisting of the 288 datasets from the ICPR landscape competition (Macià *et al.* (2010)). For this experiment, we employed the leave-one-dataset-out (LODO) procedure, where, at each iteration, one dataset (**Q**) is left for testing, and the remaining ones are used to construct the meta-dataset MT used for training the meta-models M_m. In other words, for each simulation, 287 datasets were considered for training the meta-learning framework, while one was considered as the test dataset **Q** used to evaluate its generalization performance. For each test dataset, we conduct 10 replications following the experimental protocol defined in Chapter3.

The main comparison in this experimental study is between the performance of the meta-learning recommendation system and the performance of DS methods when using either a predetermined

set of pools of classifiers (PP) or a predetermined set of DS (PDS), in order to know whether the proposed framework succeeds in his tasks of providing a suitable recommendation for each task type. It is important to mention that this is the first research work that tries to recommend the pool generation scheme and DS algorithm based on dataset characteristics and meta-learning. As such, there are no other related works in meta-learning for this task to compare to other than these baseline models.

To further assess the advantages of the proposed meta-learning recommendation framework and its improvements over existing methods, we utilized a pairwise statistical analysis through the Sign test as recommended by Demšar (2006). This non-parametric test evaluates the performance outcomes of the recommended method, designated as the control algorithm, against a baseline according to the number of wins, ties, and losses this algorithm obtained (i.e., superior performance, equivalent performance, lower performance). For Scenario I, the baseline in this context pre-established pool generation scheme across all datasets for DS methods. For Scenario II, the baseline is a pre-defined DS method using different pool generation schemes. For Scenario III, the baseline is all possible 49 configurations (7 pool generation schemes $\times$ 7 DS algorithms).

The analysis operates on two fundamental hypotheses: the null hypothesis ($\mathbf{H}_0$) and the alternative hypothesis ($\mathbf{H}_1$). Rejecting $\mathbf{H}_0$ implies that the DS technique in question exhibits a statistically significant enhancement in performance compared to the baseline method. To apply this test accurately, it is imperative to calculate the critical value (n_c) through the formula presented in Equation 5.1. This calculation determines the threshold for statistical significance between the control and compared techniques.

$$n_c = \frac{n_{exp}}{2} + z_\alpha \sqrt{\frac{n_{exp}}{2}} \tag{5.1}$$

where n_{exp} is the total number of experiments, and z_α is the critical value of the standard normal distribution for a significance level of $\alpha = 0.05$. In this study, with a total of 288 tests and a

significance level of 0.05, the calculated critical value (n_c) is 164. Therefore, if the number of wins is greater than or equal to 164, the null hypothesis that the technique being tested is no better than the baseline can be rejected with a significance level $\alpha = 0.05$ (i.e., a confidence level of 95%) (Demšar (2006)).

5.4.1 Scenario I: meta-learning for recommending the best pool generation scheme

As the first recommendation of the research, the meta-learning recommendation system suggests a pool generation scheme, C, while a DS method is fixed. This is useful when a specific DS method should be used, and an optimized pool generation scheme, C, is needed for a given dataset, $\mathbf{Q}$.

Three algorithms, including Random Forests (RF), K Nearest Neighbors (KNN), and Support Vector Machine (SVM), were initially considered as meta-models for the meta-learning recommendation system. The selection of these specific meta-models, M_m, was determined based on previous studies (Bilalli, Abelló Gamazo & Aluja Banet (2017); Khan *et al.* (2020)). These articles emphasize these options as the best-performing learning algorithms to be used as meta-learners.

To find which technique is more effective as a meta-model, M_m, in this scenario, multiple runs were conducted with different hyperparameter configurations. For K Nearest Neighbors, values of K ranging from 2 to 6 were tested, while for Random Forests, Max Depth values of 2 to 5 were tried. The hyperparameter chosen for the Support Vector Machine was gamma equal to 'scale' (i.e., $1/(n\_features \times X.var())$) and the cost, $c = 1$ (Pedregosa *et al.* (2011)). The results of the runs and the corresponding hyperparameters are presented in Table 5.1. Since the RF with a maximum depth of 5 obtained be overall best results, we used this classifier to build the meta-model in our system.

Table 5.1 The results of meta-learning prediction with different hyperparameters for scenario I

Algorithm	Hyperparameter	KNORA-E	META-DES	KNORA-U	DES-MI	DES-P	MLA	OLA
RF	Max_depth = 5	**78.47**	**70.83**	**69.44**	**78.81**	**71.87**	**67.70**	**62.84**
RF	Max_depth = 4	70.83	62.15	59.72	73.61	59.72	53.81	54.51
RF	Max_depth = 3	60.76	51.38	50.34	63.19	51.38	47.22	49.65
RF	Max_depth = 2	52.77	40.62	41.66	52.08	43.40	41.31	48.26
KNN	k = 2	65.62	62.50	64.93	70.13	69.09	60.06	**62.84**
KNN	k = 3	59.72	57.63	58.33	63.88	60.06	54.86	56.94
KNN	k = 4	61.11	54.51	56.59	61.80	59.37	50.34	58.33
KNN	k = 5	58.33	50.34	53.81	59.72	55.55	47.91	55.90
KNN	k = 6	55.55	48.26	51.73	59.02	53.47	46.87	52.77
SVM	gamma='scale', $c = 1$	36.11	31.25	36.11	41.66	37.84	28.47	48.26

Table 5.2 compares the performance of DS methods while they are using a pool generation scheme recommended by MLRS-P with the average performance of DS methods using a predetermined pool generation scheme (DS, PP) among 288 datasets. Each row in this table is named using the fixed DS method. Also, the average number of wins the corresponding technique obtained is indicated in parentheses. For example, KNORA-E, by using the pool generation scheme recommended by the MLRS-P, had the best performance in 228 datasets out of 288 which corresponds to 79.16% of the datasets. In contrast, the average number of wins of KNORA-E across different combinations, including (KNORA-E, BDT), (KNORA-E, BP), (KNORA-E, BSDT), (KNORA-E, BSP), (KNORA-E, RF), (KNORA-E, FLT), (KNORA-E, LIT) was 21.92%, with the average number of wins among these combinations being 43. For DS methods, the performance of our proposed MLRS-P is significantly higher than the baseline, as well as the random prediction (which, in this multi-class classification problem, is 1/7). Thus, we can conclude that the MLRS-P can model the relationship between the meta-features to predict the best pool generation scheme for a DS method.

Table 5.2 The results of DS methods that used the pool generation scheme proposed by the meta-learning recommendation system (DS, MLRSP) compared with the performance of DS methods that used a predetermined pool generation scheme (DS, PP). Note that (PP) refers to the predetermined pool generation scheme. And (Wins) denotes the number of wins out of a total of 288 datasets. (Ave_Wins) denotes the average number of the datasets with the best performance among 288 datasets between 7 pools of classifiers

Algorithm	MLRS-P (Wins)	(DS, PP)(Ave_wins)
KNORA-E	79.16(228)	21.92(43)
META-DES	71.87(207)	24.84(44.86)
KNORA-U	70.13(202)	25.04(48.29)
DES-MI	78.12(225)	22.26(42.86)
DES-P	71.52(206)	25.24(45.29)
MLA	66.66(192)	24.65(48.14)
OLA	62.84(181)	20.83(42.71)

The Sign test was conducted to compare the MLRS-P against the baseline. The outcomes depicted are for wins, losses, and ties shown in figures 5.3 to figure 5.9 for each specific DS model.

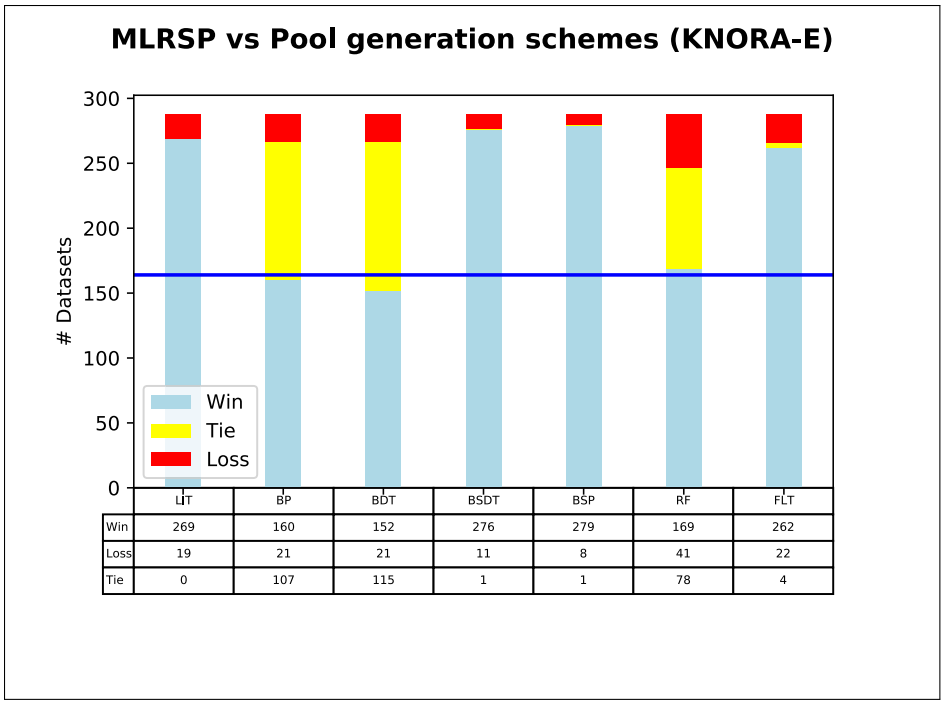


Figure 5.3 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The KNORA-E was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

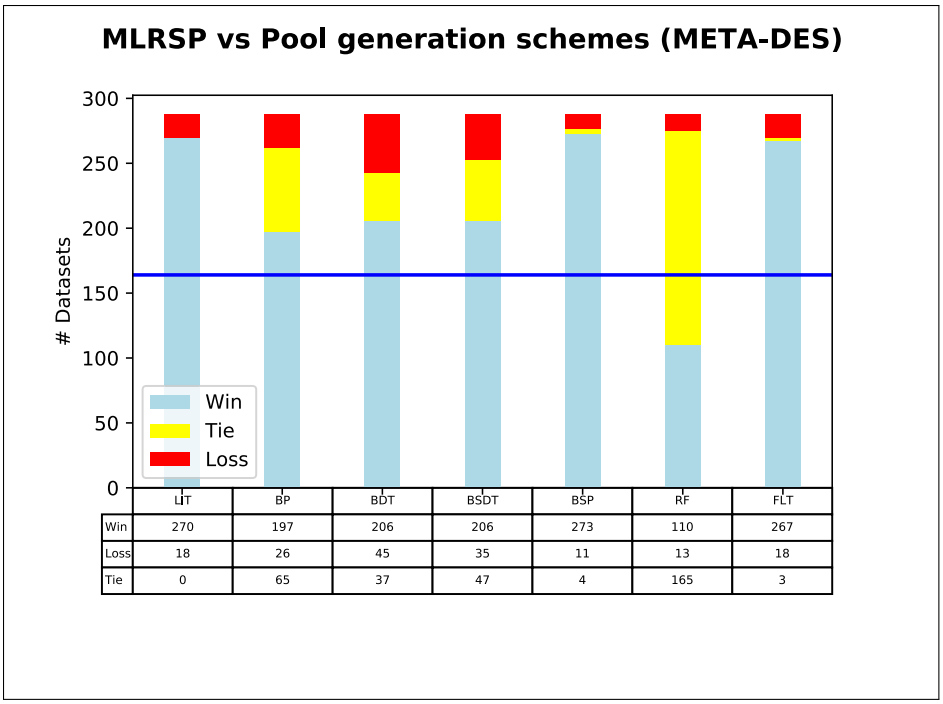


Figure 5.4 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The META-DES was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

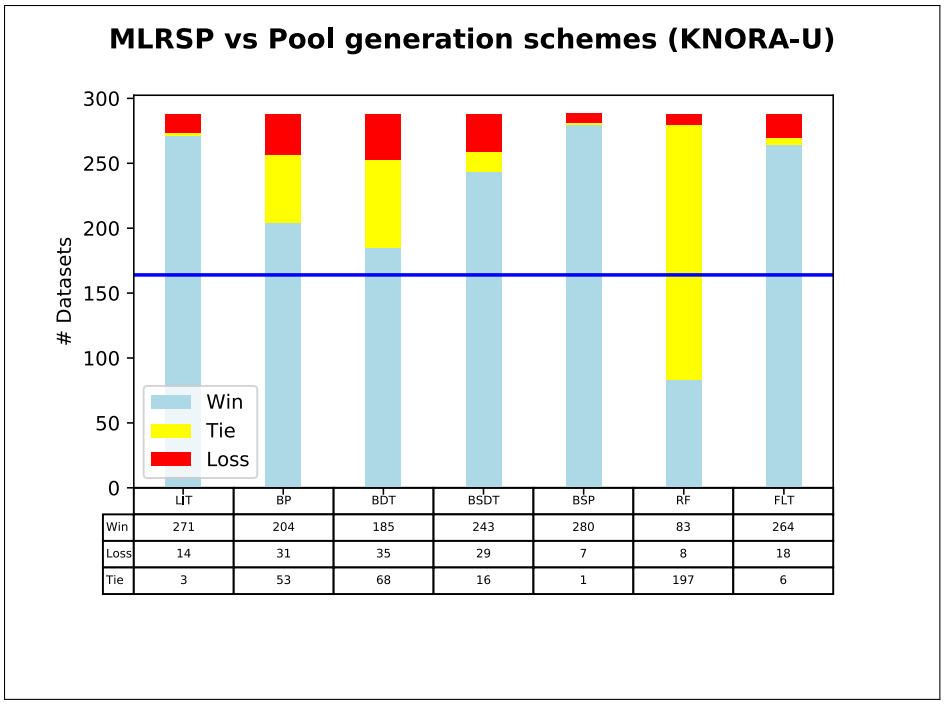


Figure 5.5 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The KNORA-U was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

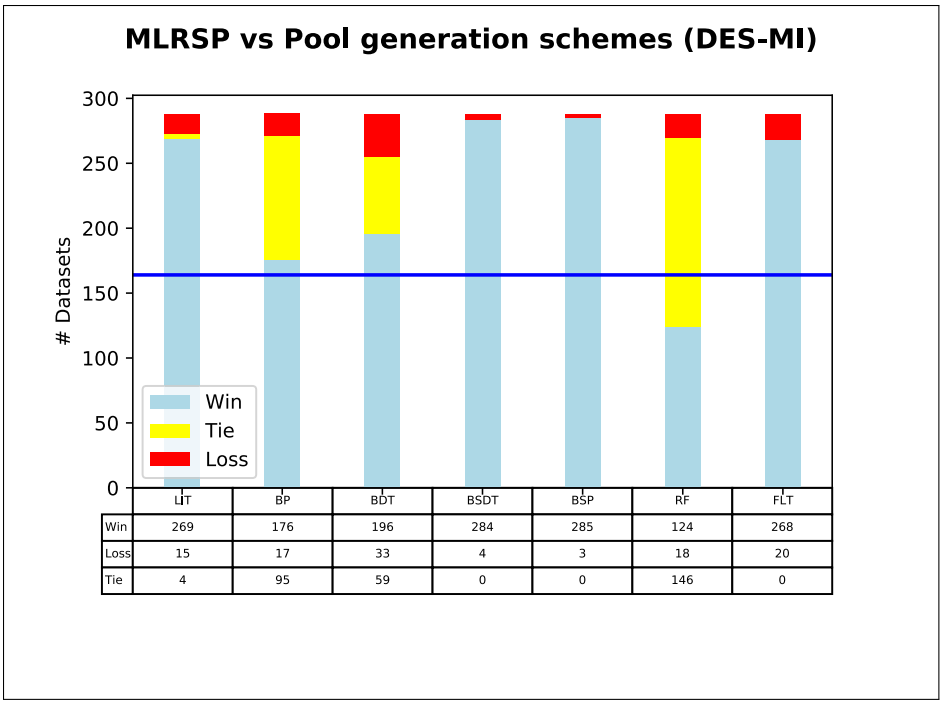


Figure 5.6 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The DES-MI was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

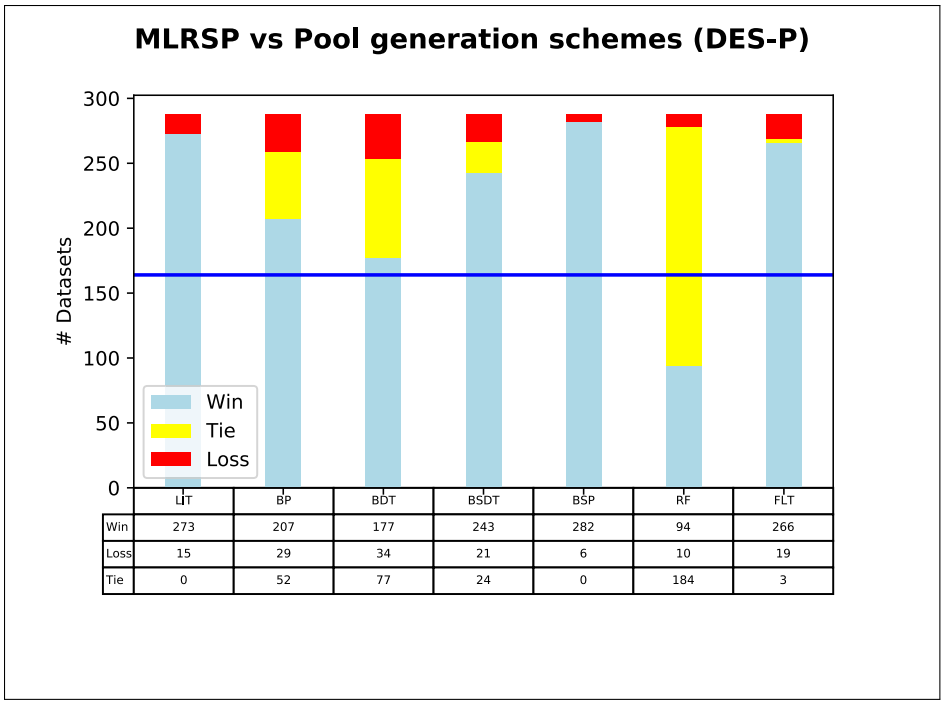


Figure 5.7 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The DES-P was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

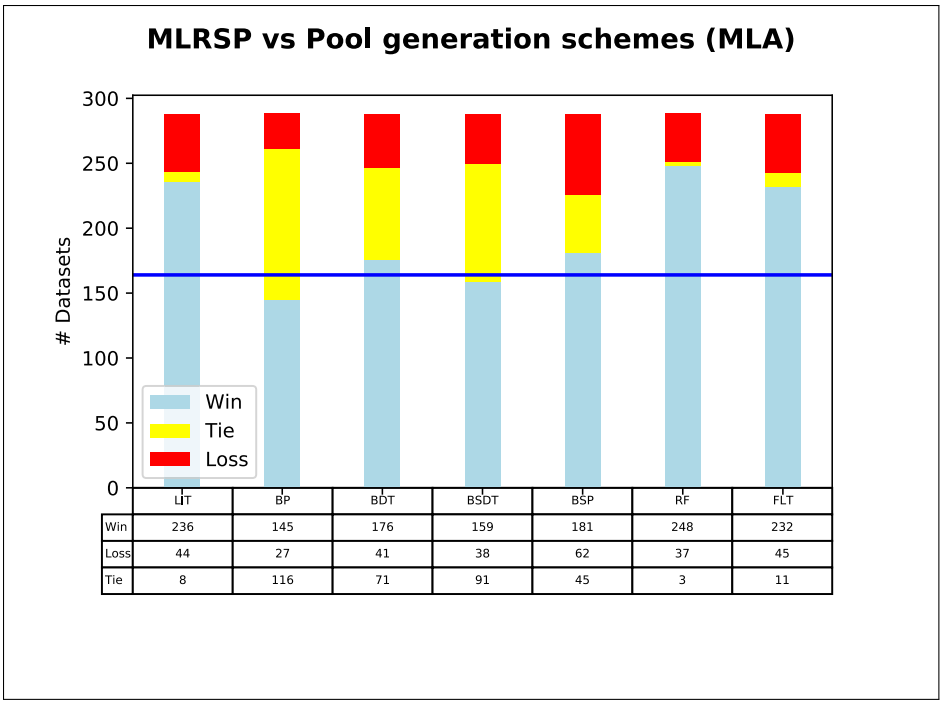


Figure 5.8 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The MLA was considered as the predefined DS method (PDS) The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

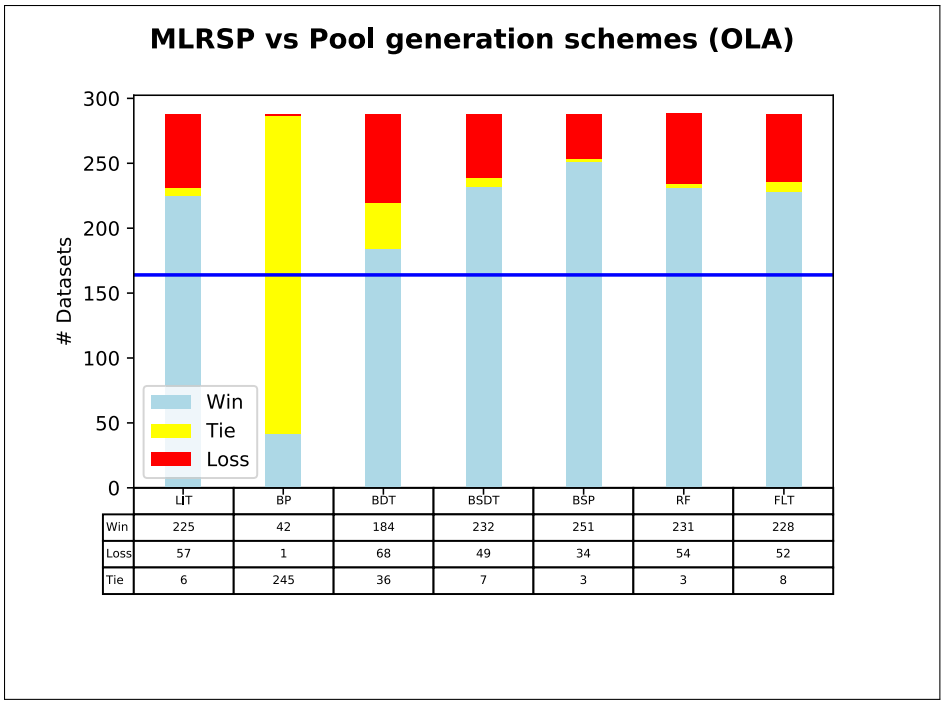


Figure 5.9 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the pool generation scheme recommendation and other pool generation schemes. The OLA was considered as the predefined DS method (PDS). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

These figures demonstrate that pools recommended by MLRS-P generally achieve a significantly higher number of wins compared to the baselines. In most cases, these wins exceed the critical value, suggesting that the proposed framework effectively recommends the most suitable pool generation method. The notable exception occurs with RF as the pool generation scheme, where a considerable number of ties are observed. This phenomenon can be attributed to RF's overall superiority as a pool generation method, as detailed in Chapter 4.

Since the Sign test incorporates half of the number of ties when comparing with the critical value, solely relying on the win-tie-loss diagram might lead to inconclusive interpretations. Therefore, we have consolidated the outcomes of all pairwise comparisons using this test in Table 5.3. This table uses the symbols +, =, and - to indicate whether MLRS-P statistically outperforms, matches, or underperforms relative to each baseline. It is evident that MLRS-P surpasses all

baselines in this statistical test, reinforcing its efficacy in recommending the most suitable pool generation schemes for any given DS method. These results underline the robustness of the meta-learning recommendation model, affirming its capability to model the relationship between problem characteristics and the selection of the appropriate pool.

Table 5.3 Comparison of MLRS-P Performance Against Baseline Methods. "+" indicates MLRS-P wins, "=" denotes a tie, and "-" signifies a loss

	LIT	BP	BDT	BSDT	BSP	RF	FLT
KNORA-E	+	+	+	+	+	+	+
META-DES	+	+	+	+	+	+	+
KNORA-U	+	+	+	+	+	+	+
DES-MI	+	+	+	+	+	+	+
DES-P	+	+	+	+	+	+	+
MLA	+	+	+	+	+	+	+
OLA	+	+	+	+	+	+	+

Moreover, we present the distributions of the recommended pool by MLRS-P while fixing different dynamic selection methods. Each figure corresponds to a specific fixed dynamic selection method. Figure 5.10 showcases the distribution when KNORA-E is fixed, while Figure 5.11 displays the distribution with META-DES fixed. Similarly, Figures 5.12 through 5.16 represent the distributions when KNORA-U, DES-MI, DES-P, OLA, and MLA are fixed, respectively.

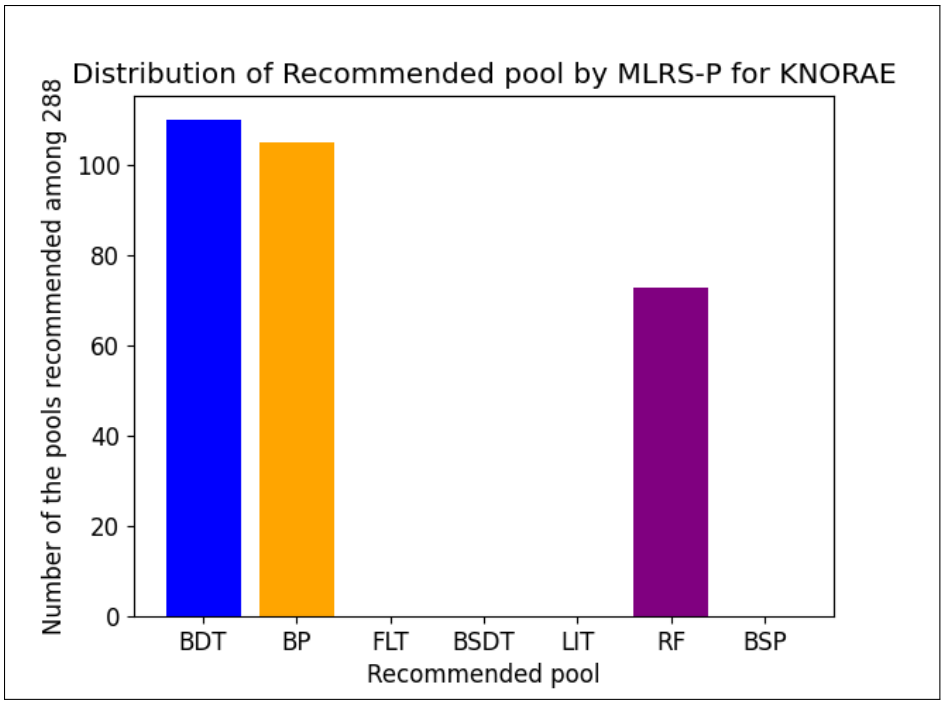


Figure 5.10 The distribution of the recommended pool by MLRS-P while KNORA-E is fixed

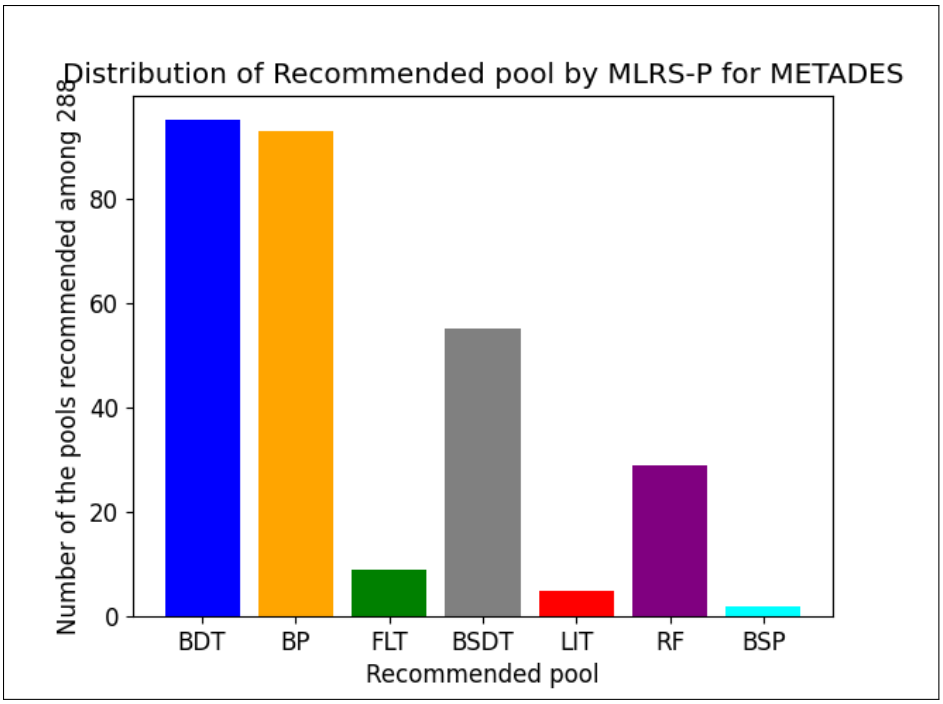


Figure 5.11 The distribution of the recommended pool by MLRS-P while META-DES is fixed

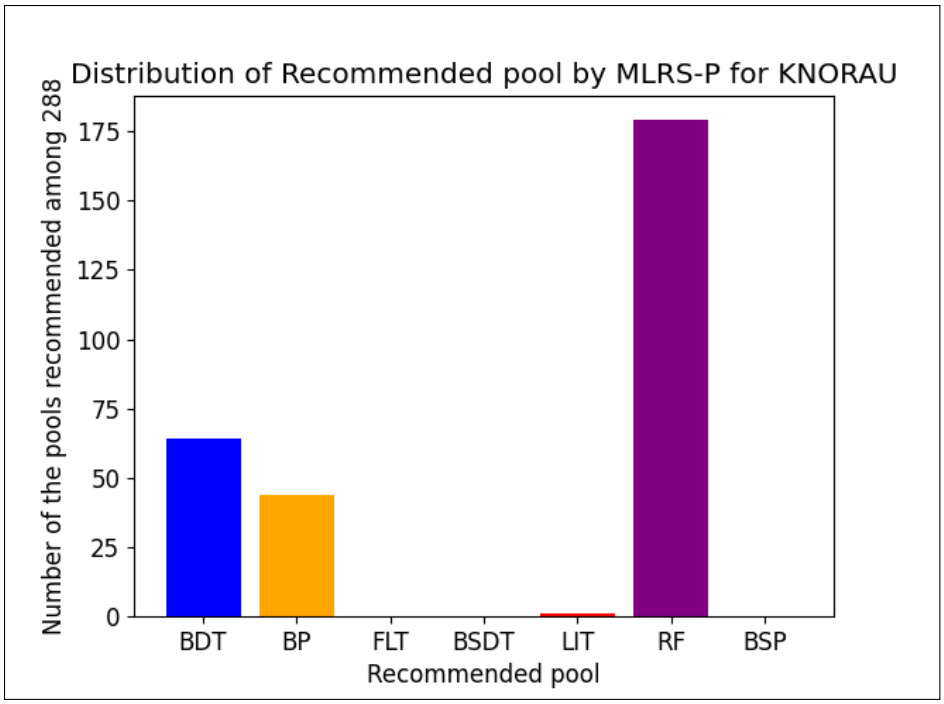


Figure 5.12 The distribution of the recommended pool by MLRS-P while KNORA-U is fixed

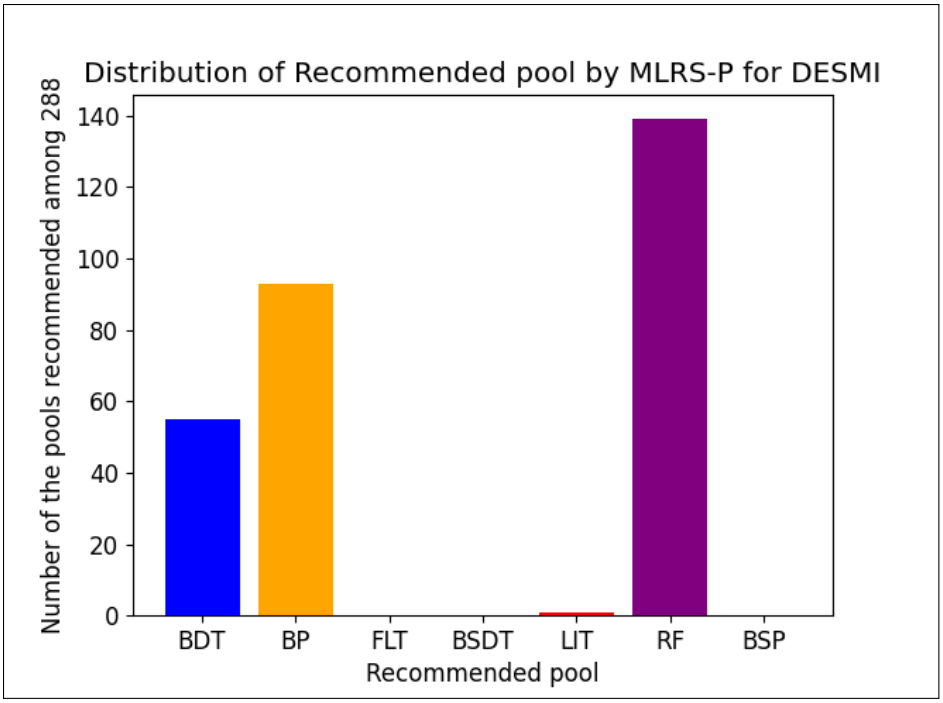


Figure 5.13 The distribution of the recommended pool by MLRS-P while DES-MI is fixed

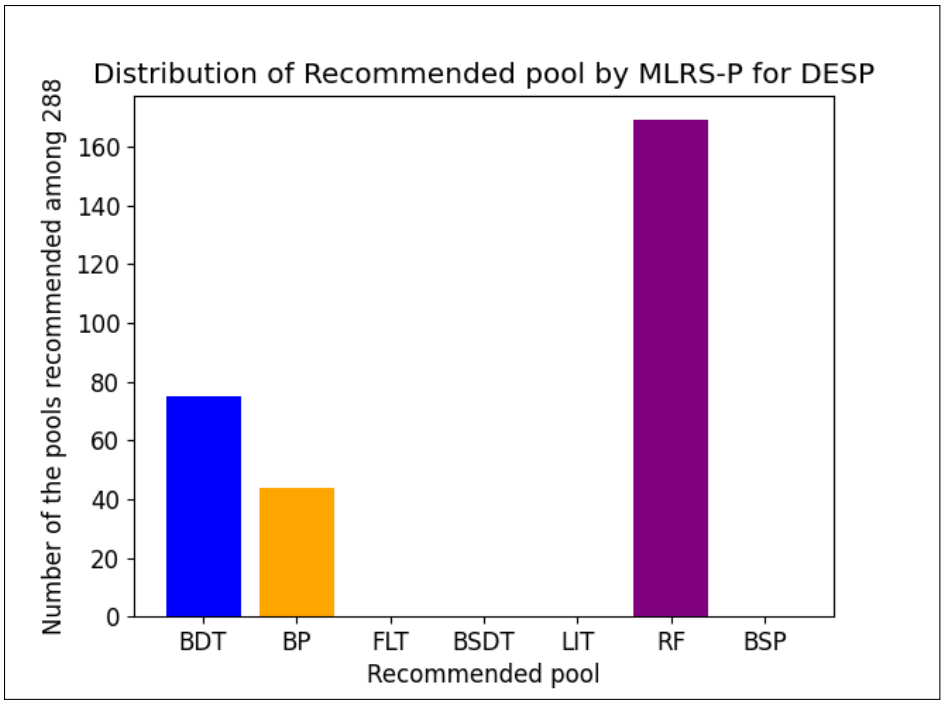


Figure 5.14 The distribution of the recommended pool by MLRS-P while DES-P is fixed

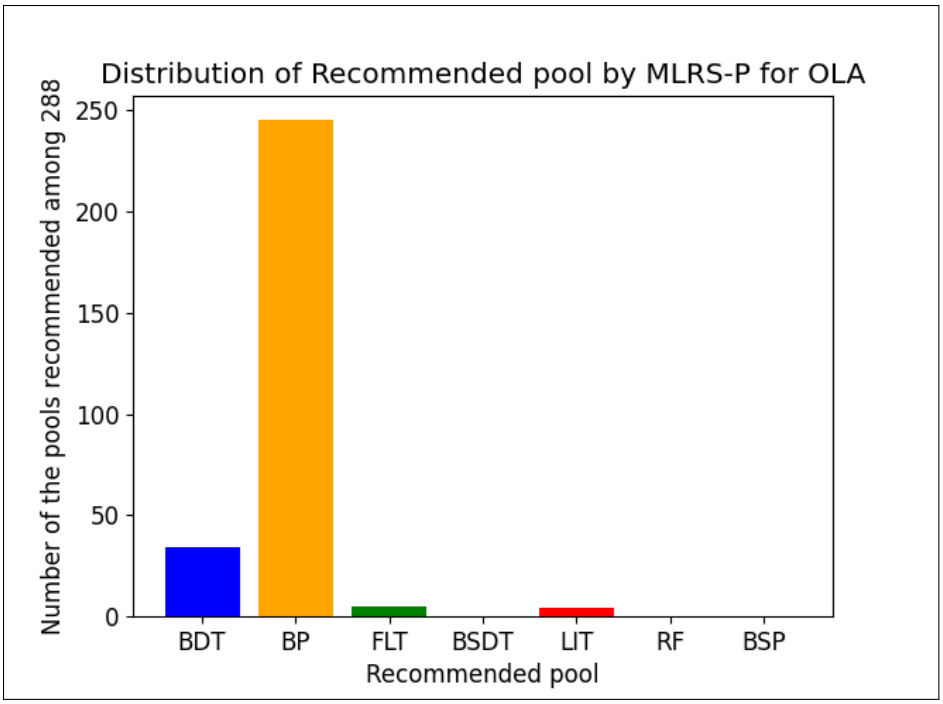


Figure 5.15 The distribution of the recommended pool by MLRS-P while OLA is fixed

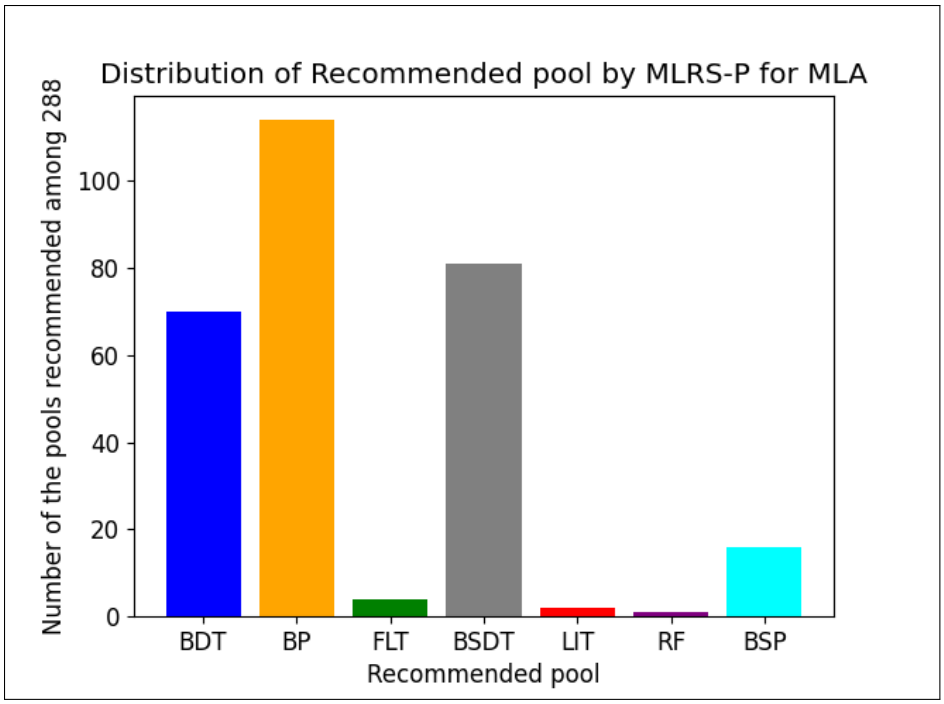


Figure 5.16 The distribution of the recommended pool by MLRS-P while MLA is fixed

The distributions of the recommended pool by MLRS-P show that the most recommended pool for different DS methods was not the same. The results indicate that MLRS-P suggests BDT most of the time for KNORA-E and META-DES both. BP stands as the second most recommended among 288 datasets for these methods. The most suggested pool generation scheme was RF for KNORA-U, DES-MI, and DES-P. As a result, since all pools suggested by MLRS-P for DES are based on DT, this underscores the significance of DT in the process. Additionally, BP was the most recommended pool for both DCS methods including MLA and OLA. Thus, we can conclude that our proposed MLRS-P can indeed model the relationship between the best dataset characteristics and the DS method used in order to recommend the best pool generation scheme.

5.4.2 Scenario II: meta-learning for recommending the best DS model

As the second recommendation of the research, the meta-learning recommendation system suggests a DS algorithm while a pool generation scheme, C, is fixed. To find which learning algorithm is more suitable as the meta-classifier, we evaluate several hyperparameter configurations that are more effective as a meta-model, M_m, in this meta-learning recommendation scenario. The results of these multiple runs are presented in table 5.4.

Based on this analysis we can conclude that the KNN with the k equal to 2 is the best approach to be used as the meta-classifier for Scenario II (MLRS-DS). This is interesting since it contrasts with the RF with max-depth 5 found from scenario I, indicating that each meta-learning recommendation formulation may require different models.

Table 5.4 The results of meta-learning prediction with different hyper-parameters while pool generation scheme is fixed. Numbers correspond to their average performance

Algorithm	Hyperparameters	(DS, LIT)	(DS, BP)	(DS, BDT)	(DS, BSDT)	(DS, BSP)	(DS, RF)	(DS, FLT)
RF	Max_depth = 5	**71.87**	**63.19**	62.50	61.11	**63.88**	59.37	57.63
RF	Max_depth = 4	56.59	52.77	60.76	57.98	57.63	54.86	48.61
RF	Max_depth = 3	48.26	45.48	59.37	57.63	54.16	52.43	41.31
RF	Max_depth = 2	37.15	39.93	59.02	57.29	50.34	50.69	37.15
KNN	K = 2	63.19	60.06	**69.79**	**67.70**	63.19	**67.01**	**62.5**
KNN	K = 3	57.63	51.04	64.93	62.84	53.12	60.06	54.16
KNN	K = 4	52.08	50	61.45	62.50	48.95	55.55	50.69
KNN	K = 5	48.26	47.22	62.15	60.41	50.69	51.38	48.26
KNN	K = 6	47.56	45.83	63.19	60.41	48.26	50.69	47.91
SVM	gamma='scale', c = 1	32.63	34.37	59.37	57.63	34.02	50.34	31.59

Table 5.5 presents a comparison between the performance of MLRS-DS against the baseline, which is a pre-defined DS method using different pool generation schemes. Each row in the table corresponds to a fixed pool of generation scheme. Also, the average number of wins is indicated in parentheses. For instance, the performance of the MLRS-DS was 71.87% among 288 datasets (207 wins in 288 datasets). In contrast, for example, the average performance of

DS methods that use LIT, including (KNORA-E, LIT), (META-DES, LIT), (KNORA-U, LIT), (DES-MI, LIT), (DES-P, LIT), (MLA, LIT), (OLA, LIT) was 21.87%, with the average number of wins among these combinations being 42.71.

Table 5.5 The results of MLRS-DS compared with the average performance of baseline among 288 datasets. Note that (Wins) denotes the number of wins out of a total of 288 datasets. (Ave_wins) denotes the average number of the datasets with the best performance among 288 datasets between 7 pools of classifiers

pool generation scheme	MLRS-DS(Wins)	(PDS, P)(Ave_wins)
LIT	71.87(207)	21.87(42.71)
BP	63.19(182)	30.25(66.57)
BDT	62.50(180)	26.98(49.14)
BSDT	61.11(176)	22.17(45.14)
BSP	63.88(184)	20.83(43)
RF	59.37(171)	27.03(52.85)
FLT	57.63(166)	23.90(43.28)

Figures 5.17 to 5.23, presents the win-tie-loss diagrams showing all pairwise comparison between MLRS-DS and the baselines method. We can observe that in the vast majority of cases, the MLRS-DS obtains a number of wins that is higher than the critical value (delimited by the blue line), except for a few cases when considering the stronger DS techniques such as the META-DES (which are considered the overall best DS), and KNORA-E where a significant number of ties can be observed. As the sign test also considers the half of the number of ties in the comparison with the critical value, we summarize the results of all pairwise comparisons in Table 5.6. In this table, the +, = and - signs illustrates whether the proposed MLRS-DS obtained statistically better, equivalent or worse results compared to the corresponding baseline.

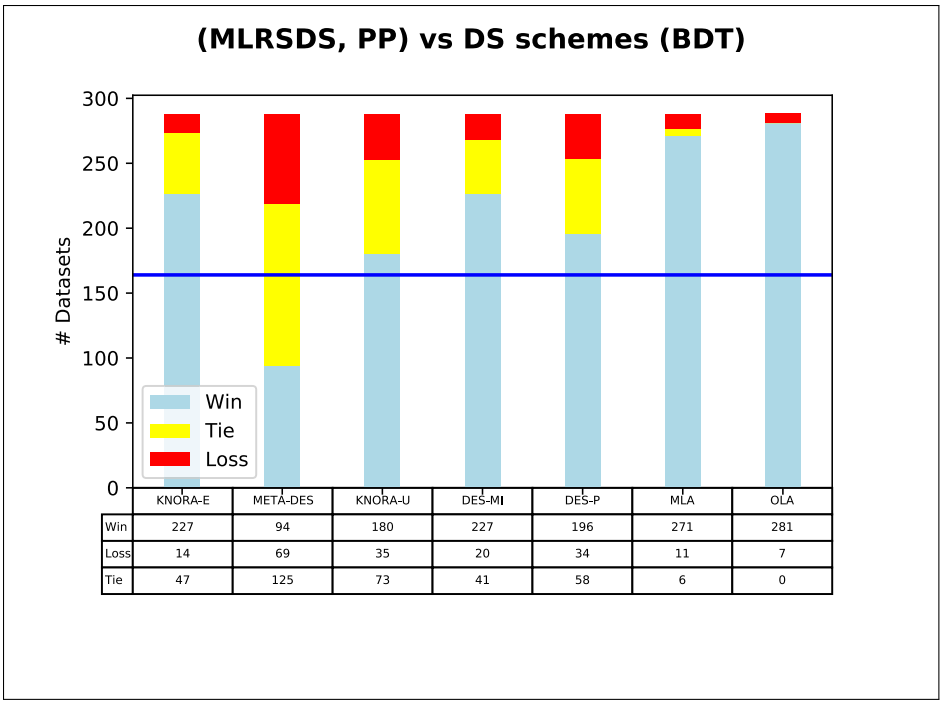
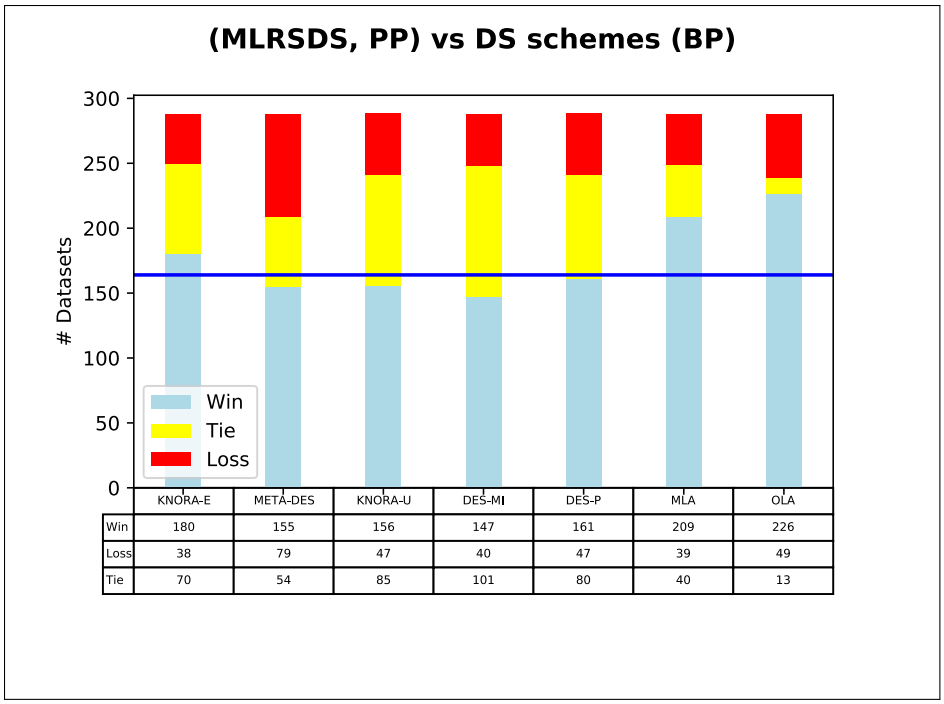


Figure 5.17 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The BDT was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$



	KNORA-E	META-DES	KNORA-U	DES-MI	DES-P	MLA	OLA
Win	180	155	156	147	161	209	226
Loss	38	79	47	40	47	39	49
Tie	70	54	85	101	80	40	13

Figure 5.18 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The BP was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

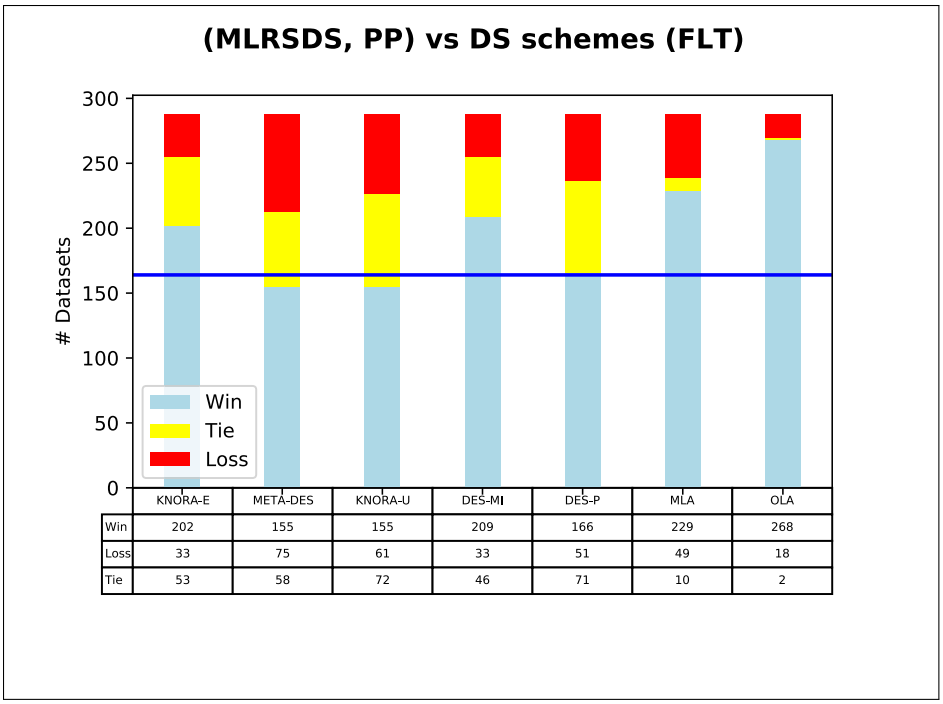


Figure 5.19 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The FLT was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

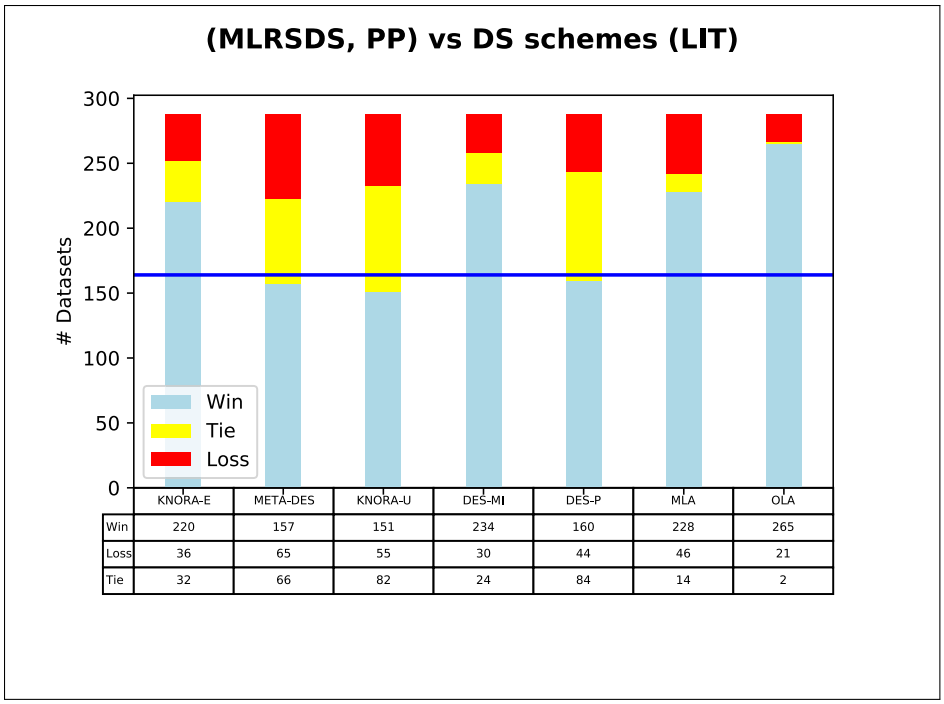


Figure 5.20 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The LIT was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

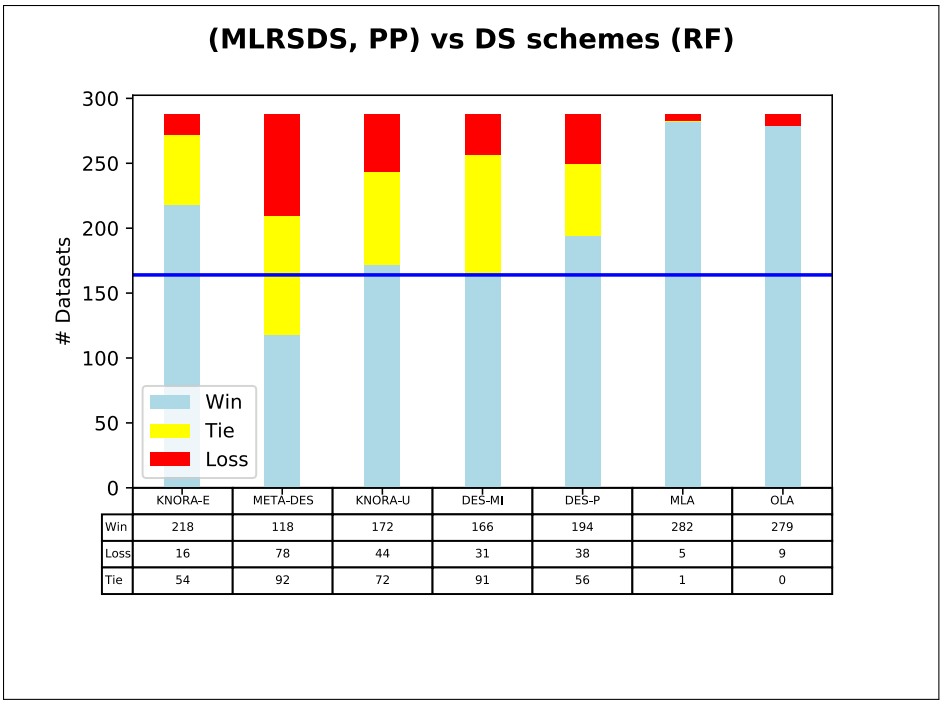


Figure 5.21 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The RF was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

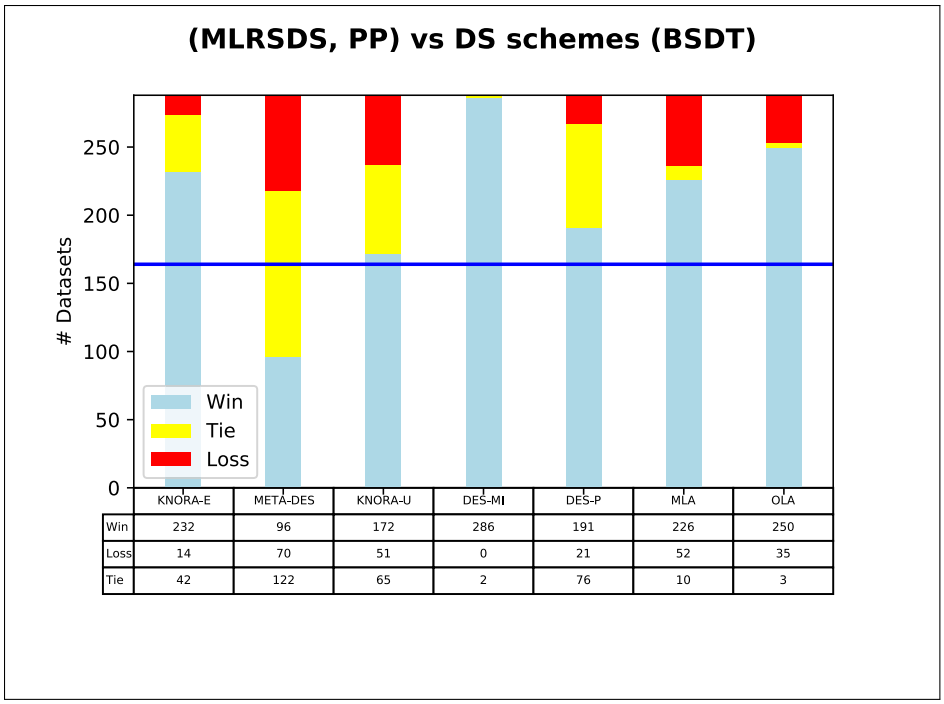


Figure 5.22 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The BSDT was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

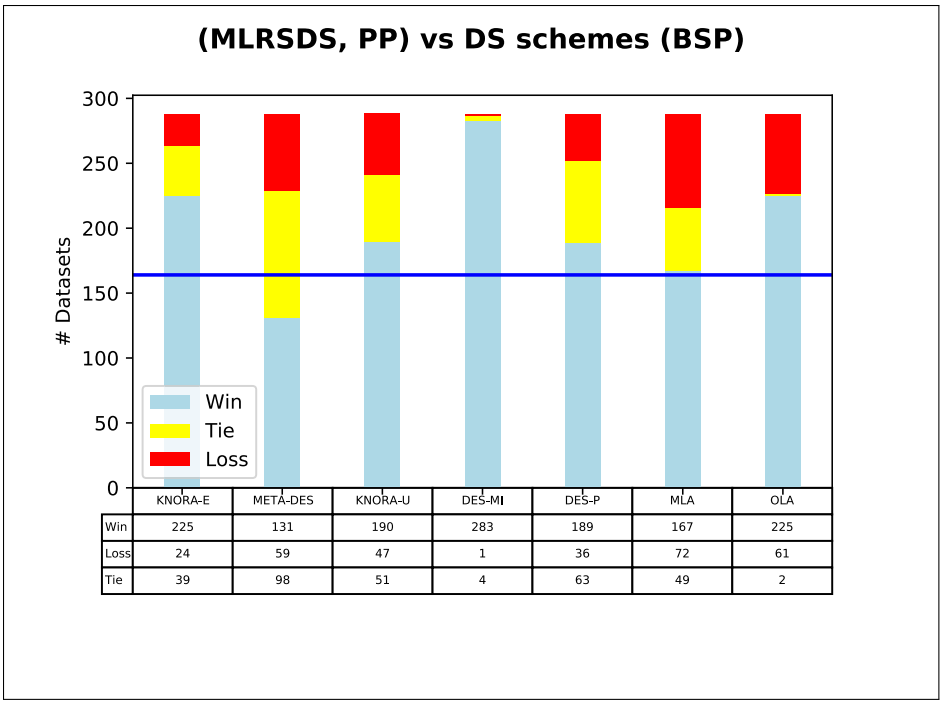


Figure 5.23 Pairwise comparison using the Sign Test between the meta-learning recommendation system for the DS method (Scenario II) and other DS schemes. The BSP was considered as the predefined pool method (PP). The horizontal line illustrates the critical values considering a confidence level of $\alpha = 0.05$

Table 5.6 Comparison of MLRS-DS Performance Against Baseline Methods. "+" indicates MLRS-DS wins, "=" denotes a tie, and "-" signifies a loss

	KNORA-E	META-DES	KNORA-U	DES-MI	DES-P	MLA	OLA
BDT	+	=	+	+	+	+	+
BP	+	+	+	+	+	+	+
FLT	+	+	+	+	+	+	+
LIT	+	+	+	+	+	+	+
RF	+	+	+	+	+	+	+
BSDT	+	=	+	+	+	+	+
BSP	+	+	+	+	+	+	+

These results further confirm that the proposed MLRS-DS statistically outperformed the majority of the predetermined DS methods (PDS) among 288 datasets, being only deemed statistically equivalent to the META-DES when the pool generation scheme is generated using

the BDT technique. Hence, we can conclude that our MLRS-DS also succeeded in its task of recommending the most suitable DS method for a given dataset and pool generation scheme.

Moreover, we analyzed the distributions of the recommended DS methods by MLRS-DS while a pool generation scheme is fixed. Each figure corresponds to a specific fixed pool generation scheme. Figure 5.24 illustrates the distribution when BDT is fixed, while Figure 5.25 depicts the distribution with BP fixed. The subsequent figures, 5.26 through 5.30, represent the distributions when BSDT, BSP, RF, FLT, and LIT are fixed, respectively.

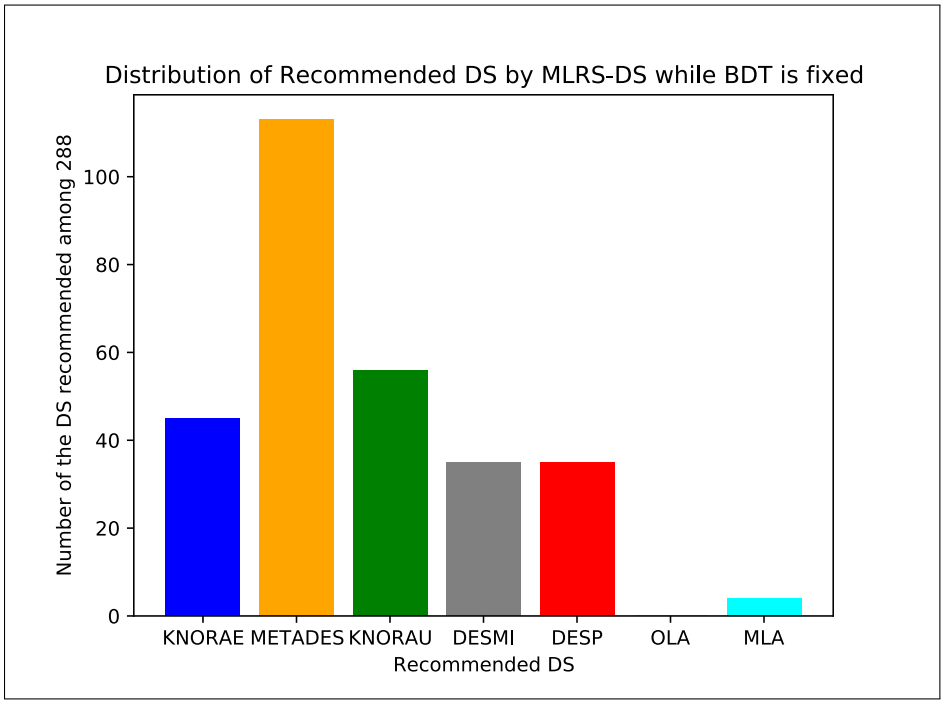


Figure 5.24 The distribution of the recommended DS method by MLRS-DS while BDT is fixed

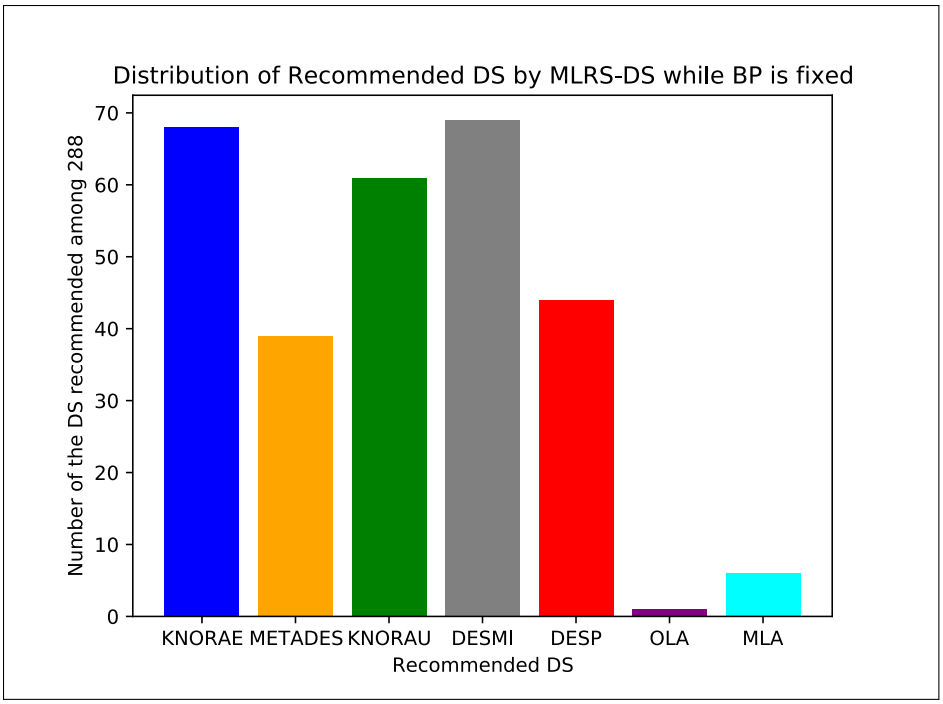


Figure 5.25 The distribution of the recommended DS method by MLRS-DS while BP is fixed

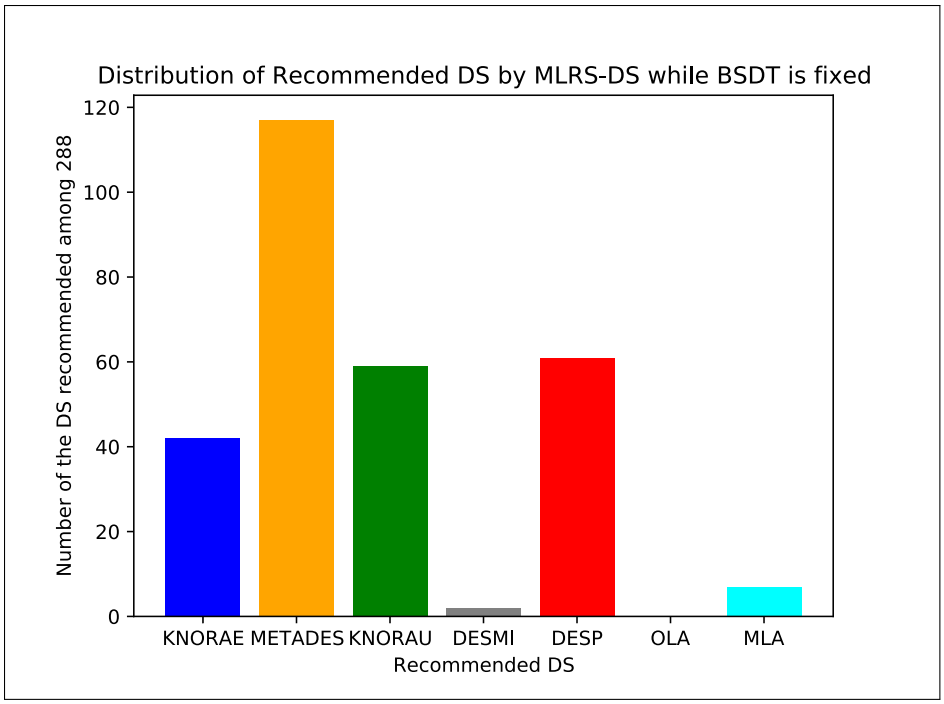


Figure 5.26 The distribution of the recommended DS method by MLRS-DS while BSDT is fixed

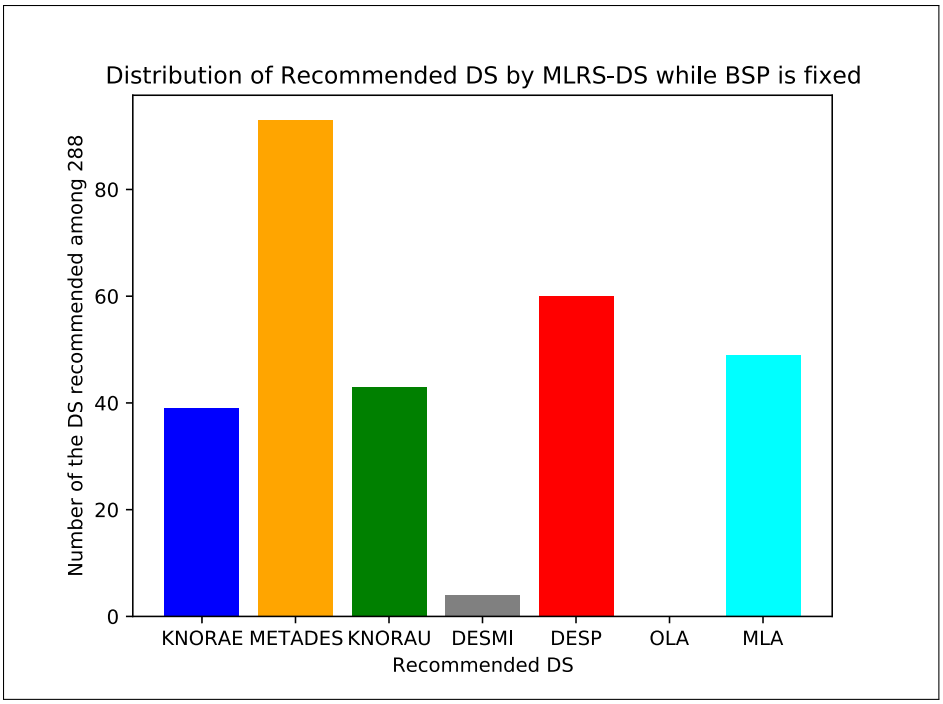


Figure 5.27 The distribution of the recommended DS method by MLRS-DS while BSP is fixed

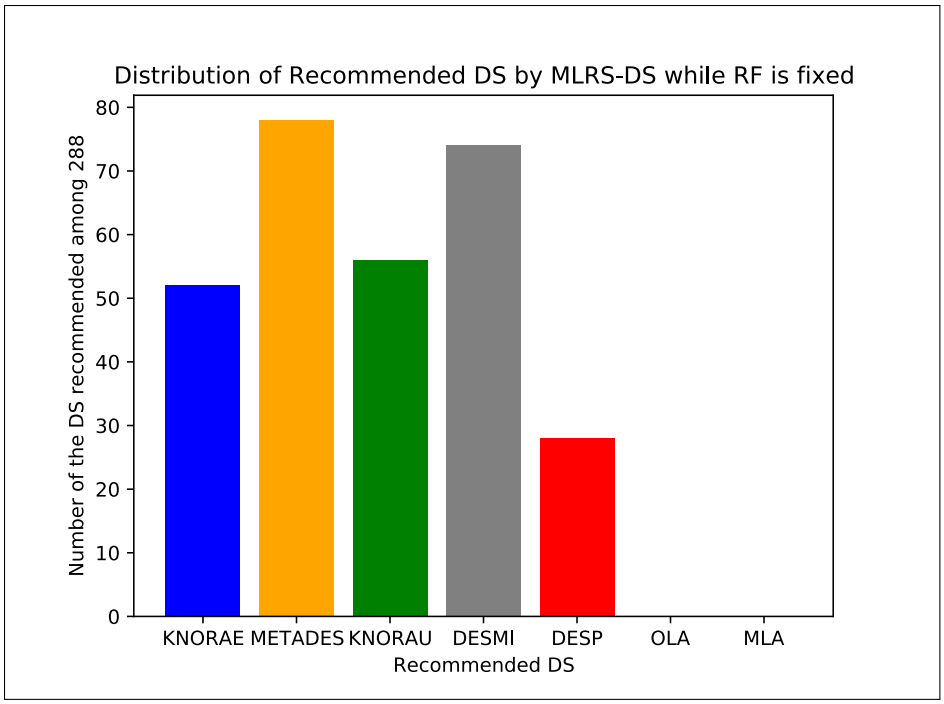


Figure 5.28 The distribution of the recommended DS method by MLRS-DS while RF is fixed

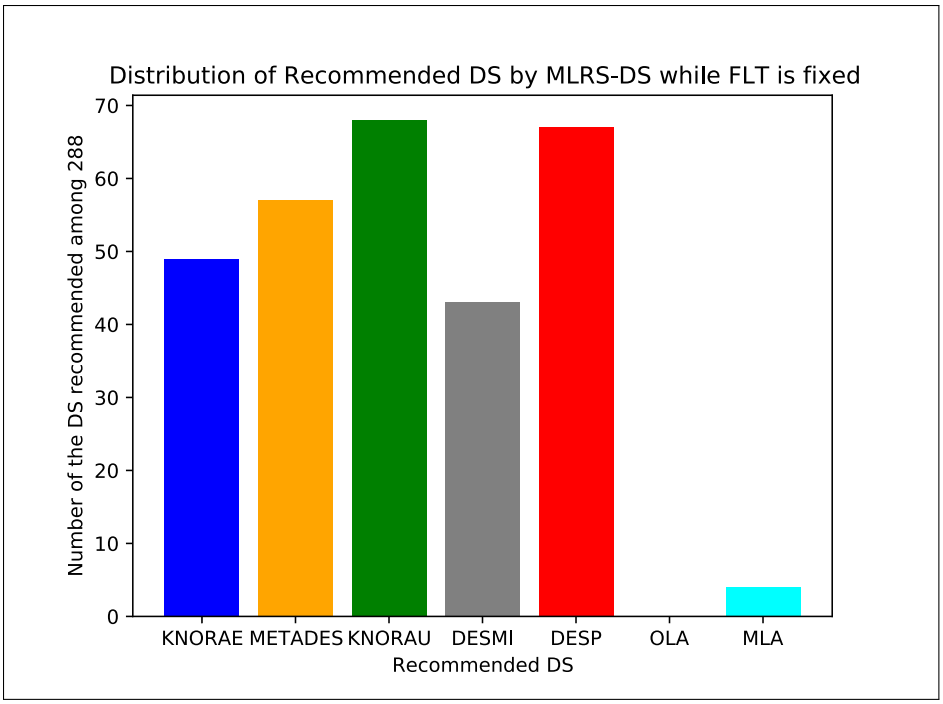


Figure 5.29 The distribution of the recommended DS method by MLRS-DS while FLT is fixed

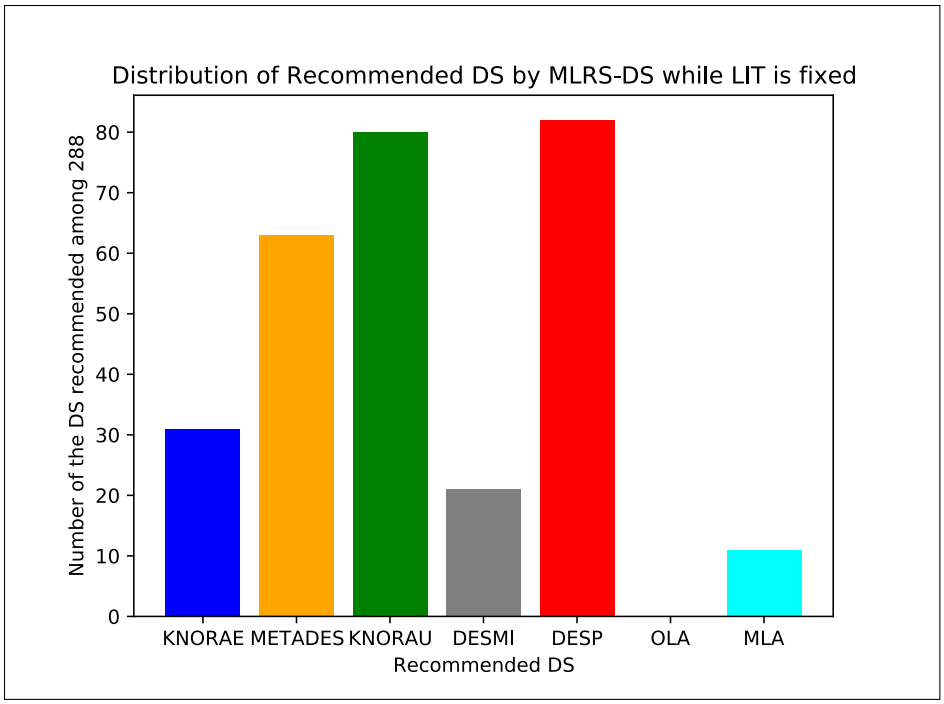


Figure 5.30 The distribution of the recommended DS method by MLRS-DS while LIT is fixed

The distributions of the recommended DS method by MLRS-DS indicate that MLRS-DS suggests META-DES most of the time while BDT is used. Moreover, META-DES remains the most recommended DS method when considering BSDT, BSP, and RF. The most suggestion was DES-MI while BP is used as a pool generation scheme. When FLT is used as a fixed pool generation scheme, the recommended DS is KNORA-U. Also, DES-P is the most suggestion while we are using LIT as a pool. However, even if META-DES is the most recommended for DS methods, we can see a diversity on this prediction, corroborating with the fact that the proposed MLRS-DS adapts its recommendation according to the dataset characteristics as well as the fixed pool of classifiers.

5.4.3 Scenario III: meta-learning for recommending the pool and DS algorithm

Scenario III is designed to recommend both the pool and DS method simultaneously and fully automated, based on the meta-features extracted from the test dataset **Q**. As mentioned

previously, meta-learning performs a chained recommendation where it first recommends the more suitable pool generation scheme according to the meta-feature and then recommends the DS method conditional to the first choice.

Similar to the previous scenarios, three algorithms, including Random Forests (RF), K Nearest Neighbors (KNN), and Support Vector Machine (SVM), were initially considered as meta-models for the MLRS-PDS. To compare the performance of the proposed MLRS-PDS, the Jaccard similarity coefficient score is used since it is commonly used in multi-label classification problems (Rinartha & Suryasa (2017)). It is a widely used metric to evaluate the similarity between two sets. It measures the intersection over the union of the sets and ranges from 0 to 1. Higher values indicate greater similarity. We utilized Jaccard similarity values scaled to a range of 0 to 100. This scaling facilitates a more intuitive interpretation, allowing us to represent the degree of similarity as a percentage.

The Jaccard similarity index mainly focuses on global ratings. It is the ratio of the proportion of the cardinality of co-rated items to the cardinality of all items rated by both users. The Jaccard similarity is calculated using Equation 5.2 where I_u and I_v are the sets of items rated by users u and v respectively.

$$\text{Jaccard}_{uv} = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} \tag{5.2}$$

As an example consider two lists of pools, u and v. The list u is $\{RF, BP, FLT, LIT\}$, and user v rated items $\{RF, BP, BSDT, BSP\}$. The Jaccard similarity index between u and v is calculated as follows:

$$\text{Jaccard}_{uv} = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} = \frac{|\{RF, BP\}|}{|\{RF, BP, FLT, LIT, BSDT, BSP\}|} = \frac{2}{6} = \frac{1}{3}$$

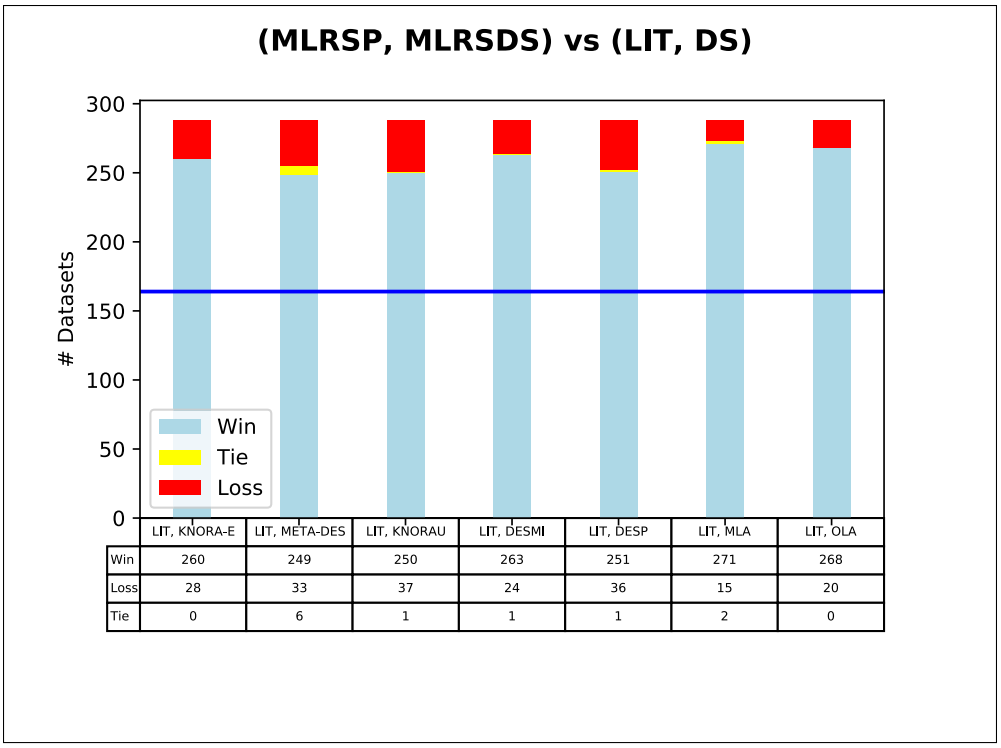
Table 5.7 presents the Jaccard similarity according to the different classifier algorithms used as meta-learners in this scenario. It can be observed that the KNN with K equal to 2 obtains a

higher score when used as the meta-model. Therefore, it was selected as our meta-classifier for the following analyses.

Table 5.7 The results of meta-learning prediction, MLRS-PDS with different hyper-parameters

Algorithm	Hyperparameters	(DS, Pool)
RF	Max_depth = 5	60.18
RF	Max_depth = 4	50.69
RF	Max_depth = 3	46.64
RF	Max_depth = 2	40.27
KNN	k = 2	**64.93**
KNN	k = 3	52.31
KNN	k = 4	49.07
KNN	k = 5	43.98
KNN	k = 6	42.82
SVM	gamma='scale', $c = 1$	33.79

A pairwise comparison using the Sign Test for the third phase of the research is conducted to compare MLRS-PDS against all possible pairs (7 pool generation scheme *times* 7 DS models) giving a total of 49 possible configurations of predetermined DS method and pool (PDS, PP). These results are presented in multiple figures (Figures 5.31 to 5.37), each one corresponding to all possible configurations regarding a particular pool generation method, used as pivot, due to the number of experiments.



	LIT, KNORA-E	LIT, META-DES	LIT, KNORAU	LIT, DESMI	LIT, DESP	LIT, MLA	LIT, OLA
Win	260	249	250	263	251	271	268
Loss	28	33	37	24	36	15	20
Tie	0	6	1	1	1	2	0

Figure 5.31 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use LIT as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

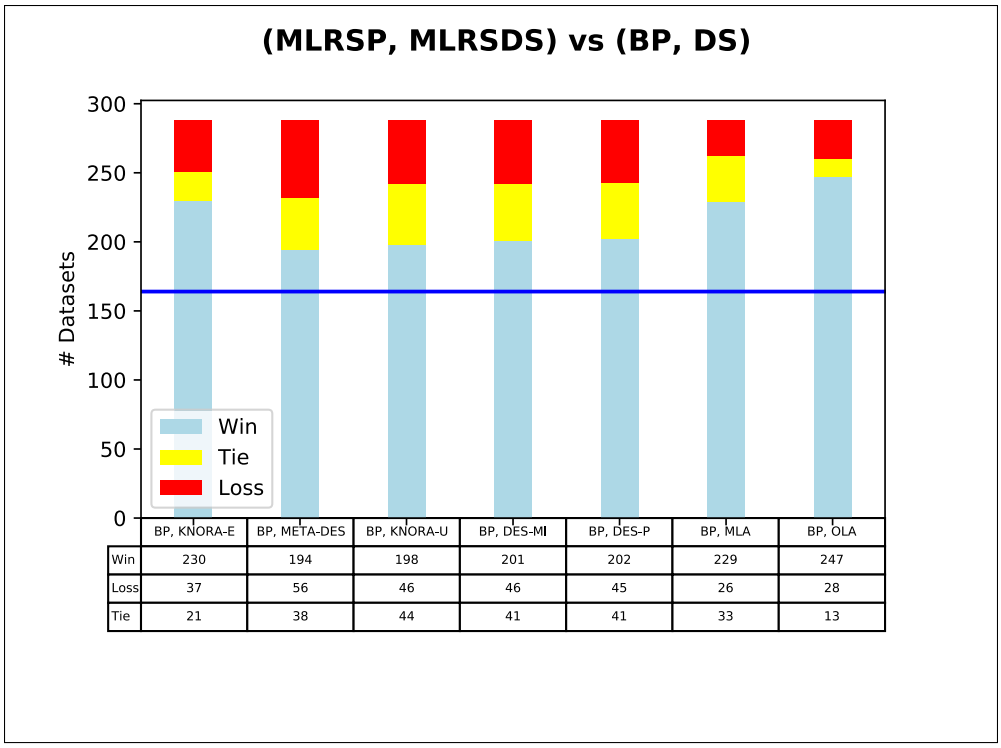


Figure 5.32 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use BP as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

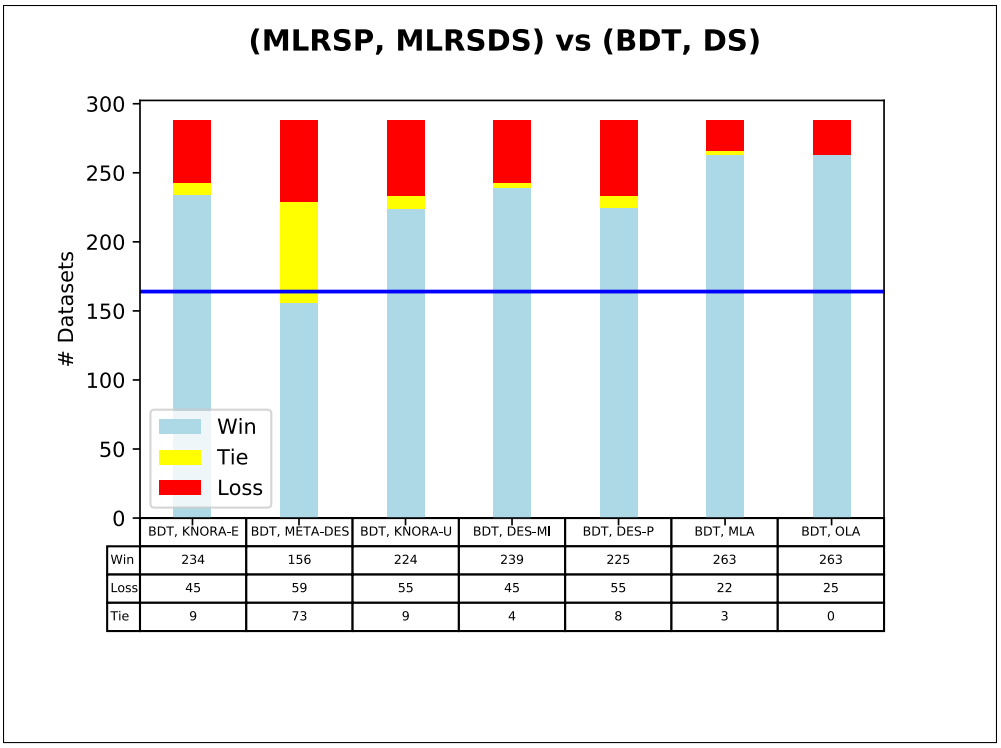


Figure 5.33 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use BDT as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

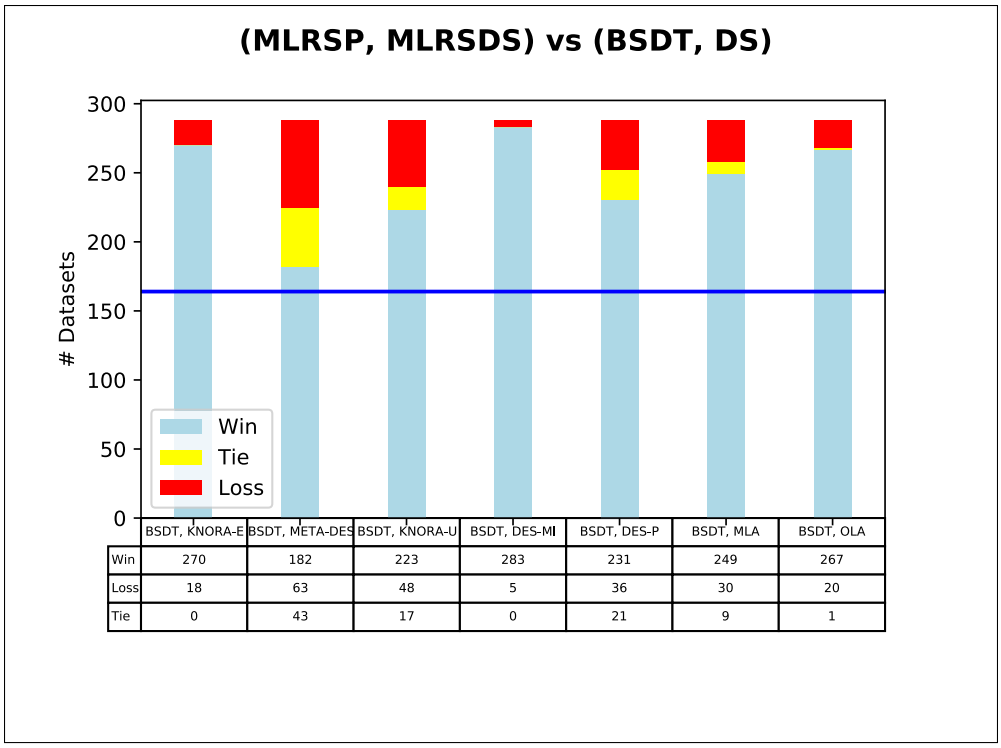


Figure 5.34 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use BSDT as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

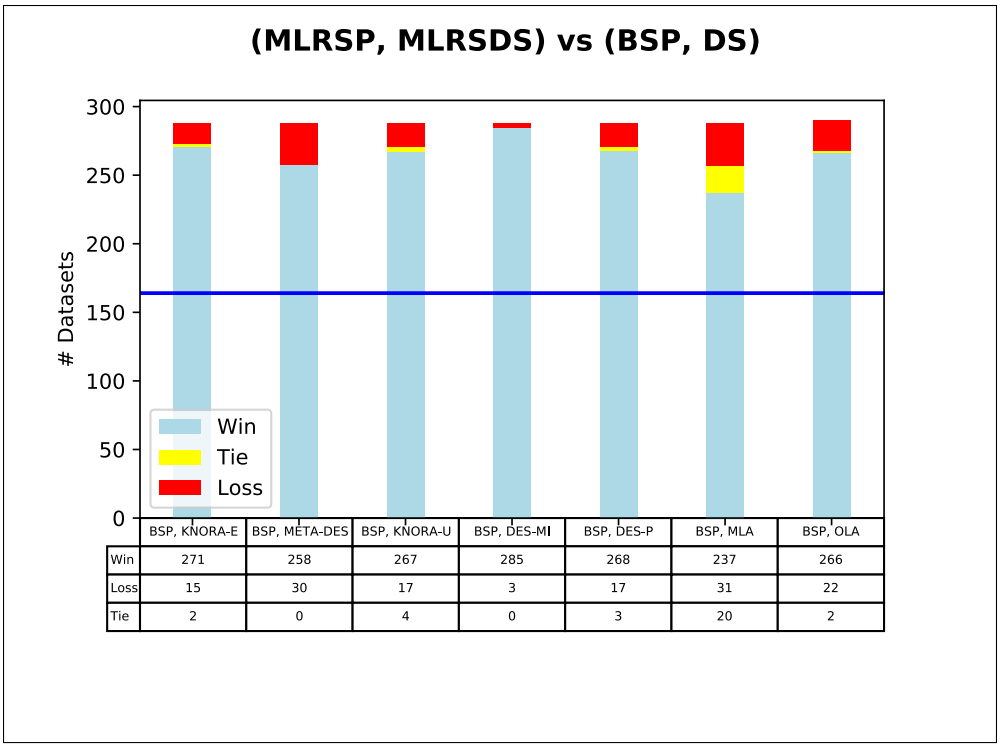
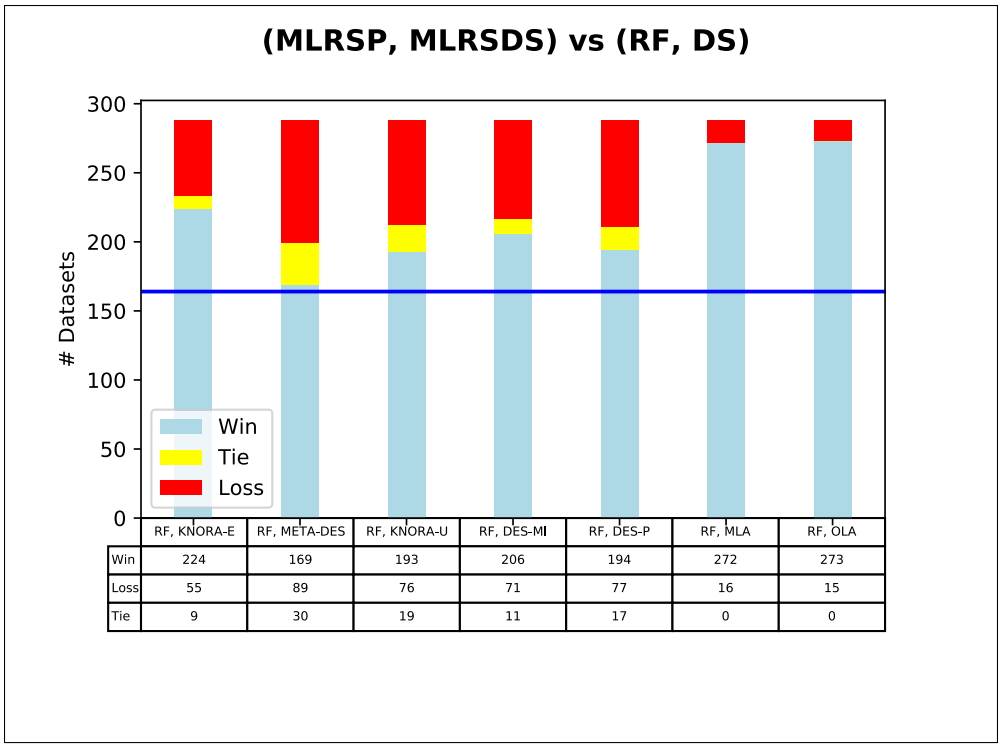


Figure 5.35 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use BSP as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)



	RF, KNORA-E	RF, META-DES	RF, KNORA-U	RF, DES-MI	RF, DES-P	RF, MLA	RF, OLA
Win	224	169	193	206	194	272	273
Loss	55	89	76	71	77	16	15
Tie	9	30	19	11	17	0	0

Figure 5.36 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use RF as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

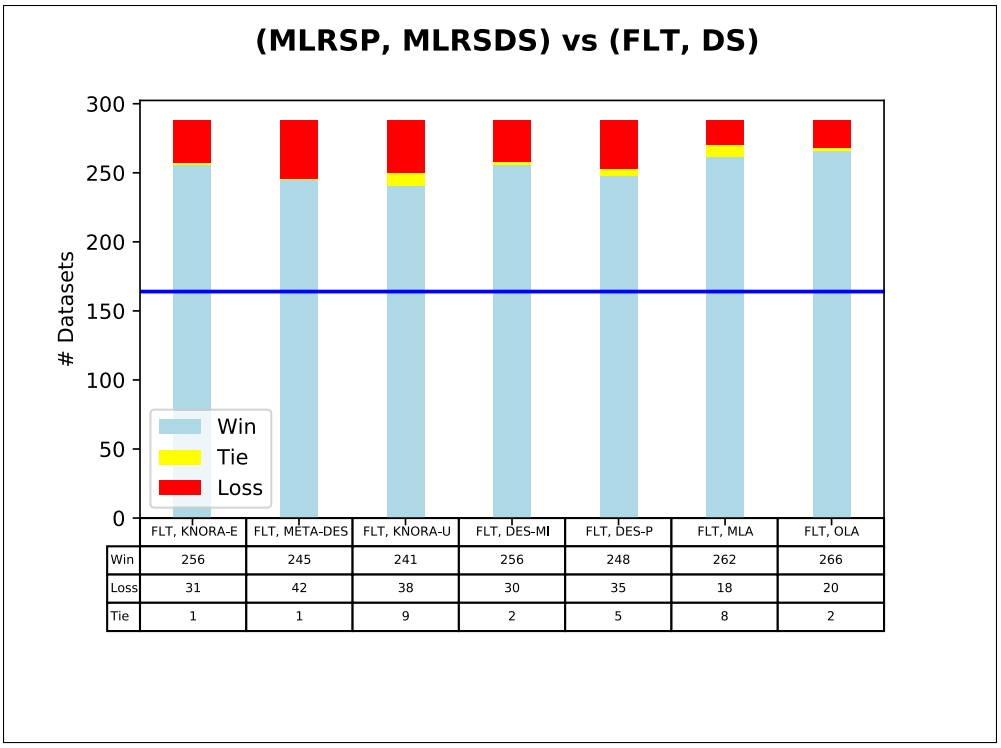


Figure 5.37 This figure illustrates the number of wins, ties and losses obtained by the MLRS-PDS against the baselines. In this case all pairs possible configurations that use FLT as the pool generation scheme is presented as baseline. The horizontal line marks the critical value threshold, which is determined based on a 95% confidence level ($\alpha = 0.05$)

Through the win-loss-tie diagrams presented in the figures as well as the summary of the Sign test presented in table 5.8, we can conclude that our fully automated solution, MLRS-PDS significantly outperforms all possible configurations. The chained recommendation scheme in MLRS-PDS that initially selects the optimal pool generation scheme (a simpler task than choosing the best DS, as shown in earlier subsections) and then recommending the most suitable DS model based on meta-features and the chosen pool, demonstrates robust performance.

Table 5.9 showcases a comprehensive comparison of MLRS-PDS against MLRS-P, MLRS-DS, and various baselines across 288 datasets. Each row in the table represents a specific (Pool, DS) pair, with the average number of first-rank results noted in parentheses. Notably, in the case of

H]

Table 5.8 Comparison of MLRS-PDS Performance Against Baseline Methods. "+" indicates MLRS-PDS wins, "=" denotes a tie, and "-" signifies a loss

	KNORA-E	META-DES	KNORA-U	DES-MI	DES-P	MLA	OLA
LIT	+	+	+	+	+	+	+
BP	+	+	+	+	+	+	+
BDT	+	+	+	+	+	+	+
BSDT	+	+	+	+	+	+	+
BSP	+	+	+	+	+	+	+
RF	+	+	+	+	+	+	+
FLT	+	+	+	+	+	+	+

MLRS-P for this analysis, the selection was based on the fixed META-DES, identified as the most effective dynamic selection scheme across all datasets. Similarly, for MLRS-DS, the RF model was chosen as the constant pool generation scheme due to its superior performance in accruing the highest number of wins among other methods. Additionally, the top 4 (Pool, DS) algorithm combinations, as identified in our previous analysis in Chapter 4, are included for reference.

Table 5.9 The comparison between the three versions of MLRS and the baselines among 288 datasets. Two recommended pairs of MLRS are presented here based on different meta-models in the third recommendation. (META-DES, MLRS-P) represents the pool recommended by MLRS-P while META-DES is fixed as the DS method. (MLRS-DS, RF) represents the DS method recommended by MLRS-DS while RF is fixed as a pool generation scheme.

Algorithm	Accuracy (wins)
MLRS-PDS	64.93(187)
MLRS-P with META-DES	10.06(29)
MLRS-DS with RF	27.08(78)
(RF, META-DES)	21.52(62)
(BP, DES-MI)	11.80(34)
(BP, META-DES)	10.06(29)
(BSDT, KNORA-U)	4.51(13)

Several conclusions can be drawn from these analyses. Firstly, when seeking the optimal solution, it is advantageous to opt for the multi-label formulation (MLRS-PDS) and predict both stages of the pipeline. The results clearly indicate that in a significant number of instances, either fixing the best pool (27.08%) or the best DS (10.06%) leads to a sub-optimal outcome, as it limits the search space to just one decision step. Therefore, to maximize performance, it is essential to model the entire process through meta-learning.

Secondly, even relying on robust, pre-existing pairs proves to be insufficient. In the best-case scenario of this analysis (RF, META-DES), achieving the optimal solution occurs in only about $\frac{1}{5}$ of the cases. This analysis underscores the potential of meta-learning in this field. This work demonstrates that meta-learning can be an effective approach for defining machine learning

solutions, particularly in complex scenarios like the one at hand, where there is significant interdependence between the components (in this case, the pool and DS method).

Moreover, we also explore the recommendations provided by MLRS-PDS in order to know which methods it is recommending. Figure 5.38 showcases the distribution of the recommended pools suggested by MLRS-PDS. In Figure 5.39, we shift the focus to the distribution of the recommended DS methods by MLRS-PDS.

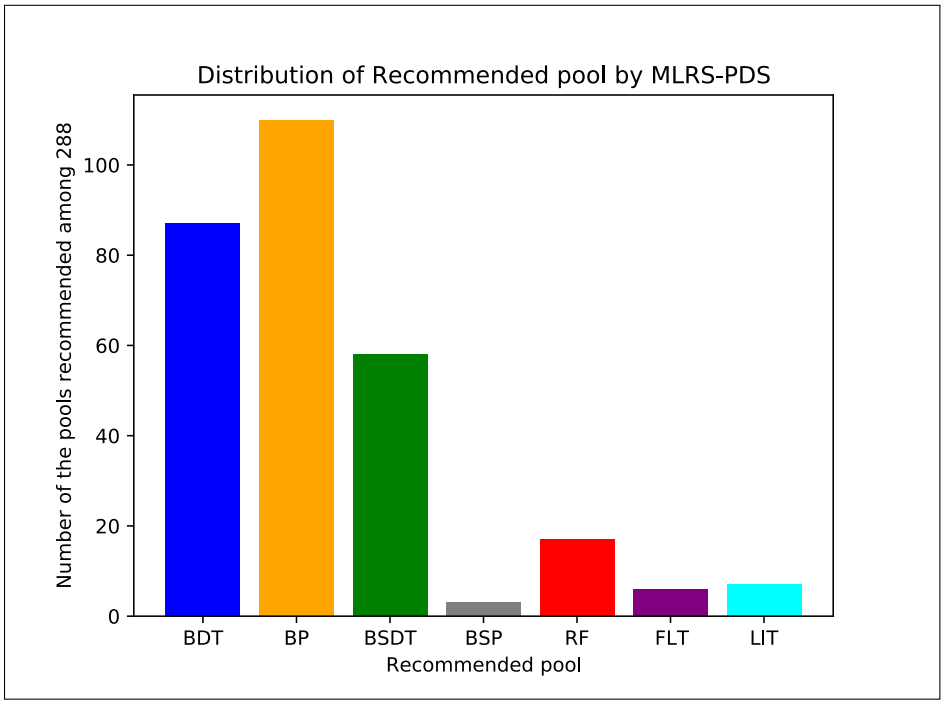


Figure 5.38 The distribution of the recommended pool by MLRS-PDS

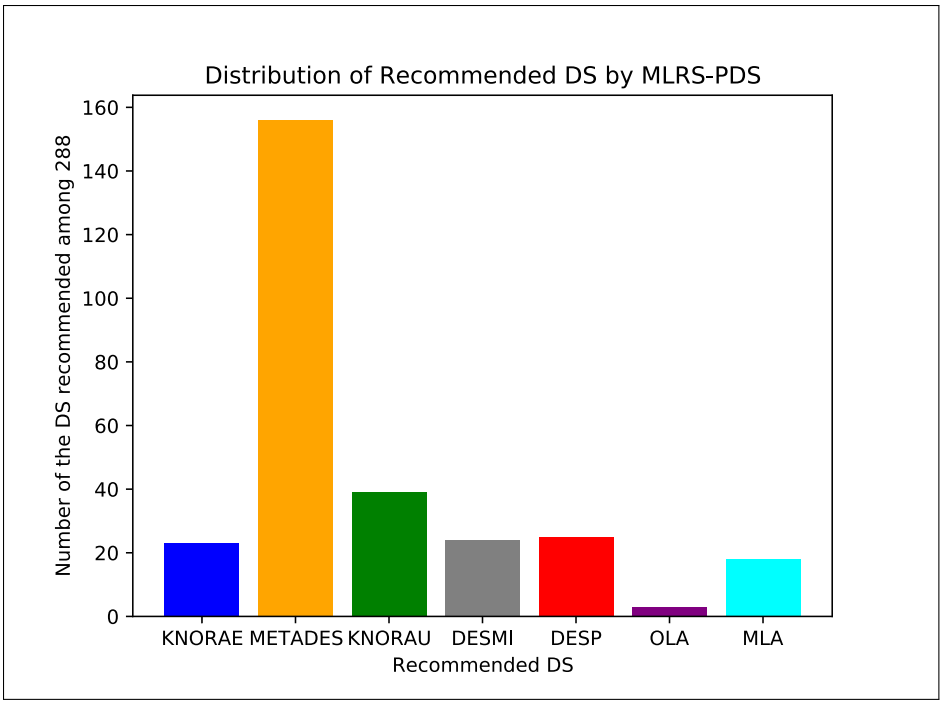


Figure 5.39 The distribution of the recommended DS method by MLRS-PDS

The distributions of the recommended pool by MLRS-PDS show that the most suggestion was BP while BDT and BSDT have also been recommended in considerable numbers in the second and third categories. The most of the time, the DS method recommended by MLRS-PDS is META-DES which shows the importance of this method. Nevertheless, we can see that there is a diversity in the recommendations, showing that our model indeed changes its recommendation based on the dataset characteristics.

5.5 RQ5: Dependency of selection of a pool generation scheme for DS techniques based on data characteristics

The last research questions from this work, RQ5, examines whether the choice of a pool generation scheme for DS methods depends on the characteristics of the datasets. The selection of the optimal pool generation scheme is determined based on the results gathered in tables 5.3 to 5.9 for Scenario I, revealing a dependence on data characteristics. Also table 5.2 reveals that

for all DS methods using the pool recommended by MLRS at least double the number of datasets that perform best. And since MLRS is based on data characteristics by extracting meta-features, it shows the selection of an optimized pool indeed depends on data characteristics, and also that we can model this relationship using meta-learning. Nevertheless, the analysis conducted in this chapter demonstrate that this relationship is more complex than just taking into account the dataset characteristics, and the choice of the best pool generation scheme or DS algorithm also depends on each other.

5.5.1 Conclusion

A novel meta-learning framework for DS methods is presented in this chapter. The meta-learning recommendation system consists of two distinct processes, namely meta-training and generalization. During the meta-training phase, meta-features are extracted from the training datasets. Then A meta-model is constructed by using the meta-target, y', which is the optimal performance obtained from a set of classifiers. During the generalization phase, the meta-features of the test dataset, are extracted, and the meta-model is utilized to recommend the best configuration.

In this chapter, we propose three different scenarios for the meta-learning reecommendation: the first, MLRS-P suggest a pool generation scheme while using a fixed DS method. This type of recommendation is useful when the user already have a particular DS they want to use, but want to obtain its best possible performance. The second, MLRS-DS the meta-learning recommendation algorithm suggests the DS method while the pool is fixed. It solves the use case when a user already have a pool model pre-trained and want to identify which DS technique would be more likely to obtain the best performance. Lastly we propose MLRS-PDS that provides the pair (Pool, DS) just by looking at the meta-features. Hence, fully automating the design decisions.

A total of 288 datasets were used in experiments to evaluate the effectiveness of our proposal to the baseline methods. The number of times the meta-learning recommendation system

predicts the correct recommendation among 288 datasets is more than the number of times a fixed pool generation scheme performs well. Similar observations were made for the second and third recommendations. Hence, the experimental analysis indicates that the proposed meta-learning recommendation system performs better in all three recommendation scenarios, obtaining statistically significance performance. Furthermore, since this analysis was conducted considering a diverse set of datasets that was particularly created for filling the whole complexity space of problems, we believe the proposed approach generalize well to datasets that were not considered in this work. As such, we believe the proposed MLRS recommendation framework, in its three versions, is a significant contribution for the dynamic selection community. Furthermore, the code of MLRS is in the process to be integrated in the next versions of DESlib, to assist future researchers in this topic.

CONCLUSION AND RECOMMENDATIONS

In this project, we have explored the impact of the local and global pool generation schemes on the performance of DS algorithms. We have examined the importance of selecting an appropriate pool of classifiers for DS algorithms and how it can impact their performance. Most publications used global perspective pool generation schemes, and those techniques have shown some limitations, such as instability and generating redundant classifiers for global perspectives. So we considered local pool generation schemes for the DS algorithm as well. Throughout the process, we conducted an analysis that compared the use of local and global pool generation schemes for DS algorithms on the same datasets and the same experimental protocols to observe which pool generation scheme would perform better when used as input of DS algorithms.

In Chapter 4, a comparative study has shown that The local pool generation schemes did not yield better results compared to the global perspective methods. The results show that no pool generation scheme outperforms others. Also, the selection of a pool generation scheme may differ according to the dataset and depends on the specific DS technique employed.

The results obtained in the first analysis indicate that we need to take into account the dataset characteristics in order to know which pool generation scheme should be used. To reach this goal, we propose a meta-learning recommendation technique for solving the aforementioned problem in chapter 5. The results illustrate that by using a meta-learning recommendation system, the selection of the pool generation scheme depends on data characteristics. We proposed a novel automated recommendation system by constructing a meta-model using the meta-features and the meta-target to predict a proper pool generation scheme for DS techniques, the best DS method, and a pair of recommendations of (DS, Pool) for a given dataset.

We conducted another experimental study to evaluate the performance of the Meta-learning recommendation system. As the first recommendation of the research, the meta-learning recommendation system suggests a pool of classifiers while a DS method is fixed. The

experimental study shows that the meta-learning recommendation system performs better than selecting the best pool of classifiers from the study in chapter 4 across 288 datasets. As the second recommendation of the research, the meta-learning recommendation system suggests a DS method, while a pool of classifiers is fixed. The experimental study has shown that the meta-learning recommendation system performs better than selecting a DS technique based on the study in chapter 4. As the third recommendation, the meta-learning recommendation system suggests the best pair of (DS, Pool) for a given dataset. The experimental study has shown that the meta-learning recommendation system performs better than selecting a DS technique and a pool of classifiers based on the study in chapter 4.

The experimental study conducted over 288 datasets demonstrates that, on average, the meta-learning recommendation improves over the baseline for DS algorithms. This project contributes to the field of dynamic ensemble selection by empirically demonstrating the impact of the pool generation method on the model's performance and its dependency on the problem characteristics. In addition, the meta-learning recommendation of the best pool generation and DS algorithm according to the problem's characteristics automate the process of finding a suitable dynamic selection pipeline for classification problems.

While this study has provided valuable insights into automated recommendations for pool generation schemes and selecting a DS method for a given data, there remain several intriguing opportunities for future research and enhancements. Here are some future works in this area:

- Creating a novel pool generation scheme taking into account DS models and their competence estimation in the process.
- Expanding the scope of our analysis to encompass a broader range of classification problems stands as a promising avenue for future research.
- Contributing Meta-learning recommendation system to the DESlib library.

- The evaluation of the developed multi-label meta-learning recommendation system for other types of tasks. These include regression problems and time-series forecasting.

APPENDIX I

LIST OF META-FEATURES USED IN THE PROPOSED APPROACH

As mentioned earlier, this work considered an extensive and diverse set of meta-features, coming from different groups in the conception of the meta-learning model. The list describing all meta-features considered in this work is presented in Tables A.I-1 to A I-4.

Table-A I-1 Meta-features present in the meta-dataset

	Meta feature	Group	Description
1	attr_to_inst	general	Compute the ratio between the number of attributes.
2	cat_to_num	general	Compute the ratio between the number of categoric and numeric features.
3	freq_class	general	Compute the relative frequency of each distinct class.
4	inst_to_attr	general	Compute the ratio between the number of instances and attributes.
5	nr_attr	general	Compute the total number of attributes.
6	nr_bin	general	Compute the number of binary attributes.
7	nr_cat	general	Compute the number of categorical attributes.
8	nr_class	general	Compute the number of distinct classes.
9	nr_inst	general	Compute the number of instances (rows) in the dataset.
10	nr_num	general	Compute the number of numeric features.
11	num_to_cat	general	Compute the number of numerical and categorical features.
12	precompute_general_class	general	Precompute distinct classes and its frequencies from y.
13	can_cor	Statistical	Compute the canonical correlations of data.
14	cor	Statistical	Compute the absolute value of the correlation of distinct dataset column pairs.
15	cov	Statistical	Compute the absolute value of the covariance of distinct dataset attribute pairs.
16	eigenvalues	Statistical	Compute the eigenvalues of covariance matrix from dataset.
17	g_mean	Statistical	Compute the geometric mean of each attribute.
18	gravity	Statistical	Compute the distance between minority and majority classes center of mass.
19	h_mean	Statistical	Compute the harmonic mean of each attribute.
20	iq_rang	Statistical	Compute the interquartile range (IQR) of each attribute.
21	kurtosis	Statistical	Compute the kurtosis of each attribute.
22	lh_trace	Statistical	Compute the Lawley-Hotelling trace.
23	mad	Statistical	Compute the Median Absolute Deviation (MAD) adjusted by a factor.
24	max	Statistical	Compute the maximum value from each attribute.
25	mean	Statistical	Compute the mean value of each attribute.
26	median	Statistical	Compute the median value from each attribute.
27	min	Statistical	Compute the minimum value from each attribute.
28	nr_cor_attr	Statistical	Compute the number of distinct highly correlated pairs of attributes.
29	nr_disc	Statistical	Compute the number of canonical correlations between each attribute and class.
30	nr_norm	Statistical	Compute the number of attributes normally distributed based in a given method.
31	nr_outliers	Statistical	Compute the number of attributes with at least one outlier value.
32	p_trace	Statistical	Compute the Pillai's trace.
33	range	Statistical	Compute the range (max-min) of each attribute.
34	roy_root	Statistical	Compute the Roy's largest root.
35	sd	Statistical	Compute the standard deviation of each attribute.
36	sd_ratio	Statistical	Compute a statistical test for homogeneity of covariances.
37	skewness	Statistical	Compute the skewness for each attribute.
38	sparsity	Statistical	Compute (possibly normalized) sparsity metric for each attribute.
39	t_mean	Statistical	Compute the trimmed mean of each attribute.
40	var	Statistical	Compute the variance of each attribute.

Table-A I-2 Meta-features present in the meta-dataset

	Meta feature	Group	Description
41	w_lambda	Statistical	Compute the Wilks' Lambda value.
42	precompute_can_cors	Statistical	Precompute canonical correlations and its eigenvalues.
43	precompute_statistical_class	Statistical	Precompute distinct classes and its abs.
44	precompute_statistical_cor_cov	Statistical	Precomputes the correlation and covariance matrix of numerical data.
45	attr_conc	Information theory	Compute concentration coef.
46	attr_ent	Information theory	Compute Shannon's entropy for each predictive attribute.
47	class_conc	Information theory	Compute concentration coefficient between each attribute and class.
48	class_ent	Information theory	Compute target attribute Shannon's entropy.
49	eq_num_attr	Information theory	Compute the number of attributes equivalent for a predictive task.
50	joint_ent	Information theory	Compute the joint entropy between each attribute and class.
51	mut_inf	Information theory	Compute the mutual information between each attribute and target.
52	ns_ratio	Information theory	Compute the noisiness of attributes.
53	precompute_class_freq	Information theory	Precompute each distinct class (absolute) frequencies.
54	precompute_entropy	Information theory	Precompute various values related to Shannon's Entropy.
55	extract_table	Model-based	Bookkeep some information table from the t_model into an array.
56	leaves	Model-based	Compute the number of leaf nodes in the DT model.
57	leaves_branch	Model-based	Compute the size of branches in the DT model.
58	leaves_corrob	Model-based	Compute the leaves corroboration of the DT model.
59	leaves_homo	Model-based	Compute the DT model Homogeneity for every leaf node.
60	leaves_per_class	Model-based	Compute the proportion of leaves per class in DT model.
61	nodes	Model-based	Compute the number of non-leaf nodes in DT model.
62	nodes_per_attr	Model-based	Compute the ratio of nodes per number of attributes in DT model.
63	nodes_per_inst	Model-based	Compute the ratio of non-leaf nodes per number of instances in DT model.
64	nodes_per_level	Model-based	Compute the ratio of number of nodes per tree level in DT model.
65	nodes_repeated	Model-based	Compute the number of repeated nodes in DT model.
66	tree_depth	Model-based	Compute the depth of every node in the DT model.
67	tree_imbalance	Model-based	Compute the tree imbalance for each leaf node.
68	tree_shape	Model-based	Compute the tree shape for every leaf node.
69	var_importance	Model-based	Compute the features importance of the DT model for each attribute.
70	precompute_model_based_class	Model-based	Precompute the DT Model and some information related to it.
71	best_node	Landmarking	Performance of the best single decision tree node.
72	elite_nn	Landmarking	Performance of Elite Nearest Neighbor.
73	linear_discr	Landmarking	Performance of the Linear Discriminant classifier.
74	naive_bayes	Landmarking	Performance of the Naive Bayes classifier.
75	one_nn	Landmarking	Performance of the 1-Nearest Neighbor classifier.
76	random_node	Landmarking	Performance of the single DT node model induced by a random attribute.
77	worst_node	Landmarking	Performance of the single DT node model induced by the worst informative attribute.
78	precompute_landmarking_kfolds	Landmarking	Precompute k-fold cross-validation related values.
79	precompute_landmarking_sample	Landmarking	Precompute subsampling landmarking subsample indices.
80	group_mtf_by_summary	Relative Landmarking	Group meta features by its correspondent summary method.

Table-A I-3 Meta-features present in the meta-dataset

	Meta feature	Group	Description
81	postprocess_landmarking_relative	Relative Landmarking	Generate Relative Landmarking from Landmarking metafeatures.
82	ch	Clustering	Compute the Calinski and Harabasz index.
83	int	Clustering	Compute the INT index.
84	nre	Clustering	Compute the normalized relative entropy.
85	pb	Clustering	Compute the Pearson correlation between class matching and instance distances.
86	sc	Clustering	Compute the number of clusters with sizes smaller than a given size.
87	sil	Clustering	Compute the mean silhouette value.
88	vdb	Clustering	Compute the Davies and Bouldin Index.
89	vdu	Clustering	Compute the Dunn Index.
90	precompute_class_representatives	Clustering	Precomputations related to cluster representative instances.
91	precompute_clustering_class	Clustering	Precompute distinct classes and its frequencies from y.
92	precompute_group_distances	Clustering	Precompute distance metrics between instances.
93	precompute_nearest_neighbors	Clustering	Precompute the n_neighbors Nearest Neighbors of every instance.
94	cohesiveness	Concept	Improved weighted distance that captures how dense or sparse is the example distribution.
95	conceptvar	Concept	Compute the concept variation that estimates the variability of class labels among examples.
96	impconceptvar	Concept	Compute the improved concept variation that estimates the variability of class labels among examples.
97	wg_dist	Concept	Compute the weighted distance and that captures how dense or sparse is the example distribution.
98	precompute_concept_dist	Concept	Precompute some useful things to support complexity measures.
99	one_itemset	Itemset	Compute the one itemset meta-feature.
100	two_itemset	Itemset	Compute the two itemset meta-feature.
101	precompute_binary_matrix	Itemset	Precompute the binary representation of attributes.
102	c1	Complexity	Compute the entropy of class proportions.
103	c2	Complexity	Compute the imbalance ratio.
104	cls_coef	Complexity	Clustering coefficient.
105	density	Complexity	Average density of the network.
106	f1	Complexity	Maximum Fisher's discriminant ratio.
107	f1v	Complexity	Directional-vector maximum Fisher's discriminant ratio.
108	f2	Complexity	Volume of the overlapping region.
109	f3	Complexity	Compute feature maximum individual efficiency.
110	f4	Complexity	Compute the collective feature efficiency.
111	hubs	Complexity	Hub score.
112	l1	Complexity	Sum of error distance by linear programming.
113	l2	Complexity	Compute the OVO subsets error rate of a linear classifier.
114	l3	Complexity	Non-Linearity of a linear classifier.
115	lsc	Complexity	Local set average cardinality.
116	n1	Complexity	Compute the fraction of borderline points.
117	n2	Complexity	Ratio of intra and extra class nearest neighbor distance.
118	n3	Complexity	Error rate of the nearest neighbor classifier.
119	n4	Complexity	Compute the non-linearity of the k-NN Classifier.
120	t1	Complexity	Fraction of hyperspheres covering data.

Table-A I-4 Meta-features present in the meta-dataset

	Meta feature	Group	Description
121	t2	Complexity	Compute the average number of features per dimension.
122	t3	Complexity	Compute the average number of PCA dimensions per point.
123	t4	Complexity	Compute the ratio of the PCA dimension to the original dimension.
124	precompute_adjacency_graph	Complexity	Calculate values associated with the nearest neighboring instances.
125	precompute_complexity	Complexity	Precompute some useful things to support feature-based measures.
126	precompute_complexity_svm	Complexity	Init a Support Vector Classifier pipeline (with data standardization.)
127	precompute_nearest_enemy	Complexity	Precompute instances nearest enemy related values.
128	precompute_norm_dist_mat	Complexity	Precompute normalized n and pairwise distance among instances.
129	precompute_pca_tx	Complexity	Precompute PCA to support dimensionality measures.

APPENDIX II

EXPERIMENTAL RESULTS OF CHAPTER4

Table II-1 provides the average ranking across 288 datasets for various combinations of 7 pool generation schemes and 7 dynamic selection (DS) methods. Each cell in the table represents the average rank of a specific pool-DS method combination. For instance, FLT achieved average ranks of 26.96 (KNORA-E), 21.40 (META-DES), 21.43 (KNORA-U), 27.54 (DES-MI), 22.89 (DES-P), 31.69 (MLA), and 33.82 (OLA). These statistics for LIT are 28.19(KNORA-E), 21.84(META-DES), 21.73(KNORA-U), 28.76(DES-MI), 22.68(DES-P), 31.84(MLA), 33.35(OLA). The best average ranking belongs to (META-DES, RF) with an average rank of 6.12, while the worst belongs to (DES-MI, BSDT) with an average rank of 47.23.

Table-A II-1 Summary of the average rank among 288 datasets between a combination of 7 pools and 7 DS methods

(META-DES, RF)	6.12	(KNORA-E, FLT)	26.96
(KNORA-U, RF)	8.67	(DESP, BSDT)	27.18
(DES-P, RF)	9.50	(MLA, BSDT)	27.43
(META-DES, BDT)	9.56	(DES-MI, FLT)	27.54
(DES-MI, RF)	10.12	(MLA, BP)	27.96
(META-DES, BSDT)	11.53	(KNORA-E, LIT)	28.19
(KNORA-U, BDT)	12.55	(MLA, BSP)	28.32
(DES-P, BDT)	13.07	(DES-MI, LIT)	28.76
(META-DES, BP)	14.64	(KNORA-U, BSP)	30.82
(KNORA-E, RF)	14.92	(OLA, BDT)	30.89
(DES-MI, BDT)	16.08	(MLA, FLT)	31.69
(KNORA-E, BDT)	17.58	(DES-P, BSP)	31.74
(DES-MI, BP)	19.07	(MLA, LIT)	31.84
(KNORA-U, BP)	19.70	(OLA, BSP)	32.02
(DES-P, BP)	19.77	(OLA, BSDT)	32.08
(KNORA-E, BP)	20.32	(MLA, BDT)	32.70
(KNORA-U, BSDT)	20.45	(KNORA-E, BSDT)	33.08
(META-DES, FLT)	21.40	(OLA, LIT)	33.35
(KNORA-U, FLT)	21.43	(OLA, FLT)	33.82
(KNORA-U, LIT)	21.73	(KNORA-E, BSP)	35.57
(META-DES, LIT)	21.84	(OLA, RF)	36.70
(DES-P, LIT)	22.68	(MLA, RF)	38.04
(DESP, FLT)	22.89	(DES-MI, BSP)	46.86
(OLA, BP)	23.61	(DES-MI, BSDT)	47.23
(META-DES, BSP)	26.01		

Table II-2 illustrates the average rank of each pool among 288 datasets including the results of majority voting. (META-DES, RF) performed better between 56 pairs of (DS, Pool) and majority voting results. The base classifiers of the top 4 of this ranking are RF which shows the importance of this method. Furthermore, the top 10 pools of classifiers are composed of DT.

Table-A II-2 Average rank of each pools

RF_METADES	7.343	BP_OLA	27.631
RF_KNORAU	9.993	LIT_MV	28.256
RF_DESP	10.913	BSP_METADES	30.434
RF_MV	11.500	FLT_KNORAE	31.489
BDT_METADES	11.517	BSDT_DESP	31.649
RF_DESMI	12.101	BSDT_MLA	32.156
BSDT_METADES	13.781	FLT_DESMI	32.243
BDT_KNORAU	14.736	BP_MLA	32.663
BDT_DESP	15.222	LIT_KNORAE	32.875
BDT_MV	15.725	BSP_MLA	33.097
BP_METADES	17.104	LIT_DESMI	33.576
RF_KNORAE	17.920	BDT_OLA	35.753
BDT_DESMI	19.072	BSP_KNORAU	35.916
BSDT_MV	20.503	BSP_MV	36.583
BDT_KNORAE	20.979	FLT_MLA	36.958
BP_DESMI	21.916	BSP_OLA	37.079
BP_KNORAU	22.423	BSP_DESP	37.083
BP_DESP	22.687	LIT_MLA	37.152
BP_KNORAE	23.913	BSDT_OLA	37.239
BSDT_KNORAU	24.024	BDT_MLA	37.993
BP_MV	24.184	BSDT_KNORAE	38.031
FLT_METADES	25.246	LIT_OLA	38.607
FLT_KNORAU	25.333	FLT_OLA	39.152
LIT_KNORAU	25.659	BSP_KNORAE	40.743
LIT_METADES	25.809	RF_OLA	42.093
LIT_DESP	26.847	RF_MLA	43.854
FLT_MV	26.902	BSP_DESMI	53.559
FLT_DESP	27.052	BSDT_DESMI	53.836

1. Comparison of a local and global pool of classifiers performance

A major question in this study is whether local pool generation schemes yield better results in comparison with global pool generation schemes. The answer to this research question would lead us to use just the global pool of classifiers, just the local pool of classifiers, C, or the solution

would be a combination of those to optimize the performance of DS methods. Table II-3 and **??** shows the average rank of FLT and LIT for different DS methods.

Table-A II-3 Average rank of FLT on 7 DS methods among 288 datasets

FLT_METADES	2.725
FLT_KNORAU	2.826
FLT_DESP	3.101
FLT_KNORAE	4.048
FLT_DESMI	4.190
FLT_MLA	5.093
FLT_OLA	5.569

Table-A II-4 Average rank of LIT on 7 DS methods among 288 datasets

LIT_METADES	2.642
LIT_KNORAU	2.701
LIT_DESP	3.003
LIT_KNORAE	4.246
LIT_DESMI	4.430
LIT_MLA	5.097
LIT_OLA	5.607

Table II-5 shows the numbers of wins (1st rank) for each DS method among 288 datasets. For example for METADES, 109 times RF reached the 1st rank among other pools of classifiers among 288 datasets.

Table-A II-5 the number of the 1st rank for each DS method and MV among 288 datasets

	KNORAE	MERADES	KNORAU	DESMI	DESP	MLA	OLA
LIT	6	5	5	7	5	14	27
BP	89	68	64	87	69	77	139
BDT	101	57	84	75	90	65	64
BSDT	6	60	37	0	34	74	16
BSP	2	8	1	0	2	67	17
RF	87	109	132	121	126	16	12
FLT	10	7	11	10	11	24	24

2. Average rank of the pools for each DS method

Figures II-1a to II-1g show the average rank of each pool generation scheme for each DS method.

a) KNORAE

RF_KNORAE	2.059
BDT_KNORAE	2.447
BP_KNORAE	3.034
FLT_KNORAE	4.322
LIT_KNORAE	4.524
BSDT_KNORAE	5.420
BSP_KNORAE	5.927

c) KNORAU

RF_KNORAU	1.958
BDT_KNORAU	2.659
BP_KNORAU	3.958
BSDT_KNORAU	4.229
FLT_KNORAU	4.357
LIT_KNORAU	4.517
BSP_KNORAU	5.850

b) METADES

RF_METADES	2.135
BDT_METADES	2.795
BSDT_METADES	3.281
BP_METADES	3.385
FLT_METADES	5.173
LIT_METADES	5.347
BSP_METADES	5.538

d) DESMI

RF_DESMI	1.798
BDT_DESMI	2.440
BP_DESMI	3.024
FLT_DESMI	3.857
LIT_DESMI	3.982
BSP_DESMI	6.045
BSDT_DESMI	6.708

e) DESP

RF_DESP	1.975
BDT_DESP	2.572
BP_DESP	3.607
LIT_DESP	4.430
FLT_DESP	4.458
BSDT_DESP	4.881
BSP_DESP	5.586

f) MLA

BP_MLA	3.243
BSDT_MLA	3.298
BSP_MLA	3.406
BDT_MLA	3.843
LIT_MLA	4.100
FLT_MLA	4.121
RF_MLA	5.486

g) OLA

BP_OLA	2.586
BDT_OLA	3.684
BSDT_OLA	3.864
LIT_OLA	3.979
BSP_OLA	4.142
FLT_OLA	4.190
RF_OLA	5.236

LIST OF REFERENCES

Aguiar, G. J., Santana, E. J., de Carvalho, A. C. & Junior, S. B. (2022). Using meta-learning for multi-target regression. *Information Sciences*, 584, 665–684.

Alcobaça, E., Siqueira, F., Rivolli, A., Garcia, L. P., Oliva, J. T. & De Carvalho, A. C. (2020). MFE: Towards reproducible meta-feature extraction. *The Journal of Machine Learning Research*, 21(1), 4503–4507.

Almeida, P. R., Oliveira, L. S., Britto Jr, A. S. & Sabourin, R. (2018). Adapting dynamic classifier selection for concept drift. *Expert Systems with Applications*, 104, 67–85.

Armano, G. & Tamponi, E. (2018). Building forests of local trees. *Pattern Recognition*, 76, 380–390.

Bashbaghi, S., Granger, E., Sabourin, R. & Bilodeau, G.-A. (2017). Dynamic ensembles of exemplar-SVMs for still-to-video face recognition. *Pattern recognition*, 69, 61–81.

Batista, L., Granger, E. & Sabourin, R. (2012). Dynamic selection of generative–discriminative ensembles for off-line signature verification. *Pattern Recognition*, 45(4), 1326–1340.

Beigy, H. et al. (2009). Dynamic classifier selection using clustering for spam detection. *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 84–88.

Biau, G. (2012). Analysis of a random forests model. *The Journal of Machine Learning Research*, 13(1), 1063–1095.

Bilalli, B., Abelló, A., Aluja-Banet, T. & Wrembel, R. (2016). Automated data pre-processing via meta-learning. *Model and Data Engineering: 6th International Conference, MEDI 2016, Almería, Spain, September 21-23, 2016, Proceedings 6*, pp. 194–208.

Bilalli, B., Abelló Gamazo, A. & Aluja Banet, T. (2017). On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4), 697–712.

Bischl, B., Mersmann, O., Trautmann, H. & Weihs, C. (2012). Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation*, 20(2), 249–275.

Brazdil, P., Carrier, C. G., Soares, C. & Vilalta, R. (2008). *Metalearning: Applications to data mining*. Springer Science & Business Media.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5–32.

Britto Jr, A. S., Sabourin, R. & Oliveira, L. E. (2014). Dynamic selection of classifiers—a comprehensive review. *Pattern recognition*, 47(11), 3665–3680.

Brun, A. L., Britto, A. S., Oliveira, L. S., Enembreck, F. & Sabourin, R. (2016). Contribution of data complexity features on dynamic classifier selection. *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4396–4403.

Cavalin, P. R., Sabourin, R. & Suen, C. Y. (2012). LoGID: An adaptive framework combining local and global incremental learning for dynamic selection of ensembles of HMMs. *Pattern recognition*, 45(9), 3544–3556.

Cavalin, P. R., Sabourin, R. & Suen, C. Y. (2013). Dynamic selection approaches for multiple classifier systems. *Neural computing and applications*, 22(3), 673–688.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

Choi, Y.-R. & Lim, D.-J. (2021). DDES: A Distribution-Based Dynamic Ensemble Selection Framework. *IEEE Access*, 9, 40743–40754.

Cruz, R. M., Cavalcanti, G. D. & Ren, T. I. (2011). A method for dynamic ensemble selection based on a filter and an adaptive distance to improve the quality of the regions of competence. *The 2011 International Joint Conference on Neural Networks*, pp. 1126–1133.

Cruz, R. M., Cavalcanti, G. D., Tsang, R. & Sabourin, R. (2013). Feature representation selection based on classifier projection space and oracle analysis. *Expert Systems with Applications*, 40(9), 3813–3827.

Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2015a). A DEEP analysis of the META-DES framework for dynamic selection of ensemble of classifiers. *arXiv preprint arXiv:1509.00825*.

Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2015b). META-DES.H: a dynamic ensemble selection technique using meta-learning and a dynamic weighting approach. *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

Cruz, R. M., Sabourin, R., Cavalcanti, G. D. & Ren, T. I. (2015c). META-DES: A dynamic ensemble selection framework using meta-learning. *Pattern recognition*, 48(5), 1925–1935.

Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2017). META-DES. Oracle: Meta-learning and feature selection for dynamic ensemble selection. *Information fusion*, 38, 84–103.

Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2018a). Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41, 195–216.

Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2018b). Prototype selection for dynamic classifier and ensemble selection. *Neural Computing and Applications*, 29, 447–457.

Cruz, R. M., Oliveira, D. V., Cavalcanti, G. D. & Sabourin, R. (2019). FIRE-DES++: Enhanced online pruning of base classifiers for dynamic ensemble selection. *Pattern Recognition*, 85, 149–160.

Cruz, R. M., Hafemann, L. G., Sabourin, R. & Cavalcanti, G. D. (2020). DESlib: A Dynamic ensemble selection library in Python. *The Journal of Machine Learning Research*, 21(1), 283–287.

Cruz, R. M., de Sousa, W. V. & Cavalcanti, G. D. (2022). Selecting and combining complementary feature representations and classifiers for hate speech detection. *Online Social Networks and Media*, 28, 100194.

Cvetkovic, V. M. & Martinović, J. (2020). Innovative solutions for flood risk management. *International Journal of Disaster Risk Management*, 2(2), 71–100.

Davtalab, R., Cruz, R. M. O. & Sabourin, R. (2022). Dynamic Ensemble Selection Using Fuzzy Hyperboxes.

Davtalab, R., Cruz, R. M. & Sabourin, R. (2024). A scalable dynamic ensemble selection using fuzzy hyperboxes. *Information Fusion*, 102, 102036.

De Almeida, P. R. L., Oliveira, L. S., Britto, A. D. S. & Sabourin, R. (2016). Handling concept drifts using dynamic selection of classifiers. *2016 IEEE 28th international conference on tools with artificial intelligence (ICTAI)*, pp. 989–995.

de Almeida, P. R. L., da Silva Júnior, E. J., Celinski, T. M., de Souza Britto, A., de Oliveira, L. E. S. & Koerich, A. L. (2012). Music genre classification using dynamic selection of ensemble of classifiers. *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 2700–2705.

de Amorim, L. B., Cavalcanti, G. D. & Cruz, R. M. (2023). The choice of scaling technique matters for classification performance. *Applied Soft Computing*, 133, 109924.

de Araujo Souza, M., Sabourin, R., da Cunha Cavalcanti, G. D. & e Cruz, R. M. O. (2023). GNN-DES: A new end-to-end dynamic ensemble selection method based on multi-label graph neural network. *International Workshop on Graph-Based Representations in Pattern Recognition*, pp. 59–69.

Deb, K. (2012). Advances in evolutionary multi-objective optimization. *Search Based Software Engineering: 4th International Symposium, SSBSE 2012, Riva del Garda, Italy, September 28-30, 2012. Proceedings 4*, pp. 1–26.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7, 1–30.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *International workshop on multiple classifier systems*, pp. 1–15.

Dorogush, A. V., Ershov, V. & Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*.

Dos Santos, E. M., Sabourin, R. & Maupin, P. (2008). A dynamic overproduce-and-choose strategy for the selection of classifier ensembles. *Pattern recognition*, 41(10), 2993–3009.

Elmi, J. & Eftekhari, M. (2020). Dynamic ensemble selection based on hesitant fuzzy multiple criteria decision making. *Soft Computing*, 24(16), 12241–12253.

Elmi, J. & Eftekhari, M. (2021). Multi-Layer Selector (MLS): Dynamic selection based on filtering some competence measures. *Applied Soft Computing*, 104, 107257.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M. & Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Freund, Y. (2001). An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3), 293–318.

Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119–139.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4), 367–378.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200), 675–701.

Gabrys, B. & Ruta, D. (2006). Genetic algorithms in classifier fusion. *Applied soft computing*, 6(4), 337–347.

Garcia, L. P., Lorena, A. C., de Souto, M. C. & Ho, T. K. (2018). Classifier recommendation using data complexity measures. *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 874–879.

García, S., Zhang, Z.-L., Altalhi, A., Alshomrani, S. & Herrera, F. (2018). Dynamic ensemble selection for multi-class imbalanced datasets. *Information Sciences*, 445, 22–37.

Gemp, I., Theocharous, G. & Ghavamzadeh, M. (2017). Automated data cleansing through meta-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(2), 4760–4761.

Ghosh, A., Shankar, B. U., Bruzzone, L. & Meher, S. K. (2010). Neuro-fuzzy-combiner: An effective multiple classifier system. *International Journal of Knowledge Engineering and Soft Data Paradigms*, 2(2), 107–129.

González, S., García, S., Del Ser, J., Rokach, L. & Herrera, F. (2020). A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion*, 64, 205–237.

Hastie, T., Rosset, S., Zhu, J. & Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, 2(3), 349–360.

Ho, T. K. & Basu, M. (2000). Measuring the complexity of classification problems. *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, 2, 43–47.

Ho, T. K. & Basu, M. (2002). Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, 24(3), 289–300.

Hollander, M., Wolfe, D. A. & Chicken, E. (2013). *Nonparametric statistical methods*. John Wiley & Sons.

Hou, W.-h., Wang, X.-k., Zhang, H.-y., Wang, J.-q. & Li, L. (2020). A novel dynamic ensemble selection classifier for an imbalanced data set: An application for credit risk assessment. *Knowledge-Based Systems*, 208, 106462.

Hutter, F., Kotthoff, L. & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature.

Islam, M. Z., Liu, J., Li, J., Liu, L. & Kang, W. (2019). A semantics aware random forest for text classification. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1061–1070.

Jiao, B., Guo, Y., Gong, D. & Chen, Q. (2022). Dynamic ensemble selection for imbalanced data streams with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*.

Kalousis, A. & Hilario, M. (2001). Feature selection for meta-learning. *Advances in Knowledge Discovery and Data Mining: 5th Pacific-Asia Conference, PAKDD 2001 Hong Kong, China, April 16–18, 2001 Proceedings 5*, pp. 222–233.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Khan, I., Zhang, X., Rehman, M. & Ali, R. (2020). A literature survey and empirical study of meta-learning for classifier selection. *IEEE Access*, 8, 10262–10281.

Kittler, J., Hatef, M., Duin, R. P. & Matas, J. (1998). On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3), 226–239.

Ko, A. H., Sabourin, R. & Britto Jr, A. S. (2008). From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5), 1718–1731.

Krawczyk, B. & Woźniak, M. (2016). Dynamic classifier selection for one-class classification. *Knowledge-Based Systems*, 107, 43–53.

Krijthe, J. H., Ho, T. K. & Loog, M. (2012). Improving cross-validation based classifier selection using meta-learning. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 2873–2876.

Kubat, M., Matwin, S. et al. (1997). Addressing the curse of imbalanced training sets: one-sided selection. *Icml*, 97(1), 179.

Kuncheva, L. I. (2000). Clustering-and-selection model for classifier combination. *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516)*, 1, 185–188.

Kuncheva, L. I. (2014a). Combining pattern classifiers: methods and algorithms.

Kuncheva, L. I. (2014b). *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons.

Kuncheva, L. I. & Rodríguez, J. J. (2007). An experimental study on rotation forest ensembles. *International workshop on multiple classifier systems*, pp. 459–468.

Kurzynski, M., Krysmann, M., Trajdos, P. & Wolczowski, A. (2016). Multiclassifier system with hybrid learning applied to the control of bioprosthetic hand. *Computers in biology and medicine*, 69, 286–297.

Lee, C.-Y. & Antonsson, E. (2000). Dynamic partitional clustering using evolution strategies. *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies*, 4, 2716–2721.

Li, D., Wen, G., Li, X. & Cai, X. (2019). Graph-based dynamic ensemble pruning for facial expression recognition. *Applied Intelligence*, 49, 3188–3206.

Lorena, A. C., Maciel, A. I., de Miranda, P. B., Costa, I. G. & Prudêncio, R. B. (2018). Data complexity meta-features for regression problems. *Machine Learning*, 107, 209–246.

Lorena, A. C., Garcia, L. P., Lehmann, J., Souto, M. C. & Ho, T. K. (2019). How complex is your classification problem? a survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5), 1–34.

Lysiak, R., Kurzynski, M. & Woloszynski, T. (2011). Probabilistic approach to the dynamic ensemble selection using measures of competence and diversity of base classifiers. *International Conference on Hybrid Artificial Intelligence Systems*, pp. 229–236.

Macià, N., Ho, T. K., Orriols-Puig, A. & Bernadó-Mansilla, E. (2010). The landscape contest at ICPR 2010. *International Conference on Pattern Recognition*, pp. 29–45.

Mendialdua, I., Martínez-Otzeta, J. M., Rodriguez-Rodriguez, I., Ruiz-Vazquez, T. & Sierra, B. (2015). Dynamic selection of the best base classifier in one versus one. *Knowledge-Based Systems*, 85, 298–306.

Monteiro, M., Britto, A. S., Barddal, J. P., Oliveira, L. S. & Sabourin, R. (2021). Classifier pool generation based on a two-level diversity approach. *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2414–2421.

Muñoz, M. A., Sun, Y., Kirley, M. & Halgamuge, S. K. (2015). Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317, 224–245.

Pavel, Y. & Soares, B. C. (2002). Decision tree-based data characterization for meta-learning. *IDDM-2002*, 111.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12, 2825–2830.

Pereira, M., Britto, A., Oliveira, L. & Sabourin, R. (2018). Dynamic ensemble selection by K-nearest local Oracles with Discrimination Index. *2018 IEEE 30th International conference on tools with artificial intelligence (ICTAI)*, pp. 765–771.

Pimentel, B. A. & De Carvalho, A. C. (2019). A new data characterization for selecting clustering algorithms using meta-learning. *Information Sciences*, 477, 203–219.

Pinto, F., Cerqueira, V., Soares, C. & Mendes-Moreira, J. (2017). autobagging: Learning to rank bagging workflows with metalearning. *arXiv preprint arXiv:1706.09367*.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3), 21–45.

Reid, S. (2007). A review of heterogeneous ensemble methods. *Department of Computer Science, University of Colorado at Boulder*.

Reif, M., Shafait, F., Goldstein, M., Breuel, T. & Dengel, A. (2014). Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17, 83–96.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers* (vol. 15, pp. 65–118). Elsevier.

Rinartha, K. & Suryasa, W. (2017). Comparative study for better result on query suggestion of article searching with MySQL pattern matching and Jaccard similarity. *2017 5th International Conference on Cyber and IT Service Management (CITSM)*, pp. 1–4.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J. & de Carvalho, A. C. (2018). Characterizing classification datasets: a study of meta-features for meta-learning. *arXiv preprint arXiv:1808.10406*.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J. & de Carvalho, A. C. (2022). Meta-features for meta-learning. *Knowledge-Based Systems*, 240, 108101.

Ross, A., Pan, W., Celi, L. & Doshi-Velez, F. (2020). Ensembles of locally independent prediction models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04), 5527–5536.

Roy, A., Cruz, R. M., Sabourin, R. & Cavalcanti, G. D. (2016). Meta-learning recommendation of default size of classifier pool for META-DES. *Neurocomputing*, 216, 351–362.

Ruta, D. & Gabrys, B. (2005). Classifier selection for majority voting. *Information fusion*, 6(1), 63–81.

Sáez, J. A., Galar, M., Luengo, J. & Herrera, F. (2013). Tackling the problem of classification with noisy data using multiple classifier systems: Analysis of the performance and robustness. *Information Sciences*, 247, 1–20.

Schapire, R. E. & Freund, Y. (2013). Boosting: Foundations and algorithms. *Kybernetes*.

Sergio, A. T., de Lima, T. P. & Ludermir, T. B. (2016). Dynamic selection of forecast combiners. *Neurocomputing*, 218, 37–50.

Skurichina, M. & Duin, R. P. (1998). Bagging for linear classifiers. *Pattern Recognition*, 31(7), 909–930.

Smith, M. R., Mitchell, L., Giraud-Carrier, C. & Martinez, T. (2014). Recommending learning algorithms and their associated hyperparameters. *arXiv preprint arXiv:1407.1890*.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1), 1–25.

Smits, P. C. (2002). Multiple classifier systems for supervised remote sensing image classification based on dynamic classifier selection. *IEEE Transactions on Geoscience and Remote Sensing*, 40(4), 801–813.

Soares, C., Petrak, J. & Brazdil, P. (2001). Sampling-based relative landmarks: Systematically test-driving algorithms before choosing. *Progress in Artificial Intelligence: Knowledge Extraction, Multi-agent Systems, Logic Programming, and Constraint Solving 10th Portuguese Conference on Artificial Intelligence, EPIA 2001 Porto, Portugal, December 17–20, 2001 Proceedings 10*, pp. 88–95.

Soares, R. G., Santana, A., Canuto, A. M. & de Souto, M. C. P. (2006). Using accuracy and diversity to select classifiers to build ensembles. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pp. 1310–1316.

Song, Q., Wang, G. & Wang, C. (2012). Automatic recommendation of classification algorithms based on data set characteristics. *Pattern recognition*, 45(7), 2672–2689.

Souza, M. A., Cavalcanti, G. D., Cruz, R. M. & Sabourin, R. (2017). On the characterization of the oracle for dynamic classifier selection. *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 332–339.

Souza, M. A., Cavalcanti, G. D., Cruz, R. M. & Sabourin, R. (2019). Online local pool generation for dynamic classifier selection. *Pattern Recognition*, 85, 132–148.

Souza, M. A., Sabourin, R., Cavalcanti, G. D. & Cruz, R. M. (2023). OLP++: An online local classifier for high dimensional data. *Information Fusion*, 90, 120–137.

Sun, Q., Liu, Y., Chua, T.-S. & Schiele, B. (2019). Meta-transfer learning for few-shot learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 403–412.

Todorovski, L., Brazdil, P. & Soares, C. (2000). *Report on the experiments with feature selection in meta-level learning*.

Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.-L. A., Elkhatib, Y., Hussain, A. & Al-Fuqaha, A. (2019). Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE access*, 7, 65579–65615.

Wang, H., Fan, W., Yu, P. S. & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 226–235.

Woloszynski, T. & Kurzynski, M. (2009). On a new measure of classifier competence applied to the design of multiclassifier systems. *Image Analysis and Processing–ICIAP 2009: 15th International Conference Vietri sul Mare, Italy, September 8-11, 2009 Proceedings 15*, pp. 995–1004.

Woloszynski, T. & Kurzynski, M. (2011). A probabilistic model of classifier competence for dynamic ensemble selection. *Pattern Recognition*, 44(10-11), 2656–2668.

Woloszynski, T., Kurzynski, M., Podsiadlo, P. & Stachowiak, G. W. (2012). A measure of competence based on random classification for dynamic ensemble selection. *Information Fusion*, 13(3), 207–213.

Woods, K., Kegelmeyer, W. P. & Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, 19(4), 405–410.

Xiao, H., Xiao, Z. & Wang, Y. (2016). Ensemble classification based on supervised clustering for credit scoring. *Applied Soft Computing*, 43, 73–86.

Xiao, J., Xie, L., He, C. & Jiang, X. (2012). Dynamic classifier ensemble model for customer classification with imbalanced class distribution. *Expert Systems with Applications*, 39(3), 3668–3675.

Zagatti, F. R., Silva, L. C., Silva, L. N. D. S., Sette, B. S., de Medeiros Caseli, H., Lucrédio, D. & Silva, D. F. (2021). MetaPrep: Data preparation pipelines recommendation via meta-learning. *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1197–1202.

Zhang, B. & Pham, T. D. (2011). Phenotype recognition with combined features and random subspace classifier ensemble. *BMC bioinformatics*, 12(1), 1–13.

Zhou, Z.-H. (2012). Ensemble methods: foundations and algorithms.

Zhu, J. & Hovy, E. (2007). Active learning for word sense disambiguation with methods for addressing the class imbalance problem. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 783–790.

Zhu, X., Yang, X., Ying, C. & Wang, G. (2018). A new classification algorithm recommendation method based on link prediction. *Knowledge-Based Systems*, 159, 171–185.

Zhu, X., Wu, X. & Yang, Y. (2004). Dynamic classifier selection for effective mining from noisy data streams. *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 305–312.