ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

HENRY EMANUEL LEAL CAGNINI

EVOLUTIONARY ALGORITHMS FOR LEARNING ENSEMBLES OF INTERPRETABLE CLASSIFIERS

Porto Alegre
2022

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

EVOLUTIONARY ALGORITHMS FOR LEARNING ENSEMBLES OF INTERPRETABLE CLASSIFIERS

HENRY EMANUEL LEAL CAGNINI

Doctoral Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Rodrigo Coelho Barros
Co-Advisor: Prof. Alex Alves Freitas

Porto Alegre
2022

Ficha Catalográfica

C131e   Cagnini, Henry Emanuel Leal

Evolutionary Algorithms for Learning Ensembles of Interpretable Classifiers / Henry Emanuel Leal Cagnini. – 2022.
184.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.
Coorientador: Prof. Dr. Alex Alves Freitas.

1. evolutionary algorithms. 2. ensemble learning. 3. machine learning. 4. interpretability. 5. classification. I. Barros, Rodrigo Coelho. II. Freitas, Alex Alves. III. , . IV. Título.

HENRY EMANUEL LEAL CAGNINI

EVOLUTIONARY ALGORITHMS FOR LEARNING ENSEMBLES OF INTERPRETABLE CLASSIFIERS

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on March 22, 2022.

COMMITTEE MEMBERS:

Dr. Duncan Dubugras Alcoba Ruiz (PPGCC/PUCRS)

Dra. Karin Becker (PPGC/UFRGS)

Dr. Hélio José Corrêa Barbosa (PGCC/UFJF)

Prof. Alex Alves Freitas (University of Kent- Co-Advisor)

Prof. Rodrigo Coelho Barros (PPGCC/PUCRS - Advisor)

“Eventually everything connects – people, ideas, objects. The quality of the connections is the key to quality per se.”
(Charles Eames)

ACKNOWLEDGMENTS

Agradeço aos meus pais, Marlene Leal Cagnini e Luiz Sadí Cagnini, por terem investido na minha educação desde cedo. Foi o seu esforço e sacrifício que permitiu que eu chegasse até aqui.

Agradeço à minha noiva, Juliana de Oliveira Mozzaquatro, por ter estado ao meu lado durante todos os momentos, alegres ou difíceis, e não ter me deixado desistir nunca.

Agradeço aos meus amigos de longa data, e amigos que fiz durante o período de mestrado e doutorado na PUCRS, por todos os momentos que vivemos juntos.

Agradeço ao meu orientador, Rodrigo Coelho Barros, e co-orientador, Alex Alves Freitas, pelas oportunidades à que tive acesso, tenham sido no Brasil ou no exterior, e pelas discussões necessárias para que meu trabalho chegasse ao nível apresentado aqui.

Agradeço ao secretário do Programa de Pós-Graduação em Ciência da Computação, Régis Escobal da Silva, pelo ótimo trabalho desempenhado nesta instituição, essencial para o correto funcionamento do programa.

Finalmente, mas não menos importante, agradeço às instituições: à Pontifícia Universidade Católica do Rio Grande do Sul, e à Unviersity of Kent, pelo acolhimento que tive durante o período do meu doutorado; e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES, por desempenhar seu trabalho com maestria, mesmo em tempos sombrios.

ALGORITMOS EVOLUTIVOS PARA O APRENDIZADO DE *ENSEMBLES* DE CLASSIFICADORES INTERPRETÁVEIS

RESUMO

Classificação é a tarefa de Aprendizado de Máquina que visa categorizar instâncias em classes. Existem diversos algoritmos na literatura que realizam classificação, com diferentes graus de sucesso. Nos últimos anos, o desempenho preditivo foi o objetivo priorizado entre praticantes de Aprendizado de Máquina e a comunidade acadêmica. Todavia, mais recentemente, interpretabilidade tem ganhado cada vez mais atenção. Uma área de aprendizado de máquina que pode se beneficiar de um ganho em interpretabilidade é a de *ensemble learning*. *Ensemble learning* visa reunir modelos que, quando agrupados em comitês, podem fornecer alto grau de desempenho preditivo, mesmo que os classificadores que façam parte do grupo não sejam (em média) muito melhores que preditores aleatórios. Doravante, os benefícios são duplos: *ensembles* podem melhorar o desempenho preditivo de modelos interpretáveis caixa branca (que são, em média, piores que modelos caixa preta); e o uso de modelos caixa-branca aumenta a interpretabilidade de *ensembles*. Nesta tese, através do projeto de algoritmos evolutivos, uma poderosa classe de algoritmos de *soft computing*, desenvolvemos dois métodos para aprendizado de *ensembles* interpretáveis: EDNEL e PUMA. Enquanto os dois métodos são semelhantes, a diferença entre eles ainda assim é significativa: PUMA aprende *ensembles* de classificadores sem levar a interação entre variáveis em consideração, enquanto EDNEL calcula a correlação das variáveis. Todavia, nos experimentos que conduzimos para avaliar o desempenho dos métodos, detectamos que a abordagem mais simples de PUMA gerou *ensembles* com melhor desempenho preditivo em média do que EDNEL, enquanto aquele é estatisticamente equivalente à dois bem-estabelecidos métodos de aprendizado de *ensembles*, Adaboost e Random Forests.

Palavras-Chave: algoritmos evolutivos, *ensemble learning*, aprendizado de máquina, interpretabilidade, classificação, aprendizado supervisionado, regressão.

EVOLUTIONARY ALGORITHMS FOR LEARNING ENSEMBLES OF INTERPRETABLE CLASSIFIERS

ABSTRACT

Classification is the machine learning task of categorizing instances into classes. There are several algorithms in the literature that perform classification, with varying degrees of success. For the most part, predictive performance was the pursued objective among practitioners and the academic community regarding the design of novel classification algorithms. More recently, however, interpretability has been gaining more and more attention. One area of machine learning that can benefit from increased interpretability is that of ensemble learning. Ensemble learning aims to reunite models that, when ensembled, can provide a high degree of predictive performance, even though the individual classifiers of the ensemble are often not much better at predicting classes than random guessing. Hence, the benefits are twofold: ensembles can improve predictive performance of interpretable (white-box) models that perform, on average, worse than black-box models; and the use of white-box models improves the interpretability of ensembles. In this thesis, we design two evolutionary algorithms (a powerful soft computing technique) to develop two ensemble learning methods, EDNEL and PUMA. PUMA learns ensembles of classifiers in a univariate strategy, assuming independence among variables, while EDNEL takes into account variable dependence through correlation analysis. However, in the thorough experimental analysis performed, we found that PUMA performs better than EDNEL with regards to average rank, whilst it is statistically equivalent to two well-established ensemble learning algorithms, Adaboost and Random Forests.

Keywords: evolutionary algorithms, ensemble learning, machine learning, interpretability, classification, supervised learning, regression.

LIST OF ACRONYMS

EA – Evolutionary Algorithms

GA – Genetic Algorithm

NSGA-II – Non-dominated Sorting Genetic Algorithm II (NSGA-II)

GP – Genetic Programming

PSO – Particle Swarm Optimization

DE – Differential Evolution

EDA – Estimation of Distribution Algorithm

GM – Probabilistic Graphical Model

LIST OF SYMBOLS

ϕ – Fitness of an (unspecified) individual. 32

$|S|$ – Number of individuals in an evolutionary algorithm's population 33

G – Number of generations in an evolutionary algorithm run. 34

s_i – i-th solution/individual from a population 34

Φ – number of objectives in a multi-objective problem 34

$\Phi_j(s_i)$ – quality of the i-th solution for the $j-th$ objective 34

$U(a, b)$ – A random number sampled from a uniform distribution in the interval $[a, b]$. 37

$\mathcal{A}$ – Set of algorithms 50

$A^{(k)}$ – k-th algorithm in a set of algorithms 50

$\Lambda^{(k)}$ – Hyper-parameter space of the k-th algorithm $A^{(k)}$ 50

N – Number of instances 50

M – Number of attributes 50

$\mathbf{X}$ – Predictive data with size $N \times M$ 50

Y – Set of labels associated to $\mathbf{X}$ 50

$y^{(i)}$ – Label of the i-th instance 50

X_j – j-th attribute 50

$X^{(i)}$ – i-th instance 50

$x_j^{(i)}$ – value of the i-th instance in the j-th attribute 50

λ_l – The l-th hyper-parameter of an algorithm 51

Λ_l – The domain of the l-th hyper-parameter of an algorithm 51

$\boldsymbol{\Lambda}$ – Set of domains for an algorithm 51

$\boldsymbol{\lambda}^*$ – The best set of hyper-parameters for a given algorithm. 51

B – Number of base learners in an ensemble. 83

C – Number of classes in a dataset. 83

CONTENTS

1 INTRODUCTION 23
1.1 OBJECTIVES 25
1.2 THESIS' CONTRIBUTIONS 26
1.3 THESIS' OUTLINE 27

2 BACKGROUND 29
2.1 TRADITIONAL NON-EVOLUTIONARY ENSEMBLE LEARNING METHODS 29
2.1.1 BOOSTING 29
2.1.2 BAGGING 30
2.1.3 STACKING 30
2.1.4 RANDOM FORESTS 31
2.2 EVOLUTIONARY ALGORITHMS 31
2.2.1 THE FOUR COLOR THEOREM 32
2.2.2 GENETIC ALGORITHMS 33
2.2.3 GENETIC PROGRAMMING 36
2.2.4 PARTICLE SWARM OPTIMIZATION 37
2.2.5 DIFFERENTIAL EVOLUTION 39
2.2.6 ESTIMATION OF DISTRIBUTION ALGORITHMS 41
2.3 INTERPRETABILITY 42
2.3.1 WHITE-BOX MODELS 45
2.3.2 BLACK-BOX MODELS 48
2.3.3 EVALUATING INTERPRETABILITY 49
2.4 AUTO-MACHINE LEARNING 50
2.4.1 FORMAL DEFINITION 50
2.4.2 ADDRESSING THE CASH PROBLEM 51
2.5 SUMMARY 52

3 ENSEMBLE LEARNING WITH EVOLUTIONARY ALGORITHMS 55
3.1 METHODOLOGY 56
3.2 TAXONOMY 58
3.3 THE GENERATION STAGE OF ENSEMBLE LEARNING 59
3.3.1 INSTANCE SELECTION 60

3.3.2 ATTRIBUTE SELECTION 61
3.3.3 MODEL OPTIMIZATION 62
3.4 THE SELECTION STAGE OF ENSEMBLE LEARNING 64
3.4.1 STATIC SELECTION 66
3.4.2 DYNAMIC SELECTION 67
3.5 THE INTEGRATION STAGE OF ENSEMBLE LEARNING 68
3.5.1 LINEAR MODELS 69
3.5.2 EXPRESSION TREES 70
3.5.3 GENETIC FUZZY SYSTEMS 70
3.5.4 INDUCED ORDERED WEIGHTED AVERAGING (IOWA) 71
3.5.5 ERROR CORRECTING OUTPUT CODES (ECOC) 71
3.5.6 NEURAL NETWORKS 73
3.5.7 EVOLUTIONARY ALGORITHMS FOR SELECTING META-COMBINERS 73
3.6 FITNESS FUNCTIONS 73
3.6.1 EFFECTIVENESS, DIVERSITY, COMPLEXITY AND EFFICIENCY 73
3.6.2 SINGLE VS. MULTI-OBJECTIVE OPTIMIZATION 76
3.7 TYPES OF EVOLUTIONARY ALGORITHMS 77
3.8 TYPES OF BASE LEARNERS 78
3.9 APPLICATION DOMAINS 78
3.10 SUMMARY OF FINDINGS 80

4 EEL: ESTIMATION OF DISTRIBUTION ALGORITHMS FOR ENSEMBLE LEARNING 83
4.1 PROPOSED METHOD 83
4.1.1 FITNESS COMPUTATION 84
4.1.2 UPDATING THE PROBABILISTIC GRAPHICAL MODEL 84
4.1.3 COMPLEXITY ANALYSIS 86
4.2 EXPERIMENTAL SETUP 87
4.2.1 BASELINE ALGORITHMS 87
4.2.2 DATASETS 88
4.3 EXPERIMENTAL RESULTS 88
4.3.1 EXECUTION ANALYSIS 90
4.4 DISCUSSION AND FINAL REMARKS 92

5 PUMA: PROBABILISTIC UNIVARIATE ESTIMATION OF DISTRIBUTION ALGORITHM FOR ENSEMBLE LEARNING 93

5.1 PROPOSED METHOD . . . 94
5.1.1 INDIVIDUALS . . . 94
5.1.2 FITNESS EVALUATION . . . 96
5.1.3 PUMA'S PROBABILISTIC GRAPHICAL MODEL . . . 97
5.1.4 EARLY-STOPPING AND TERMINATION . . . 99
5.1.5 COMPLEXITY ANALYSIS . . . 100
5.2 EXPERIMENTAL SETUP . . . 100
5.2.1 DATASETS . . . 101
5.2.2 MODIFICATIONS TO PUMA . . . 102
5.2.3 BASELINE ALGORITHMS AND HYPER-PARAMETER OPTIMIZATION . . . 102
5.2.4 HARDWARE SPECIFICATIONS AND SOURCE CODE . . . 105
5.3 EXPERIMENTAL RESULTS . . . 106
5.3.1 FAIR ANALYSIS ON INTERPRETABILITY . . . 111
5.4 DISCUSSION AND FINAL REMARKS . . . 113

6 EDNEL: ESTIMATION OF DEPENDENCY NETWORKS FOR ENSEMBLE LEARNING . . . 115
6.1 PROPOSED METHOD . . . 116
6.1.1 INDIVIDUALS . . . 116
6.1.2 PROBABILISTIC GRAPHICAL MODEL . . . 120
6.1.3 SAMPLING INDIVIDUALS . . . 123
6.1.4 FITNESS EVALUATION . . . 125
6.1.5 POPULATION SELECTION AND ELITISM . . . 125
6.1.6 UPDATING THE GM'S STRUCTURE . . . 125
6.1.7 UPDATING THE GM'S PROBABILITIES . . . 127
6.1.8 EARLY STOP, TERMINATION, AND VALIDATION SET . . . 128
6.2 EXPERIMENTS . . . 129
6.2.1 NESTED CROSS-VALIDATION EXPERIMENTAL SETUP . . . 129
6.2.2 NESTED CROSS-VALIDATION EXPERIMENTAL RESULTS . . . 135
6.2.3 HOLDOUT EXPERIMENTAL SETUP . . . 145
6.2.4 HOLDOUT EXPERIMENTAL RESULTS . . . 146
6.3 DISCUSSION AND FINAL REMARKS . . . 153

7 CONCLUSIONS . . . 157
7.1 LIMITATIONS . . . 158
7.2 FUTURE WORK . . . 158

REFERENCES 161

1. INTRODUCTION

Classification is the task of supervised machine learning that aims at classifying instances into categories (classes). Based on instances with already-known classes, a machine learning algorithm can predict the classes of unknown instances by building models that capture the rationale behind the data. Take for example a database comprised of x-rays from lung cancer patients. The task is to, given a new x-ray image, predict whether this person shows signs of lung cancer.

There are several algorithms in the literature that tackle the classification task, with varying degrees of success. Often, predictive performance is not the only desirable feature of a model; interpretability also plays a part [20, 35, 98, 227]. White-box models, for example, show the underlying classifying process to a user. Black-box models, on the other hand, provide only a predicted class label, and not an explanation. Looking back at the x-rays example, a black-box model would say that a patient could have lung cancer signals, but without specifying which features it is using to do so.

Black-box models have been favored in the literature, at the deprecation of white-box models, due to their predictive performance superiority [35]. Deep neural networks [111] are a notorious example on this front: by chaining several linear models (expressed within the neurons of the network), this class of algorithm achieves unparalleled accuracy on image classification tasks [35]. Interpretability, however, is a desirable feature for certain application domains, such as finance, medicine, churn prediction, and bioinformatics, just to name a few. In medical applications, for example, it is not sufficient to have a model with good predictive performance if the predictions of such a model cannot be explained – and hence, trusted [98].

One could argue that there are methods to explain predictions, such as SHAP [176], which explain how much each attribute contributed to the classification outcome, or, for image classification tasks, saliency maps [227], that show which regions in an image contribute the most for the model to assign a particular class. However, the trick here is that these post-model explanations build *explanation models*, which are not necessarily the same as the *classification models*. For example: if an *explanation model* correctly explains predictions only 90% of the time, should we trust the *predictive model* 90% of the time too? [227]

Moreover, black-box models are vulnerable to misunderstand the data, and their obliqueness makes impossible for a user to detect such misunderstandings. Deep neural networks, when applied to image classification, for example, are vulnerable to one-pixel attacks: a trick that changes the intensity of a single pixel, but which is sufficient to change the opinion of the network on the class of an image [245].

One way to improve performance of white-box models is to ensemble them. Supervised ensemble learning – sometimes referred to as mixture of experts, classifier ensembles, or multiple classifier system [184, 230] — is the machine learning paradigm that aims at integrating multiple base supervised learners in order to produce better predictive models than simply learning a single strong model. An ensemble typically performs its predictions by using a voting mechanism that computes the mean or the mode of the predictions output by the ensemble's members (base learners). Ensemble learning methods have won several academic and industrial machine learning competitions [228], and such methods have been extensively deployed in real-world AI applications [199, 246]. Examples of successful applications of supervised ensemble learning include intrusion detection [91, 184], wind speed forecasting [269, 279], and power grid transformers fault prediction [206, 207].

Ensembles present several advantages over a single learner: (i) it is usually computationally cheaper to integrate a set of surrogate models that approximate a given function than to induce a single complex model [153]; (ii) base learners that are only marginally better than a random classifier, when properly integrated, can produce predictions that are comparable to those of a strong classifier [99, 127, 171]; (iii) base learners can be trained to become specialized in certain regions of the high-dimensional input space, making their consensus more flexible and effective when dealing with complex problems [99]. Indeed, there is both theoretical and empirical evidence demonstrating that a good ensemble can be obtained by combining individual models that make distinct errors (e.g., errors on different parts of the input space) [114, 116, 136, 154, 197].

Ensemble learning comprises three distinct stages, whose names vary in the literature: generation, selection, and integration [26, 36, 167]; pre-gate, ensemble-member, and post-gate [67]; or generation, pruning, and fusion [201]. Because of this multi-stage framework, it is common to have multiple variables involved during the design of novel ensemble approaches. There are, for example, different methodologies regarding the composition of ensembles: base learners can be from (1) different paradigms, e.g., mixing models such as decision trees and neural networks; (2) same paradigm, e.g., all models are neural networks; and (3) present differences within the same paradigm, e.g., mixing neural networks with different activation functions and/or architectures.

There are three main motivations to combine multiple learners [71]: representational, statistical, and computational. The representational motivation is that combining multiple base learners may provide better predictive performance than a single strong learner. For example, it was empirically shown that the generalization ability of a neural network can be improved by using it as base learner within an ensemble [39]. In theory, no base learner will have the best predictive performance for all problems, as stated by the *No Free Lunch* theorem [268]; and in practice, selecting the best learner for any given dataset is a very difficult problem [84, 144], which can be addressed by integrating several good learners into an ensemble.

The statistical motivation lies in the fact that it is statistically possible to avoid poor performance by averaging the outputs of several base learners. Averaging multiple good solutions may not produce the overall global optimum, but it can at least avoid generating a poor ensemble [120] by eliminating uncorrelated base-learner errors [58]. This is particularly the case where few data points are available, and so overfitting is more likely to occur. Finally, the computational motivation is that some algorithms require several runs with distinct initializations in order to avoid falling into bad local minima. Gradient descent, for example, often requires several runs and further evaluation on a validation set in order to avoid being trapped into local minima. Thus, it seems reasonable to integrate these already-trained intermediate models into an ensemble, stabilizing and improving the system's overall performance [58, 255].

Currently there is an extensive literature on ensemble learning, as surveyed by Sagi and Rokach [228] and Dietterich [71] specifically for traditional, greedy, local-based search strategies (e.g., boosting [233], bagging [23], and stacking [267]); and as surveyed by Yao and Islam [274] for evolutionary algorithms (e.g., [159, 167]). However, there is still no consensus on which approach is better suited for this task, even though evolutionary algorithms (EAs) have some interesting characteristics: i) the global-search characteristic means that this class of algorithm is less likely to get trapped in local minima; ii) EAs are easy to adapt to multi-objective problems; and finally, iii) paralellization is an option to improve computational performance. Hence, as it will be shown in this thesis, it is not surprising that a large number of EAs for supervised ensemble learning have been proposed in the literature in the past few years.

The use of evolutionary algorithms for ensemble learning can be configured as an Auto-Machine Learning (Auto-ML) application [271]. Auto-ML is concerned in selecting the best algorithms for a given task (which, in the case of this thesis, will be classifiers), while also optimizing their hyper-parameters. When considering that one of the tasks of the generation stage of ensemble learning comprises hyper-parameter optimization, while the whole selection stage of ensemble learning is indeed recommending base classifiers, it is evident that an algorithm that learns ensembles can also perform Auto-ML.

1.1 Objectives

Considering the topics discussed above, the overall objective of this thesis is to develop new evolutionary algorithms for learning ensembles of interpretable classifiers. By automatically selecting interpretable classifiers, and optimizing their hyper-parameters, we believe that we can provide models with predictive performance comparable to black-box models, while still being interpretable. If successful, this thesis could be among present and future work that integrate the so-called Explainable Artificial Intelligence (XAI) [35] track, which aims to shift the focus from developing black-box models with ever-increasing performance to more accountable, explainable models.

Among the several types of evolutionary algorithms present in the literature (on which a brief, non-extensive review is presented in Section 2.2), we choose to employ Estimation of Distribution Algorithms (EDAs). In the extensive literature review conducted by us on ensemble learning with evolutionary algorithms, presented in Chapter 3, we detected that there are more genetic algorithms (GAs) than Estimation of Distribution Algorithms for this task. However, we choose to employ EDAs because they use probabilistic graphical models, a structure that explicitly manipulates variables in the problem. Because of that, it is possible to bias the initial probabilities of the variables, a resource that was used on all of this thesis' proposed algorithms. Such processing could not be performed using, e.g., genetic algorithms, since in the latter variables are implicitly encoded, with their values manifesting only within individuals.

The first specific objective of this thesis is to develop an Estimation of Distribution Algorithm for learning ensembles of interpretable classifiers that does not take probabilistic relationship between variables into account (in this thesis, variables are the hyper-parameter values of base classifiers), while still having comparable predictive performance to traditional, non-evolutionary, black-box, ensemble learning algorithms.

The second objective is similar to the first, but this time we are interested in learning relationships between variables. How relationships will be learned? Which type of probabilistic graphical model is more appropriate for this task, while also taking interpretability into account?

Finally, the third and last specific objective aims to perform an extensive experimental comparison between both EDAs. Which EDA would perform better, the one that learns relationships between variables or the one that does not? And more importantly, due to what factors can we attribute the experimental outcome?

1.2 Thesis' Contributions

We performed a survey on evolutionary algorithms for ensemble learning, a work that, up to our knowledge, is the first in its kind. The survey proposes a new taxonomy for classifying the literature on the subject. The survey reviews what are the ensemble learning steps, how evolutionary algorithms can be employed in each one of them, and discusses pros and cons on some topics, such as model selection – a topic that has yet to see consensus on the academic community. The survey is presented in Chapter 3.

The main contribution of this thesis is to propose new evolutionary algorithms for inducing ensembles of interpretable classifiers. The first algorithm is called PUMA – Probabilistic Univariate Estimation of Distribution Algorithm for Ensemble Learning, and is described in Chapter 5. The second algorithm is called EDNEL – Estimation of Dependency Networks for Ensemble Learning, which in turn is presented in Chapter 6. As contributions of this thesis, we can enumerate:

- Developing EEL (chapter 4), an Estimation of Distribution Algorithm for adjusting the voting weights of Adaboost post-classification, thus increasing its predictive performance;

- Developing evolutionary algorithms for ensemble learning that work on the three stages of ensemble learning: generation, selection, and integration. We are unaware of other EAs that operate simultaneously on these three stages;

- Evolving ensembles of classifiers with the explicit objective of providing interpretable models to users (specialists or not);

- Developing an evolutionary ensemble learning algorithm (PUMA, Chapter 5) that does not take variable relationships into account, but nonetheless presents competitive predictive performance when compared to traditional, non-evolutionary, black-box ensemble learning algorithms;
- Developing an evolutionary ensemble learning algorithm (EDNEL, Chapter 6) that explains the relationship between base classifiers' hyper-parameter values for a given learning process (e.g. "how does the pruning policy of base learner A affects the search procedure of base learner B?");
- Developing evolutionary ensemble learning algorithms that automatically learn the best-suited strategy for each application domain at hand, achieving predictive performance comparable to those of powerful, well-established black-box models.

1.3 Thesis' Outline

This thesis is structured in 7 chapters, as follows.

Chapter 2 (Background) describes the background topics regarding evolving interpretable ensembles of classifiers: evolutionary algorithms, remarks on interpretability, and automated machine learning.

Chapter 3 (Ensemble Learning with Evolutionary Algorithms) performs an extensive literature review on ensemble learning with evolutionary algorithms. This chapter outlines where evolutionary algorithms have been used by practitioners and researchers for generating ensembles, while also proposing a new taxonomy for classifying such work. It also briefly reviews traditional, non-evolutionary methods for ensemble learning.

Chapter 4 (EEL: Adjusting Adaboost Weights with an EDA) describes one of the early work of this thesis, a post-hoc evolutionary algorithm to Adaboost for adjusting its voting weights in order to maximize predictive performance.

Chapter 5 (PUMA: Probabilistic Univariate Estimation of Distribution Algorithm for Ensemble Learning) introduces PUMA, an univariate Estimation of Distribution Algorithm for ensemble learning that considers at most five white-box classifiers.

Chapter 6 (EDNEL: A Multivariate EDA for Inducing Interpretable Ensembles) introduces EDNEL, the next iteration of PUMA. EDNEL uses a multivariate Dependency Network instead of a univariate Bayesian Network for the same task.

Chapter 7 (Conclusion) summarizes the contributions and limitations of this thesis, as well as the opportunities we envision for future work.

2. BACKGROUND

This chapter covers topics that are required to better understand the methods proposed in the following chapters. We briefly review non-evolutionary methods for performing ensemble learning (Section 2.1); evolutionary algorithms, not necessarily for ensemble learning, but rather showing how they tackle a simple toy problem (Section 2.2), as means to explain how they differentiate among themselves; comment on interpretability, and ways to objectively measure it (Section 2.3); and finally in Section 2.4 we briefly review auto-machine learning, the type of problem that this thesis addresses.

2.1 Traditional non-Evolutionary Ensemble Learning Methods

Ensemble learning became popular during the 1990's [58], with some of the most important work arising around that time: stacking in 1992 [267]; boosting in 1995 and 1996 [99, 100, 233]; bagging in 1996 [23]; and random forests in the early 2000's [24]. We call these methods *traditional* to differentiate them from EA-based ensembles, though they are also referred to as *preprocessing-based ensemble methods* in the EA literature [146]. Table 2.1 presents a brief overview of the methods that will be covered in this section.

Table 2.1: Comparison of traditional ensemble learning methods. Adapted from [177].

Algorithm	Sampling	Base learner	Integration strategy
Bagging [23]	instance	Unstable learner trained over re-sampled instance subsets	Majority voting
Boosting [99]	instance	Weak learner re-weighted at every iteration	Weighted majority voting
Stacking [267]	None	Any	Meta-model
Random forests [24]	instance; attribute	Decision trees	Majority voting

2.1.1 Boosting

Boosting refers to the technique of continuously improving the performance of a weak learner [146]. We present here the popular AdaBoost algorithm, proposed by Freund and Schapire [99]. Given a set of predictive attributes $\mathbf{X}$ and a set of class labels $Y, y \in Y = \{-1, +1\}$, in its first iteration AdaBoost assigns equal importance (weights) to each instance in the training set, $D_1(i) = 1/N$, $i = 1, \dots, N$, where N is the number of instances. For a given number of iterations G, AdaBoost trains a weak classifier based on the distribution D_g, and then computes its error $\epsilon_g = \mathrm{P}_{i \sim D_g}[h_g(x_i) \neq y_i]$. Instances that are harder to classify will have their weights increased, so it becomes more rewarding to the model to classify them correctly. This process is done by computing the voting strength of the g^{th} iteration classifier by

$$\alpha_g = \frac{1}{2} \ln \left(\frac{1 - \epsilon_g}{\epsilon_g} \right) \tag{2.1}$$

and then adjusting the weight of the instances by

$$D_{g+1}(i) = \frac{D_g(i)\exp(-\alpha_g y_i h_g(x_i))}{Z_g} \tag{2.2}$$

where $Z_g = \sum_{i=1}^{N} D_{g+1}(i)$, which is obtained after calculating all the N new instance weights. After the G iterations are completed, the prediction for a new instance is given by

$$H(x) = \text{sign}\left(\sum_{g=1}^{G} \alpha_g h_g(x)\right) \tag{2.3}$$

Classifiers with greater voting weight will have greater impact in the final prediction.

Candidate algorithms for boosting must support assignment of weights for instances. If this is not possible, then a set of instances can be sampled from D_g and supplied to the g^{th} learner. Although boosting has been shown to improve the predictive performance of a weak classifier, its performance suffers when faced with noisy instances, since failing to correctly classify those instances will iteratively improve their importance and hence lead the learner to overfitting [159].

2.1.2 Bagging

Bootstrap **agg**regat**ing**, or simply *bagging*, aims at reducing training instability when a learner is faced with a given data distribution [23]. It consists of generating B subsets of size N from the original training distribution $D(i) = 1/N$, $i = 1, \dots, N$ with replacement, causing some instances to be present in more than one subset. As a result some base learners have a tendency to favor such instances, having more opportunities to correctly predict their values. The predictions of all trained B learners are combined by computing their mean (regression task) or mode (classification task).

Bagging implicitly injects diversity within the ensemble, whereas boosting explicitly does this by weighting the data distribution to focus the base learners' attention into more difficult instances [112, 159].

2.1.3 Stacking

Compared to bagging and boosting, *stacked generalization* or simply *stacking* [267] is a more flexible strategy for ensemble learning. The practitioner can choose one or more types of base learners to be used in the ensemble (e.g. using only decision trees, or mixing them with artificial neural networks [235]). Then, each base learner will output a prediction, and all learners' predictions will be combined by a meta-learner (which can also be chosen) to produce a single output. Popular traditional meta-learners for stacking include linear regression (for regression) and logistic regression (for classification). Stacking often improves the overall predictive performance of ensembles, making it a popular method [184, 235].

2.1.4 Random Forests

Random forests [24] is a type of ensemble learning method where both the base learner and data sampling are pre-determined: decision trees and random sampling of both instances and attributes. First, the random forest algorithm randomly samples with replacement B subsets of training instances, one for building each of B decision trees that will compose the ensemble. For each inner node within a decision tree, the algorithm first randomly samples without replacement a subset of m attributes, and then it selects, among those attributes, the one that minimizes the local class impurity for that node. This procedure is applied to each inner node in the current decision tree within the ensemble, and it is repeated until the tree achieves maximum class purity for all leaf nodes.

Random forests can sometimes perform better than boosting algorithms, while being resilient to outliers and noise, faster to train than bagging and boosting techniques (depending on the respective base learner), and being easily parallelized. However, it can require very many decision trees to provide an acceptable predictive performance, depending on the dataset at hand.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are robust optimization techniques that perform global search in the space of candidate solutions. Apart from being used in a wide range of applications (e.g., finance, health, natural language processing), EAs are simple to implement (requiring little domain knowledge) and can produce several good solutions to the same problem due to their population-based nature [105].

Regarding ensemble learning, EAs have several desirable characteristics due to their capability of producing a set of solutions that can be readily integrated into an ensemble [74, 159]; supporting multi-objective optimization (e.g., based on Pareto dominance), allowing the generation of solutions that cover distinct aspects of the input space [159]; and removing the need to manually optimize some hyper-parameters (e.g., the base learner's hyper-parameters, the number of ensemble members, etc.). However, EAs will most likely increase the computational cost of ensemble learning due to their robust global-search behavior that usually considers tens or hundreds of possible solutions at each iteration (generation). Nonetheless, parallelization is an option to mitigate such problem [167].

Since ensemble learning comprises at least three main optimization steps, which are sometimes referred to as generation, selection, and integration [26, 36, 167]; pre-gate, ensemble-member, and post-gate [67]; or generation, pruning, and fusion [201]; there are several tasks that are prone to optimization by evolutionary algorithms [87]. In the literature on EA-based ensembles for supervised learning, the most common approach is to optimize a single step, though less frequently some studies optimize two of them (e.g., [43, 190]).

There are several types of evolutionary algorithms, and it is difficult to describe most of them with a single explanation. For this reason, the rest of this section reviews some of the most popular evolutionary algorithms, according to our literature review of evolutionary algorithms for ensemble learning (presented in Chapter 3). As a means to facilitate the reader's understanding, we present a toy application (namely the map coloring problem), and then show how each of the presented EAs would solve the problem.

2.2.1 The Four Color Theorem

The Four Color Theorem [7] is a mathematical theorem that states that any map can be colored with four colors preventing that two neighboring regions share the same color. For example, if we consider the map of Brazilian Federative States (26 states + 1 Federal district), we can prevent that two neighboring states have the same color with a palette with only four colors. The mathematical proof behind this statement requires the conversion of maps to graphs: each region becomes a vertex (or node), and neighborhoods are defined by edges.

Nonetheless, it is interesting to investigate how simpler graphs behave with varying number of colors. Suppose we have a graph that contains a set of edges and vertices. An edge connects two neighboring vertices. If we use a sufficiently limited number of colors to a relatively complex graph structure, it will not always be possible to avoid neighbors sharing the same color. For this reason we define a function to measure how good the chosen color assignments are: this function is called fitness ϕ, and the measure of quality is the number of edges that connect different color vertices. Colored graphs with few same-color neighbors will result in better fitness values. Figure 2.1 depicts an arbitrary coloring of four vertices and three colors, whereas Figure 2.2) depicts the solution space for such a graph.

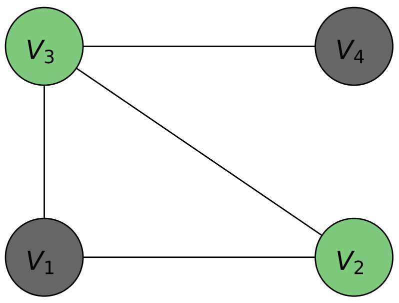
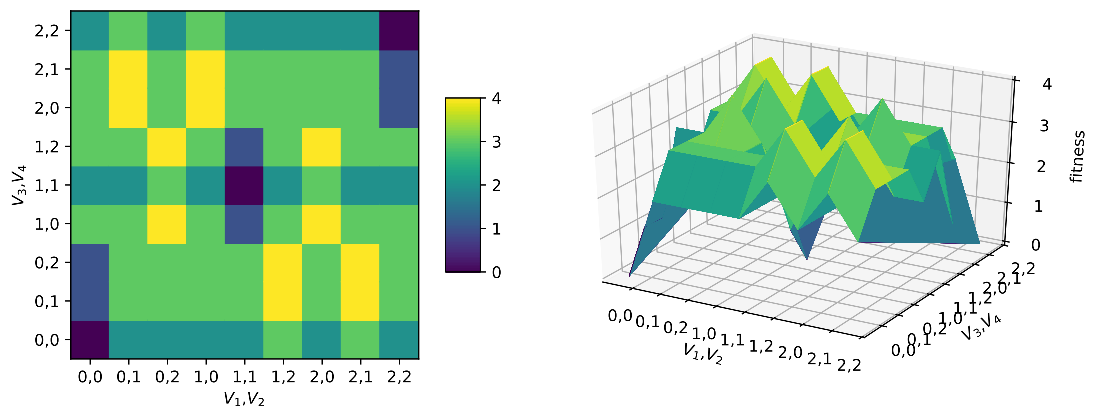


Figure 2.1: Input space for the graph coloring problem with four vertices and two (out of three allowed) colors. This solution has fitness $\phi = 3$ (out of 4), since three edges (the ones that connect vertices V_1 to V_2, V_3 to V_4, and V_1 to V_3) connect different-color vertices.



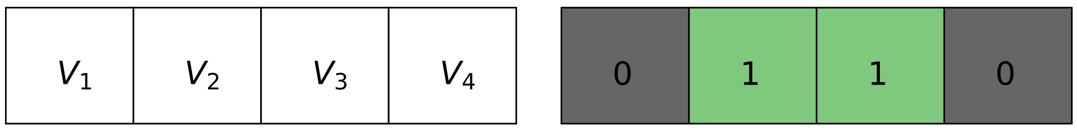
(a) bidimensional (b) tridimensional

Figure 2.2: 2D and 3D solution spaces for the graph coloring problem with 4 vertices and 3 colors. Dark blue-ish tiles represent bad arrangements, whereas light yellow-ish tiles represent good arrangements. Each axis represents two variables (e.g., V_1 and V_2 in the horizontal axis, V_3 and V_4 in the vertical axis), and each tick indicates which colors are assigned to those vertices (0 through 2 since we only have 3 available colors). The lower-left position of Figure 2.2a, $(0, 0)$, $(0, 0)$, is the case where all vertices have the same color, thus yielding a bad arrangement (blue-ish).

2.2.2 Genetic Algorithms

Genetic Algorithms (GAs) were first introduced by John Holland in the book *Adaptation in natural and artificial systems* [122]. They take inspiration from Darwinan evolution and chromosomal interactions, with operators such as crossover and mutation.

Genetic algorithms encode individuals – candidate solutions to the problem – as arrays. The individual encoding may be more intuitive in some cases than others, depending on the problem at hand. For the graph coloring problem, each array position may be representative of a node in the graph, and the value for that array position is the color assigned to that node in that individual. Let V be the set of all array positions, $|V|$ the number of positions in the array (i.e., string length), V_i the set of possible values for the i-th position, and $v_{i,k}$ the k-th value available for the i-th position. For the graph coloring problem, $|V| = 4$, $V_i \in \{\text{gray}, \text{green}, \text{blue}\} \forall i, i = 1, \ldots, |V|$. Figure 2.3 depicts an arbitrary encoding for the graph coloring problem.



(a) Encoding (b) Individual

Figure 2.3: On the left, encoding adopted by a GA designed to the graph coloring problem with 4 nodes. On the right, one valid individual randomly generated as a member of the initial population.

Genetic algorithms adopt the following pipeline. After defining a solution encoding, a given number of individuals $|S|$ are generated at random. The individuals have their fitness assessed by a fitness function (in this application domain, a fitness function that makes sense counts the number of edges that connect two vertices of different colors). The individual from Figure 2.3b, for example, has fitness = 2. A selection strategy (e.g., tournament selection) will select the parents for crossover, generating new offspring, following the Darwinian ideas of selective pressure within populations of species. Some of the individuals from the current population may also undergo mutation, where some of their array values may be changed at random, helping the genetic algorithm to explore the neighborhood of good solutions in the solution space. Figure 2.4 depicts the crossover and mutation operators.

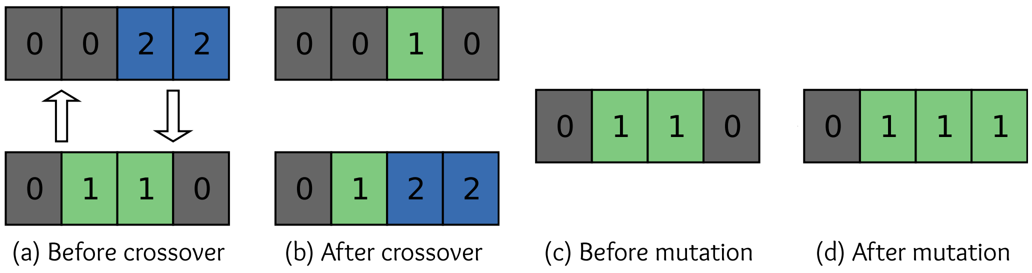


Figure 2.4: Crossover and mutation operators used in genetic algorithms.

This process continues until a termination criterion is met, usually when a predefined number of G generations have elapsed. Albeit popular, genetic algorithms suffer from several problems, including how to define the best selection strategy for mating, the most suitable strategy for crossover (one point, two points, etc), and the implicit modelling performed [117], which may prove difficult to analyze once the evolutionary process is over. In the graph coloring problem, it is easy to see how variables (i.e., vertices) interact with one another. However, for other applications the effective analysis of interactions between variables in the problem may be impractical.

Non-dominated Sorting Genetic Algorithm II (NSGA-II)

Although single-objective evolutionary algorithms may be sufficient to solve several problems, for other application domains it is sometimes required to optimize a second objective. Such an objective may be surrogate with regard to another: when designing predictive models, predictive accuracy is the first and most important objective, while interpretability may be the second. Assigning a weight for each objective, or performing a lexicographic search, may be sufficient to find a satisfactory solution [97]. However, for other problems (e.g., designing a car engine), the first objective may be energy efficiency, with the second one being engine power. In this example, no objective is surrogate to the other.

For that kind of problem, an algorithm that considers that multiple objectives have the same importance should be employed. This concept is based on the Pareto dominance of solutions [190]. Let $\Phi_j(s_i)$ be the quality of the solution s_i in the j-th objective, and $|\Phi|$ the number of objectives. In a problem in which the higher the fitness values the better, we have that:

Definition 1. Pareto-dominance – A solution s_i is said to dominate another solution s_k if $\forall j = 1, \dots, |\Phi|$, $\Phi_j(s_k) \leq \Phi_j(s_i)$, and $\exists j \in \{1, \dots, |\Phi|\}$ such that $\Phi_j(s_k) < \Phi_j(s_i)$ holds.

Definition 2. Pareto-optimal – A solution s_i is called Pareto-optimal if there does not exist any other solution that dominates it. A set of Pareto-optimal solutions is called Pareto-front.

For GAs, the most popular implementation of multi-objective optimization is the Non-dominated Sorting Genetic Algorithm II, or NSGA-II [65]. NSGA-II uses the same encoding than the GA presented in Section 2.2.2. It starts by sampling a random initial population and then assessing its performance via a set of fitness functions $\Phi_j, j \in \{1, \dots, |\Phi|\}$. It will then sort individuals based on their non-dominated rank [16].

This sorting yields a set of Pareto Fronts. Within each Pareto Front, none of the individuals dominates another, but any individual in this particular front may dominate or be dominated by individuals in other fronts. Figure 2.5 depicts a typical output for an NSGA-II iteration.

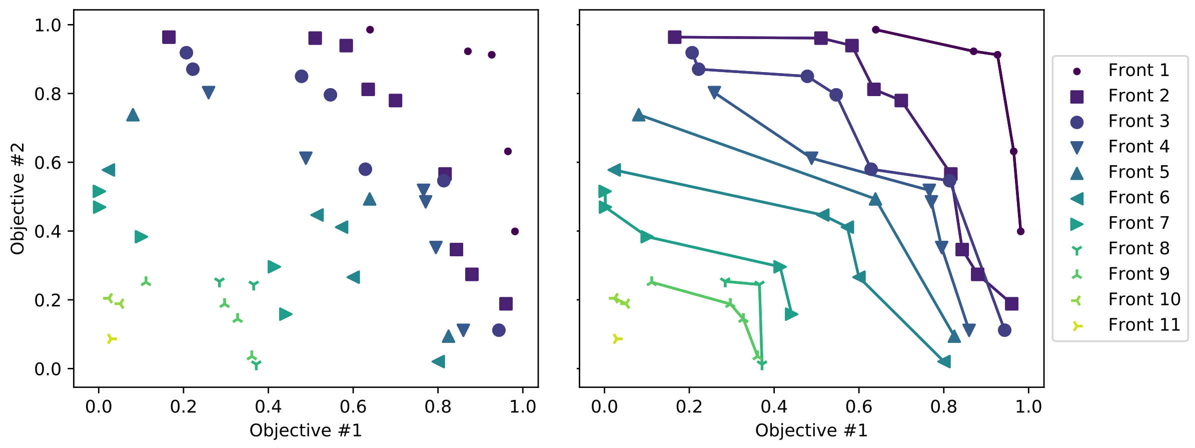


Figure 2.5: Projection of individuals regarding their performance in two objectives with NSGA-II. Individuals within the same Pareto Front are not better than their neighbors, only equivalent.

The selection algorithm for mating follows special rules in multi-objective optimization. Individuals from least-dominated fronts are preferred over most-dominated fronts, and mating individuals from the same front are also encouraged. If two individuals are in the same front, they are further sorted by the region that they inhabit within that front, with a preference for individuals that are more spread-out from the others (i.e., that are located in the least dense region). The authors use a crowding distance to determine whether regions are more or less populated, with bigger crowding distances being preferred.

Finally, once the mating process is done, individuals are selected based on their front position (e.g., first front, second front, etc.) to be a part of the next generation. This is done since elitism is present in NSGA-II, thus some individuals may go unaltered to the following generation, yielding a pool of candidates greater than the size of the population.

2.2.3 Genetic Programming

Differently from the other algorithms presented in this section, the main objective of genetic programming (GP) [83] is to generate a function or a small computer program. Even though GPs can be adapted to classification problems, the explanation of GPs given in this section does not use the coloring problem as example, using a problem that is more appropriate to GPs capabilities.

Genetic programming algorithms typically model individuals after n-ary trees, even though the implementation may be done using a traditional data structure (e.g., an array for a binary heap tree). A set of operators and operands must be defined beforehand. Assume one desires to approximate the function $2x + 1$. The defined set of operators could be $\{+, -, /, *\}$, and the set of operands could be a combination of scalar values and the input, e.g., $\{1, 2, 5, x\}$. With a set of pairs x, y given to the problem as $\{(0, 1), (1, 3), (2, 5), (3, 7), (4, 9)\}$, the GP algorithm must then randomly generate an initial population, with the maximum tree depth being an hyper-parameter. Figure 2.6 depicts two valid individuals for this initial population.

Once generated, individuals will undergo the same procedures adopted by a genetic algorithm: crossover and mutation. The mutation operator swaps one or more nodes in the tree with other operators or operands, while crossover swaps portions of the trees from two individuals. After either of the operations is performed, the GP algorithm must maintain the general rules of evolution (e.g., replacing operators that may have been placed in terminal nodes after crossover). Figure 2.7 depicts a crossover operation between the two individuals from Figure 2.6.

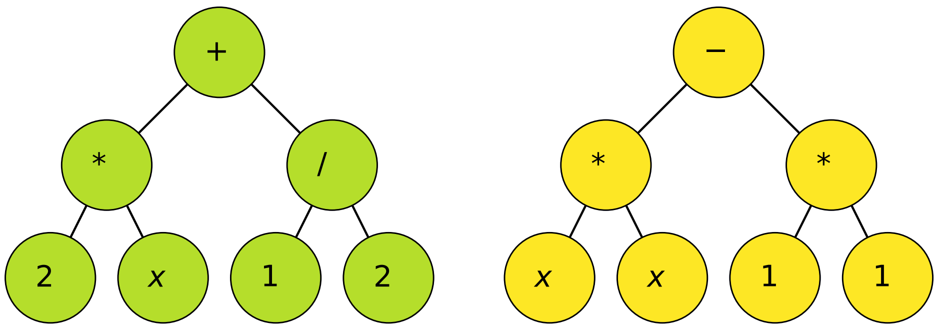


Figure 2.6: Individuals randomly generated for the first iteration of a GP with the purpose of approximating a mathematical function. Only operators are allowed in inner nodes, and only operands are allowed in terminal nodes.

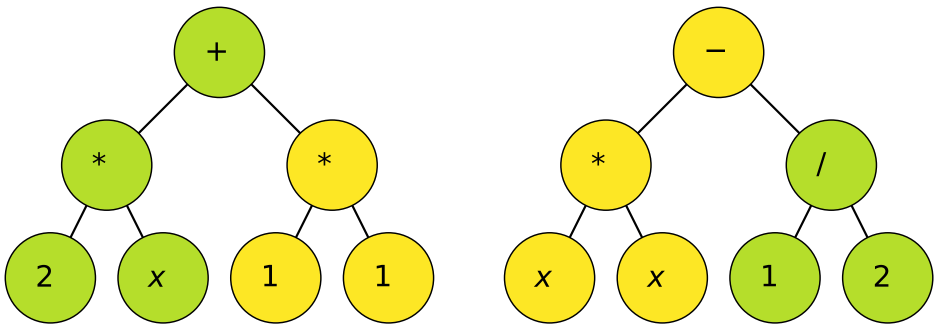


Figure 2.7: Individuals from the first population after crossover. Notice that, while not syntactically identical, the left individual is semantically equal to the objective function.

2.2.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based meta-heuristic algorithm developed by Kennedy and Eberhart in 1995 [77, 133]. PSO's individuals interact with one another, simulating how flocks of birds behave.

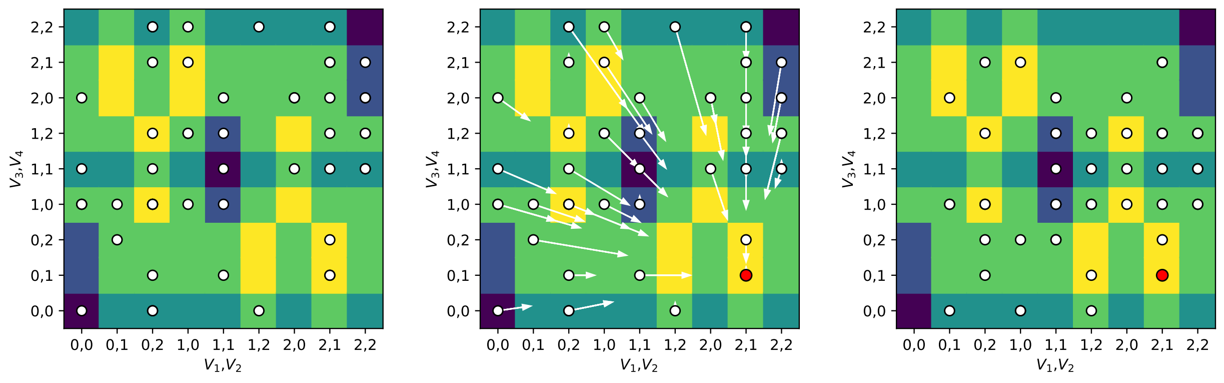
Expressions such as *individual*, *solution*, and *particle* are interchangeable while explaining how PSOs works. For the graph coloring problem, a typical PSO can use the same encoding adopted by the genetic algorithm described in subsection 2.2.2. Along the fitness of individuals, PSO also stores other metadata, such as individuals' velocity (updated at each generation); the former best position for each individual; and the position of the best overall individual so far.

The velocity of individuals is updated following the equation

$$\begin{aligned} velocity(i) = & W \times velocity(s_i) + \\ & U(0,1) \times (past\_best(s_i) - s_i) + \\ & U(0,1) \times (global\_best - s_i) \end{aligned} \tag{2.4}$$

where W is a decrease factor, s_i is the position of the current individual, $velocity(s_i)$ is the velocity of the i-th individual, $past\_best(s_i)$ the former best position of the i-th individual, $global\_best$ is the best individual found so far, and $U(0, 1)$ samples a value from the interval $[0, 1]$. The position of the i-th individual is updated by simply accelerating it towards the velocity vector. Figure 2.8 presents the updating process of a PSO solving the graph coloring problem.

This characteristic of moving individuals toward the global allows PSO to employ an effective exploration and exploitation strategy within the solution space [207]. However, it is sensitive to the hyper-parameters choice, which may increase the computational cost due to the necessity of performing a search for the best set of hyper-parameters.



(a) Initial population (b) Calculating velocity vector (c) Moving population

Figure 2.8: PSO updating its population. Since in the initial generation there is no information about the last best position for each individual, all particles move towards the global best. Notice that some individuals may be stacked upon each other.

Multi-Objective Particle Swarm Optimization

Multi-Objective Particle Swarm Optimization (MOPSO) [49] extends PSO to deal with multiple objectives that present the same importance. It does so by adopting a Pareto Dominance concept, explained in subsection 2.2.2.

Along all the variables stored by the single-objective PSO (described in subsection 2.2.4), MOPSO also stores a repository of non-dominated solutions. Particles are allowed to pursue its past best position and an arbitrary non-dominated solution, selected at each generation. The velocity of the particle is updated by

$$\begin{aligned} velocity(s_i) = & W \times velocity(s_i) \times \\ & U(0,1) \times (past\_best(s_i) - s_i) + \\ & U(0,1) \times (non\_dominated(s_j) - s_i) \end{aligned} \tag{2.5}$$

where W is a decrease factor, $velocity(s_i)$ is the velocity of the i-th particle, s_i is the current position of this particle, $U(0,1)$ samples a value from the interval $[0,1]$, $past\_best(s_i)$ is the best position found so far by the i-th particle, and $non\_dominated(s_j)$ is the j-th non-dominated particle, stochastically selected, with preference for particles in sparser regions [49]. $past\_best(s_i)$ is updated according to the Pareto Dominance of the current position and the last best position, with preference for storing the least dominated position. The set of $non\_dominated$ particles is updated at each generation.

2.2.5 Differential Evolution

Differential evolution was first introduced by Storn and Price in 1995 [244]. It primarily deals with continuous search spaces, but can be adapted for discrete domains. The encoding of individuals follows the same principle as the one presented by genetic algorithms (Section 2.2.2), using an array of size $|V|$.

Differential Evolution uses three hyperparameters: $\mathbf{CR} \in [0, 1]$, the probability that an individual will undergo crossover; $\mathbf{F} \in [0, 2]$, called differential weight; and the population size, $|S| \geq 4$. As well as other evolutionary algorithms, differential evolution samples individuals from an uniform distribution over all possible solutions. After assessing their fitness, two structures are generated for each individual, a mutation vector m and a trial vector t. The mutation vector is generated by

$$m_i = s_{r_1} + F \cdot (s_{r_2} - s_{r_3}) \tag{2.6}$$

where m_i is the i-th mutation vector, r_1, r_2 and r_3 are randomly sampled numbers from $\{1, \dots, |S|\} - \{i\}$ without replacement, and F is the differential weight. The trial vector is generated by

$$t_{ij} = \begin{cases} m_{ij} \text{ if } U(0,1) \leq \mathbf{CR} \text{ or } j = U([1, |V|]) \\ s_{ij} \text{ if } U(0,1) > \mathbf{CR} \text{ and } j \neq U([1, |V|]) \end{cases} j = 1, \dots, |V| \tag{2.7}$$

where m_{ij} is the j-th position of the i-th mutation vector, s_{ij} is the j-th position of the i-th solution, $U(0, 1)$ yields a real-valued number within interval $[0, 1]$, $\mathbf{CR}$ is the crossover probability (an hyperparameter defined by the user), and $U([1, |V|])$ samples an integer from the interval $[1, |V|]$. If the trial vector yields a better fitness than the target vector s_i, then the trial vector will replace s_i in the following generation. If this is not the case, then the target vector s_i is kept. Figure 2.9 depicts the structure generation procedure.

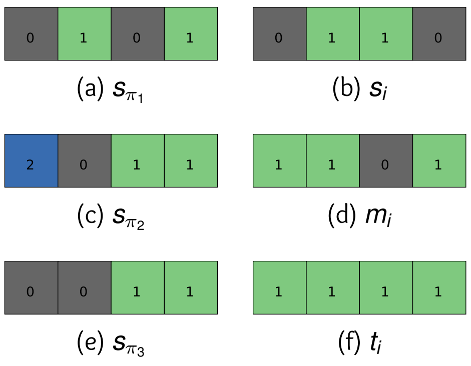


Figure 2.9: Generation of mutation and trial vectors for an individual. Mutation vector is generated by randomly selecting three other individuals s_{π_1}, s_{π_2}, s_{π_3} and applying Equation 2.6. Trial vector is generated by following Equation 2.7. For this particular case, the sampled indices were $\{2, 3, 0, 3\}$, the random values were $\{0.07, 0.29, 0.57, 0.90\}$, and with hyper-parameters $\mathbf{CR} = 0.1$ and $F = 0.5$.

Multi-Objective Differential Evolution

There are several ways to modify the single-objective differential evolution to deal with multiple objectives, as pointed out by Mezura-Montes, Reyes-Sierra, and Coello in their paper [183]. However, in this subsection we will briefly describe the first proposed approach. In the work of Chang, Xu and Quek [37], a MODE starts by uniformly sampling individuals among all possible solutions. The MODE keeps track of two subpopulations: the general subpopulation S, and a subpopulation of nondominated solutions Ψ. The later subpopulation is fulfilled as follows: in the first generation, a random individual is added to Ψ. The algorithm then proceeds to compare all other individuals from S to this single individual. If an individual s_i from S dominates any solution in Ψ, then the dominated solutions in Ψ are removed and s_i is added to Ψ. This procedure is repeated at each generation. For comparing individuals, MODE uses the Pareto Dominance concept, explained in subsection 2.2.2. In practice, what the algorithm does is to advance the first Pareto Front towards better solutions, until a termination criterion is met.

Although this procedure keeps track of the best nondominated solutions, it does not guarantee diversity among this subpopulation. This is further achieved by using the fitness sharing mechanism: individuals that lie in densely-populated areas in the solution space will have its fitness decreased, while individuals in sparsely-populated areas will have their fitness increased. How close neighbors must be to this modification to take effect is regulated through a hyper-parameter σ_{share}. Figure 2.10 depicts this concept.

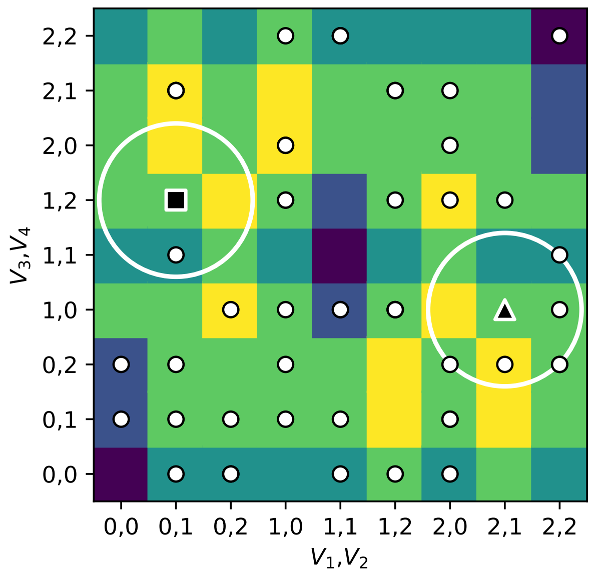
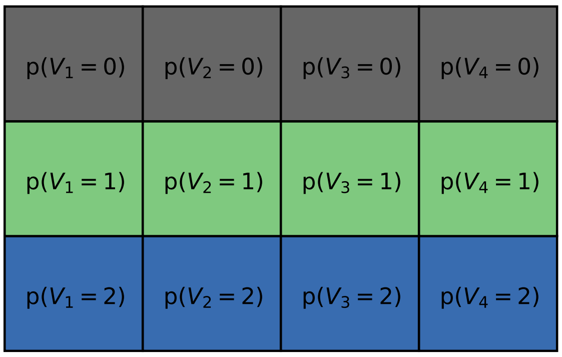


Figure 2.10: Definition of a vicinity of an individual, with radius = σ_{share}, shown here for a single objective. Although both the black square and black triangle have the same fitness, the square individual lies in a less crowded region, and thus will perform better during the selection procedure.

2.2.6 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms, or EDAs for short, were first proposed by Larrañaga and Lozano [163]. The main difference between EDAs and genetic algorithms [122] is that while GAs perform an implicit propagation of characteristics throughout evolution (i.e., by carrying-on high-quality individuals from one generation to another), EDAs will do this explicitly, by encoding those characteristics in a probabilistic graphical model [117]. A graphical model is a directed acyclic graph modeled after a Bayesian network. While graphical models are usually initiated with uniform distributions of probabilities, it is also possible to set custom probabilities that come from a previous search procedure, or based on knowledge provided by a domain expert. The structure of a graphical model is depicted in Figure 2.11.

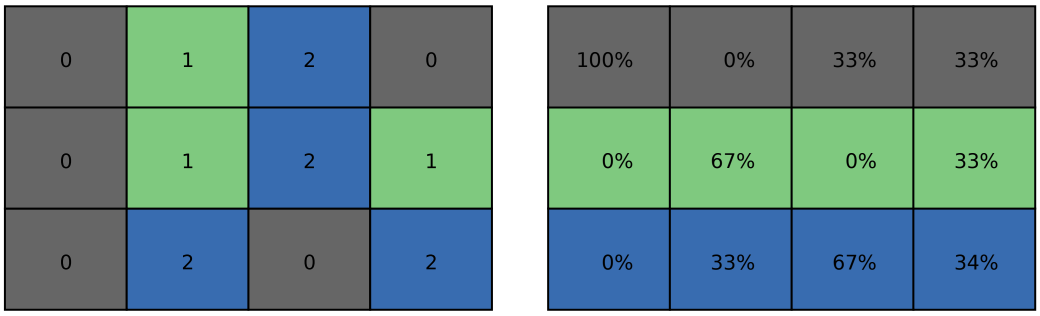


p($V_1 = 0$)	p($V_2 = 0$)	p($V_3 = 0$)	p($V_4 = 0$)
p($V_1 = 1$)	p($V_2 = 1$)	p($V_3 = 1$)	p($V_4 = 1$)
p($V_1 = 2$)	p($V_2 = 2$)	p($V_3 = 2$)	p($V_4 = 2$)

Figure 2.11: Graphical model for the graph coloring problem. A probability is assigned for each node assuming a given color. Probabilities for each node sum up to 100%, and for a new search procedure, are initiated uniformly, i.e. $p(V_i = j) = \frac{1}{3}, \forall i = 1, \dots, |V|, \forall j = 1, 2, 3$.

A typical Estimation of Distribution Algorithm starts by initializing its graphical model and sampling individuals from it. Since the probability for any given node assuming a color is initially uniform, all possible solutions are equally likely to be sampled in the first generation. The EDA then assesses the quality of each solution using a fitness function, selecting the individuals that have quality above the median fitness (or other selection scheme) to update the graphical model. The probability of a given color being sampled for a given node in the fittest population will be broadcasted to the graphical model, so in the next generation higher-quality solutions will be sampled, in theory. EDAs may have additional mechanisms for inferring the relationship between variables (i.e., drawing edges between graphical model variables). Thus, for the graph coloring problem, an EDA with this mechanism can infer the direct relationship of variables in the problem (e.g., automatically detecting that changing the color of node V_1 affects the selected color for node V_2).

This process of graphical model updating and population sampling continues until a termination criterion is met: a sufficient number of generations have elapsed; no improvement in the quality of individuals is verified; or a sufficiently high-quality individual is found. Figure 2.12 depicts the process of probability updating for the univariate graphical model (where no relationship among variables is assumed).



(a) Elite population (b) Updated graphical model

Figure 2.12: From all individuals available in a given generation, the ones in Figure 2.12a were selected for updating the graphical model. The likelihood of a color occurring in a given node in the elite population will be broadcasted to the graphical model of the next generation (Figure 2.12b).

Multi-Objective Estimation of Distribution Algorithms

There is a set of strategies for dealing with multiple objectives with Estimation of Distribution Algorithms, as shown in the survey paper of Hauschild and Pelikan [117]. The shown strategies, however, do not substantially differ from NSGA-II (presented in Section 2.2.2) and MODE (Section 2.2.5): whether using a custom-made selection strategy for preserving diversity among population, or keeping track of a subpopulation of Pareto-optimal solutions. One novel strategy, however, is presented in the Multi-Objective Hierarchical Bayesian Optimization Algorithm (mohBOA) [208]. After sampling, solutions are ranked based on their crowding distance (i.e., good solutions that share little similarities with other solutions are preferred). After using the same selection criterion as NSGA-II, selected solutions are clustered with k-means. Each cluster will generated a probabilistic graphical model, and new solutions will be sampled from these graphical models, replacing solutions marked for replacement.

2.3 Interpretability

Interpretability, in the context of machine learning, is the capability of a model to allow its predictive process to be explained. Other terms used in the literature include comprehensibility, understandability, and explainability; although interpretability is more frequently used, and all terms are often used interchangeably [35].

Interpretability of machine learning models is an important feature for a wide range of applications, such as medicine, credit scoring, bioinformatics, and churn prediction [98]. For those end-users, being able to trust predictions of generated models often comes in hand with the ability to understand them. This feature is critical in scenarios where models can lead to life-or-death decisions (such as in medicine), or influence decisions that can put several lives at risk, such as the use of recommendation algorithms in a nuclear power plant [98]. More recently, being able to interpret machine learning models also gained attention from different segments of society, from the academic community (in the form of workshops on model comprehensibility) to governments (e.g., the *Data Protection Regulation* from the European Union, which includes a “right to explanation" in its text) [103].

There is no mathematical definition of interpretability [35]. Non-mathematically, literature surveyed by Carvalho et al. [35] seems to agree that interpretability is defined in human terms: the easier for a human to grasp a model’s predictive process, the more interpretable the model. According to Belle and Papantonis [20], transparency (which here we will treat as a synonymous to interpretability) can be viewed according to three approaches: simulatability, decomposability, and algorithmic transparency. Simulatability refers to the capability that a human can simulate the predictive process of a model, given its inner workings. This concept is elastic, since not all decision trees, for example, would be computable by a human by thought [20]. On the other hand, a neural network with no hidden layers could fall into this category. Decomposability “denotes the ability to break down a model into parts (input, parameters and computations) and then explain these parts” [20]. Finally, Algorithmic transparency refers to “the ability to understand the procedure the model goes through in order to generate its output” [20]. Neural networks, for example, do not satisfy this requirement: even experts may struggle to run a single backward procedure, a process that is repeated several times within a neural network training, and that is used to adjusted the neural network’s weights.

Interpretability, while already-experimented in machine learning for some time (CART [25] and C4.5 [213] decision tree algorithms are, respectively, from the 80s and 90s), remains as one of the least researched characteristics of machine learning algorithms [35], partly because it is associated with white-box models, which are not always the interest of research for several groups [103]. A greater focus is devoted to producing black-box models with ever-increasing predictive performance, disregarding interpretability [35]. However, interpretability is finally gaining more traction in the machine learning community, with more congresses and journals focused on XAI (eXplainable Artificial Intelligence) being proposed in recent years [35]. Additionally, the fact that machine learning systems are deployed to high-stakes applications, with severe consequences in case of misjudgments, calls for greater accountability. Examples of high-stakes applications include “people incorrectly denied parole, incorrect bail decisions leading to the release of potentially dangerous criminals, pollution models stating that dangerous situations are safe”, just to name a few [35].

When researched, interpretability is often measured in terms of model complexity – e.g., the number of rules in a rule-based classifier, or the depth of a decision tree [103]. However, this is not what an end-user might seek in a model to deem it comprehensible. Let us take the example of decision trees, a popular white-box model for the problem of predicting whether a person is eligible for being a goalkeeper in a soccer club. For a decision tree with height 3, using parallel decision boundaries (that is, using a single attribute, e.g., age ≤ 22, height $\geq 185cm$) may increase human comprehensibility though at the expense of predictive performance. Conversely, oblique trees can use the combination of two or more attributes (e.g., (age ≤ 22) $\wedge$ (height $\geq 185cm$)) to have trees with the same height and better predictive performance, but at the expense of comprehensibility. In this scenario, tree height is certainly not a good proxy to model comprehensibility.

On the other hand, enforcing monotonicity constraints (provided by either the user or a domain expert) may lead to better models [98, 103]. Monotonic relationships describe the relation between increasing the value of a given attribute (for example, the age of a patient) to the probability of the instance with that attribute value belonging to a given class (for example, the probability that the same patient will present some kind of cancer). For a decision tree, enforcing that the tree is monotone (i.e., it respects all the monotonicity constraints) can be in the form that no branching test leads to counter-intuitive results (in the above example, stating that an *increase* in age leads to a *decrease* in the probability of having cancer). These constraints may be used in a soft-constraint fashion if breaking the constraints leads to better accuracy.

As defined by Carvalho et al. [35], interpretability can be pursued in three ways: pre-model, in-model, and post-model. Pre-model is achieved through an interpretation of the dataset; in-model refers to using models that are inherently interpretable (the so-called white-box models); finally, post-model interpretability refers to (i) interpreting the inner workings of a white-box model, such as analyzing the weights of a logistic regression; or (ii) applying a model-agnostic method to interpret model predictions.

We will briefly review white-box models, and strategies to make black-box models interpretable, as these are the main methods employed in this thesis. The interested reader is referred to the surveys of Belle and Papantonis [20], and Carvalho et al. [35] for an in-depth review of these approaches. Figure 2.13 shows the taxonomy proposed by Belle and Papantonis [20] on approaches to develop interpretable systems.

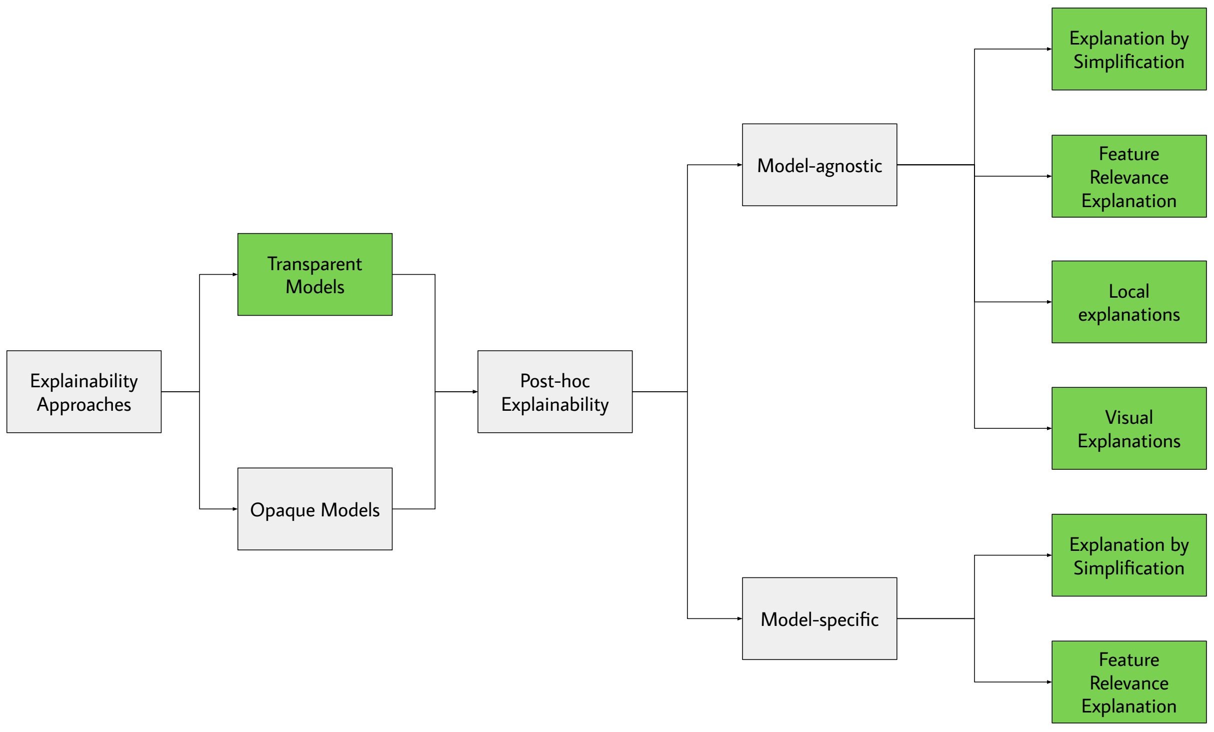

Explainability Approaches
Transparent Models
Opaque Models
Post-hoc Explainability
Model-agnostic
Model-specific
Explanation by Simplification
Feature Relevance Explanation
Local explanations
Visual Explanations
Explanation by Simplification
Feature Relevance Explanation

Figure 2.13: Taxonomy proposed by Belle and Papantonis on approaches to develop interpretable machine learning systems. Green darker boxes denote interpretable models. Adapted from [20].

2.3.1 White-box models

White-box models are generated by classifiers that fulfill at least one of the three transparency dimensions proposed by Belle and Papantonis [20]: simulatability, decomposability, and algorithmic transparency. One of the challenges of pursuing interpretability with white-box models (e.g., rule lists, decision trees) is that they present, on average, inferior predictive performance than black-box models (e.g., neural networks, Support Vector Machines) [34]. This presents a challenge: while increased predictive accuracy is required in mission-critical applications, such as healthcare [98], the ability to review the predictive process is also important.

Even though Belle and Papantonis [20] make a distinction between decision trees and rule-based classifiers – and we agree with that distinction –, we also consider that several classifiers steer from the same generic framework: decision trees can be, after all, translated as a sequence of rules, as it is done with the PART algorithm [96]. Also, decision tables can also be converted into rules [123].

The play tennis dataset, shown in Table 2.2, will be used as model for all reviewed white-box models, allowing the reader to draw a comparison between data representation between them.

Table 2.2: Play Tennis dataset.

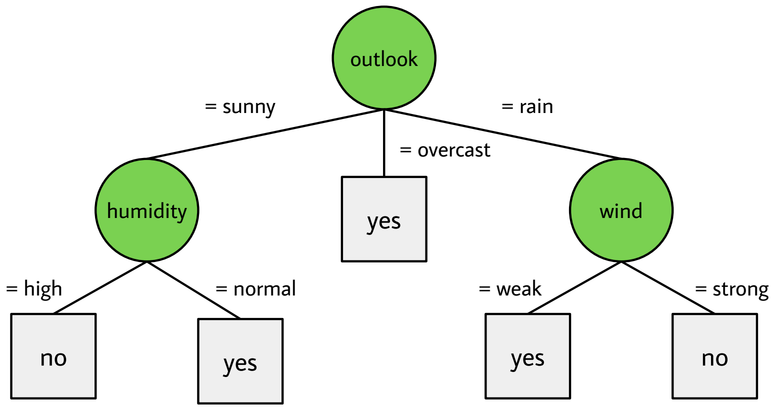
Outlook	Temperature	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Decision trees

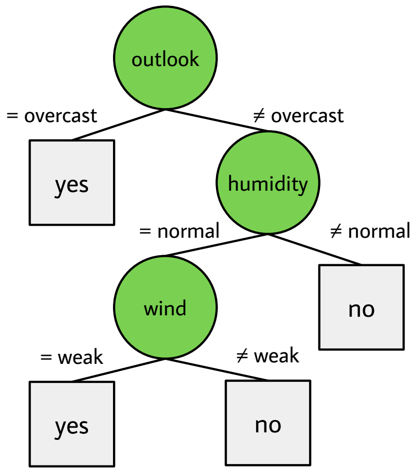
Decision trees can be used for both classification and regression. Decision trees, while differing in the branching factor (binary, *n*-ary), decision functions in inner nodes (parallel, oblique), purity function (Gini Index, Entropy, Information Gain), and a few other hyper-parameters, have all the same basic properties: a root node, which splits the training data using a rule over features (according to a purity metric); inner nodes, which further split the training data according to other rules over features; and leaf nodes, the nodes that will assign a value (for regression) or label (for classification) to incoming data.

Decision trees are widely used in applications where interpretability is paramount, in fields such as medicine and finance [123]. Unsurprisingly, decision trees fulfill most, if not all, of the transparency criteria of Belle and Papantonis [20]: if a decision tree is sufficiently small (i.e., few leaf nodes, where few is subject to the human capability of memorizing decision paths), then it is simulatable; if the tree is large but its features are understandable by a human user, then it is decomposable; finally, if the features are not understandable, then it is still algorithmically transparent.

In this thesis, we use two decision tree algorithms: C4.5 [213] and CART [25]. The main differences between the two, apart from specific implementation details, are the splitting criteria (C4.5 uses the gain ratio while CART uses the Gini Index) and the splitting arity (C4.5 makes binary splits for numeric attributes, and *n*-ary splits for categorical features, whereas CART makes binary splits for all types of attributes) [211]. Both algorithms support pruning. C4.5 is also referred to as J48 in the Weka Toolkit implementations [113], while CART is known as SimpleCart. In Figure 2.14, we depict a comparison of decision trees induced by J48 and SimpleCart for the play tennis dataset.



(a) J48



(b) SimpleCart

Figure 2.14: Decision trees induced by J48 and SimpleCart algorithms of the Weka Toolkit [113].

Rule-based classifiers

There are two types of rule-based classifiers: rule lists and rule sets. While both types of algorithms iteratively build models by adding rules to the system, the *triggering* mechanism is different. In a rule-list procedure, the learning algorithm starts with the entire training set and finds a rule that is both (i) the simplest, (ii) covers most examples, and (iii) has the best metric value (e.g., accuracy). This rule is then added to the list, and all covered examples are removed from the training set (note that covered examples will be removed regardless of being correctly labeled). This procedure is repeated until all examples are covered, even if a default rule needs to be added at the end of the list. The list can be pruned afterwards by a different procedure.

A rule-set building algorithm, on the other hand, does not simply finds the rule that covers most examples, but instead the rule that covers most *positive* examples, the positive class being iterated over all class labels. It also does not remove all covered examples, but only those correctly classified. This procedure is repeated until all examples are covered. As it can be seen, more than one rule can cover different examples, hence a triggering policy is defined: it can be either the first rule to be triggered (which prioritizes simpler rules), or assigning a voting weight to each rule and averaging out the class.

In this thesis, we use two rule-list algorithms as base classifiers: RIPPER [50] (also known as JRip in the Weka Toolkit [113] implementation) and PART [96]. The main difference between them is that, while RIPPER builds a rule list as described above, PART does so by converting a C4.5 decision tree to a rule list. Using RIPPER [50] on the play tennis dataset generates the rule list of Figure 2.15.

- **if** *outlook* = *overcast* **then** *play* = *yes*
- **if** *humidity* = *high* **then** *play* = *no*
- **default:** *play* = *yes*

Figure 2.15: Rule list generated by RIPPER over the play tennis dataset.

EDNEL, one of the contributions of this thesis (described in Chapter 6) also uses a modified version of the CN2 algorithm [47] for generating unordered rule sets from decision trees and decision tables. The procedure is similar to the one explained above, except for the fact that rules are directly extracted from these algorithms instead of being evolved.

Decision tables

Decision Tables [143] are quite different from rule-based classifiers, even though they steer from the same learning paradigm, rule-based learning. Decision tables have four quadrants, which will be explained using Figure 2.16 as an example. In the leftmost top quadrant, the variables used for classification are depicted – in this example, outlook and humidity. The rightmost top quadrant has values paired for each variable within the rows. A dash (–) denotes that any value for that variable is acceptable. In the leftmost bottom quadrant, the class attribute values are depicted: yes (do play tennis today) and no. Finally, in the rightmost bottom quadrant, the check marks denote which class would be assigned for an instance that meets all the criteria in the header rows. Since we are using single-hit decision tables, there are at most one check mark for each column. If no check mark is present for a given column, then the default rule should be assumed. A valid conversion from the decision table of Figure 2.16 would be the rule list of Figure 2.15.

Outlook	Sunny	Overcast	Rain	–	–
Humidity	–	–	–	Normal	High
play = yes		✓			
play = no					✓

Default: play = yes

Figure 2.16: A decision table built for the play tennis dataset. Note that it provides a default rule.

2.3.2 Black-box models

Black box models are, by definition, non-interpretable. A machine learning practitioner can, for example, inspect the neurons, weights, and connections of a neural network, but that alone will help in interpreting the decisions performed by the model.

However, this limitation does not forbid one from trying to understand the inner workings of a model that falls in this category. As defined by Belle and Papantonis [20], several techniques can be applied to explain black-box models. We will briefly detail two of them: explanation by simplification and feature relevance explanation.

Explanation by simplification consists in converting a black-box model to a white-box model. We will shortly explain how the LIME algorithm works [219], which is a model-agnostic technique. After inducing a black box model, a white-box model (e.g., decision tree, linear model) is approximated for a given region of the input space that needs explanation (i.e., the white-box model does not approximate all predictions from the black box model, only a part of the entire space). Once trained, this white-box model, which has features that are not necessarily the ones used for training the black-box model, can explain a portion of the input space that is close to the instance that is being explained.

For Random Forests, neural networks, and support vector machines, one can extract a rule set, thus turning an opaque model into a transparent one [123]. EDNEL, one of our evolutionary algorithms, performs this kind of adaptation (as explained in Section 6.1.1) to decision trees and decision tables. We also perform this procedure on Random Forest, explained in Section 6.2.1. One should note that the predictive performance of Random Forests decreases with the adoption of this technique, as it is shown in Section 6.2.2. Also note that there is no boundary in the number of rules to be extracted, which can decrease the interpretability of the provided white-box model [123].

Extracting explanations (or converting black-box models to white box) might not be appropriate for all approaches. Consider an white-box model that explains a Random Forests model 90% of the time. When the explanation model agrees with the predictive model, there are no problems; however, what should we do when they disagree? Should we scrap the predictions, since we do not understand them, or should we trust the explanation, even though it is not explaining anything present in the predictive model? [227]

Feature relevance seeks to verify how much a feature is contributing to the prediction of a given instance. Currently, one of the most popular methods is SHAP (SHapley Additive exPlanations) [176], which in turn is similar to LIME [219], in that it also builds a linear model around the instance that is required to be explained. SHAP is grounded in Game Theory, and presents Shapley values – roughly speaking, how much a feature contributes to the model's prediction given all combinations of features [20].

An alternative strategy, dependent on the model, is using Random Forests for verifying how much each attribute contributes to the general predictions. This is a built-in functionality of the algorithm. For each feature in the dataset, we check on how many internal nodes (over all decision trees in the forest) it is present, and average how much using that feature decreases the impurity of the split. We then rank attributes based on this average, which explains which attributes make the most contributions to classifying instances.

Regarding ensemble learning, there are some studies that focus on explaining the predictive process of such models. In [89], an ensemble is reduced to its most representative model, i.e., the base model that shares most characteristics with the entire ensemble. Evidently, the base learner must be comprehensible in essence, and for this reason the authors use decision trees. Similarly, in [257], after an ensemble is built, a decision tree is trained with the objective of encoding the knowledge of the ensemble, while also being comprehensible.

2.3.3 Evaluating Interpretability

As discussed earlier, there is no mathematical definition of interpretability. This is not to say that some authors have not tried to design proxy measures to evaluate it. Regarding decision trees for classification, Piltaver et al. [209] perform an empirical evaluation by applying a survey to human users with different trees in order to understand which metrics best describe interpretability. The aspects that are analyzed include branching factor (how many children nodes an inner node spawns), maximum depth, and number of leaf nodes. The authors find that, for moderately large trees (e.g., trees with 4 to 20 nodes, among other characteristics) the features that contribute the most to time-to-answer (a proxy metric to interpretability) are the number of leaves (the fewer the better), branching factor (also smaller is better), and the depth of the deepest leaf node. Regarding answer-correctness, the depth of the deepest node and using negated statements in inner nodes has the most negative impact (i.e., decrease in accuracy).

Similarly, Vanthienen et al. [123] evaluated decision tables, rule-based classifiers, and decision trees, also finding out that simpler models (number of rules, number of inner nodes, and number of rows/columns for rule-based classifiers, decision trees, and decision tables, respectively) imply in faster time-to-answer when used by humans, while also implying in greater confidence in the models, and accuracy. Additionally, the authors found that, among the visual models (i.e., decision tables and decision trees), decision tables were preferred among human users. The authors speculate that this may be because tables make a better use of computer screens than decision trees.

Piltaver et al. [209] and Vanthienen et al. [123] empirically demonstrate that simpler models are good proxies to interpretable models. In the algorithms proposed in this thesis, however, we do not enforce this simplicity due to the following reasons: (i) while a larger model may be less interpretable than a smaller model, the comparison is more complicated when two models with the same number of leaves, for instance, are compared. Additionally, enforcing simpler models will have an impact on the predictive performance of the models, which might not be desired in all application domains; and (ii) there is no way to bring human participants to subjectively evaluate interpretability during an evolutionary algorithm run. Instead, it appears to be simpler to let practitioners decide which models are more useful to them once the evolutionary algorithm finishes its process. Since the evolutionary algorithms proposed in this thesis (PUMA and EDNEL, presented in Chapters 5 and 6, respectively) allow the user to select between two models at the end of the process, the user can then decide which one suits best their needs.

2.4 Auto-Machine Learning

Automated (or Automatic) Machine Learning, known simply as Auto-ML, is the machine learning area concerned with performing automatic selection/recommendation of algorithms, as well as optimization of their hyper-parameters [271] in order to provide a model that has the best possible generalization performance for a given scenario. Auto-ML has recently gained attention due to the capability of sparing the end user from a manual optimization of hyper-parameters in a scenario of several machine learning algorithms, which can be a repetitive and tiresome task, while often requiring advanced domain-specific knowledge [90,194,271]. Auto-ML has been addressed under the perspective of Bayesian optimization [90,249] and more recently with evolutionary algorithms [15,193,194,271].

2.4.1 Formal Definition

Auto-ML can be formally defined as a CASH problem: combined algorithm selection and hyper-parameter optimization [90, 249]. As the name denotes, it consists of two sub-problems. The first one, model selection, is based on having a set of algorithms $\mathcal{A} = \{A^{(1)}, A^{(2)}, \dots, A^{(k)}\}$, annotated input data $(\mathbf{X}, Y) = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(N)}, y^{(N)})\}$, and the need of selecting algorithm

$A^* \in \mathcal{A}$ that provides the best generalization performance. The generalization performance is approximated by training the algorithm in a training set $\mathbf{X}_{\text{train}}$ and evaluating its predictive performance in a validation set $\mathbf{X}_{\text{val}}$, with training and validation sets being two disjoint sets of the full set. The model selection problem is then defined as

$$A^* = \underset{A \in \mathcal{A}}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A, \mathbf{X}_{\text{train}}, \mathbf{X}_{\text{val}}) \tag{2.8}$$

where $\mathcal{L}(A, \mathbf{X}_{\text{train}}, \mathbf{X}_{\text{val}})$ is the loss function (e.g., error rate) of A when trained on the training set and evaluated on the validation set, and k is the number of folds of a cross-validation procedure.

Selecting the hyper-parameter set $\lambda \in \Lambda$ of a given algorithm A that optimizes its predictive performance is similar to performing model selection, with some small differences: hyper-parameters may be continuous, may fall onto a high dimensional space, and also may be correlated among themselves [249]. Considering that an algorithm may have l hyper-parameters $\lambda_1, \dots, \lambda_l$ with associated domains $\Lambda_1, \dots, \Lambda_l$, the hyper-parameter space $\mathbf{\Lambda}$ is a subset of combinations of these domains, $\mathbf{\Lambda} \subset \Lambda_1 \times \cdots \times \Lambda_l$. Then, the problem of selecting the best set of hyper-parameters $\boldsymbol{\lambda}^*$ can be defined as

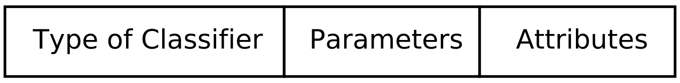
$$\boldsymbol{\lambda}^* = \underset{\lambda \in \Lambda}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_\lambda, \mathbf{X}^{(\mathbf{i})}_{\text{train}}, \mathbf{X}^{(\mathbf{i})}_{\text{val}}) \tag{2.9}$$

Once both model selection and hyper-parameter optimization have been defined, the CASH problem is reduced to simply finding the best algorithm for a given task, given that it is already using its best set of hyper-parameters:

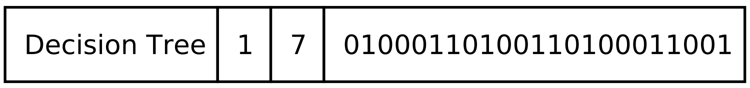
$$\mathbf{A}^*\boldsymbol{\lambda}^* = \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \mathbf{\Lambda}^{(j)}}{\arg\min} \frac{1}{k} \mathcal{L}(\mathbf{A}^{(j)}_{\boldsymbol{\lambda}}), \mathbf{X}^{(i)}_{\text{train}}, \mathbf{X}^{(i)}_{\text{val}}) \tag{2.10}$$

2.4.2 Addressing the CASH problem

Historically, automatic optimization has been performed in a localized fashion: only some aspects of an algorithm are optimized, as opposed to performing a combined, global-orientated search. Hyper-parameter optimization, for example, has gone from brute-force grid-search to random exploration of the solution space [193]. Conversely, in ensemble learning most reviewed work (presented in Chapter 3) optimize strategies of ensemble induction individually. Some exceptions include the work of [229], in which model selection, hyper-parameter optimization, and feature selection are performed in an integrated way for identifying cancerous micro-RNA markers. Individuals are generated in a Pareto-front fashion and are combined via majority voting. A depiction of the individuals' encoding of this algorithm is presented in Figure 2.17.



(a) Encoding



(b) Individual

Figure 2.17: (a) Encoding and (b) individual used in [229]: model selection, hyper-parameter optimization, and attribute selection are performed simultaneously.

While [229] proposes an algorithm that addresses two ensemble learning stages (namely selection and generation), it does not address model integration, resorting to a simple majority voting scheme. To the best of our knowledge, even though there are several EA-based systems that address this problem by considering a search space with many types of supervised learning algorithms (e.g., [62, 193]), there are only two studies using EAs to address this problem by considering a search space focused on ensembles [144, 271], as detailed below.

In [271], different base learners can be chosen from a determined set of learners, and then have their hyper-parameters tuned using an Estimation of Distribution Algorithm. However, the work automates only part of the ensemble learning procedure, more specifically the generation stage; it does not address whether strategies from other stages (namely selection and integration) can improve the predictive performance of the entire ensemble. In [144], a genetic programming strategy is used to generate what the authors call templates: configurations of base learners (logistic regressors; neural networks; support vector machines; naïve bayes classifiers; decision trees; and k-nearest neighbors base learners) and ensembling strategies (bagging; boosting; stacking; cascade generalization; delegating; cascading; arbitraring). With this strategy, a boosting ensemble can be interpreted as the base model of a higher-degree bagging ensemble, for example. Though this work employs a wide variety of base learners and ensemble algorithms, those algorithms are already pre-defined from a fixed list; for example, boosting performs only instance sampling, and it is not capable of performing attribute selection, even if that would be beneficial for the entire system.

2.5 Summary

This chapter briefly reviewed the main topics that permeate this thesis. Considering that we propose new evolutionary algorithms for inducing ensembles of interpretable classifiers, which is an auto-machine learning problem, the topics covered by this chapter were:

- Ensemble learning and some well-established non-evolutionary ensemble learning algorithms, reviewed in Section 2.1;

- A brief review on some of the most popular evolutionary algorithms in the literature according to our survey on evolutionary algorithms for ensemble learning (Chapter 3), including Estimation of Distribution Algorithms, which is the type of evolutionary algorithm used in all of our proposed methods (Section 2.2);
- Interpretability and correlated topics: how to measure it, examples of interpretable models, discussion on new research trends, etc. (Section 2.3);
- Auto-machine learning, which is the problem that is being addressed by this thesis; formal definitions and an non-exhaustive review of related work of evolutionary algorithms for ensemble learning that address this problem (Section 2.4).

The next chapter (Chapter 3) will present an extensive review on related work that make use of evolutionary algorithms for ensemble learning, which is more akin to the algorithms proposed in this thesis as opposed to the more generic topics covered in this chapter.

3. ENSEMBLE LEARNING WITH EVOLUTIONARY ALGORITHMS

This chapter presents a literature review that was conducted to evaluate how evolutionary algorithms could be employed for supervised ensemble learning. Several comprehensive surveys have been published aiming at analyzing ensemble learning from different perspectives. Regarding the specific analysis of EAs for supervised ensemble learning, Yao and Islam [274] are the only authors to present a review of EAs for designing ensembles, though they focus only on artificial neural networks as the base learners to be combined. Sagi and Rokach [228], as well as Dietterich [71] present a general review of ensemble learning studies, discussing the challenges and trends of traditional non-evolutionary methods. Rokach [223], Kotsiantis [145], Tabassum and Ahmed [246], in turn, focused on reviewing ensembles designed only for classification tasks. Similarly, Mendes-Moreira et al. [182] and Vega-Pons et al. [260] review ensemble methods focusing only on regression and clustering tasks, respectively. There are also papers on specific domain applications of ensembles, such as the work of Athar et al. [9], which reviews the leading ensemble approaches for sentiment analysis; and the studies by Gomes et al. [110] and Krawczyk et al. [147], which review ensemble learning for data stream classification and regression.

Despite the relevant contributions of the previously cited literature, we deemed necessary to perform a new survey since it is, up to our knowledge, the first review to focus on general-application EAs for supervised ensemble learning in a comprehensive fashion. In particular, we highlight the following contributions: i) we provide a general overview of EAs for supervised ensemble learning, not exclusively focusing on any specific EA or any given type of supervised model, but presenting an in-depth analysis of the different algorithms proposed for each stage of ensemble learning, with their respective advantages and pitfalls; and ii) we provide a detailed taxonomy to properly categorize supervised evolutionary ensembles, helping the reader to filter the literature and understand the possibilities when designing EAs for this task. Note that reviewing EAs for ensemble learning in unsupervised settings (e.g., the clustering task) is out of the scope of this survey.

The rest of this chapter is organized as follows. Section 3.1 presents the methodology adopted for searching related work in well-known online repositories. Section 3.2 presents the novel taxonomy to categorize EAs designed for supervised ensemble learning. Sections 3.3, 3.4, and 3.5 review the EAs employed for the three stages of ensemble learning: generation, selection, and integration. Section 3.6 details common fitness functions employed by the EAs for optimizing the ensembles. Section 3.7 summarizes the types of EAs used in the aforementioned stages, and Section 3.8 points to the most common base learners within the proposed approaches. Section 3.9 presents the application domains in which EAs for supervised ensemble learning are often employed.

3.1 Methodology

The main objective of the survey is to identify published work that apply evolutionary algorithms to learn ensembles of predictive models, for supervised machine learning. The objective is expressed from the research questions presented in Table 3.1. These questions aim to analyze the relevant work both in the context of evolutionary algorithms used and the characteristics of the ensemble that are optimized.

Table 3.1: Research questions of the survey.

ID	Research Question	Description
RQ1	What are the existing approaches that apply evolutionary algorithms in learning ensembles for supervised machine learning?	General question that aims to identify existing work that apply evolutionary algorithms in the context of ensemble learning.
RQ2	What are the evolutionary algorithms used to learn the ensembles?	Aims to identify which evolutionary algorithms are applied for ensemble learning.
RQ3	What stages of ensemble learning are addressed by the evolutionary algorithm?	Aims to categorize the approaches according to the ensemble optimization step (generation, selection or integration).
RQ4	Which objective functions are optimized by the evolutionary algorithm?	Since fitness function is an essential component of EAs, and given the complexity of the ensembles where several objectives can be optimized, this question aims to analyze how these functions are employed in ensemble learning task.
RQ5	What are the base learners used?	Finally, this survey aims to analyze the relevant work from the point of view of the base learners that are used to compose the ensembles.

We have searched the following repositories: Scopus[1], Science Direct[2], IEEE Xplore[3] and ACM Digital Library[4]. Based on the main objective, we select keywords that are likely to be present in most of the work that proposes evolutionary algorithms for ensemble learning; and from these keywords we compose a search string. Synonyms of each term were incorporated using the Boolean operator *OR*, whereas the Boolean operator *AND* was used to link the terms. The generic search string derived is

```
'ensemble' AND
(('classification' OR 'classifier' OR 'classifiers') OR
('regression' OR 'regressor' OR 'regressors')) AND
('evolutionary' OR 'evolution')
```

A list of search strings used for each search engine is presented in Table 3.2.

The search is divided into two rounds. The first round was executed in the year of 2017, and the second round in the year of 2018, the latter to cover papers published during the first reviewing round. In the first round, 680 papers matched the keywords. All those papers had their abstract

[1]Available at https://www.scopus.com/home.uri. Accessed June 12 2017.
[2]Available at http://www.sciencedirect.com. Accessed June 12 2017.
[3]Available at http://ieeexplore.ieee.org/Xplore/home.jsp. Accessed June 12 2017.
[4]Available at http://dl.acm.org. Accessed June 12 2017.

reviewed, and from this reading 343 were deemed relevant to the survey elaboration, as shown in the third column of Table 3.3.

Table 3.2: Search string used in the repositories.

Scopus

```
TITLE-ABS-KEY("ensemble") AND (
    (
        TITLE-ABS-KEY("classification") OR
            TITLE-ABS-KEY("classifier") OR
            TITLE-ABS-KEY("classifiers")
    ) OR (
        TITLE-ABS-KEY("regression") OR
        TITLE-ABS-KEY("regressor") OR
        TITLE-ABS-KEY("regressors")
    )
) AND (
    TITLE-ABS-KEY("evolutionary") OR
    TITLE-ABS-KEY("evolution")
)
```

ACM Digital Library

```
"ensemble" AND (
    (
        "classification" OR
        "classifier" OR
        "classifiers"
    ) OR (
        "regression" OR
        "regressor" OR
        "regressors"
    )
) AND (
    "evolutionary" OR
    "evolution"
)
```

IEEE Xplore

```
"Abstract":ensemble AND (
    (
        "Abstract":classification OR
        "Abstract":classifier OR
        "Abstract":classifiers
    ) OR (
        "Abstract":regression OR
        "Abstract":regressor OR
        "Abstract":regressors
    )
) AND (
    "Abstract":evolutionary OR
    "Abstract":evolution
)
```

ScienceDirect

```
title-abstr-key("ensemble") AND (
    (
        title-abstr-key("classification") O
        title-abstr-key("classifier") OR
        title-abstr-key("classifiers")
    ) OR (
        title-abstr-key("regression") OR
        title-abstr-key("regressor") OR
        title-abstr-key("regressors")
    )
) AND (
        title-abstr-key("evolutionary") OR
        title-abstr-key("evolution")
)
```

Since Scopus is the largest database, some of the papers indexed in the remaining three repositories were already present in Scopus. The amount of already indexed papers is counted in column *Indexed* from Table 3.3. The number of reviewed papers in the second round is shown in Table 3.4.

Table 3.3: Number of papers which had its abstract reviewed in the first round.

Repository	Search Date	Relevant	Irrelevant	Indexed	Total
Scopus	April 26 2017	343	337	——	680
ACM Digital Library	June 7 2017	14	4	34	52
IEEE Xplore	June 7 2017	4	4	121	129
Science Direct	June 8 2017	4	39	78	121
Total		**365**	384	233	982

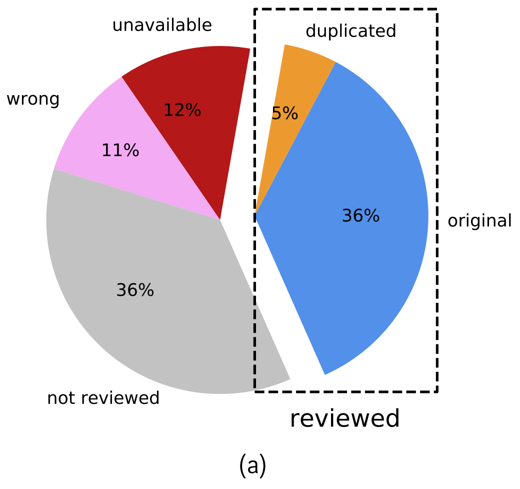
Table 3.4: Number of papers which had its abstract reviewed in the second round.

Repository	Search Date	Relevant	Irrelevant	Indexed	Total
Scopus	March 29 2018	23	99	——	122
ACM Digital Library	April 27 2018	2	7	0	9
IEEE Xplore	April 4 2018	8	4	6	18
Science Direct	June 16 2018	5	12	12	29
Total		**38**	122	18	178

Among both rounds, 403 papers were deemed relevant for the survey. From those, 163 were reviewed and included in the survey. Additionally, one work authored by us and published in 2018 was also included, and is reviewed in greater detail in Chapter 4.

From the 164 reviewed papers, 20 were duplicated and fell into one of the following categories: (i) the algorithms were published in conferences and had expanded versions in journals; (ii) different application domains but the same algorithm; or (iii) slightly different implementations (for example, changing the number of layers and/or activations in a neural network).

Reasons for papers not being added in the survey include unavailability (paper not available in any online repository, or papers available only under payment) (50 papers); and wrong topic (on further review, papers that did not cover the surveyed topic) (43 papers). Finally, 147 papers were found but not reviewed due to a truncation in the reviewing process (that is, we deemed that only papers before a given date in 2017 would be reviewed). An overall summary of the papers is presented in Figure 3.1a, while the distribution of papers among the years they were published is presented in Figure 3.1b.



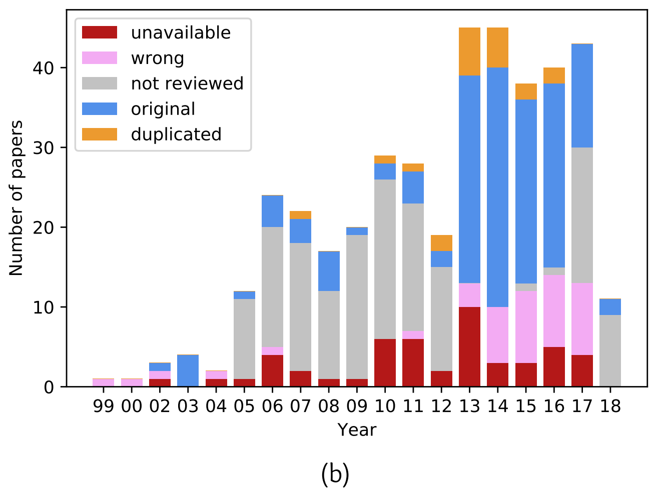


Figure 3.1: (a) From the 404 papers selected for review, 164 were added to the survey. Among these, 20 were duplicated (e.g. expanded work), and 144 original work. (b) Distribution of papers over the years they were published.

3.2 Taxonomy

We provide a taxonomy to categorize the EA-based approaches for supervised ensemble learning (Figure 3.2). All surveyed studies are focused on supervised problems, i.e., no unsupervised approach is reviewed.

We divide the surveyed studies according to the well-established main stages of supervised ensemble learning [26, 36, 167]: generation, selection, and integration. The approaches most often used in each stage are presented at the second level of the taxonomy. For example, attribute selection, model tuning, and instance selection are the three most common approaches for the generation stage. Further divisions in the taxonomy are presented at the next levels, whenever it is the case.

Note that taxonomies vary depending on the aspect being analyzed – e.g. Gu [112] is concerned with the generation stage, and hence proposes a taxonomy exclusively for that step. To the best of our knowledge, our taxonomy is one of the broadest with regard to EAs for ensemble learning, with the closest reference being the one proposed by Cruz et al. [55]. While the description of generation and selection stages in [55] is identical to ours, we are more specific regarding the strategies for the integration stage. In addition, while the authors propose a two-level taxonomy, we present a more detailed and thorough four-level taxonomy.

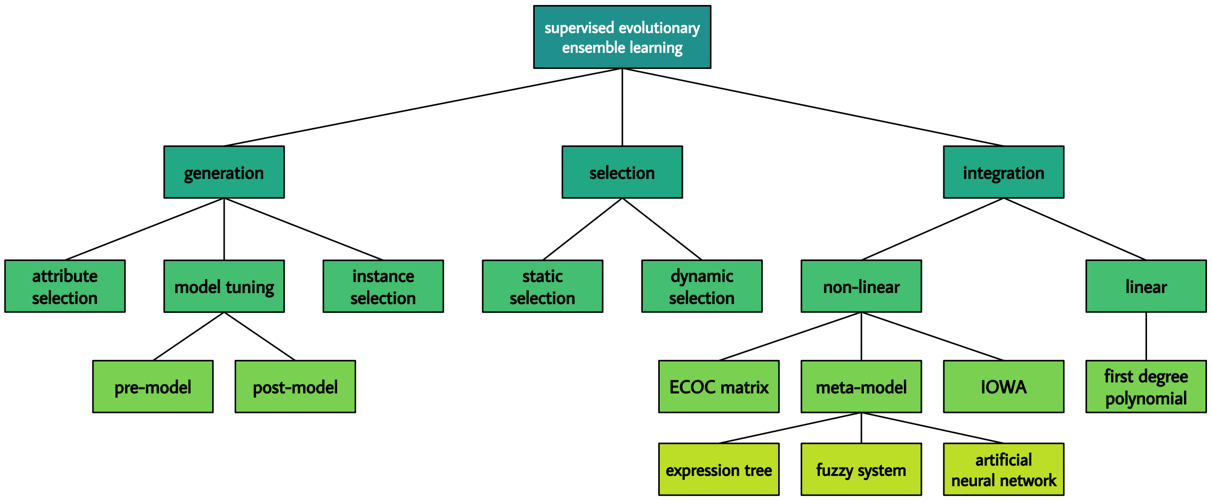


Figure 3.2: The proposed taxonomy for evolutionary algorithms employed in ensemble learning.

3.3 The Generation Stage of Ensemble Learning

In this stage the algorithm generates a pool of trained models. Those models may come be from: (i) different paradigms (e.g., Naïve Bayes, Support Vector Machines, and Neural Networks [207]); (ii) the same paradigm; or (iii) differences within the same paradigm (e.g., neural networks of different topologies and/or activation functions [279]).

The main objective in this stage is to generate a pool of both accurate and diverse base learners. Base learners must be diverse in order to provide source material for the selection and integration steps to work with. A diverse pool of base learners has more chances to commit errors in different data instances, thus correctly predicting more instances [200].

An example of an ensemble algorithm that focuses on the generation phase is Random Forest [24, 252], considering that it selects distinct subsets of both attributes and instances for building different decision trees, resulting in an ensemble of trees that is more robust than a single tree.

We have identified in the literature three distinct ways of generating pools of learners that are both diverse and accurate: (i) providing distinct training sets for each base learner (instance selection); (ii) providing the same training set for all learners but with distinct sets of attributes (attribute selection); and (iii) optimizing the model by modifying the hyper-parameters and/or the structure of the base learners. Table 3.5 summarizes the work on EAs for the generation step of supervised ensemble learning, based on our proposed taxonomy. The remaining of this section will explain in greater detail the different methods of generating ensemble members.

Table 3.5: Studies that employ EAs in the generation stage of supervised ensemble learning organized by generation technique.

Method	Related work
Instance selection	[1, 6, 58, 104, 112, 132, 146, 150, 151, 261]
Attribute selection	[5, 11, 41, 43, 46, 56, 59–61, 66, 68, 112, 139, 155, 167, 170, 181, 189, 206, 207, 217, 229, 236–241, 247, 250, 256, 266, 276]
Pre-model optimization	[6, 51–53, 59–61, 69, 130, 131, 134, 166, 167, 174, 190, 200, 217, 224, 225, 229, 242, 256, 266, 269]
Post-model optimization	[10, 21, 22, 32, 33, 36, 42, 57–60, 64, 66, 68, 74, 75, 79–82, 86, 87, 92–95, 109, 124, 129, 137, 140, 153, 158, 159, 164, 175, 178–180, 190, 218, 222, 231, 234, 242, 243, 250, 253, 259, 262, 264]

3.3.1 Instance Selection

Instance selection, also known as prototype selection or data randomization [2, 261] consists of providing different (not necessarily disjoint) subsets of training instances for different base learners [6, 224]. This approach is well-suited for homogeneous sets of base learners which are sensitive to changes in the instance distribution (e.g., decision trees [120]).

Instance selection can also be used to reduce training time by finding a subset of representative instances for each class [6, 224]. This is also beneficial for problems with high class imbalance, given that resampling instances with replacement makes it possible to simulate a uniform distribution among classes. Indeed that is one of the capabilities of the traditional bagging algorithm [23]. Thus, “bagged" EAs are likely to present the same benefits as non-EAs which were also bagged: improved noise tolerance and reduced overfitting risk [261]. A method for selecting instances is needed since random sampling can lead to information loss and poor model generalization [132]. By using an EA, both tasks of undersampling the majority class and oversampling the minority class are possible in parallel.

This section mainly focuses on instance selection techniques, since instance generation is more scarce. The former simply selects a subset of instances from the original training data, while the latter creates new artificial instances to better adjust the decision boundaries of the classes, though being more prone to overfitting. The best performing models are EA-based techniques such as differential evolution [261]. Only one work uses a hybrid selection-and-generation strategy [261].

Instance selection is also susceptible to overfitting, if it leads to one class having many more instances than other classes. An approach to avoid overfitting is to assign different misclassification costs to different classes. Typical cost-sensitive learning techniques take misclassification costs into account during model construction, and do not modify the imbalanced data distribution directly [32].

Instance selection methods can be divided into wrapper and filter methods. Wrapper methods are by far more common in the literature, since they tend to provide more accurate ensembles and they measure the quality of instance subsets using trained models [181]. There is a direct link between high-quality instance subsets and a high-quality pool of base learners [181]. Filter approaches break this link, evaluating the quality of an instance set in a way independent from the overall base learner pool [132, 181]. A study showed that genetic algorithms (GAs) with error rate as fitness function are capable of outperforming greedy wrapper methods in terms of ensemble accuracy [181].

In EAs for instance selection, usually the training set is encoded as either a binary or real-valued chromosome of N positions (the number of instances). In the binary case, each bit encodes the presence or absence of an instance in the solution encoded by the current individual. In a real-valued case, each gene encodes the probability that the respective instance will be present in that solution. To address class imbalance, in [104] only majority class instances are encoded in a binary string – minority class instances are always sampled. In [6], a GA is used to optimize k-means groups, finding evenly-distributed groups (based on the class distribution) and then using each group as the training set of a neural network base classifier.

It is also possible to perform instance selection together with other techniques. In [224], the multi-objective problem consists in optimizing both the learner's hyper-parameters and the instance set which will be used for training each model. This also fits well with weight optimization: in [146], evolutionary under-sampling and boosting are used in a C4.5 decision-tree classifier to iteratively optimize its performance in grading breast cancer malignancy.

For an extensive review on instance selection techniques, Olvera-López provides a survey of both evolutionary and non-evolutionary methods proposed until 2010 [195].

3.3.2 Attribute selection

Attribute selection, also known as feature selection or variable subset selection [236], offers distinct subsets of attributes to different base learners in order to artificially induce diversity among base models. By removing irrelevant and redundant attributes from the data, attribute selection can improve the performance of base learners [239]. Reducing the number of attributes also reduces the complexity of learned base models, and may improve the efficiency of the ensemble system.

Attribute selection also performs dimensionality reduction, and is an efficient approach to build ensembles of base learners [171]. Like in instance selection, there is no need to provide disjoint sets of attributes to different learners. The base learners that are used must be sensitive to modifications in the data distribution. Support Vector Machines, for example, were reported to be little affected by attribute selection [256].

There are three approaches to perform attribute selection: the filter, wrapper, or hybrid approach, as follows.

Filter methods analyze the relevancy of attributes regardless of the predictive performance of base learners. Filter methods are normally faster than wrapper ones, but they may lead to poor predictive performance, particularly if the filter ignores interactions among attributes. Examples of traditional filter methods are the Correlation-based Approach, Information Gain, t-test, Markov Blanket Filter, among others [181].

Wrapper methods are by far the most common type of EAs for attribute selection. However, there are also traditional search methods for wrapping, be it deterministic or stochastic. A wrapper method provides a reduced subset of attributes to a learning algorithm, and then the predictive performance of the model trained with those attributes is used as a measure of the quality of the selected attributes. The random subspace method, for example, is a traditional approach for wrapping algorithms that randomly selects different attribute subsets to construct different base learners. Although this method is usually much faster than EAs, its performance is sensitive to the number of attributes and ensemble size [171]. By contrast, EAs can improve stability and provide more accurate ensembles [171], though at the expense of computational time. Other examples of traditional methods include sequential forward selection, sequential backward selection, beam search, etc. [181].

Finally, hybrid approaches attempt to combine the filter and wrapper approaches in order to overcome their shortcomings. Various combinations have been successfully implemented in the literature [181].

Two concepts relevant for attribute selection are sparsity and algorithmic stability. An attribute selection algorithm is called sparse if it finds the sparsest or nearly-sparsest set of attributes subject to performance constraints (e.g. small generalization error) [256]. An algorithm is called stable if it produces similar outputs when fed with similar inputs – i.e., it selects similar attribute sets for two similar datasets [273]. As noted in [256, 273], stability and sparsity constitute a trade-off. An algorithm that is sparse may be incapable of selecting similar sets of attributes across runs [256].

EAs for attribute selection vary on the number of objectives, integration with other stages, and distribution of base learners. In [206, 207] a multi-objective Particle Swarm Optimization algorithm provided different attribute subsets to heterogeneous base learners. In [239–241], both model and attribute selection were also performed in a multi-objective fashion.

The encoding adopted in [139] considers each individual as an ensemble of classifiers. Classifiers are trained differently based on the features provided to them. Each classifier competes with its neighbors within the same ensemble; and at a higher level, ensembles compete among themselves based on their predictive accuracy.

3.3.3 Model optimization

Models may have their hyper-parameters and/or structure modified while creating a pool or ensemble of base learners. We divide this category of our taxonomy into two groups: pre-model and post-model optimization.

Pre-model optimization involves fine-tuning the hyper-parameters of base learners that will generate base models. We call these approaches *pre-model* because the optimization happens prior to model generation. Examples include tuning the neural network's learning rate; the Support Vector Machine's type of kernel function [224], regularization parameter C [224], L2 regularization [269], Radial Basis Function kernel free parameter [224, 269], or degree of polynomial [224]); and random forests' number of trees [229].

Pre-model approaches may support heterogeneous sets of base learners. In [229] the authors select both the type of base learner and their respective hyper-parameters, coupled with a set of attributes that will be assigned to a given learner. They use NSGA-II [65], and the one-point crossover keeps base models and hyper-parameters together, only allowing to swap the selected attributes for each model. The encoding is depicted in Figure 3.3.

Type of Classifier	Parameters	Attributes

(a) Encoding

Decision Tree	1	7	01000110100110100011001

(b) Individual

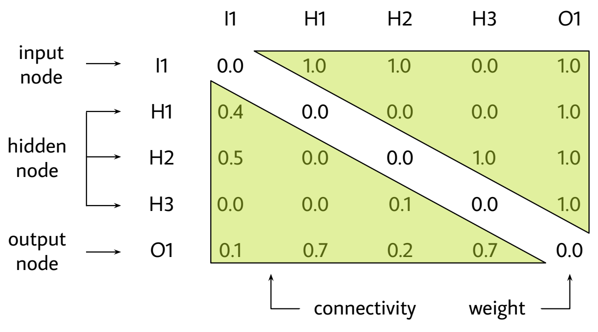
Figure 3.3: Type of encoding and individual in [229] for pre-model optimization. The first part of the chromosome represents the type of classifier, followed by its respective hyper-parameters and a string denoting which features are available for that classifier.

Post-model approaches try to improve an existing model. Examples are layout and inner node selection for decision trees [10, 180, 264], and topology, weight, and activation function optimization in neural networks [87, 190, 190]. Weights are also optimized in [153], where an ensemble of heterogeneous parametric models are optimized by differential evolution.

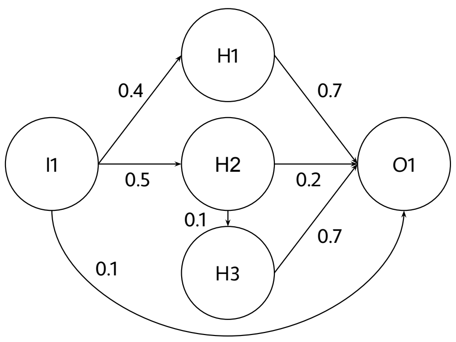
Post-model encoding depends on the type of base learner being used, and hence are more common on homogeneous sets of base learners. In [137], the weights of artificial neural networks are modified by a niching-based GA. The authors adopt a matrix of size $W \times W$ (Figure 3.4), where W is the number of neurons in the entire network. The upper diagonal encodes whether two given neurons are connected, and the lower diagonal encodes the weights associated with those connections.

There are studies that perform both pre- and post-model optimization. In [190], first the topology of a neural network is evolved by using NSGA-II. The best found topology then has its parameters (e.g., weights and activation functions) adjusted by a multi-objective Differential Evolution method. In the end, the final population is submitted to a voting scheme optimized by another EA.

Attribute selection is often coupled with model optimization. In [250], both post-model optimization of Radial Basis Function Neural Networks and attribute selection were used, by performing both approaches in two subpopulations of the Cooperative Coevolutionary EA. In [217], solutions for both tasks were placed within the same chromosome. With a 132-wide chromosome array, 88 bits are designated for attribute selection, 10 bits represent parameter *nu* and threshold (integer and decimal part), 14 bits correspond to the gamma value and 20 bits are used for the parameter C of a nu-SVR learner.



(a) Encoding



(b) Individual

Figure 3.4: Encoding and resulting individual adopted in [137] for post-model optimization.

3.4 The Selection Stage of Ensemble Learning

From the pool of generated base learners, model selection (or model pruning [201]) is performed in order to define the final set of base models for the ensemble. This stage is optional and frequently not performed by traditional methods (e.g., boosting [99], bagging [23]) or EA-based ones (e.g., [33, 276]). Selection may consist of simply selecting the Φ most accurate learners, or using a (greedy or EA-based) algorithm for choosing models.

Whether or not to perform selection is an issue for debate, with some authors proposing to bypass this stage (i.e., using the entire pool of models as ensemble) [251]. Lacy et al. [159] argue that model selection is irrelevant for ensemble learning, and that simply selecting the Φ best models from the pool is more effective than building an ensemble based on diversity measures. They also claim that their argument is consistent with other studies that simply select the most accurate learners instead of employing diversity measures [100], and that there is little correlation between measures of ensemble diversity and accuracy [24, 198, 201]. On the other hand, some authors argue the opposite: e.g., for regression, Wang and Alhamdoosh [263] argue that the Φ best neural networks may not produce an ensemble with better Mean Squared Error (MSE). This is also stated by Liu et al. [172], adding that simply selecting the most accurate models may result in loss of predictive performance given that most of those models may be strongly correlated, leaving the opinion of the minority of the committee underrepresented.

Although Lacy et al. [159] and Liu et al. [172] have different opinions on the utility of model selection, both agree that diversity measures are not a good proxy for ensemble quality, with Liu et al. [172] suggesting that accuracy on a validation[5] set is sufficient. The rationale for using diversity measures is that by sacrificing individual accuracy for group diversity, one can achieve better group accuracy [36, 201]. Diversity in this case should not be measured at the genotype level (e.g., individuals encoding different attributes for the same base model), but rather measured based on the predictive performance of the algorithms decoded from the individuals. Diversity metrics can be of two types: pairwise or group-wise [120]. A pairwise diversity metric often outputs a matrix of values denoting how diverse one base model is from another. Then, algorithms may select models that are, e.g., more diverse to the other already-selected models. By contrast, group-wise metrics validate how diverse a group of base models is, thus requiring a previous strategy for composing groups. A review of diversity measures is presented by Hernández et al. [120].

The motivations for using EAs for model selection are as follows. First, finding the optimal model subset within a large set is unfeasible with exhaustive search (the search space size is $\approx 2^M$, where M is the number of base models). By contrast, EAs perform a robust, global-search for the near-optimal set of base models [201]. There is evidence that smaller ensembles can indeed outperform larger ones [252]. However, in practice, the optimal ensemble size varies across types of ensembles, types of base learners, and datasets.

Model selection can be further divided into two categories: static and dynamic selection [55, 58, 126, 127]. In static selection, regions of competence are defined at training time and are never changed [55,126,127]. In dynamic selection the regions are defined during classification time, through the use of a competence estimator [126, 127, 255]. Figure 3.5 puts both strategies in perspective.

Some studies say they perform dynamic selection (e.g. [6] via k-means to define regions of competence), but in fact they perform static selection, since the assignment of classifiers is done during training time and does not change after that.

Table 3.6 shows the distribution of the surveyed EAs into the static and dynamic selection categories. The remaining of this section will explain in greater detail the two selection methods.

Table 3.6: Categorization of EAs for the selection stage of supervised ensemble learning.

Method	Related work
Static selection	[6, 8, 16, 17, 36, 39, 43, 45, 48, 54, 61, 63, 72, 73, 75, 76, 120, 126, 127, 135, 136, 138, 142, 155, 166, 177, 184, 188, 189, 201–204, 210, 214, 215, 225, 229, 235, 241, 242, 248, 263]
Dynamic selection	[54, 167, 252]

[5]In supervised learning it is common to divide a dataset into three disjoint sets: training, validation and test. The validation set is used to evaluate the quality of models *while* training, and helps to prevent overfitting to the training data. The test set is used for the final model evaluation *after* training.

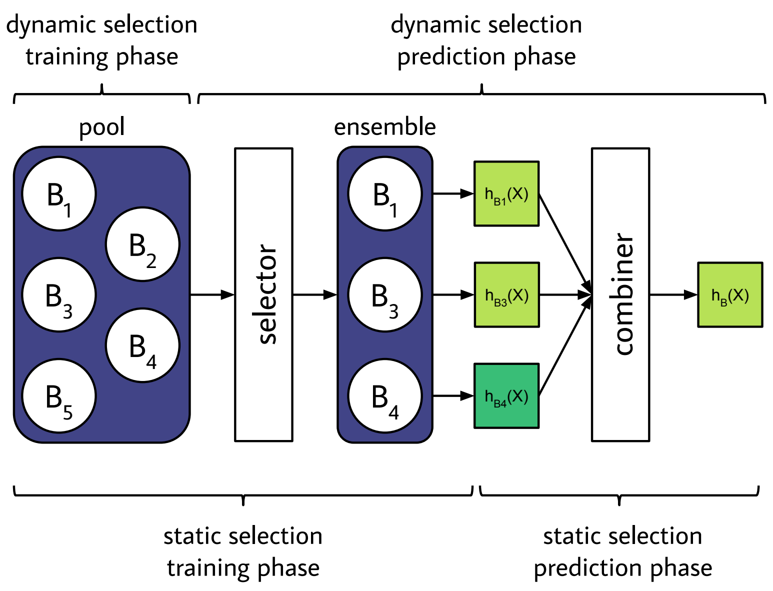

dynamic selection training phase
dynamic selection prediction phase
pool
ensemble
B1
B2
B3
B4
B5
selector
B1
B3
B4
hB1(X)
hB3(X)
hB4(X)
combiner
hB(X)
static selection training phase
static selection prediction phase

Figure 3.5: Difference between static and dynamic selection strategies. While in static selection the competence estimator assigns regions to base learners during the training phase, in dynamic selection this is done during the prediction phase. Dynamic selection can also have a selector (e.g. oracle) that assigns a single base learner to regions of competence.

3.4.1 Static selection

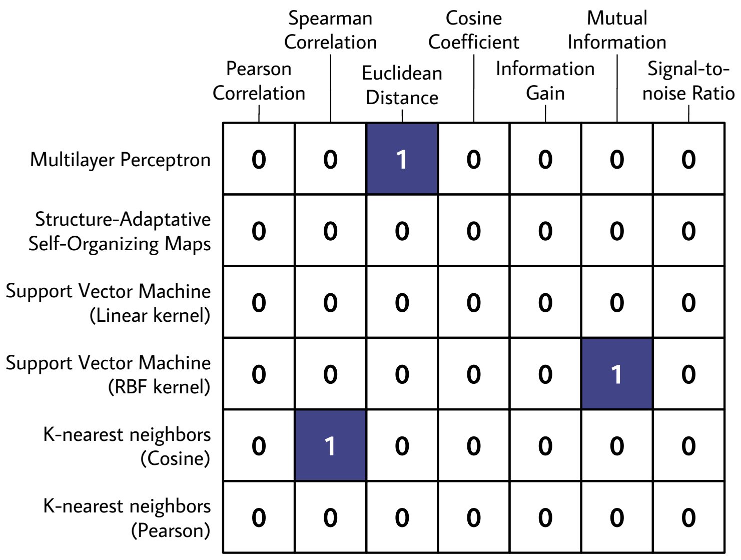
In the *overproduce-and-select* strategy [54, 130], an algorithm first generates a large pool of base models using a generation method (see Section 3.3). Then, the base models are selected from this pool and their votes are combined according to an integration scheme (see Section 3.5). The rationale is that some of the models may perform poorly or have strongly-correlated predictions, making some of these safe for exclusion from the final ensemble [251].

A second strategy for static selection, known as *clustering-and-selection*, uses a clustering algorithm to assign models to distinct regions of competence in the training phase. In the testing phase, a new instance is submitted to the base model that covers the region closer to that instance. Studies using this strategy include [76, 214, 215]. In [127], a GA was used for selecting the number of partitions in which the input space is divided. It then assigns an ensemble of classifiers to each partition, optimizing the voting weight of each base learner.

In [263] a hill-climbing strategy was used for increasing the size of the ensemble. By starting with only two classifiers (Extreme Learning Machines), the number of ensemble members is increased by adding classifiers that reduce the overall ensemble's error rate. In [73], the authors investigate the impact of combining error rate (effectiveness), ensemble size (efficiency), and 12 diversity measures on the quality of static selection by using pairs of objectives. The authors also study conflicts between objectives, such as error rate/diversity measures and ensemble size/diversity measures. They argue that, among diversity measures, difficulty, inter-rate agreement, correlation coefficient, and double-fault are the best for combining with error rate, ultimately producing the best ensembles.

Studies that use the overproduce-and-select strategy often employ a chromosome encoding as a binary string, where 0 denotes the absence of that model in the final ensemble and 1 the presence [43]. However, in [210] the chromosome size was doubled by using two values for each model: one for the aforementioned task, and another to determine the strength of that model's output in the final ensemble's prediction.

In [136], attribute and model selection were performed at the same time. The authors use a binary matrix chromosome where each row represents a different base learner and each column a filter-based attribute selection approach. In this sense, if a bit is active somewhere within the individual's genotype, it means that the base learner of the corresponding row will be trained with the attributes selected by the filter approach of the corresponding column, as shown in Figure 3.6.



	Pearson Correlation	Spearman Correlation	Euclidean Distance	Cosine Coefficient	Information Gain	Mutual Information	Signal-to-noise Ratio
Multilayer Perceptron	0	0	1	0	0	0	0
Structure-Adaptative Self-Organizing Maps	0	0	0	0	0	0	0
Support Vector Machine (Linear kernel)	0	0	0	0	0	0	0
Support Vector Machine (RBF kernel)	0	0	0	0	0	1	0
K-nearest neighbors (Cosine)	0	1	0	0	0	0	0
K-nearest neighbors (Pearson)	0	0	0	0	0	0	0

Figure 3.6: Encoding used for static selection in [136].

3.4.2 Dynamic selection

In dynamic selection, a single model or a subset of most competent learners is assigned to predict an unknown-class instance [55] (hereafter, unknown instance for short). This strategy was reported to perform better than boosting and static selection strategies [55]. However, work on dynamic selection is much less frequent than work on static selection. Dynamic selection is also more computationally expensive, since estimators are required to define regions of competence for all predictions, which can be unfeasible in some scenarios [60].

One approach for dynamic selection is to use random oracles [54,252]. A random oracle is a mini-ensemble with only two base learners that are randomly assigned to competence regions [252]. At prediction time, the oracle decides which base learner to use for providing predictions for unknown instances.

Another strategy is to train a meta-learner. In [167], generation strategies of feature selection and pre-model optimization were coupled with an initial overproduce-and-select strategy for generating a diverse pool of base learners. Next, a meta-learner was trained for selecting the best subset of models for predicting the class of unseen instances.

Dynamic selection is also known as classifier pruning or ensemble pruning and can be considered as an optimization problem with two objectives, classification accuracy and diversity, where both need to be maximized. When the size of a classifier ensemble is relatively large, classifier pruning is computationally expensive or even prohibitive [201].

3.5 The Integration Stage of Ensemble Learning

The last step of ensemble learning concerns the integration of votes (for classification) or value approximation (for regression) in order to maximize predictive performance. Ensemble integration, also called learner fusion [251] or post-gate stage [67], is the final opportunity to fine-tune the ensemble members in order to correct minor faults, such as giving more importance to a minority of learners that are however making correct predictions. Integration is an active and diverse research area in ensemble learning [251]. Similarly to the selection stage, this is another stage where using a validation set has shown to be useful, since reusing the training set that was employed to generate base models can lead to overfitting.

Traditional strategies for fusing ensemble predictions include majority voting and weighted majority voting (for classification), and also average, weighted average, maximum, minimum, sum, and product rules (for regression) [141,158,181]. More sophisticated strategies include Dempster-Shafer, Naïve Bayes, artificial neural networks, entropy weighting, and distribution summation [158,177].

For classification, the most popular method is weighted majority voting, which allows to weight the contribution of each individual classifier to the prediction according to its competence via voting weights [251]:

$$h_B(X^{(i)}) = \underset{j}{\operatorname{argmax}} \left(\sum_{b=1}^{B} w_{b,j} \times [h_b(X) = c_j] \right) \tag{3.1}$$

where B is the number of classifiers, $w_{b,j}$ the weight associated with the b^{th} classifier for the j^{th} class, and $[h_b(X) = c_j]$ outputs 1 or 0 depending on the result of the Boolean test. This strategy has been shown to perform better than majority voting and averaging [159]. A simplification of that function sets all weights to 1, which turns this method into a simple majority voting, another popular approach [282]. For instance, bagging uses a simple majority voting scheme, whereas boosting uses weighted majority voting [281].

For regression, the most popular is the simple mean rule, which averages the predictions of base regressors, $h_B(X^{(i)}) = \frac{1}{B}\sum_{b=1}^{B} h_b(X^{(i)})$, where B is the number of regressors and $h_b(X^{(i)})$ is the prediction for the b^{th} regressor. Simple aggregation strategies are better suited for problems where all predictions have comparable performance, however those methods are extremely vulnerable to outliers and unevenly-performing models [177].

A common integration strategy is to use meta-models to learn the proper weights for averaging the performance of the base models, as in stacking [181, 255]. Ensembles that use stacking are referred to as two-tier (or two-level) ensembles [255]. Those ensembles are well-suited, e.g., for incremental learning [255]. E.g., when updating an existing ensemble model to consider new data, we may need to train only a few novel base models covering the new data and then re-train the meta-model with the both the novel and the previous base models. This is more efficient than re-training all existing base models in a single-level ensemble [255]. Two-tier ensembles were reported to perform better than simple weighting strategies in larger datasets [186]. As disadvantages, two-tier ensembles are more susceptible to overfitting when compared to traditional integration methods, and also increase the training time of the entire ensemble [177]. In practice, whether stacking or traditional aggregation methods are better is heavily influenced by the input data [186].

To summarize, Table 3.7 shows the categorization of studies using EAs in the integration stage of ensemble learning, according to the type of integration method. The rest of this section briefly explains each one of these methods.

Table 3.7: Categorization of studies that employ EAs in the integration stage of ensemble learning.

Method	Related work
First Degree Polynomial	[16, 28, 38, 57, 79–82, 84, 102, 104, 115, 125–128, 136, 138, 146, 148–152, 158, 170, 173, 188, 190–192, 196, 210, 230, 232, 236–240, 270, 278–282]
Expression trees	[4, 78, 91, 158, 159, 169, 172, 254]
Genetic Fuzzy System	[54, 251, 255]
Error Correcting Output Codes	[31]
Artificial Neural Network	[158]
IOWA	[19]
Meta-learner selection	[235]

3.5.1 Linear models

We focus now on using EAs to learn the voting weights. A wide variety of methods were proposed for this task, such as using genetic algorithms [146, 190], particle swarm optimization [230], flower pollination [279], differential evolution [236, 280, 281], etc. Those methods can be applied to both homogeneous [38, 280, 281] and heterogeneous [138, 192, 282] base learner sets. For classification, methods may also differ in the number of voting weights, either by using one voting weight *per* classifier (e.g. [188, 282]) or one voting weight *per* classifier *per* class (e.g. [57, 84, 240]).

For a thorough experimental analysis of linear and non-linear voting schemes, the reader is referred to the work of Lacy et al. [158], which presents the most comprehensive experimental comparison of EA-based combining methods to date. Notwithstanding, in the next sections we present a broader review of EAs proposed for this task, as well as methods that were not presented in [158].

3.5.2 Expression trees

Instead of optimizing weights, one can use non-linear models for integrating predictions. This may better exploit classifiers' diversity and accuracy properties [78]. One of the most popular EA-based methods are expression trees [4, 78, 91, 158, 159, 169, 172, 254]. Expression trees resemble decision trees in their structure, but with models in their leaves and combination operators in their inner nodes. As an example, Figure 3.7 depicts the expression tree used in [91].

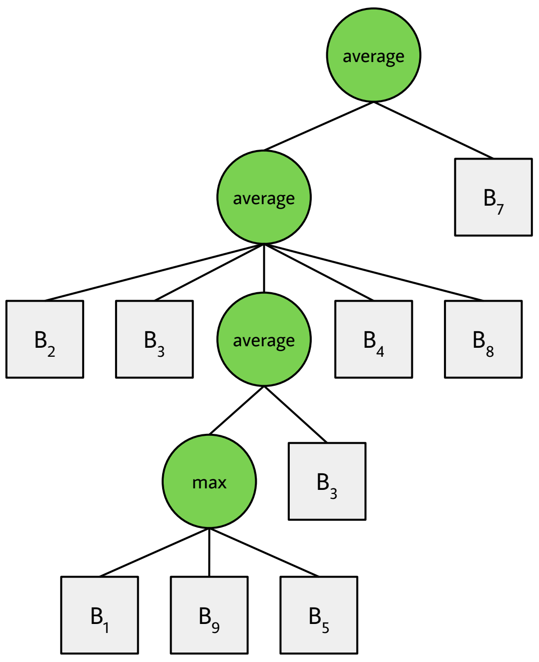


Figure 3.7: Expression tree induced by a Cellular Genetic Programming Algorithm used in [91]. Leaf nodes contain base classifiers, whereas inner nodes contain combination rules.

For the problem of microarray data classification, in [169, 172] some decision trees (initially trained with bagging) are fed to a Genetic Programming algorithm, which then induces a population of expression trees (each allowed to have at most 3 levels) for combining the base classifiers' votes. The GP uses three operators: minimum (for a binary class problem, the negative class is selected, if any model predicts the instance as negative), maximum (the positive class is selected, if any model predicts the instance as positive) and average (which works as a majority voting operator). After the evolutionary process is completed, expression trees with accuracy higher than the average are selected by a forward-search algorithm to compose the final meta-committee, which will predict the class of unknown instances. Figure 3.8 depicts the voting scheme adopted by the authors in in [169, 172].

3.5.3 Genetic Fuzzy Systems

Genetic fuzzy systems are popular in ensemble learning, where fuzzy systems optimized by EAs are used to predict the class of unknown instances. A study reports that fuzzy combiners

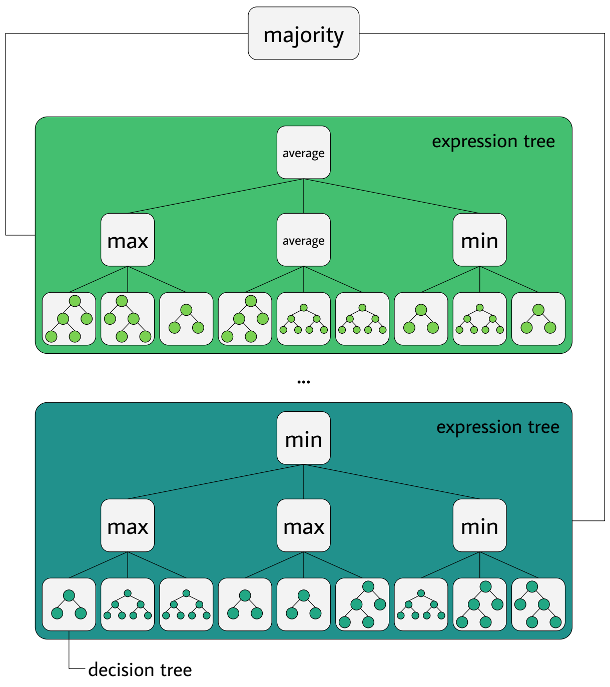


Figure 3.8: Voting scheme adopted by Liu et al. [169, 172]. A pool of decision trees is shared among several meta-models (in this case, expression trees) which will have their votes combined by a simple majority voting rule.

can outperform crisp combiners in several scenarios [251]. There are several steps in the induction of fuzzy systems where EAs may be used: from tuning fuzzy membership functions to inducing rule bases [54, 255]. For instance, in [54, 251], a GA was used with a sparse matrix for codifying features and linguistic terms; and in [255] a GP algorithm was used to evolve combination structures of a grammar-free fuzzy system.

3.5.4 Induced Ordered Weighted Averaging (IOWA)

The application of EAs for other non-linear structures include approximating the weights via Induced Ordered Weighted Averaging (IOWA). This is done in [19] through the use of a Multi-Objective EA based on Decomposition (MOEA-D). The IOWA operator is a mean-type aggregation operator, generalizing several aggregation functions such as weighted arithmetic mean and OWA operators. IOWA introduces a reordering step for its weights, adding non-linearity to the aggregation system.

3.5.5 Error Correcting Output Codes (ECOC)

Error Correcting Output Codes (ECOC) [18] is a meta-method which combines many binary classifiers in order to solve multi-class problems [11]. It is an alternative to other multi-class strategies for binary classifiers [31] – such as one-vs-one, which learns a classifier for each pair of classes; and one-vs-all, which learns one classifier *per* class, discriminating instances from that class (positives) from all other instances (negatives). The ECOC strategy provides meta-classes to its classifiers (i.e. positive and negative classes are in fact combinations of instances from one or more classes). An example of ECOC is shown in Figure 3.9.

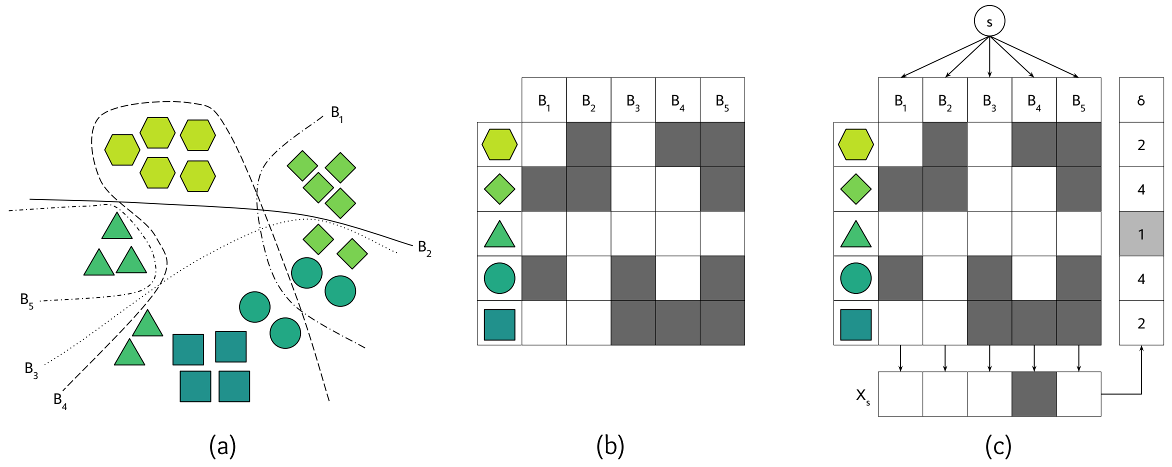


Figure 3.9: (a) Feature space and trained boundaries of base classifiers. (b) Coding matrix, where black and white cells correspond to positive and negative classes, respectively, denoting the two partitions to be learned by each base classifier. (c) Decoding step, where the predictions of classifiers $\{b_1, b_2, \dots, b_5\}$ for sample s are compared to the codewords $\{y_1, \dots, y_N\}$ and s is labeled as the class codeword at minimum distance. Adapted from [18].

ECOC comprises two steps: encoding and decoding. The aim of encoding is to design a discrete decomposition matrix (codematrix) for the given problem [11]. A study reports that larger matrices (with regard to number of classifiers) improve predictive performance [11]. In the decoding phase, each classifier casts a vote to a meta-class for an unknown instance. The predicted class is computed by comparing the distance of the outputed codeword for that instance with the codeword from each real class via a similarity metric.

Though in classification we wish to reach top predictive accuracy, other measures should also be considered for evaluating the ECOC matrix, such as row separation and column diversity [31]. By using an indicator-based selection EA (IBEA), in [31] the ensemble accuracy, individual classifier accuracy, and hamming distance were used as objectives for optimizing the layout of ECOC matrices, by manipulating the distribution of classes among base classifiers. In [11], on the other hand, an attribute selection strategy was used to generate classifiers to be integrated by an ECOC scheme; hence, this work is labeled as a generation technique instead of an integration one.

3.5.6 Neural Networks

In an experimental work comparing several integration methods [158], a multilayer perceptron neural network was used as a combination strategy. The output from base classifiers was used as input for the neural network, with an EA used for optimizing the weights of connections between neurons.

3.5.7 Evolutionary Algorithms for selecting meta-combiners

In [235], besides using the Artificial Bee Colony (ABC) algorithm for selecting base classifiers, the authors also use another ABC for selecting the meta-learner that will combine the votes of ensemble members.

3.6 Fitness functions

First, we review four types of objective (or fitness) functions, broadly categorized with regard to their nature: effectiveness, efficiency, diversity, and complexity. Next, we review multi-objective optimization approaches.

3.6.1 Effectiveness, Diversity, Complexity and Efficiency

An objective function measures *effectiveness* when it evaluates the ensemble's predictive performance. This aspect is essential to ensemble learning and is addressed by all surveyed studies. The most popular objectives within this category are accuracy (or its dual, error rate) for classification tasks and mean squared error for regression tasks. The well-known accuracy measure is given by:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

where TP, TN, FP and FN are the numbers of true positives, true negatives, false positives, and false negatives, respectively. The error rate is simply: $1 - accuracy$. Note that accuracy and error rate are not strong metrics, since they present poor performance on highly imbalanced class distributions [164].

The mean squared error is given by:

$$\text{MSE}(X^{(i)}) = \frac{1}{N}\sum_{i=1}^{N}(h_B(X^{(i)}) - Y^{(i)})^2 \tag{3.3}$$

which computes the difference between the value predicted by the ensemble ($h_B(X^{(i)})$) and the real value $Y^{(i)}$, for all N instances. Other effectiveness measures include exponential squared loss [128], geometric mean [32,33,217,261], imbalance ratio [261] (for imbalanced classes), and confidence [136, 138, 242], to name just a few.

A diversity metric evaluates how diverse an ensemble's members (base learners) are. A diversity measure is often used as an objective in the selection stage of ensemble learning (Section 3.4); and it can also used in the generation stage (Section 3.3). We refer the reader to [36] for a review on diversity measures for generating models; and next we discuss the controversial issue of using diversity as an objective in general, regardless of the ensemble learning stage.

Several researchers defend diversity as a valid objective (e.g. [43, 44, 92, 243]), stating that it contributes to ensemble accuracy [44]. De Stefano et al. [243] state that, as the number of base learners increase, so does the probability that a minority of correct base learners will be overrun by a majority of wrong base learners, and thus the need for using diversity measures to reverse that effect. Also, in EAs, genetic material from well-performing solutions tend to be propagated to their offspring, often compromising diversity [74].

However, other researchers do not see the utility of diversity measures (e.g. [142, 159, 172]), stating that the correlation between ensemble accuracy and diversity is not as strong as expected [252]. Some authors also note that classic ensemble learning methods (e.g. bagging, boosting, and random subspace) introduce diversity in an ensemble without directly measuring it [73]. We can conclude, from this debate, that the relationship between ensemble effectiveness and diversity is not fully understood yet [73, 252].

Diversity can be measured based on the ensemble's characteristics encoded in an individual's genotype, or based on the predictions made by each base learner. In the latter case, a set of base learners is said to be diverse when their errors are not correlated [230]. Examples of diversity measures in this category include Yule's Q statistic [250], average residual correlation coefficient [254], and negative correlation [21,22]. The most popular measure seems to be the Kohavi-Wolpert variance [45, 214, 215], given by:

$$KW = \frac{1}{NB^2} \sum_{j=1}^{N} L(X^{(j)})(B - L(X^{(j)})) \tag{3.4}$$

where B is the number of classifiers, N is the number of instances in the (training or validation) evaluation set, and $L(X^{(j)})$ is the number of classifiers within the ensemble that correctly predict the class of instance $X^{(j)}$.

Diversity measures can be divided into pairwise and group measures. The latter evaluate diversity among all classifiers in the ensemble, whereas pairwise metrics evaluate diversity between two classifiers, and require an averaging technique for obtaining a group measure from all classifier-pairwise measures [120]. This is performed by the disagreement measure. The pairwise disagreement measure [36] is defined as:

$$\mathrm{Diff}(B_i, B_j) = \frac{L^{01} + L^{10}}{L^{00} + L^{01} L^{10} + L^{11}} \tag{3.5}$$

where B_i and B_j are respectively the i-th and j-th classifiers within the ensemble, L^{10} is the number of instances correctly classified by B_i and wrongly classified by B_j, and so on for the remaining indices L^{01}, L^{00}, L^{11}. Pairwise disagreement varies from 0 to 1, with 0 indicating no disagreement (i.e. equal predictions) and 1 maximum disagreement. The plain disagreement measure [170] simply averages the overall disagreement among the members of the ensemble:

$$\mathrm{PSM} = \sum_{i=1}^{B} \sum_{j=i+1}^{B} \sum_{k=1}^{N} \frac{\mathrm{Diff}(B_i, B_j)}{((B-1) \times B \times N)} \tag{3.6}$$

where B is the number of classifiers, N the number of instances in the training or validation set.

For an extensive list of diversity measures for ensemble learning, the reader is referred to [120, 142, 157].

Complexity metrics evaluate how complex the classifiers in the ensemble, or the ensemble as a whole, are. The most popular complexity metrics are the number of activated classifiers (for classifier selection approaches) [54, 73, 124, 135, 136, 203, 251, 252] and the number of attributes used by the models induced by the base learners [5,39,41,167,217,240,247,266,276]. Other complexity metrics include the number of nodes in flexible neural trees [190] and the number of hidden neurons in a neural network [53, 167]; the structured minimization principle [109]; the number of support vectors in a Support Vector Machine model [217]; and the length of fuzzy rules [124].

Efficiency is a desired objective when an ensemble must be fast, during training and/or testing (prediction) phase. Training efficiency is obviously important in very large datasets. In addition, both training and testing efficiency are especially important in data stream scenarios, where a continuous flow of incoming data is presented to the system and predictions must be made in a real-time basis.

Complexity and efficiency metrics are related since, broadly speaking, reducing the complexity of the base learners or the ensemble as a whole leads to more efficient ensemble learning systems – e.g., reducing the number of base learners (a complexity metric) leads to faster ensembles, for a fixed type of base learner. Note, however, that the number of base learners is not directly a measure of efficiency, since efficiency depends on both the number and the type of base learners. For example, an ensemble with a given number N of decision tree algorithms would probably be trained faster than an ensemble with $N/2$ neural networks, since the later type of base learner is much slower than the former. In addition, it is possible to improve efficiency without directly reducing the complexity of the models in the ensemble – e.g., by reducing the number of instances fed to the ensemble learning system.

Among surveyed work, only two studies optimize efficiency, one measuring prediction time reduction [189], and the other measuring training set size reduction (for instance selection) [224], as a proxy for training time. None of the surveyed EAs employed training time *per se* as a metric.

3.6.2 Single vs. Multi-Objective Optimization

Ensemble learning methods performing single-objective optimization are, evidently, constrained to optimize effectiveness. However, ensemble learning may be naturally viewed as a multi-objective task, involving also other types of objectives, as discussed earlier. Figure 3.10 shows the distribution of other objectives that were optimized along effectiveness in studies that employed multiple objectives.

In this survey we follow the taxonomy of multi-objective optimization approaches proposed in [97], where approaches are categorized into three types: (i) weighted fitness functions, where each objective is assigned a user-defined (typically, very ad-hoc) weight indicating that objective's importance; (ii) the lexicographic approach, where the user only ranks the objectives in terms of their priorities (no ad-hoc numerical weights), and then the EA selects individuals for reproduction by trying to optimize the objectives in their decreasing order of priority; and (iii) the Pareto dominance approach, where the EA evolves a set of non-dominated solutions in the Pareto sense – i.e., a solution is non-dominated if it is not worse than any other according to each objective *and* it is better than others according to at least one objective.

In the surveyed papers, the least popular approach was the lexicographic one ([167]), followed by weighted fitness functions ([48, 61, 120, 135, 136, 138, 166, 201–203, 266, 278]), then the single-objective approach (see the single-objective entry in Table 3.8) and finally the Pareto dominance approach as the most popular one (all the papers that were not cited in this paragraph and are within the multi-objective entry in Table 3.8).

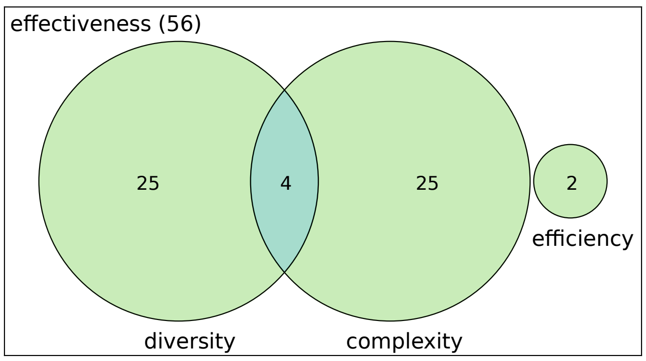


Figure 3.10: Distribution of objectives across EAs using multiple objectives. Effectiveness (predictive performance) is optimized in all 56 studies. In addition, only four studies [54, 73, 190, 252] optimize three objectives (effectiveness, diversity and complexity), and no study optimizes all four objectives.

Table 3.8: Studies categorized by number and type of objectives employed.

Number of objectives	Nature	Related work
Single-objective	Effectiveness	[4, 6, 11, 28, 32, 33, 38, 42, 43, 46, 51, 52, 54, 57, 61, 63, 64, 69, 72, 74–76, 79–82, 84, 86, 91–95, 102, 115, 126–132, 134, 137–140, 148, 149, 152, 153, 158, 169, 170, 172–175, 177–179, 191, 192, 200, 204, 218, 222, 232, 234–237, 242, 243, 248, 253, 255, 256, 259, 262, 269, 270, 279, 282]
Multi-objective	Effectiveness	[1, 5, 8, 10, 16, 17, 19, 21, 22, 31, 36, 39, 41, 45, 48, 53, 54, 56, 58–61, 66, 68, 73, 78, 87, 104, 109, 112, 120, 124, 125, 135, 136, 138, 142, 146, 150, 151, 155, 159, 164, 166, 167, 170, 180, 181, 184, 188–190, 196, 201–203, 206, 207, 210, 214, 215, 217, 224, 225, 229–231, 238–241, 247, 250–252, 254, 261, 263, 264, 266, 276, 278, 280, 281]
	Efficiency	[189, 224]
	Diversity	[21, 22, 36, 45, 48, 54, 61, 73, 104, 112, 120, 135, 142, 146, 150, 151, 159, 170, 190, 201, 202, 206, 207, 214, 215, 250, 252, 254, 263]
	Complexity	[5, 8, 17, 39, 41, 53, 54, 73, 109, 124, 136, 138, 166, 167, 188, 190, 203, 210, 217, 225, 229, 231, 240, 247, 251, 252, 266, 276, 278]

3.7 Types of Evolutionary Algorithms

A wide variety of EAs from different paradigms have been employed for ensemble learning. While some of those paradigms are quite rare (e.g. Flower Pollination Algorithm [279], Levy-Flight Firefly Algorithm [278]), others are widely used in ensemble learning. Among them, Genetic Algorithms (GAs) seem to be the most popular, followed by Genetic Programming and Differential Evolution.

Within GAs, apart from its single-objective version, Non-dominated Sorting Genetic Algorithm II (NSGA-II) is the most popular incarnation. This choice seems due to NSGA-II's ability to deal with multiple objectives, suiting well the multi-objective nature of ensemble learning. Table 3.9 shows the distribution of the surveyed studies according to the type of EA used.

Table 3.9: Studies organized by the type of evolutionary algorithm that is employed.

Evolutionary family	Related work
Flower Pollination	[279]
Clonal Selection	[61]
Evolutionary Algorithm	[218]
Inclined Planes Optimization	[210]
Multi-Objective EA	[16, 17]
Moth-Flame Optimization	[278]
Levy-flight firefly Algorithm	[278]
Virus-Evolutionary Genetic Algorithm	[102]
Many-Objectives Evolutionary Algorithm	[8]
Evolutionary Strategy	[262, 269]
Artificial Bee Colony	[32, 33, 128, 201, 202, 235]
Estimation of Distribution Algorithm	[28, 36, 43, 79–82, 177]
Particle Swarm Optimization	[4, 42, 43, 51–53, 61, 72, 130, 131, 140, 200, 206, 207, 210, 230]
Differential Evolution	[38, 58–60, 63, 68, 69, 115, 153, 167, 196, 236–241, 256, 256, 280–282]
Genetic Programming	[4, 21, 22, 42, 64, 75, 78, 91–95, 109, 158, 159, 159, 169, 172, 175, 178, 179, 222, 243, 253–255, 259, 264]
Genetic Algorithms	[1, 5, 6, 10, 11, 16, 17, 19, 31, 39, 41, 45, 46, 48, 54, 54, 56, 57, 61, 63, 68, 72–75, 75, 76, 84, 86, 87, 104, 112, 120, 124–127, 129, 132, 134–139, 142, 146, 148–152, 155, 158, 164, 166, 170, 173, 174, 180, 181, 184, 188–192, 203, 204, 214, 215, 217, 224, 225, 229, 231, 232, 234, 247, 248, 250–253, 261, 263, 266, 270, 276]

3.8 Types of Base Learners

In the surveyed studies, the most commonly used type of base learner is artificial neural networks, used in 75 studies; followed by tree-based algorithms (e.g. decision trees, arithmetic trees), used in 48 studies; and support vector machines, used in 48 studies. The number of studies using each type of base learner algorithm is shown in Table 3.10.

Some ensemble techniques are more appropriate for different types of base learners. Support Vector Machines, for instance, are stable classifiers, making the techniques of selecting either instances, or attributes for each base learner inefficient as a diversity inductor [61].

The majority of the studies use homogeneous ensembles (117 work) instead of heterogeneous ones (51). Homogeneous ensembles are composed by models from the same base learner paradigm. However, this is not to say that all models are similar. When using neural networks, ensemble members can have distinct activation functions, or topologies. Figure 3.11 shows the base learners that were used in at least 5 studies, as well as the study's ensemble type: either homogeneous, using only one type of base learner, or heterogeneous, using several types of base learners. The specific studies using each of these types of ensemble are mentioned in Table 3.11. Note that some work use both types of ensemble.

According to Rahman and Verma [216], there are five strategies for inducing diverse models in homogeneous ensembles: (i) post-model optimization (see Section 3.3.3); (ii) manipulation of the error function; (iii) distinct attribute subsets across base learners (see Section 3.3.2); (iv) manipulation of output targets, in which some instances in the training set have their class labels switched, for inducing diversity; and (v) distinct instance subsets across base learners (see Section 3.3.1). Strategies (ii) and (iv) are not covered, though, due to the lack of relevant papers.

Heterogeneous ensembles comprise base learners from distinct paradigms. Because of this, there is no (direct) pressure for inducing diversity in the ensemble, since models induced by different learners will likely perform different predictions, and thus commit different classification errors. Among the studies using diversity measures, 19 have homogeneous set of base learners, while 9 use a heterogeneous set. These numbers include a single paper that proposes both homogeneous and heterogeneous ensembles. The larger number of papers with homogeneous sets is probably because it is simpler to work with homogeneous ensembles than with heterogeneous ones.

3.9 Application Domains

Several application domains have benefited from evolution-based ensemble learning algorithms. From wind speed prediction to cancer detection, evolutionary ensembles are employed with distinct goals, from base learner selection to optimization of stackers. While some of those fields were marginally explored as a proof-of-concept (e.g. noise by-pass detection in vehicles, stock market

Table 3.10: Studies organized according to the base learners they employ, for base learners that appear in at least 5 studies. For a complete list, please refer to the metadata at the https://github.com/henryzord/eael source code repository.

Base Learner	Related work
Bayesian Network	[115, 120, 210, 282]
Gaussian Process Regression	[19, 174, 191, 192, 266]
Linear Regression	[126, 127, 164, 191, 192]
Fuzzy rule-based Classifier	[54, 86, 124, 181, 251, 252]
Conditional Random Fields	[84, 236–241]
Logistic Regression	[78, 91, 115, 120, 196, 229, 235]
Random Forest	[4, 78, 120, 184, 225, 229, 256, 266]
Rule-based	[16, 17, 57, 66, 68, 178, 179, 181, 222, 231, 241]
Naïve Bayes	[4, 56, 78, 91, 115, 120, 125, 132, 155, 184, 196, 206, 207, 235, 276, 282]
K-Nearest Neighbor	[4, 8, 56, 72, 73, 78, 91, 115, 120, 125, 127, 135, 136, 138, 142, 153, 170, 184, 189, 203, 204, 206, 207, 210, 230, 235, 261, 266, 282]
Support Vector Machines	[4, 8, 31, 38, 48, 56, 56, 61, 78, 84, 112, 115, 120, 120, 126, 127, 130, 131, 135, 136, 138, 148, 170, 174, 191, 192, 196, 202–204, 206, 207, 214, 215, 217, 224, 225, 229, 230, 236, 237, 241, 254–256, 266, 269, 278]
Trees	[5, 8, 10, 11, 21, 22, 28, 41, 56, 64, 75, 91–95, 104, 109, 115, 120, 132, 146, 149–152, 158, 159, 164, 169, 170, 172, 175, 180, 184, 189, 225, 229, 232, 235, 243, 247, 253, 259, 262, 264, 266, 282]
Artificial Neural Network	[6, 8, 11, 32, 33, 36, 39, 41–43, 45, 46, 51–53, 56, 58–60, 63, 69, 74, 76, 78–82, 84, 87, 115, 120, 126, 127, 129, 134–140, 159, 166, 167, 173–175, 188, 190–192, 200, 203, 204, 206, 207, 210, 218, 225, 230, 234, 242, 247, 248, 250, 254, 255, 263, 266, 270, 278–281]

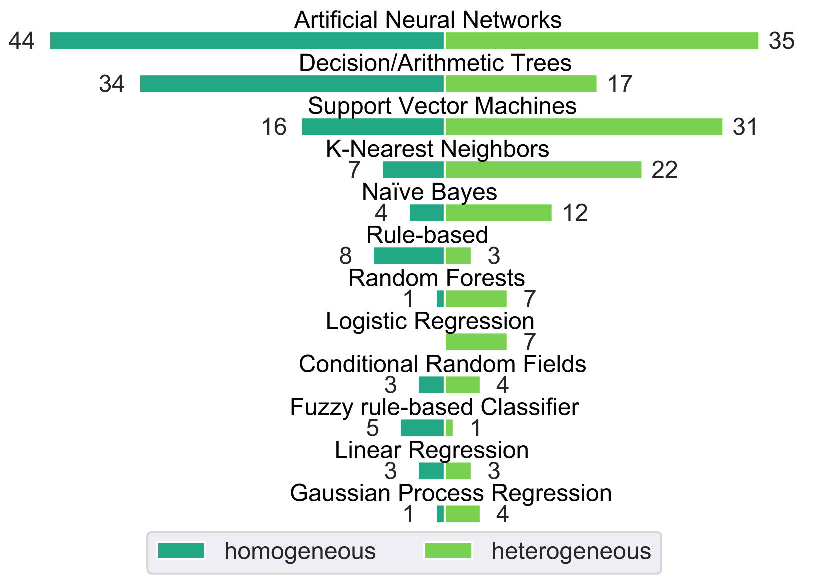


Figure 3.11: Distribution of base learners that are used in at least 5 studies, as well as the type of ensemble in which they appear: either homogeneous, using only one type of base learner, or heterogeneous, using multiple types of base learner.

Table 3.11: Studies organized according to the base learners' homogeneity/heterogeneity.

Homogeneity	Related work
Homogeneous	[1, 5, 6, 10, 17, 19, 21, 22, 28, 31–33, 38, 39, 42, 43, 45, 46, 51–54, 57–61, 63, 64, 66, 68, 69, 72–75, 79–82, 86, 87, 92–95, 102, 104, 109, 112, 124, 125, 127–132, 134, 137, 139, 140, 142, 146, 148–152, 155, 158, 164, 166, 169, 172, 173, 177–180, 188–190, 200, 202, 214, 215, 217, 222, 224, 231, 232, 234, 238–240, 242, 243, 250–253, 256, 259, 261–264, 269, 270, 276, 280, 281]
Heterogeneous	[4, 8, 11, 16, 36, 48, 56, 76, 78, 84, 91, 115, 120, 135, 136, 138, 153, 167, 170, 174, 175, 181, 184, 191, 192, 196, 201, 203, 204, 206, 207, 210, 218, 225, 229, 230, 235–237, 241, 248, 254, 255, 266, 278, 279, 282]
Both	[41, 126, 159, 247]

prediction), others have been far more researched (e.g. microarray data classification). This may be due to inherent qualities of ensembles for those fields. For instance, let us analyze the case of data-stream classification: a pool of base classifiers that is periodically updated is precisely what is needed for dealing with the non-stationary, non-repetitive, time-varying, volatile, agile, and dynamic nature of continuously-incoming data [181].

Also, there is the phenomenon of concept drift often present in data-streams, which happens when the data distribution changes due to external causes. By using a single classifier, one must deal with the task of selecting all relevant data for re-training it. This can be avoided by training new classifiers over subsets of new information, and then combining those classifiers via an integration technique [80, 181].

The majority of papers surveyed perform classification tasks. A list of papers that perform classification, regression, or both tasks is presented in Table 3.12. Table 3.13 summarizes the application domains found within research papers on evolutionary algorithms for ensemble learning.

Table 3.12: Studies organized according to the addressed task.

Task	Related work
Classification	[1, 4–6, 8, 10, 11, 16, 17, 21, 22, 28, 31–33, 36, 38, 39, 41, 43, 45, 46, 48, 51–54, 56–61, 63, 64, 66, 68, 69, 72–76, 78–82, 84, 86, 87, 91–95, 102, 104, 112, 115, 120, 124, 125, 127, 128, 130–132, 134–140, 142, 146, 148–152, 155, 158, 159, 164, 166, 167, 169, 170, 172, 173, 177–181, 184, 188, 189, 196, 200–204, 206, 207, 210, 214, 215, 224, 225, 229–232, 234–243, 247, 248, 250–253, 256, 261, 262, 264, 266, 270, 276, 278, 280–282]
Regression	[19, 42, 109, 126, 129, 153, 174, 175, 191, 192, 217, 218, 222, 254, 255, 259, 263, 269, 279]
Both	[190]

3.10 Summary of Findings

First, we discuss the main findings of the survey regarding which type of technique was found to be the most commonly used in each stage of the ensemble learning process.

The generation stage, i.e. where the ensemble members are generated, was found to be the most popular step to employ EAs, having more studies dedicated to it than the selection and integration stages combined. Wrapper methods were found to be much more common than filter ones for the instance selection and attribute selection approaches. This seems natural, considering that, unlike filters, wrappers select attributes or instances customized for the supervised learning algorithm to be used later (to induce a model), which tends to improve predictive performance. However, wrappers are normally much slower than filters. Hence, in applications with large datasets or where efficiency is a critical factor, the filter approach deserves more attention. In addition, in the model tuning approach for generation, post-model optimization (used to improve an existing model) was found to be more popular than pre-model optimization (used before learning the model).

The next stage, selection – where ensemble members are selected to be used in the testing phase – is an optional stage, which is missing in many ensemble learning systems. In this stage, static selection, where the regions of competence of ensemble members are identified at training time, was

Table 3.13: Application domains where evolutionary algorithms for ensemble learning were employed.

Domain	Papers
P300 Speller channel subset optimization	[1, 38]
Software Defect Prediction	[10, 180]
Particle dissolution prediction	[191, 192]
Noise by-pass detection in vehicles	[218]
General classification at hardware-level	[262]
Industrial Machine Fault Prediction	[61]
Internet Traffic Classification	[5]
Recommendation Systems	[250]
Stock market prediction	[42, 178, 179]
Weather forecast	[222]
Estimation of Water Chlorophyll Concentration	[19]
Typhoon Intensity Prediction	[129]
One-class web traffic anomaly detection	[202]
Intrusion Detection	[91, 95, 155, 184]
Microgrid islanding detection	[134]
Power consumption estimation	[109]
Wind Speed Forecasting	[269, 279]
Electric Transformer Fault Prediction	[206, 207]
Anaphora resolution	[237, 241]
Sentiment Analysis	[196, 266]
Spam detection	[16, 17, 253]
Text Entity Recognition	[236, 238–240]
Human Activity Recognition	[84]
Mild Laryngeal Pathology Detection	[256]
Breast Cancer Prediction	[242]
Disease Diagnosis	[57]
Motion Recognition	[41, 247]
Seizure Identification for Epilepsy Diagnosis	[69, 140]
Microarray Data	[4, 43, 135, 136, 138, 169, 170, 172, 203, 204, 217, 229]
Radar image classification	[230]
Resting-State fMRI imaging analysis for Schizophrenia Prediction	[46]
Movement Recognition for Parkinson Disease	[175]
Handwritten Digit Recognition	[234]
Posture Recognition	[188]
Diabetic Retinopathy Detection	[248]
Handwritten Digits Recognition	[63]
Facial emotion recognition	[278]
Steganalysis	[102]
Individual Recognition via Palma Dorsa Vein Pattern	[128]
Object Recognition	[11, 78]
Face recognition	[51–53, 200]
Breast Cancer Prediction	[146, 148–151]
One-class classification	[201]
Imbalanced classification	[21, 22, 32, 33, 104, 231, 261]
General data stream-based classification and regression	[79–82, 93, 94, 125, 130, 131, 181]
General batch-based classification and regression	[6, 8, 11, 28, 31, 36, 39, 41, 45, 48, 54, 56, 58–60, 64, 66, 68, 72–76, 86, 87, 92, 112, 115, 120, 124, 126, 127, 132, 137, 139, 142, 152, 153, 158, 159, 164, 166, 167, 173, 174, 177, 189, 190, 210, 214, 215, 224, 225, 232, 235, 243, 247, 251, 252, 254, 255, 259, 263, 264, 270, 276, 280–282]

found to be much more popular than dynamic selection, where those regions are identified at testing (prediction) time. This seems partly due to the greater simplicity and computational efficiency of the former, since dynamic selection in general requires a more time-consuming process of identifying regions of competence of ensemble members for each testing instance.

In the integration stage, where the predictions of the base learners are integrated into a final prediction for each instance, by far the most popular approach among the surveyed studies was the use of a first degree polynomial – a simple linear approach. Among the non-linear integration techniques, the most common was the use of expression trees, using a genetic programming algorithm. It seems that more research is needed on non-linear techniques for integration, in order to determine whether or not their higher computational complexity could be justified by a significant increase in predictive performance.

Regarding the number of objectives in the fitness function, multi-objective EAs were found to be much more common than single-objective ones. This seems natural, given the multi-objective nature of the ensemble learning problem. In terms of specific types of objectives, effectiveness (predictive performance) is used by all surveyed EAs, since it is essential. Diversity and complexity share a second place, despite diversity being a controversial objective, as discussed earlier. Finally, efficiency, the capacity to generate ensembles that are computationally fast, is optimized in only two studies.

Regarding the main types of EAs used in the surveyed studies, the most popular one was by far Genetic Algorithms (often NSGA-II, a multi-objective GA), followed by Genetic Programming and Differential Evolution.

Regarding the main type of base learner, the most popular one was Artificial Neural Networks (ANNs), followed by decision trees and Support Vector Machines (SVMs). The popularity of ANNs as base learners seems partly due to a long history of interaction in the EA and ANN research areas, and partly due to the nature of ANNs, whose performance can often be improved when using ensembles. However, learning ensembles of ANNs or SVMs tends to be very computationally expensive. This problem is mitigated when learning an ensemble of decision trees (much faster base learners).

4. EEL: ESTIMATION OF DISTRIBUTION ALGORITHMS FOR ENSEMBLE LEARNING

While reviewing related work for the survey (described in Chapter 3), we developed our own solution for inducing ensembles with evolutionary algorithms. It was published in 2018 at the IEEE Congress on Evolutionary Computation and is called EEL – Estimation of Distribution Algorithms for Ensemble Learning.

Seeking to use the best practices from classic ensemble-learning algorithms, while also performing a robust global optimization, we propose to integrate several decision trees for classification from a prior AdaBoost execution into one ensemble. Decision trees are one of the most popular models due to their robustness to noise, speed regarding both training and prediction, and ability to deal with redundant attributes [14, 15, 29]. There are exponentially many decision trees that can be built from the same dataset, with different levels of predictive quality and compactness. Indeed, inducing decision trees is a combinatorial problem, with complexity NP-hard for generating an optimal decision-tree, and NP-complete for generating a minimal binary decision-tree [29].

EEL comprises two steps: generation of base classifiers, and subsequent integration. AdaBoost originally uses two sets of weights: instance weights (denoting the importance of correctly classifying that instance in that given iteration of the AdaBoost process), and classifier weights (the overall vote weight of that classifier in the final ensemble prediction). AdaBoost does not take into account that some classifiers might perform differently among classes (i.e., that a classifier performs better at detecting the positive class than the negative class, or vice-versa). It is possible to induce a set of voting weights, with one weight *per* classifier *per* class, to overcome that limitation. EEL is capable of starting a search procedure from a previous high-quality region in the fitness landscape – a characteristic inherited from Estimation of Distribution Algorithms. We test the proposed strategy in 15 UCI datasets, improving AdaBoost predictive performance to up to $\approx 11\%$.

The rest of this chapter is organized as follows. Section 4.1 goes into further detail on the implementation of the algorithm. Section 4.2 presents the experimental setup, while Section 4.3 presents the experimental results. We draw our conclusions in Section 4.4.

4.1 Proposed Method

EEL, being an Estimation of Distribution Algorithm, has a probabilistic graphical model (GM), a structure used to encode variables in the problem. The GM is a matrix of size $C \times B$, where B is the number of decision trees and C the number of classes. Each cell in this matrix is a Gaussian with mean $\bar{x}$ and standard deviation σ, as depicted in Figure 4.1.

EEL makes use of a univariate EDA, which means its GM assumes that there is no correlation between the voting weights of two base classifiers. Due to the nature of the problem, in which we

	b_1	b_2	b_3	b_4	b_5
c_1	0.96	0.97	1.18	1.07	1.06
c_2	0.96	0.97	1.18	1.07	1.06
c_3	0.96	0.97	1.18	1.07	1.06

Figure 4.1: Initial probabilistic graphical model used in the EDA optimization step, with 5 base classifiers and 3 classes. Each cell comprises the mean of a Gaussian distribution for that voting weight. Initial values come from former AdaBoost classifier weights. Note that there is no restriction for a single classifier to sum its votes to 1 (i.e., the rows do not sum up to 1). Also note that since we use one weight *per* classifier *per* class, in the first generation of EEL the weights are repeated among classes, since they were not learned yet.

have several classifiers casting votes that are further integrated, we are aware that this is a naïve assumption, but in practice univariate EDAs can provide sufficiently high-quality solutions while also being computationally efficient [117].

From this initial GM we sample S solutions. A solution is a matrix $B \times C$ representing the an ensemble's voting weights. Note that each classifier within the ensemble never changes its predictions with regard to the training set; the aim of our method is to change the whole ensemble prediction by adjusting the voting weights of each classifier, $w_{b,c} \forall b \in [1, B], c \in [1, C]$.

4.1.1 Fitness Computation

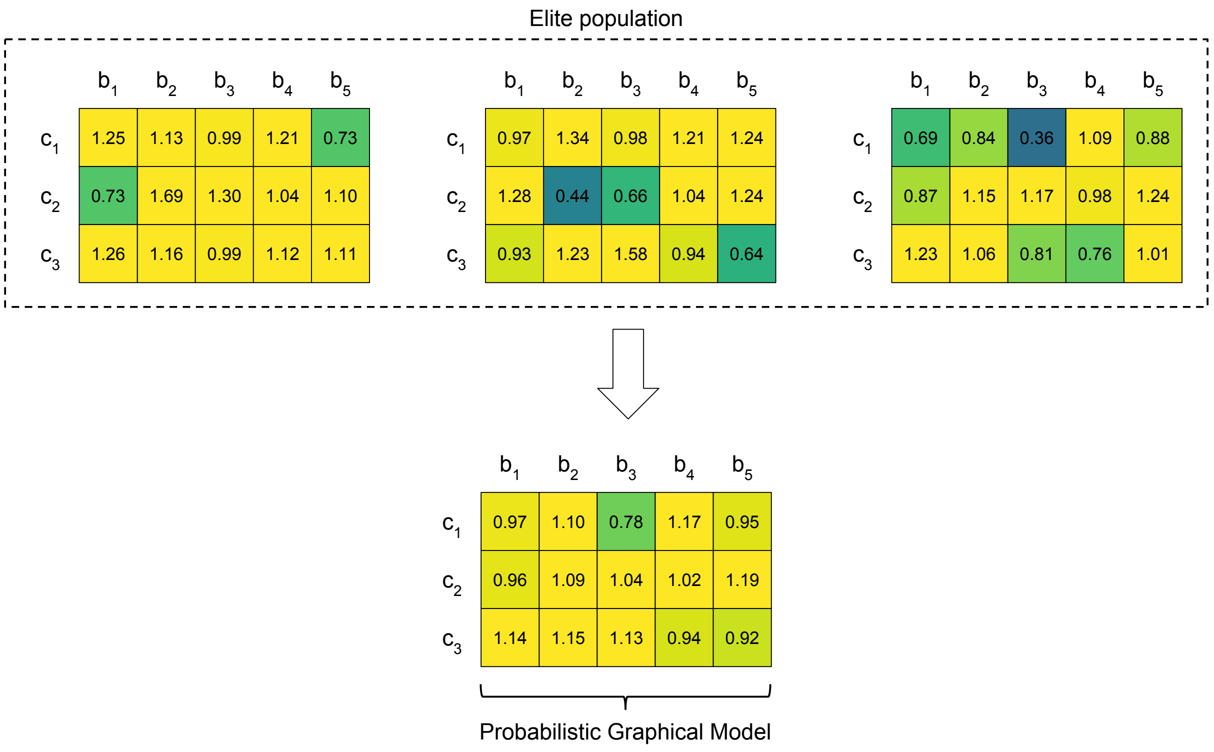
We compute the fitness of individuals (ensembles) as follows. Each individual outputs a $C \times N$ matrix, where N is the number of instances in the training set and C the number of classes. The highest score in each column denotes the prediction for that instance. We sum the scores of the **incorrectly** predicted instances and use that as the individual's fitness. Hence, the objective of the EDA is to decrease the median fitness throughout evolution, thus decreasing the level of certainty in incorrect predictions. Figure 4.2 depicts the fitness computation for a single individual.

4.1.2 Updating the Probabilistic Graphical Model

After computing the fitness of a population, we separate the individuals into two subsets. The first subset comprises all individuals that surpass the median fitness of the current population (elite), whereas the second comprises individuals with fitness below or equal to the median. The former will update the probabilistic graphical model (GM), and will be preserved for the following generation, as depicted in Figure 4.3. The rest of the population will be sampled from the updated GM.

	$X^{(1)}$	$X^{(2)}$	$X^{(3)}$	$X^{(4)}$
c_1	**25.65**	14.71	22.98	24.61
c_2	2.34	**15.12**	17.95	22.18
c_3	0.55	5.48	13.67	**28.53**
Y	c_2	c_1	c_1	c_1

Figure 4.2: Ensemble scores for instances in a given training set, along with truth labels (Y). Each row denotes a class, and each column an instance. Grey cells indicate the prediction of the ensemble for that instance, with bold values denoting **incorrect** predictions. Fitness for the depicted individual is 25.65 + 15.12 + 28.53 = 69.3, and the aim of EEL is to decrease the fitness of individuals.



	b_1	b_2	b_3	b_4	b_5
c_1	1.25	1.13	0.99	1.21	0.73
c_2	0.73	1.69	1.30	1.04	1.10
c_3	1.26	1.16	0.99	1.12	1.11

	b_1	b_2	b_3	b_4	b_5
c_1	0.97	1.34	0.98	1.21	1.24
c_2	1.28	0.44	0.66	1.04	1.24
c_3	0.93	1.23	1.58	0.94	0.64

	b_1	b_2	b_3	b_4	b_5
c_1	0.69	0.84	0.36	1.09	0.88
c_2	0.87	1.15	1.17	0.98	1.24
c_3	1.23	1.06	0.81	0.76	1.01

	b_1	b_2	b_3	b_4	b_5
c_1	0.97	1.10	0.78	1.17	0.95
c_2	0.96	1.09	1.04	1.02	1.19
c_3	1.14	1.15	1.13	0.94	0.92

Figure 4.3: Example of an update on GM probabilities, using three elite individuals. The new GM values are simply the mean value found in the elite population.

The GM comprises Gaussian distributions that are updated by computing the mean value among elite individuals – each cell is a Gaussian mean. The standard deviation is stored in another variable, which is initially set to σ (hyper-parameter), and is iteratively decreased by a factor of τ (another hyper-parameter) at every generation, as recommended by [117].

EEL repeats the process of sampling, calculating fitness, and updating the GM until a sufficient number of generations have been achieved, or the median fitness has not improved over δ for Δ generations. Once the evolutionary procedure halts, we use the fittest individual from the last generation as the selected solution. The pseudocodes for training and prediction phases are shown in Figures 4.4 and 4.5, respectively.

```
1: function TRAIN($B$, $G$, $\delta$, $\Delta$, $\bar{x}$, $\sigma$, $\mathbf{X}$, $Y$)
2:     run AdaBoost with $\mathbf{X}$ and $Y$
3:     use AdaBoost base classifiers for initializing the EDA
4:     initialize the GM using $\bar{x}$ and $\sigma$
5:     $g \leftarrow 0$
6:     while $g < G$ and GM has not yet converged do
7:         sample population $S$ from GM
8:         assess the population fitness, as shown in Figure 4.2
9:         if the median fitness improves $< \delta$ for $\Delta$ generations then
10:            Early stop
11:        use the individuals with fitness $>$ median to update GM
12:        resample individuals replacing those with fitness $\leq$ median
13:        $\sigma \leftarrow \sigma - \tau$
14:        $g \leftarrow g + 1$
15:    return best individual from last generation
```

Figure 4.4: EEL pseudocode for training phase.

```
1: function PREDICT($\mathbf{X}$)
2:     $H \leftarrow 0_N$
3:     for $i \in [1, N]$ do
4:         $\eta \leftarrow 0_C$
5:         for $b \in [1, B]$ do
6:             $\eta_{h_b(\mathbf{X}^{(i)})} \leftarrow \eta_{h_b(\mathbf{X}^{(i)})} + w_{b,c}$
7:         $H_i \leftarrow \text{argmax}(\eta)$
8:     return $H$
```

Figure 4.5: EEL pseudocode for prediction phase.

4.1.3 Complexity Analysis

The complexity of inducing a decision tree is $O(\eta MN \log N)$, with η being the number of nodes, M the number of attributes and N the number of instances in the training set. AdaBoost runs this process for B iterations, thus $O(B\eta MN \log N)$.

Generating the matrix of predictions (of size BN) has complexity $O(BHN)$, with B being the number of base classifiers, H the height of the current decision tree and N the number of training instances. This matrix does not change overtime, and will later be used to perform the predictions.

For each solution in the EDA, sampling new weights from the probabilistic graphical model has complexity of $O(BC)$, with B being the number of base classifiers and C the number of weights *per* classifier. This is done for $S - \Phi$ individuals (i.e., those that have to be replaced for the following generation). The whole ensemble prediction procedure, including the sum of votes of incorrect instances, has complexity of $O(BN)$, since it requires only looking up the prediction table for each classifier and for each training instance (as shown in line 6 of Figure 4.5). Updating the probabilistic graphical model based on the Φ fittest individuals has complexity of $O(\Phi BC)$.

Since the EDA evolves S solutions and repeats its process for G generations, the complexity of the EDA optimization procedure is $O((BHN) + G \times ((S - \Phi)(BC + BN) + \Phi BC))$ (with Φ being zero in the first generation). With the former AdaBoost complexity and after some simplification we have:

$$O((BN(H + \eta M \log N)) + (G((S - \Phi)(B(N + C(1 + \Phi)))) \tag{4.1}$$

4.2 Experimental Setup

In this section we describe the experimental setup conducted on EEL to asses its predictive performance, particularly regarding datasets used, baselines to compare, and metric used to assess quality.

4.2.1 Baseline Algorithms

We compare our proposed method with the original implementation of AdaBoost, as well as two other variations. Comparing to AdaBoost is intuitive, since our algorithm further optimizes its ensemble learning procedure by expanding the set of voting weights available for prediction – that is, using multiple weights per classifier, as opposed to AdaBoost single-weight strategy. The other two baselines are modifications to AdaBoost. The first modification, AdaBoost-ones, sets all voting weights to one (i.e., every classifier has the same importance). This strategy may reduce AdaBoost's tendency to overfit, since it originally gives more importance to classifiers that can correctly predict outliers. The second modification, AdaBoost-normal, samples the voting weights from a normal distribution with $\bar{x} = 1$ and $\sigma = 0.25$. This version serves as a "null hypothesis" (or random classifier) regarding the weight optimization strategy: if this variation performs reasonably well in most datasets, then it would indicate that voting weights may have no significant impact in instance prediction and thus can be safely discarded. All algorithms are tested with $B = 50$ decision trees in their ensemble. The hyper-parameters for EEL are described in Table 4.1, and no effort was made to optimize these values.

Table 4.1: Hyper-parameters used for EEL.

Parameter	Description	Value
B	Number of decision trees	50
G	Number of generations	50
S	Number of individuals	100
δ	Fitness threshold	0.01
Δ	Generation threshold	5
$\bar{x}$	Gaussian Mean	AdaBoost weights
σ	Gaussian Standard deviation	0.25
τ	Standard deviation decrease	0.005

4.2.2 Datasets

We evaluate EEL and baseline methods in 15 UCI Datasets [165], shown in Table 4.2. All datasets present only numeric attributes and no missing data. The restriction on numeric attributes is due to the implementation of decision trees we are using, which is from Python's scikit-learn package [205], though it is not a conceptual limitation of our method *per se*. We perform a three-fold cross-validation, with $2/3$ folds being used as training set and $1/3$ as test set, for each run. We repeat the three-fold cross validation 10 times.

Table 4.2: Datasets used in our experiments.

Dataset	Instances	Attributes	Min class	Max class	classes
diabetes	768	8	268	500	2
ecoli	336	7	2	143	8
glass	214	9	9	76	6
hayes roth	160	4	31	65	3
ionosphere	351	33	126	225	2
iris	150	4	50	50	3
KDD synth control	600	60	100	100	6
liver disorders	345	6	145	200	2
segment	2310	18	330	330	7
semeion	1593	265	158	1435	2
sonar	208	60	97	111	2
vehicle	846	18	199	218	4
wine	178	13	48	71	3
winequality red	1599	11	10	681	6
winequality white	4898	11	5	2198	7

4.3 Experimental Results

We use a Friedman aligned ranks [121] test for evaluating if algorithms within a group are statistically different among themselves. As stated by Demšar [70], the original Friedman Test [101] requires more than five classifiers, and more than ten datasets to yield a significant analysis, under the risk of being too conservative on the results. Since we are here comparing four methods, we use Friedman aligned ranks, which is more appropriate for smaller sets [108].

If the Friedman aligned ranks test detects a statistical difference within the group of algorithms, we proceed the analysis with a Nemenyi post-hoc test [185]. Both tests were conducted with a significance level of 0.05, which is standard for most machine learning experiments. We use the STAC site [221] [1] for statistical tests, and Orange Python library [2] for generating critical difference graphics. The exact script used is available in our Github repository[3].

The Friedman Aligned ranks yields a *p*-value of $5e-5$, which rejects the null hypothesis (that is, that two or more algorithms within the group present the same average predictive performance) with great confidence. We can then proceed with the Nemenyi test. The critical difference graph is depicted in Figure 4.6. EEL is the best algorithm on average, followed by Adaboost, Adaboost-ones, and finally Adaboost-normal. We can draw the following conclusions: first, simply guessing voting weights (Adaboost-normal) is not sufficient to solve the problem. Secondly, there is a reason why Adaboost assigns different voting weights; not all classifiers are the same, which is what Adaboost-ones is performing. EEL is not statistically equivalent to any one of these algorithms, which reinforces our opinion that a careful optimization must be performed to extract the best voting weights from Adaboost.

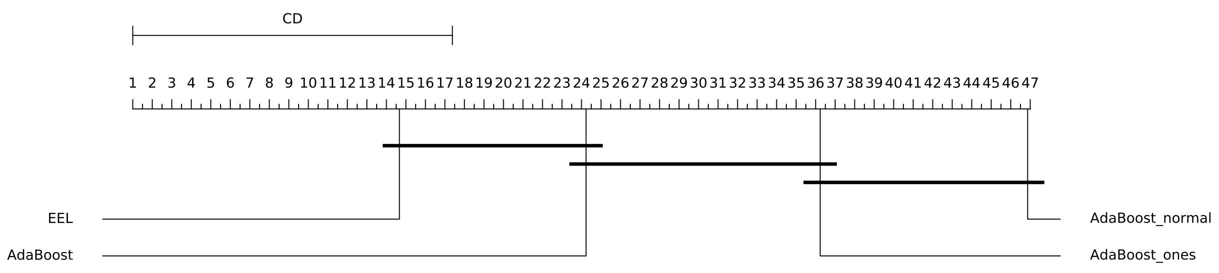


Figure 4.6: Critical difference graph for Nemenyi post-hoc test, on compared algorithms. Recall that ranks are aligned.

We proceed the analysis, now comparing EEL to Adaboost. It is true that both methods were found statistically equivalent; however EEL performs better on average than Adaboost. As shown in Table 4.3, EEL is the best algorithm in 9 out of 15 datasets, with Adaboost comparatively being the best in only one – Adaboost-ones performs better, with 5 wins. Also, Adaboost performs better than EEL in only one dataset, tying with it in another two (or, in other words, Adaboost manages to be better or at least as good as EEL on only three occasions). Based on this data, from all compared algorithms, we recommend EEL as the best algorithm in terms of predictive performance.

[1]Available at http://tec.citius.usc.es/stac/index.html. Accessed July 17 2021.

[2]Available at https://orange3.readthedocs.io/projects/orange-data-mining-library/en/latest/index.html, accessed July 22 2021.

[3]https://github.com/henryzord/thesis_experiments

Table 4.3: Mean accuracy across 10 runs of a three-fold cross-validation procedure. Bold values indicate the best algorithm for that dataset.

Dataset	Adaboost	Adaboost-ones	Adaboost-normal	EEL (ours)
diabetes	0.755 ± 0.000	0.727 ± 0.000	0.716 ± 0.013	$\mathbf{0.756 \pm 0.004}$
ecoli	0.705 ± 0.000	0.720 ± 0.000	0.661 ± 0.089	$\mathbf{0.818 \pm 0.003}$
glass	0.579 ± 0.000	0.561 ± 0.000	0.487 ± 0.034	$\mathbf{0.598 \pm 0.004}$
hayes roth	0.594 ± 0.000	$\mathbf{0.600 \pm 0.000}$	0.598 ± 0.003	0.598 ± 0.003
ionosphere	0.917 ± 0.000	$\mathbf{0.926 \pm 0.000}$	0.920 ± 0.004	0.917 ± 0.006
iris	0.933 ± 0.000	$\mathbf{0.940 \pm 0.000}$	0.933 ± 0.004	0.933 ± 0.000
KDD synth control	0.745 ± 0.000	0.733 ± 0.000	0.717 ± 0.016	$\mathbf{0.784 \pm 0.003}$
liver disorders	0.725 ± 0.000	0.635 ± 0.000	0.654 ± 0.019	$\mathbf{0.730 \pm 0.006}$
segment	0.818 ± 0.000	0.777 ± 0.000	0.691 ± 0.065	$\mathbf{0.866 \pm 0.000}$
semeion	0.964 ± 0.000	$\mathbf{0.967 \pm 0.000}$	0.954 ± 0.007	0.966 ± 0.001
sonar	$\mathbf{0.815 \pm 0.002}$	0.797 ± 0.007	0.797 ± 0.014	0.809 ± 0.017
vehicle	0.621 ± 0.000	0.603 ± 0.000	0.589 ± 0.012	$\mathbf{0.664 \pm 0.009}$
wine	0.944 ± 0.000	$\mathbf{0.961 \pm 0.000}$	0.950 ± 0.004	0.953 ± 0.006
winequality red	0.548 ± 0.000	0.477 ± 0.000	0.441 ± 0.018	$\mathbf{0.566 \pm 0.003}$
winequality white	0.473 ± 0.000	0.415 ± 0.000	0.408 ± 0.032	$\mathbf{0.494 \pm 0.001}$
Wins (including ties)	1	5	0	9
Average rank	2.50	2.37	3.53	1.60
Average rank (aligned)	24.23	36.23	46.87	14.67

4.3.1 Execution Analysis

In this section we analyze how EEL conducted its evolutionary search procedure. Figure 4.7 depicts the mean of the Gaussian distributions throughout EEL's procedure, in a given run on the *ecoli* dataset. In the first generation (i.e., lowermost line), weights come from a previous Adaboost execution. Recall that Adaboost uses one weight *per* classifier (that is, it yields $B = 50$ weights at the end of its execution), whereas EEL uses one weight *per* classifier *per* class (i.e., it must start with $B \times C = 50 \times 7$ weights, with B as the number of base classifiers and C the number of classes). Due to that fact, we broadcast Adaboost weights into the first Gaussians of the GM (initial population), as explained in Figure 4.1. Note that EEL adapts to the classifiers strengths and weaknesses in certain classes, as demonstrated by the smaller line trends in Figure 4.7.

By looking at the standard deviation of EEL in Table 4.3, it is possible to see that the proposed method is stable. This could mean that using the scores of incorrect predictions as fitness function provides a smooth fitness landscape, with few local minima. To better visualize it, we present the known fitness landscape in Figure 4.8. The horizontal axis is the mean of the first $W/2$ weights, and the vertical axis the mean of the last $W/2$ weights. Note that this projection increases the likelihood that two individuals are in the same position in the projection. To counter this undesired effect, we show the mean fitness of all individuals occupying that spot in the projection. Please note that this is an oversimplification of the fitness landscape to fit in a 2D projection. With this in mind, one can verify that it appears to exist a region in the left-down portion of the landscape with better individuals. Indeed, we verified in our experiments that EEL starts with a population in the top-right corner that migrates to the bottom-left as the evolution progresses.

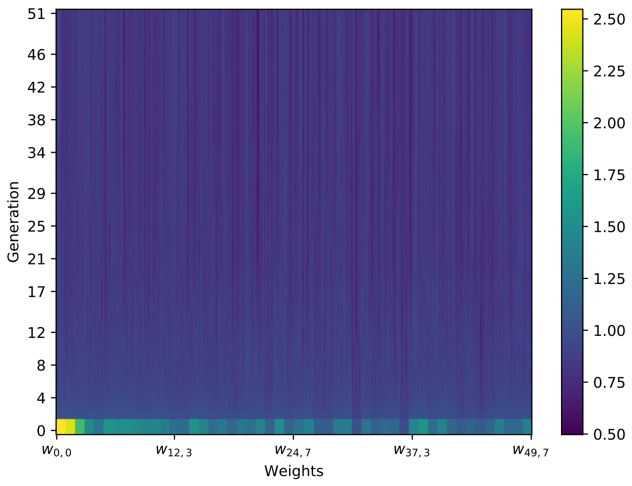


Figure 4.7: Probabilistic graphical model evolution on the *ecoli* dataset. Vertical axis denotes the generations whereas the horizontal axis denotes the $50 \times 7 = 350$ weights for that particular dataset. Brighter colors indicate more importance (heavier weights) in a given region. Note that EEL penalizes for incorrect predictions per class, as opposed to Adaboost that penalizes the entire classifier.

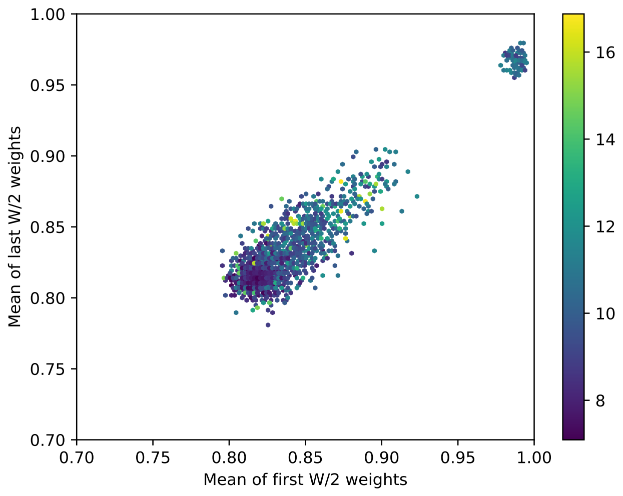


Figure 4.8: Fitness landscape of EEL. Darker spots indicate valleys (better regions), whereas brighter spots indicate peaks.

4.4 Discussion and Final Remarks

In this chapter we introduced EEL: Estimation of Distribution Algorithms for Ensemble Learning. EEL resumes the optimization pipeline started by AdaBoost. It uses AdaBoost's previous base classifiers and finds a new set of weights for those classifiers by conducting a population-based global search procedure with the aim of minimizing the error rate of ensemble predictions.

By using multiple weights, one weight *per* classifier *per* class, as opposed to AdaBoost, which uses a single weight *per* classifier, we are capable of enhancing AdaBoost in 12 cases, and producing the best results in 9 out of 15 datasets.

As future work for this specific algorithm, we would like to investigate whether a Pareto approach for decreasing error rate and maximizing accuracy can yield better results than simply using a single-objective EDA. We also intend to consider a multivariate probabilistic graphical model, which may improve the quality of the produced ensembles. In addition, we would like to investigate whether Random Forests benefits from a voting weight optimization procedure, since it assigns the same importance to all of its base classifiers.

5. PUMA: PROBABILISTIC UNIVARIATE ESTIMATION OF DISTRIBUTION ALGORITHM FOR ENSEMBLE LEARNING

If we want to deploy truly interpretable classifiers in real-life problems, we need to make use of white-box models. As explained in Section 2.3, building explanation models is not enough – they could disagree with the prediction models, and for some applications (such as medicine) this may be unacceptable. Secondly, since white-box models do not perform as well as black-box models, an appropriate induction strategy must be chosen to allow the former to achieve comparable performance to the latter.

PUMA – **P**robabilistic **U**nivariate Esti**m**ation of Distribution **A**lgorithm for Ensemble Learning – is our first take on ensemble learning with evolutionary algorithms aimed specifically at building interpretable models. It was published in the Proceedings of the Brazilian Conference of Intelligent Systems [30], in 2020.

PUMA performs Auto-ML, an area that has recently gained attention due to its capability of replacing the manual optimization step of the algorithms' hyper-parameters, which can be a repetitive and tiresome task, and often requires advanced domain-specific knowledge [90, 194, 272].

There are several ways to perform Auto-ML, but we deem appropriate to employ evolutionary algorithms for such a task. Evolutionary algorithms can explore several regions in the solution space in parallel, adapting its search depending on the quality of solutions found in those regions, and performing a global-search in the space of solutions [105, 144, 161, 272].

PUMA selects the best hyper-parameters for a small set of ensemble members, all of them white-box models. Selecting the best setting (or configuration) of hyper-parameters for each base learner in an ensemble is a difficult task *per se* [90, 249], which involves testing a very large number of candidate settings in order to find the most suitable configuration for the dataset at hand. Besides optimizing hyper-parameters, our method can also choose to use a number of base learners varying within $[1, 5]$ in the final ensemble, along with the most appropriate aggregation policy (the strategy to use to aggregate vote weights for the final predictions).

We conducted experiments to verify the predictive performance of PUMA against a set of baseline algorithms, which includes Adaboost and Random Forests. PUMA, as shown later in this chapter, outperforms Adaboost while being statistically equivalent to Random Forests. We recall that neither one of these two baseline algorithms is interpretable; both methods generate ensemble members that perform predictions on very specific regions of the input space, either by being tailored to predict outliers (Adaboost) or producing trees with feature randomization (Random Forests).

The rest of this chapter is organized as follows. Section 5.1 describes the proposed method. Sections 5.2 and 5.3 present the experimental setup and results, respectively. Section 5.4 presents the conclusions and future research directions regarding this particular method.

5.1 Proposed Method

EDAs evolve a probabilistic graphic model of candidate solutions, so that each solution (individual) is sampled from that model and evaluated at each generation. In general, an EDA consists of three stages performed at each generation: (a) sampling of new individuals (candidate solutions) from the probabilistic graphic model; (b) evaluation of the new individuals' performance; and (c) updating of the probabilistic graphic model based on the best individuals selected from the current generation. Importantly, EDAs avoid the need for specifying genetic operators like crossover and mutation (and their corresponding probabilities of application). Instead of generating new individuals by applying genetic operators over pre-selected candidates, they generate new individuals by sampling from the current probabilistic graphic model, which captures the main characteristics of the best individuals (based on fitness) along the evolutionary process.

Among several EDA types, PUMA is based on PBIL: Probabilistic Incremental Learning [13]. The main characteristic of PBIL is that it assumes independence between variables in the probabilistic graphical model. Although this has the disadvantage of ignoring interactions among variables, it has an important advantage in the context of our task of evolving an ensemble of classifiers: it makes PBIL much more computationally efficient by comparison with other EDA types that consider complex variable interactions, whilst still allowing PBIL to learn ensembles with good predictive accuracy, as shown later.

Another aspect of PBIL is the use of a learning rate α hyper-parameter for updating probabilities in the graphical model, making this process smoother. Take for example two initial probabilities for a binary variable V, $P(V = 0) = 0.5$ and $P(V = 1) = 0.5$, and a learning rate of 0.3. Assume only two individuals are selected to update the graphic model's probabilities, and both have $V = 0$. In this extreme case, an EDA without learning rate would update V so that it would be $P(V = 0) = \frac{2}{2} = 1$ and $P(V = 1) = \frac{0}{2} = 0$ in the next generation. However, using a learning rate, the new probabilities for V are $P(V = 0) = (1 - 0.3) \times 0.5 + 0.3 \times \frac{2}{2} = 0.65$ and $P(V = 1) = (1 - 0.3) \times 0.5 + 0.3 \times \frac{0}{2} = 0.35$. Section 5.1.3 discusses in more detail how probabilities are updated.

PUMA keeps track of the best individual found so far in variable φ. At the end of a PUMA run, the returned solution can be the best individual stored in φ or the best individual in the last generation (these two approaches are compared in the experimental analysis). An overview of PUMA's pipeline is shown in Figure 5.1.

5.1.1 Individuals

Each individual is an ensemble and comprises five base models (each learned by a different type of base learner) and an aggregation policy. For the base learners, we choose the ones that can generate readily-interpretable models [98, 123, 209].

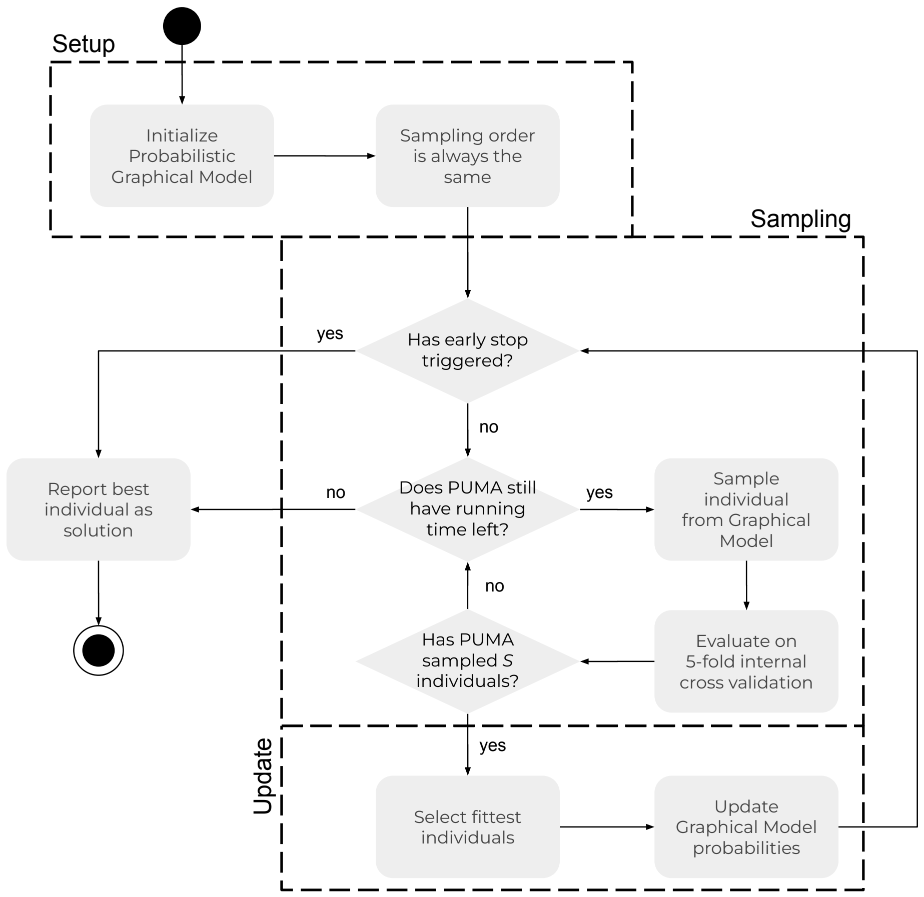

Setup
Initialize Probabilistic Graphical Model
Sampling order is always the same
Sampling
yes
Has early stop triggered?
no
Report best individual as solution
no
Does PUMA still have running time left?
yes
Sample individual from Graphical Model
no
Has PUMA sampled S individuals?
Evaluate on 5-fold internal cross validation
Update
yes
Select fittest individuals
Update Graphical Model probabilities

Figure 5.1: Overview of the processing pipeline of PUMA.

The five base learners employed are: two decision-tree induction algorithms (C4.5 [213] and CART [25]); two rule induction algorithms (RIPPER [50] and PART [96]); and a decision table algorithm [143]. We use the implementations of those algorithms in the well-known Weka Toolkit [113]. For the rest of this chapter, we will refer to them by their Weka names: J48 for C4.5, SimpleCart for CART, JRip for RIPPER, PART, and Decision Table.

An individual is encoded as an array, where each position denotes a variable and each value denotes the assigned value for that variable. Some variables may not have assigned value, because they are not used by an individual. Figure 5.2 depicts a portion of an individual's array regarding some variables of its J48 classifier. J48 has three options for tree pruning: *reduced error pruning*, *confidence factor*, and *unpruned*. In this example, *reduced error pruning* is used. For that reason, there is no need to set hyper-parameters to the *confidence factor* strategy, which are then set to *null*.

Aggregators

An aggregator is a method responsible for finding consensus among votes from base models. Consider a three-dimensional probability tensor $\mathbf{P}$, of dimensions (B, N, C) – respectively the number of base classifiers in the ensemble, number of instances, and number of classes. The objective of

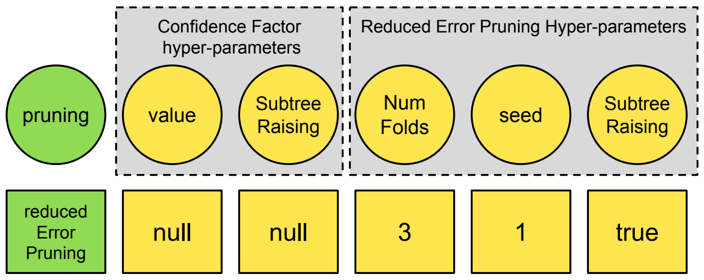


Figure 5.2: An example individual in PUMA. Although PUMA assumes probabilistic independence between variables, some values do dependent on others.

an aggregator is to transform this three-dimensional tensor into a unidimensional array of length N, where each position has the predicted class for each instance.

We use two types of aggregators: majority voting and weighted aggregators. The probabilistic majority voting aggregator uses the fusion function described in [156, p. 150]:

$$\rho_c^{(j)} = \sum_{i=1}^{B} P_c^{(i,j)} \tag{5.1}$$

$$h(X^{(j)}) = \arg\max_{c \in C} \left(\frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{5.2}$$

where $\rho_c^{(j)}$ is the sum of the probabilities that the j-th instance belongs to the c-th class over all B classifiers, with C being the number of classes. The weighted aggregator is similar to majority voting, except that individual probabilities from classifiers are weighted according to the fitness of each classifier:

$$\rho_c^{(j)} = \sum_{i=1}^{B} \psi^{(i)} P_c^{(i,j)} \tag{5.3}$$

$$h(X^{(j)}) = \arg\max_{c \in C} \left(\frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{5.4}$$

where $\psi^{(i)}$ is the fitness value of individual $S^{(i)}$.

5.1.2 Fitness evaluation

At the start of the evolutionary process, PUMA receives a training set as input. This set is split into five subsets, which are used to compute each individual's fitness by performing an internal 5-fold stratified cross validation (SCV). By keeping the subsets constant throughout all evolutionary process, we allow direct comparisons between individuals from different generations. The fitness function is the Area Under the Receiving Operator Characteristic (ROC) curve (AUC) [85], a popular predictive measure.

AUC values are within $[0, 1]$, with 0.5 representing the performance of a random classifier in the case of binary-class problems. For PUMA, regardless of the number of classes in the dataset, we calculate one AUC for each class, and then average the AUC among all classes. Hence, the fitness of an individual is actually a mean of means: first, the mean AUC among all classes, for a given fold; then, the mean AUC among all five internal folds. Figure 5.3 depicts the fitness calculation procedure.

```
1:  function COMPUTE_FITNESS($\mathbf{X}, y, C$)
2:      train $\leftarrow$ (generate_train_subsets($y$))
3:      val $\leftarrow$ (generate_validation_subsets($y$))
4:      $\Psi \leftarrow (0 | i = 1, 2, \dots, |S|)$
5:      for $i = 1, 2, \dots, |S|$ do
6:          for $k = 1, \dots, 5$ do
7:              $S^{(i)} \leftarrow$ build_model($\mathbf{X}^{(\text{train}^{(k)})}, y^{(\text{train}^{(k)})}$)
8:              $P^{(i)} \leftarrow$ predict($S^{(i)}, \mathbf{X}^{(\text{val}^{(k)})}$)
9:              $\psi \leftarrow \frac{1}{5C} \sum_{c=1}^{C} \text{AUC}(P_c^{(i)}, (y^{(j)} = c | j \in \text{val}^{(k)}))$
10:             $\Psi^{(i)} \leftarrow \Psi^{(i)} + \psi$
11:             )
12:     return $\Psi$
```

Figure 5.3: Pseudo-code for fitness calculation.

5.1.3 PUMA's Probabilistic Graphical Model

At each generation, new individuals are sampled from the probabilistic graphical model (GM), and the best individuals will update the GM's probabilities. Recall that PUMA assumes that the variables in the GM are independent, though we know (as shown in Figure 5.2) that there exist dependencies. However, this does not prevent PUMA from finding good-performing solutions, analogously to the overall good performance of the Naïve Bayes classifier, which also assumes that attributes are independent [220].

The sampling procedure is based on hierarchical relationships among the variables representing hyper-parameters in PUMA's GM, as shown in Figure 5.4, where the top-level variables are hyper-parameters of base learners that will activate or deactivate the sampling of other variables/hyper-parameters at a lower level. When sampling a new individual, higher-level variables are sampled first, and their descendants are sampled next. Using J48 as example, the variables for this algorithm are *useLaplace, minNumObj, useMDLcorrection, collapseTree, doNotMakeSplitPointActualValue, binarySplits*, and *pruning*. Since none of these variables have any descendant variable with the exception of *pruning*, the sampling proceeds to choose which type of pruning will be used by J48, and depending on the chosen option, it samples the variables descendant to that option. Unused variables are set to *null*. Once all pertinent variables are sampled, their values are fed to the base classifier constructor, which will in turn generate the model. Figure 5.4 depicts the variables in PUMA's GM.

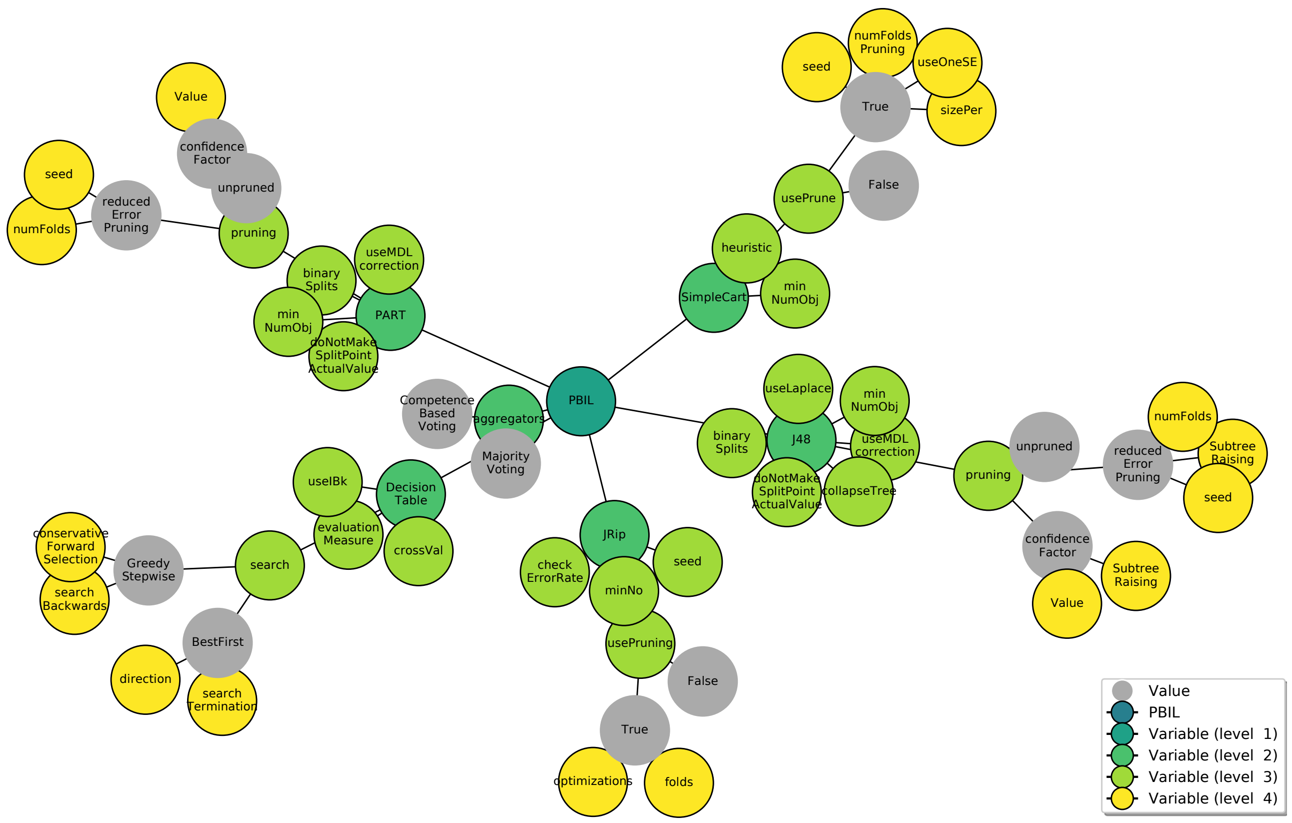


Figure 5.4: Graphical model used by PUMA. Edges denote an implicit correlation, since no probabilistic correlation is designed. Borderless nodes denote values that were sampled from the variable in the above level (not variables).

Initial values

There are two types of variables in PUMA's GM: 48 of them are discrete and 2 are continuous. Discrete variables were first introduced in the original PBIL work [13]. We use the EDA ability of biasing probabilities to increase by 10% the probability to sample values that are the base learner's default in Weka. For all other values, we set uniform probabilities. For instance, for J48's *numFolds*, the default value 3 folds has probability 20%, while each other value in $\{2, 4, 5, 6, 7, 8, 9, 10\}$ has probability 10%. Exceptionally for variable *evaluationMeasure* of Decision Table, *auc* has a 50% probability of being sampled. We do this to increase the chances that a base learner is using the same metric used as our fitness function, which for PUMA is the AUC.

For continuous variables, we use unidimensional Gaussian distributions. The mean is the default Weka value for the hyper-parameter, and the standard deviation was chosen in a way that borderline values have at least 10% chance of being sampled. Values outside the valid range are clipped to the closest valid value. The range of valid values was inferred by inspecting Weka's source code. The full list of of variables and their values is present in the source-code of our method [1].

Updating PUMA's GM

The updating of the variables' probabilities is dependent on their type. If a variable is discrete, the update follows the scheme known as PBIL-iUMDA [272, 277], shown in Equation 5.5:

$$p_{g+1}(V_j = v) = (1 - \alpha) \times p_g(V_j = v) + \alpha \times p_{\Phi,g}(V_j = v) \tag{5.5}$$

where $p_g(V_j = v)$ is the probability that variable V_j assumes discrete value v in the g-th generation (estimated by the proportion of observed occurrences of value v for variable V_j among all individuals in that generation), α is the learning rate, and $p_{\Phi,g}(V_j = v)$ is the proportion of occurrences of value v for V_j in the set of individuals Φ which were selected (based on fitness) at the g-th generation. This process is iterated over all values of a discrete variable. Note that when computing $p_g(V_j = v)$ and $p_{\Phi,g}(V_j = v)$, if some individuals do not have any value set for variable V_j, their *null* values are discarded and do not contribute at all to the updating of probabilities for V_j's values.

Equation 5.5 was adapted to deal with continuous variables, which encode the mean and the standard deviation of a normal distribution, as follows. The mean of the normal distribution is updated by

$$\mu_{g+1}(V_j) = \mu_g(V_j) + \alpha \times (\mu_g(V_j) - \mu_{\Phi,g}(V_j)) \tag{5.6}$$

where $\mu_g(V_j)$ is the mean of the normal distribution of the j-th variable V_j in the g-th generation, α is the learning rate, and $\mu_{\Phi,g}(V_j)$ is the observed mean for the variable V_j in the set of individuals Φ selected at the g-th generation, again considering only individuals where the variable V_j was used.

The standard deviation is decreased as follows:

$$\sigma_{g+1}(V_j) = \sigma_g(V_j) - \frac{\sigma_1(V_j)}{G} \tag{5.7}$$

where $\sigma_g(V_j)$ is the standard deviation of the j-th variable V_j in the g-th generation, $\sigma_1(V_j)$ the initial standard deviation for variable V_j, and G is the number of generations.

5.1.4 Early-Stopping and Termination

If the fitness of the best individual does not improve more than ϵ in ε generations, we assume that PUMA is stuck in a local optimum solution and we choose to terminate the execution of the algorithm. In our experiments, we use $\epsilon = 5 \times 10^{-4}$ and $\varepsilon = 10$. At the end of the evolutionary process, we report the best individual from the last generation as the final solution.

[1] Available at https://github.com/henryzord/PBIL

5.1.5 Complexity Analysis

Assume $T(train)$ to be the time to train an ensemble, and $T(fitness)$ to be the time to assert the fitness of the given ensemble. At every generation S new ensembles are generated. This process is repeated at most G times (assuming that the early stop mechanism of the previous section is not triggered). This procedure has complexity $GS \times (T(train) + T(fitness))$.

Sampling and updating the graphical model are procedures directly dependent on the number of variables $|V|$. Variables need first to be initialized with default values, for later sampling and update. Variables are sampled S times every generation, and are updated based on the number of fittest individuals, $|\Phi|$. For each variable, we iterate over all of its values but assuming the number of values are not significant – discrete variables have between 2 and 10 values, with 4 as average; continuous variables count as 2 values, i.e., mean and standard deviation of normal distributions. From this analysis we have $|V| \times (1 + G(S + |\Phi|))$. Thus, the overall complexity of training the proposed PUMA is

$$O(GS \times (T(train) + T(fitness)) + |V| \times G(S + |\Phi|)). \tag{5.8}$$

5.2 Experimental setup

We describe in this section the experiments performed on PUMA to assess its performance against baseline algorithms. We choose a nested cross-validation procedure coupled with a grid search in the space of PUMA's hyper-parameters. In this sense, each external fold of the nested-cross validation will have an optimized version of PUMA for that external fold. We choose grid search for simplicity, aware that this is not the state-of-the-art for hyper-parameter optimization; however we did not want to incur on a costly evaluation process, since running PUMA for a single dataset in this setup already takes $12 \times 5 \times 10 = 600$ hours at most.

The nested cross-validation procedure is described as follows. The whole dataset is divided into ten folds, which we will refer to as external folds. Nine folds are used for training, and one for testing. The nine external folds are split again into five folds, which we will refer to internal folds. Different combinations of hyper-parameters, for each classifier, are tested with these five internal folds. The predictions for each internal test fold are stored in an array. At the end of the internal five-fold cross validation, the unweighted Area Under the ROC curve is computed, and the best combination of hyper-parameters is chosen. The best combination is then trained with the nine external folds and cast its predictions on the external test fold. The procedure is described in Figure 5.5.

```
1: function NESTED_CROSS_VALIDATION($H$, $\mathbf{X}$, $Y$)
2:     for $i = 1, 2, \dots, 10$ do
3:         $\mathbf{X}^{(i)}_{\text{train}} = 9/10$ of $\mathbf{X}$
4:         $Y^{(i)}_{\text{train}} = 9/10$ of $Y$
5:         for each hyper-parameter configuration from set of all hyper-parameters configurations do
6:             for $j = 1, 2, \dots, 5$ do
7:                 $\mathbf{X}^{(i,j)}_{\text{train}} = 4/5$ of $\mathbf{X}^{(i)}_{\text{train}}$
8:                 $Y^{(i,j)}_{\text{train}} = 4/5$ of $Y^{(i)}_{\text{train}}$
9:                 train H with hyper-parameter configuration on $\mathbf{X}^{(i,j)}_{\text{train}}$, $Y^{(i,j)}_{\text{train}}$
10:                test H on $\mathbf{X}^{(i,j)}_{\text{test}}$, annotate prediction scores
11:            compute unweighted Area Under ROC curve with annotated prediction scores
12:        choose hyper-parameter configuration with best unweighted AUC
13:        train H with best hyper-parameter configuration on $\mathbf{X}^{(i)}_{\text{test}}$, annotate prediction scores of $Y^{(i)}_{\text{test}}$
14:    compute unweighted AUC with annotated prediction scores
15:    return unweighted AUC
```

Figure 5.5: Nested cross-validation procedure. All subsets follow the same class distribution as the original dataset (in other words, both the external and internal cross-validations are stratified).

5.2.1 Datasets

We selected 20 datasets from both KEEL [2] [3] and UCI Machine Learning repository[3] [165]. The selected datasets are shown in Table 5.1. We select these datasets to cover a wide range or characteristics. Datasets have from 1066 to 10992 instances, from 3 to 61 attributes, and from 2 to 13 classes.

Table 5.1: Datasets used in the experiments.

name	instances	attributes	classes
banana	5300	3	2
banknotes	1372	5	2
car	1728	7	4
contraceptive	1473	10	3
diabetic	1151	20	2
flare	1066	12	6
krvskp	3196	37	2
page_blocks	5472	11	5
penbased	10992	17	10
phoneme	5404	6	2
ring	7400	21	2
seismicbumps	2584	19	2
splice	3190	61	3
steelfaults	1941	34	2
texture	5500	41	11
thyroid	7200	22	3
titanic	2201	4	2
turkiye	5820	33	13
twonorm	7400	21	2
waveform	5000	41	3

[2]Available at https://sci2s.ugr.es/keel/datasets.php
[3]Available at https://archive.ics.uci.edu/ml/datasets

5.2.2 Modifications to PUMA

We perform adaptations to PUMA source-code to allow it to run in a nested-cross validation experiment. We refer to this modified version as PUMA-star. Instead of performing an internal five-fold cross-validation procedure, PUMA-star performs an internal holdout procedure, splitting the training set at a $80/20$ proportion for learning and validation sets, respectively. We perform this modification to avoid that datasets are split into too many smaller subsets under the nested cross-validation procedure. If we were to use an internal five-fold cross-validation, the set used to evolve ensembles would be $\frac{9}{10}\frac{4}{5}\frac{5}{6}\frac{4}{5} = 0.48$ of the whole dataset, which increases the chances that our algorithm overfits the data. By using holdout with 80% of the data for learning and 20% for validation, the data used to learn ensembles is improved to $\frac{9}{10}\frac{4}{5}\frac{8}{10} = 0.576$. We also implement two timeout hyper-parameters, both for general evolution and individual evolution. The hyper-parameters optimized by grid search for PUMA-star are shown in Table 5.2.

Table 5.2: Hyper-parameters used for PUMA-star in the nested cross-validation procedure, coupled with a grid search. Multiple values indicate that the hyper-parameter was optimized; single values were constant among all grid-search iterations.

Algorithm	Hyper-parameter	Values
PUMA-star	Learning rate	{0.13, 0.26, 0.52}
PUMA-star	Selection share	{0.3, 0.5}
PUMA-star	Population size	50
PUMA-star	Generations	100
PUMA-star	Evolution timeout	one hour
PUMA-star	Individual timeout	one minute
PUMA-star	individual to report	{best overall, best from last generation}

5.2.3 Baseline Algorithms and Hyper-parameter optimization

We describe in this section the hyper-parameters used (if fixed) or optimized by grid search (if a range of values is considered) for PUMA-star. We do not optimize population size, leaving it at a fixed rate of 50 individuals per generation, because it is analogous to Random Forests' number of trees: if we could hypothetically have an infinitely large population, then it would be guaranteed to have at least one individual with the best achievable performance, given PUMA-star's constraints (e.g., aggregation policies, algorithm behavior, etc).

In a similar sense, we do not optimize the number of generations nor the time available for overall evolutionary process, leaving these hyper-parameters at 100 generations and one hour, respectively. PUMA-star also has a time limit for evolving individuals, which is implemented in the same way: if an individual exceeds 60 seconds to sample and train its ensemble members, then it is discarded and a new individual must be sampled.

PUMA-star is compared to other algorithms in thematic groups. Table 5.7, at the end of this section, summarizes the composition of all groups. The first group comprises the base classifiers used by PUMA-star, namely J48, SimpleCart, PART, JRip, and Decision Table. The reasoning for comparing PUMA-star to these algorithms is to check whether it is not simpler to simply optimize the hyper-parameters of an already-interpretable well-established classifier. These classifiers also have their hyper-parameters optimized by grid search. The list of optimized hyper-parameters is described in Table 5.3.

Table 5.3: List of base classifiers and hyper-parameters optimized by grid search. Hyper-parameters not shown in this table were kept constant at the default Weka values.

Algorithm	Hyper-parameter	Values
J48	Minimum number of examples in leaf nodes	{2, 4, 6, 8}
J48	pruning policy	{unpruned, reduced error, confidence factor}
SimpleCart	Minimum number of examples in leaf nodes	{2, 5, 8}
SimpleCart	Apply pruning	{true, false}
SimpleCart	Use heuristic for binary splits of categorical attributes	{true, false}
JRip	Minimum number of examples to be covered by a rule	{2, 4, 6, 8}
JRip	use pruning	{true, false}
PART	Minimum number of examples to be covered by a rule	{2, 4, 6, 8}
PART	pruning policy	{unpruned, reduced error, confidence factor}
Decision Table	internal evaluation metric	{AUC, accuracy, Root Mean Squared Error, Mean Absolute Error}
Decision Table	Search Policy	{Greedy Stepwise, Best First}
Decision Table	Default rule assigns majority class or uses IBK	{majority class, IBK}

The second group of algorithms is comprised of baseline ensembles. The first algorithm is an unoptimized baseline ensemble, consisting of the five base classifiers from PUMA-star (J48, CART, JRip, PART, and Decision Table) with their default hyper-parameter configuration and a simple majority voting scheme as aggregation policy. The reasoning for doing so is to check whether there is a difference between simply ensembling these five base classifiers and optimizing their hyper-parameter configurations with an evolutionary algorithm.

The second algorithm in this group is an optimized baseline ensemble. It has the same five base classifiers and also uses majority voting, but each base classifier had its hyper-parameters individually optimized by grid search. We compare evolutionary algorithms to this optimized baseline ensemble in order to verify whether it is worth to apply an evolutionary algorithm to perform global optimization or it is simply better to individually optimize base classifiers and ensemble them with a simple majority voting policy.

The next group has only one algorithm: a random-search procedure with the same time budget of the evolutionary algorithms. We are evaluating whether it is really justified to apply an evolutionary algorithm or simply randomly guessing solutions solves the problem. Since we run PUMA-star with a fixed population of 50 individuals for 100 generations, we use the source code of EDNEL

(Introduced in Chapter 6) to perform random search in the space of solutions with a budget of $5,000$ evaluations, with no compromise to sample distinct solutions. This is achieved by running EDNEL with 5000 individuals and 1 generation. Note that the initial probabilities are the same, as described in Section 5.1 – that is, they are slightly biased towards the default Weka values. Since we are re-using the source code of EDNEL, we have to set some hyper-parameters. The hyper-parameters used are listed in Table 5.4.

Table 5.4: Hyper-parameters used on EDNEL to perform random search in the space of solutions. Hyper-parameters marked with an asterisk (*) have no effect on the outcome of the algorithm run but are listed here since EDNEL requires a value.

Algorithm	Hyper-parameter	Values
Random Search	Learning rate	0.1*
Random Search	Selection share	0.1*
Random Search	Population size	5000
Random Search	Generations	1
Random Search	Evolution timeout	no limit
Random Search	Individual timeout	no limit
Random Search	Burn-in	100
Random Search	thinning factor	0
Random Search	Early stop generations	1*
Random Search	Maximum probabilistic parents per variable (same value for all variables)	0*
Random Search	Delay Structure Learning in N generations	5*
Random Search	evaluation procedure	holdout
Random Search	individual to report	best overall*

The fourth group has six ensembles, each comprising the boosted version (using Adaboost [100]) of each one of the five aforementioned base classifiers, plus Adaboost with its default base classifier (decision stumps). We use the Weka toolkit version of Adaboost, which – translating to the original paper [100] – is the M1 version. We optimize these algorithms with grid search. It is important to note that none of the algorithms in this group is interpretable, nor their ensemble members, since the boosting process produces classifiers with hypothesis that only make sense for that particular distribution of instances. We explain boosting in greater detail in Section 2.1.1.

In Weka, only three hyper-parameters are available for customization: weight threshold ($[0, 1]$) – how much of the training data should be used for training the classifier at each iteration); number of iterations (i.e. number of classifiers in the ensemble); and type of classifier. We decide to leave weight threshold at 1 (i.e., use all training data), and since the type of classifier is not really an optimization option (we will be using six different types of base classifiers), we optimize only the number of iterations, or conversely the number of classifiers in the final ensemble. The choice of ranges are defined in Table 5.5, and we base the choice of values on the work of Rijn and Hutter [258].

Finally, the fifth and last group has also only one classifier, Random Forests [24]. Random Forests is a well-known ensemble algorithm and is in general among the best classification methods regarding predictive performance [88]. A Random Forests ensemble solely comprises decision trees. Each decision tree is learned from a different subset of instances, randomly sampled with replacement from the training set. For each internal node in each tree, a subset of M attributes is randomly sampled without replacement, and the attribute that minimizes the local class impurity is selected as splitting

Table 5.5: Hyper-parameters of Adaboost optimized with grid search. Note that we consider all entries in this table as distinct algorithms.

Algorithm	Hyper-parameter	Values
Adaboost (decision stumps)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (J48)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (SimpleCart)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (JRip)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (PART)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (DecisionTable)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}

criterion. This process is repeated recursively until no further split improves the impurity metric, when nodes are then turned into leaves. We explain Random Forests in greater detail in Section 2.1.4.

Random forests usually require a large number of trees in the ensemble to achieve good predictive performance. Also, despite using decision trees, the ensemble as a whole is not directly interpretable, since there are a very large number of trees. Even if the number of trees were small, interpreting each tree would still be problematic due to the large degree of randomness involved in learning each tree. That randomness is necessary to provide diversity to the ensemble, which improves its predictive accuracy but hinders interpretability.

We also optimize Random Forests' hyper-parameters. Based on [212], we consider the following hyper-parameters and range of values: percentage of the training set that will be used by each tree in the forest in $\{0.9, 1.0\}$; percentage of attributes to randomly sample for considering in a given internal node of a given tree, in relation to the total number of attributes in the dataset in $\{\frac{\sqrt{M}}{2}, \sqrt{M}, 2\sqrt{M}, \frac{\log_2 M}{2}, \log_2 M, 2\log_2 M\}$. We do not optimize the number of trees in the forest, leaving it always at 1000 trees, because it is not really an important hyper-parameter, and it should be left in a sufficiently-large number [212]. A summary of the hyper-parameters and their values is shown in Table 5.6.

Table 5.6: Hyper-parameters optimized (when multiple values) or used (when single value) for Random Forests using grid search. Unmentioned hyper-parameters use the default Weka values.

Algorithm	Hyper-parameter	Values
Random Forest	Number of trees in forest	1000
Random Forest	Share of training set to be used at any given tree (randomly sampled)	{90%, 100%}
Random Forest	Number of features sampled to be used at any given split of trees	$\{\frac{\sqrt{M}}{2}, \sqrt{M}, 2\sqrt{M}, \frac{\log_2 M}{2}, \log_2 M, 2\log_2 M\}$

5.2.4 Hardware specifications and source code

The hardware specifications where we ran PUMA-star are described in Table 5.8. The source code used for performing nested cross-validation is available at https://github.com/henryzord/PBIL.

Table 5.7: Groups of algorithms that were compared to PUMA-star.

Group	Algorithms
Base classifiers	J48 SimpleCart JRip PART DecisionTable
Baseline Ensembles	Unoptimized ensemble Optimized ensemble
Random Search	Random Search
Boosted	Adaboost J48 SimpleCart JRip PART DecisionTable
Random Forests	Random Forests

Table 5.8: Hardware components of the machine on which PUMA-star was run.

Specification	Z Machine
Processor	AMD Ryzen Threadripper 1950X
Core speed	3.4GHz
Cores	32
Architecture	x86_64
RAM Memory	128GB
Operating System	Ubuntu 20.04.2 LTS

5.3 Experimental results

We use an assortment of statistical tests to assess PUMA-star performance. If the group has more than five classifiers (including PUMA-star), we use the Friedman Test [70, 101]. If the group has exactly or less than five classifiers, we use the Friedman aligned ranks [121]. As stated by Demšar [70], the original Friedman Test [101] requires more than five classifiers and more than ten datasets to yield a significant analysis, under the risk of being too conservative on the results. Friedman aligned ranks is more appropriate for smaller sets [108].

If any one of the two tests detects a significant difference within the group, we proceed with a post-hoc Nemenyi test [185] for pairwise comparisons. On the occasion that a group contains only a single algorithm, we instead use a Wilcoxon test [265].

All tests were conducted with a significance level of 0.05, which is standard for most machine learning experiments. We use the STAC site [221][4] for statistical tests and the Orange Python library [5] for generating critical difference graphics. The exact script used is available in our Github repository[6].

[4]Available at http://tec.citius.usc.es/stac/index.html. Accessed July 17 2021.

[5]Available at https://orange3.readthedocs.io/projects/orange-data-mining-library/en/latest/index.html, accessed July 22 2021.

[6]https://github.com/henryzord/thesis_experiments

We start the analysis with the group of base classifiers. All base classifiers were optimized with grid search. The *p*-value for the Friedman test is virtually 0. PUMA-star places first regarding average ranking – and according to Nemenyi – is not statistically similar to any other algorithm in this group. In other words, just performing a greedy hyper-parameter optimization on interpretable classifiers is statistically worse than using an evolutionary algorithm tailored for this task. The critical difference graph is shown in Figure 5.6), while individual unweighted AUCs are shown in Table 5.9.

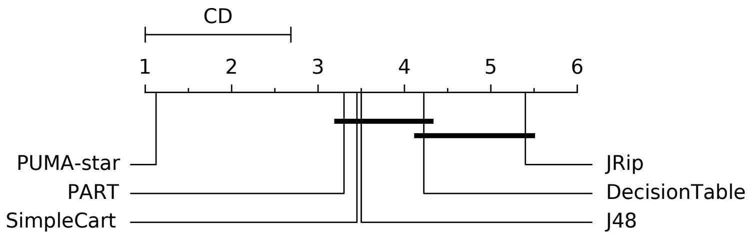


Figure 5.6: Critical difference graph for PUMA-star and optimized base classifiers in nested cross-validation experiment.

Table 5.9: Unweighted AUCs for base classifiers and PUMA-star. Best algorithm for each dataset is shown in bold.

Dataset	PUMA-star	J48	SimpleCart	JRip	PART	DecisionTable
banana	**0.9588**	0.9423	0.9559	0.8950	0.9453	0.8077
banknotes	**0.9969**	0.9876	0.9891	0.9804	0.9912	0.9825
car	**0.9973**	0.9724	0.9942	0.9533	0.9907	0.9742
contraceptive	**0.7341**	0.7040	0.7015	0.6373	0.7014	0.7126
diabetic	**0.7335**	0.7133	0.6810	0.6408	0.6955	0.7001
flare	**0.9255**	0.9109	0.9076	0.8629	0.9089	0.9193
krvskp	**0.9991**	0.9975	0.9979	0.9947	0.9960	0.9903
page-blocks	**0.9902**	0.9576	0.9477	0.9057	0.9553	0.9773
penbased	**0.9989**	0.9897	0.9889	0.9873	0.9883	0.9646
phoneme	**0.9468**	0.9061	0.9032	0.8282	0.9037	0.8735
ring	**0.9854**	0.9402	0.9334	0.9402	0.9614	0.8389
seismicbumps	**0.7739**	0.7248	0.7464	0.5252	0.7207	0.7601
splice	**0.9893**	0.9683	0.9811	0.9586	0.9756	0.9629
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	**0.9981**	0.9814	0.9808	0.9796	0.9819	0.9719
thyroid	**0.9997**	0.9977	0.9964	0.9948	0.9962	0.9982
titanic	**0.7615**	0.7100	0.7549	0.6755	0.7549	0.7389
turkiye	**0.8495**	0.8182	0.8402	0.7384	0.8345	0.8325
twonorm	**0.9828**	0.8895	0.8868	0.9216	0.9398	0.8538
waveform	**0.9518**	0.8940	0.8994	0.8842	0.9117	0.8924
Average rank	**1.125**	3.500	3.450	5.400	3.300	4.225

The second group comprises baseline ensembles: the first with default Weka hyper-parameters, and the other with individually-optimized base classifiers. The *p*-value of the Friedman Aligned ranks test is $2.28e-3$, which rejects the null hypothesis. The Nemenyi test corroborates that PUMA-star is not similar (statistically better) to (than) any of the algorithms in this group, although both baseline versions are similar between themselves. The critical difference graph is shown in Figure 5.7. This further corroborates the view that the task of selecting good sets of hyper-parameters and algorithms for classification with ensemble learning is not an easy task achievable with a greedy strategy. The unweighted AUCs for each algorithm are shown in Table 5.10.

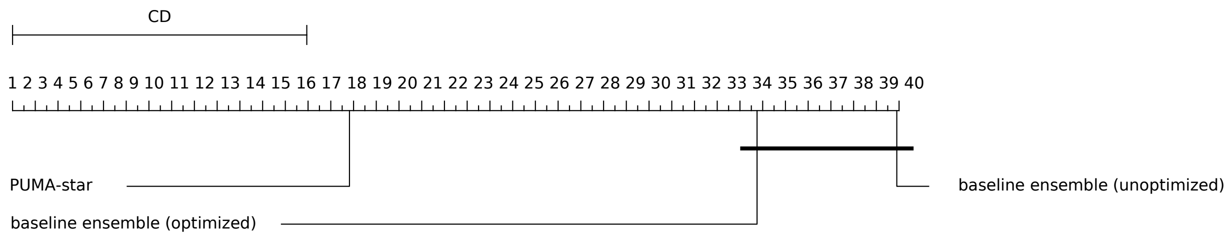


Figure 5.7: Critical difference graph for PUMA-star and both versions of baseline ensemble: one unoptimized (using base classifiers with their default hyper-parameters) and another that just ensembles the individually optimized base classifiers by means of a grid search. Note that ranks are aligned.

Table 5.10: Unweighted AUCs for PUMA-star, and two ensembles: one with its default hyper-parameters from Weka, and another with its members individually optimized by means of a grid search. Best algorithm for each dataset is shown in bold.

dataset	PUMA-star	baseline ensemble (unoptimized)	baseline ensemble (optimized)
banana	**0.9588**	0.9529	0.9582
banknotes	0.9969	**0.9980**	0.9967
car	**0.9973**	0.9959	0.9939
contraceptive	**0.7341**	0.7278	0.7280
diabetic	**0.7335**	0.7188	0.7302
flare	**0.9255**	0.9218	0.9236
krvskp	**0.9991**	0.9986	0.9989
pageblocks	0.9902	0.9885	**0.9910**
penbased	0.9989	**0.9992**	0.9988
phoneme	**0.9468**	0.9374	0.9349
ring	**0.9854**	0.9824	0.9816
seismicbumps	**0.7739**	0.6693	0.7667
splice	**0.9893**	0.9876	0.9871
steelfaults	**1.0000**	**1.0000**	**1.0000**
texture	0.9981	**0.9984**	0.9979
thyroid	**0.9997**	**0.9997**	0.9996
titanic	**0.7615**	0.7566	0.7588
turkiye	0.8495	0.8482	**0.8521**
twonorm	**0.9828**	0.9825	0.9812
waveform	**0.9518**	0.9508	0.9489
Average rank	**1.325**	2.275	2.400
Aligned rank	**17.875**	39.875	33.750

The third group is not a group *per se*, but instead a single algorithm, random search. There are no surprises in this experiment: PUMA-star performs much better than simply guessing hyper-parameter sets for base classifiers. The Wilcoxon test has a *p*-value of $1e - 4$ and predictive performance for both methods is shown in Table 5.11.

The next two groups comprise non-interpretable algorithms. The fourth group, of boosted algorithms, "distorts" the input data by applying a transformation to it. With the increase of iteration, harder-to-classify instances have an increased weight on the hypothesis decision. While this technique improves predictive performance, it harms interpretability. The best algorithm for this group, according

Table 5.11: Unweighted AUCs for PUMA-star and random search in the space of solutions. Best algorithm for each dataset is shown in bold.

dataset	PUMA-star	Random Search
banana	**0.9588**	0.8265
banknotes	**0.9969**	0.9853
car	**0.9973**	0.9459
contraceptive	**0.7341**	0.6398
diabetic	**0.7335**	0.6803
flare	**0.9255**	0.9234
krvskp	**0.9991**	0.9902
pageblocks	**0.9902**	0.9367
penbased	**0.9989**	0.9906
phoneme	**0.9468**	0.8221
ring	**0.9854**	0.9607
seismicbumps	**0.7739**	0.6092
splice	**0.9893**	0.9543
steelfaults	**1.0000**	**1.0000**
texture	**0.9981**	0.9772
thyroid	**0.9997**	0.9995
titanic	**0.7615**	0.7248
turkiye	**0.8495**	0.7342
twonorm	**0.9828**	0.9514
waveform	**0.9518**	0.8781

to average rank, is the boosted version of SimpleCart, followed by PUMA-star. The next best boosted algorithms are JRip, DecisionTable, PART, J48, and finally the original Adaboost. The *p*-value for the Friedman test is $2.86e-3$. Subsequent Nemenyi tests find statistical difference between Adaboost and the boosted version of SimpleCart, but no statistically-significant difference between all other pairs of algorithms. The critical difference graph is shown in Figure 5.8.

The fact that Adaboost places last is interesting: while the boosting framework works with any type of base classifier, this experiment demonstrates that using a stronger base classifier (any one of the five considered) is better than a single, slightly-better-than-random-guessing classifier (i.e., decision stumps). We also note that PUMA-star is found to be statistically equivalent to all other algorithms in this group, which is actually an advantage given that PUMA-star uses (i) less classifiers than the others, and (ii) all models generated by PUMA-star are interpretable. Individual unweighted AUC are shown in Table 5.12.

Finally, we compare PUMA-star with Random Forests. The *p*-value for the Wilcoxon test is 0.6008, which means that PUMA-star and Random Forests are statistically equivalent – at least for the group of datasets of this experiment. Random Forests presents an overall better performance than PUMA-star. We believe that this is – similarly to what happened to the boosted algorithms – a positive outcome. By using at most five base classifiers, all of them interpretable and without transformations on the input data, PUMA-star is able to achieve comparable performance to Random Forests with its hyper-parameters optimized by grid search, and using 1000 trees in its forest. The Unweighted AUCs for compared algorithms are shown in Table 5.13.

Table 5.12: Unweighted AUCs for PUMA-star, Adaboost, and boosted base classifiers. Best algorithm for each dataset is show in bold.

dataset	PUMA-star	Adaboost	J48 (boosted)	SimpleCart (boosted)	JRip (boosted)	PART (boosted)	Decision Table (boosted)
banana	0.9588	0.7842	0.9564	0.9385	**0.9598**	0.9589	0.8899
banknotes	0.9969	**0.9997**	0.9990	**0.9997**	0.9996	0.9993	0.9996
car	0.9973	0.8297	0.9967	**0.9997**	0.9948	0.9983	0.9937
contraceptive	**0.7341**	0.5441	0.6866	0.6887	0.6348	0.6702	0.6942
diabetic	0.7335	0.7537	0.7273	**0.7609**	0.7294	0.7231	0.7068
flare	**0.9255**	0.6707	0.8992	0.9039	0.8814	0.8997	0.8981
krvskp	**0.9991**	0.9953	0.9981	0.9975	0.9984	0.9972	0.9988
pageblocks	0.9902	0.8778	0.9857	**0.9925**	0.9843	0.9913	0.9517
penbased	0.9989	0.7085	0.9994	0.9993	**0.9996**	0.9993	0.9979
phoneme	0.9468	0.8836	0.9511	**0.9544**	0.9374	0.9365	0.9044
ring	0.9854	**0.9946**	0.9894	0.9915	0.9929	0.9900	0.9895
seismicbumps	**0.7739**	0.7486	0.5865	0.6542	0.7066	0.6099	0.7295
splice	0.9893	0.9681	0.9829	0.9884	**0.9897**	0.9846	0.9862
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9981	0.7353	0.9994	**0.9996**	0.9995	0.9990	0.9989
thyroid	**0.9997**	0.9895	0.9989	0.9980	0.9994	0.9978	0.9930
titanic	**0.7615**	0.7466	0.7571	0.7605	0.7586	0.7578	0.7559
turkiye	**0.8495**	0.7587	0.6625	0.7385	0.7316	0.6942	0.8333
twonorm	0.9828	**0.9965**	0.9910	0.9936	0.9944	0.9906	0.9946
waveform	0.9518	0.8795	0.9556	0.9599	0.9633	0.9556	**0.9651**
Average rank	3.300	5.275	4.475	**2.850**	3.325	4.400	4.375

Table 5.13: Unweighted AUCs for PUMA-star, and grid-search optimized Random Forest. Best algorithm for each dataset is show in bold.

dataset	PUMA-star	Random Forest
banana	0.9588	**0.9637**
banknotes	0.9969	**0.9999**
car	**0.9973**	0.9921
contraceptive	**0.7341**	0.7045
diabetic	0.7335	**0.7715**
flare	**0.9255**	0.909
krvskp	**0.9991**	0.9988
pageblocks	0.9902	**0.993**
penbased	0.9989	**0.9997**
phoneme	0.9468	**0.9671**
ring	0.9854	**0.994**
seismicbumps	**0.7739**	0.7543
splice	0.9893	**0.9957**
steelfaults	**1.0000**	**1.0000**
texture	0.9981	**0.9997**
thyroid	0.9997	**0.9999**
titanic	**0.7615**	0.755
turkiye	**0.8495**	0.8305
twonorm	0.9828	**0.9965**
waveform	0.9518	**0.9684**

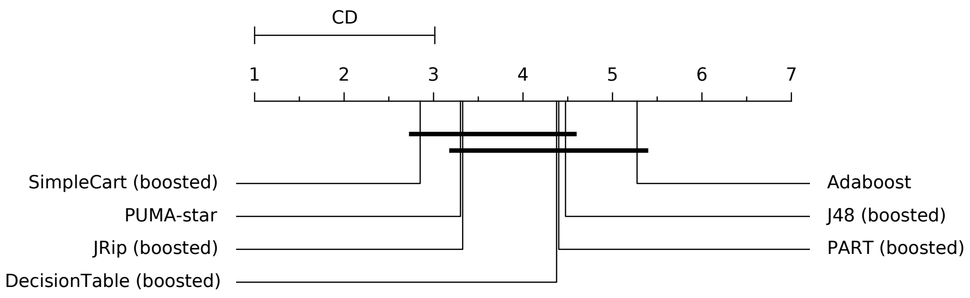


Figure 5.8: Critical difference graph for PUMA-star, AdaBoost, and boosted base classifiers, all optimized with grid search.

5.3.1 Fair Analysis on Interpretability

Let us take a deeper look on the results obtained so far. The boosted version of SimpleCart and Random Forests outperformed PUMA-star in terms of predictive performance considering the average ranking. The reader could then assume that, if the justification for using a limited amount of base classifiers in PUMA (only five) is to achieve an interpretable solution, then it could be possible to use a similar strategy by limiting the number of classifiers for both boosted SimpleCart and Random Forests, while hopefully achieving superior predictive performance than PUMA.

We recall the reader that boosted classifiers are not necessarily interpretable. The hypothesis drawn by ensemble members are custom-tailored to the instances that are harder to classify. Likewise, Random Forests intrinsically require randomizing the choice of attributes for each tree's internal node, which could hinder interpretability. Nonetheless, we carry an additional experiment in this section to compare PUMA-star to seven new baseline ensembles, namely: the five boosted base classifiers (J48, SimpleCart, PART, JRip, DecisionTable) with only five base classifiers in the ensemble; Adaboost (which uses decision stumps as default classifier) with five base classifiers; and Random Forests with five trees.

While PUMA-star and Random Forests are grid-search optimized and submitted to a nested cross-validation procedure, this is not performed on any of the six boosted classifiers, since the only hyper-parameter we had previously optimized for them is the number of ensemble members, and for this set of experiments, that is fixed on five base classifiers.

The unweighted AUCs are shown in Table 5.14. PUMA-star is now the best algorithm with an average rank of 2.225, followed by PART (3.275), JRip (3.825), J48 (3.9), SimpleCart (4.4), Random Forests (5.4), DecisionTable (5.475), and finally the original Adaboost (7.5). In other words, when given the same number of base classifiers, PUMA-star is able to outperform any other method, which indicates that the optimization performed by this algorithm is tailored to its needs, whereas boosting or the bagging procedure in Random Forests are inadequate for small-sized ensembles. Indeed, Adaboost is the worst algorithm from this group, with Random Forests being the third from last.

When this group of algorithms is submitted to a Friedman test, the *p*-value is *zero*. The critical difference graph can be viewed in Figure 5.9. Note that, since the ratio of algorithms-to-datasets is larger for this experiment than any other compared group from the last section, the Nemenyi post-hoc test is assessing some statistical equivalences that would otherwise not be detected if the number of algorithms were smaller (e.g., PUMA-star and boosted SimpleCart).

Table 5.14: Unweighted AUCs for PUMA-star, Adaboost, boosted base classifiers, and Random Forests. While PUMA-star may have from one to five base classifiers, the remaining methods all have exactly five classifiers in the ensemble. Best algorithm for each dataset is shown in bold.

dataset	PUMA-star	Random Forests (5)	Adaboost (5)	J48 (B, 5)	SimpleCart (B, 5)	JRip (B, 5)	PART (B, 5)	Decision Table (B, 5)
banana	**0.9588**	0.9389	0.6995	0.9562	0.9458	0.9579	0.9586	0.8846
banknotes	0.9969	0.9982	0.9631	0.9994	0.9986	0.9989	**0.9996**	0.9977
car	0.9973	0.9758	0.8297	0.9891	0.9985	0.9801	**0.9991**	0.9925
contraceptive	**0.7341**	0.6606	0.5441	0.6922	0.6691	0.6358	0.6695	0.6798
diabetic	**0.7335**	0.7073	0.6618	0.7107	0.6660	0.7020	0.7013	0.6938
flare	**0.9255**	0.8820	0.6707	0.8964	0.9049	0.8779	0.8979	0.8957
krvskp	0.9991	0.9974	0.9150	0.9990	0.9980	**0.9994**	0.9993	0.9987
page-blocks	**0.9902**	0.9600	0.8775	0.9791	0.9817	0.9704	0.9840	0.9405
penbased	0.9989	0.9984	0.7085	0.9992	0.9992	0.9992	**0.9993**	0.9892
phoneme	**0.9468**	0.9350	0.8095	0.9315	0.9336	0.9238	0.9167	0.8971
ring	0.9854	0.9759	0.7222	0.9832	0.9761	0.9822	**0.9890**	0.9615
seismicbumps	**0.7739**	0.6719	0.7547	0.6580	0.6487	0.7184	0.6426	0.7191
splice	**0.9893**	0.9802	0.9525	0.9852	0.9852	0.9876	0.9829	0.9757
steelfaults	**1.0000**	0.9999	0.8112	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9981	0.9965	0.7353	0.9983	0.9976	**0.9988**	0.9987	**0.9888**
thyroid	**0.9997**	0.9996	0.9823	**0.9997**	**0.9997**	0.9995	0.9991	0.9965
titanic	0.7615	0.7590	0.7396	0.7594	0.7594	0.7597	**0.7637**	0.7570
turkiye	**0.8495**	0.7790	0.7591	0.6550	0.7298	0.7315	0.6995	0.8298
twonorm	0.9828	0.9720	0.8587	0.9733	0.9733	0.9840	**0.9886**	0.9555
waveform	**0.9518**	0.9305	0.8578	0.9285	0.9231	0.9343	0.9310	0.9386
Average rank	**2.225**	5.400	7.500	3.900	4.400	3.825	3.275	5.475

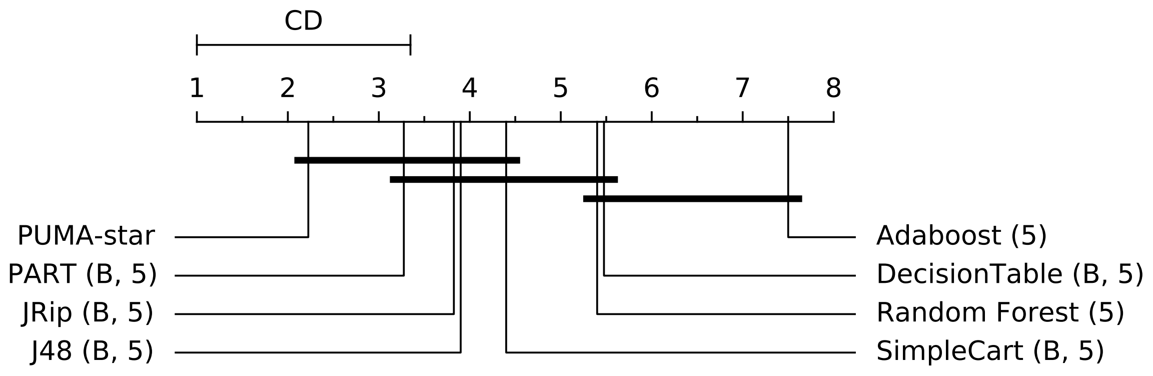


Figure 5.9: Critical difference graph for PUMA-star, Random Forests, AdaBoost, and boosted base classifiers. While PUMA-star can have between one and five classifiers, the remaining algorithms all have five base classifiers in their ensembles.

5.4 Discussion and Final Remarks

In this chapter we presented PUMA, a new evolutionary algorithm for optimizing hyper-parameters for a small number of intepretable classifiers, as well as the best aggregation policy for ensembling them. PUMA aims at maximizing predictive performance while also generating interpretable models by design. The proposed algorithm, along with Random Forests and some boosted algorithms, achieved the best predictive performance. However, among those classifiers, only PUMA generates an interpretable ensemble, composed of white-box models that do not perform any kind of transformation in the input space.

An experimental analysis of PUMA-star performance was conducted, comparing it with a thorough list of distinct baselines, namely base classifiers optimized by grid search, random search, boosted base classifiers, Adaboost, Random Forests, and baseline ensembles. PUMA-star was the best algorithm for all groups except two: boosted algorithms and Random Forests. For all groups that PUMA-star was the best algorithm, PUMA-star was also not statistically similar to any other algorithm in the groups. For the groups that PUMA-star was not the best algorithm, it was found that it was statistically similar to the best algorithm for that group. This result was expected, since boosted classifiers, and Random Forest transform the input space while also using many more base classifiers than PUMA-star was allowed to use, a constraint imposed solely by the pursuit of interpretability.

When confined to the constraints imposed on PUMA (that is, small-ish ensembles composed by interpretable classifiers), Adaboost and Random Forests are unable to match PUMA predictive performance. This implicates that PUMA is the best algorithm when interpretability is a must-have feature. However, we recall the reader that restricting the number of base classifiers from either Adaboost or Random Forests defeats the mechanisms from which these algorithms achieve good predictive performance, which relies on a sufficient number of base classifiers.

Future work involves designing a more advanced version of PUMA that takes into account dependencies among variables in the graphical model, as presented in the next chapter.

6. EDNEL: ESTIMATION OF DEPENDENCY NETWORKS FOR ENSEMBLE LEARNING

In Chapter 5 we introduced PUMA, an EDA for learning ensembles of interpretable classifiers. PUMA uses PBIL as its base EDA, which means it does not take into account probabilistic relationships between variables, only deterministic ones. In this sense, PUMA is limited: if there are correlations between variables, it will never detect (nor exploit) them. To overcome this limitation, we followed one of the topics in the future work section of PUMA: to take into account these relationships.

EDNEL – Estimation of Dependency Networks for Ensemble Learning – is an EDA much like PUMA, but with a few differences, one of them critical: its graphical model uses a Dependency Network [119] for capturing relationships among variables. Dependency Networks are different to Bayesian networks in the sense that they allow cycles and mutual relationships (that is, two variables can interact with each other at the same time).

In order to sample new values from the Dependency Network, we make use of a sampler algorithm, namely Gibbs sampling. It makes use of a starting set of values for each variable (which allows it to navigate a network that has cycles), and it updates variable values as soon as they are sampled. We discretize the two continuous-valued variables of PUMA (namely the confidence factor from J48 and PART), using a range of five evenly-distributed values for each variable in order to simplify the process of computing correlation between pairs of variables – a step introduced by the use of a Dependency Network.

Apart from graphical model modifications, we also experiment with a new aggregation policy. From base classifiers that we can extract unordered rules (namely J48, SimpleCart, and DecisionTable), we apply the CN2 rule learning algorithm [47, Section 3.2.1]. CN2 not only selects rules but also learns them; in our case, we already have the rules (they were learned by J48, SimpleCart, and DecisionTable) and we leave to CN2 to perform the second part of its process. Hence, EDNEL can choose between three aggregation policies: simple majority voting, weighted majority voting, or rule-extraction aggregation. While in the two first aggregation policies EDNEL can range from one to five base classifiers, in the third it ranges from one to at most three, since the rule-extraction aggregation will merge the rules from aforementioned three classifiers into a single rule-set classifier.

The rest of this chapter is organized as follows. Section 6.1 describes the proposed method. Section 6.2 describes the experiments performed on EDNEL, as well as our findings. Section 6.3 draws conclusions and point to future research directions.

6.1 Proposed Method

In this section we describe in more details the particularities of EDNEL. Even though EDNEL is the next iteration of PUMA, and the two methods are similar in several aspects, we explain everything concerning EDNEL even when being slightly repetitive regarding what was already explained in Section 5.1. An overview of EDNEL's pipeline is shown in Figure 6.1.

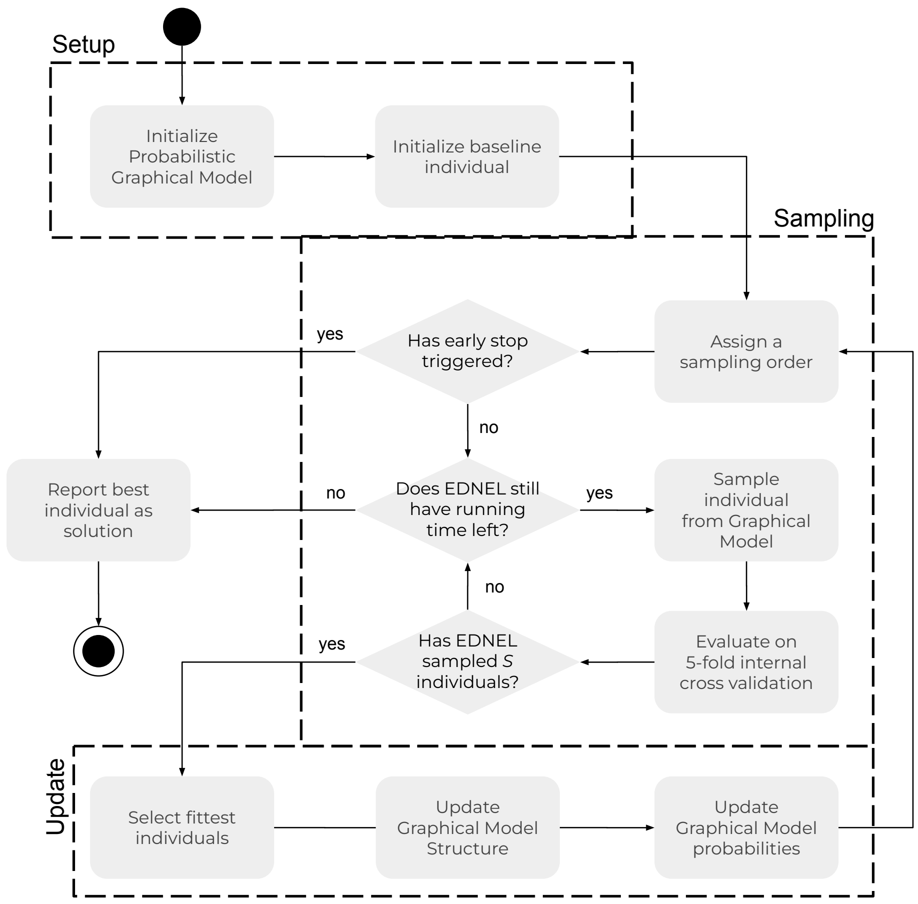

Setup
Initialize Probabilistic Graphical Model
Initialize baseline individual
Sampling
Assign a sampling order
Has early stop triggered?
yes
no
Report best individual as solution
no
Does EDNEL still have running time left?
yes
Sample individual from Graphical Model
no
Evaluate on 5-fold internal cross validation
yes
Has EDNEL sampled S individuals?
Update
Select fittest individuals
Update Graphical Model Structure
Update Graphical Model probabilities

Figure 6.1: Overview of the processing pipeline of EDNEL.

6.1.1 Individuals

Each individual is an ensemble that comprises set of base models and an aggregation policy. An ensemble can have at least one and at most five base models at the same type. Base models are generated by different base learners; that is, a base learner does not generate more than one model per individual. We choose base learners that can generate readily-interpretable models [98, 123, 209]. The recent literature on classification focuses mainly on producing classifiers with ever-increasing predictive performance, with little attention devoted to interpretability [103]. For instance, deep learning classifiers, which have received great attention lately due to obtaining high predictive accuracy in image tasks, are very difficult to interpret [103], with interested researchers shifting the focus from interpreting the models themselves to interpreting their predictions [160].

The five base learners employed are two decision-tree induction algorithms (C4.5 [213] and CART [25]); two rule induction algorithms (RIPPER [50] and PART [96]); and a Decision Table algorithm [143]. We use these algorithms' implementations in the well-known Weka Toolkit [113]. For the rest of this chapter, we will refer to them by their Weka names: J48 for C4.5, SimpleCart for CART, JRip for RIPPER, PART, and Decision Table.

At the implementation level, an individual is encoded as an array of values, where each position denotes a variable, and each value denotes the assigned value for that variable. Some variables may not have any value, since they are not used by an individual. Figure 6.2 depicts a segment from an individual's array, regarding some variables of its J48 classifier. J48 has three options for tree pruning: *reduced error pruning*, *confidence factor*, and *unpruned*. For this individual, *reduced error pruning* is used, hence there is no need to set hyper-parameters of the *confidence factor* strategy, which are then set to *null*.

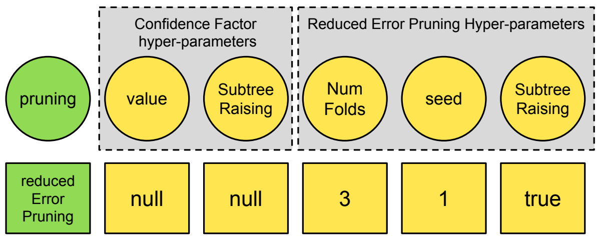


Figure 6.2: An example individual in EDNEL.

Aggregators

An aggregator is a method responsible for finding a consensus among votes from base models. We use three types of aggregators: majority voting, weighted majority voting, and CN2 rule-based aggregator. The probabilistic majority voting aggregator uses the fusion function described in [156, p. 150]:

$$h(X^{(j)}) = \arg\max_{c \in C} \left(\frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{6.1}$$

$$\rho_c^{(j)} = \sum_{i=1}^{B} P_c^{(i,j)} \tag{6.2}$$

where $\rho_c^{(j)}$ is the sum of the probabilities that the j-th instance belongs the c-th class over all B classifiers, and C is the number of classes. The weighted aggregator is similar to majority voting, except that individual probabilities from classifiers are weighted according to the fitness of each classifier:

$$\rho_c^{(j)} = \sum_{i=1}^{B} \psi^{(i)} P_c^{(i,j)} \tag{6.3}$$

$$h(X^{(j)}) = \arg\max_{c \in C} \left(\frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{6.4}$$

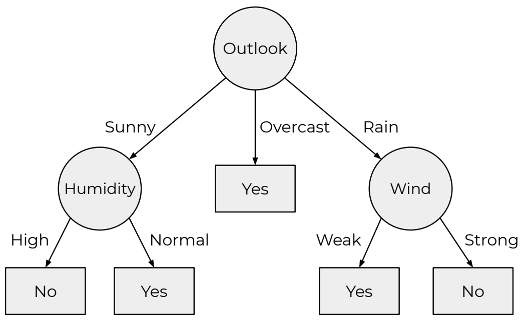
where $\psi^{(i)}$ is the fitness computed over the learning set of individual $S^{(i)}$.

Finally, the CN2 rule-based aggregator performs a post-process on the ensemble models, merging classifiers that generate unordered rules (i.e., J48, SimpleCart, and Decision Table) into a single (unordered) rule-based classifier, while keeping ordered classifiers (i.e., JRip, and PART) separated.

The process is described as follows. For learners that generated tree models (J48, SimpleCart), each path, from root to leaves, is converted to a rule as shown in Figure 6.3. For the model generated by the Decision Table learner, it is sufficient to treat each row in the table as an independent rule. Once extracted, we use a modification of the CN2 algorithm [47, Section 3.2.1] where instead of iteratively generating and selecting rules, we only select from the available rules from J48, SimpleCart, and Decision Table models. Given a dataset with predictive attributes $\mathbf{X}$ and class labels Y, and a set of rules R, it starts by creating an auxiliary array A of size N (where N is the number of instances in the training set), denoting whether a given instance is yet to be covered by a rule in rule set $\hat{R}$. At the beginning of the process, no instance is covered, hence all values in array A are set to 1. It then iteratively searches for a rule $\hat{r}$ with the best quality $\hat{q} = \mathit{precision} \times \mathit{recall}$ on the subset of yet-to-be-covered instances. The current best rule $\hat{r}$ is added to the set of rules $\hat{R}$, and has its voting weight computed based on the quality of that rule ($\mathit{precision} \times \mathit{recall}$) in the whole training set (not to be confused with the quality on yet-to-be-covered instances A). Correctly covered instances are set to 0 in A (i.e., removed), and the process is repeated until all instances are covered or no rule in R covers none of the remaining instances in A. Figure 6.4 shows the pseudocode for the CN2 rule-based aggregator.

We do not merge the models from the remaining learners in the ensemble, JRip, and PART, since these models produce ordered lists, which are dependent on one another to produce meaningful predictions (Section 2.3.1). However, the voting weight used when these models are making predictions is not the $\mathit{precision} \times \mathit{recall}$ of the entire classifier, but instead for the single rule that is activated. Figure 6.5 shows the algorithm used for assessing the quality of rules in an ordered list of rules.

During prediction, each rule that covers a new instance votes with its consequent class, with voting weight $\hat{q}$. All class votes are normalized. In an individual where all base models are present, at least three rules (one from PART, one from JRip, and at least one from J48 + DecisionTable + SimpleCart, when a rule from one of these models supersedes the two other rules that could cover that instance) and at most five rules (same as previous case but with no merging) will be activated during the prediction for any new instance.



(a) a

#1: (Outlook is Sunny) and (Humidity is High) then **Play is No**
#2: (Outlook is Sunny) and (Humidity is Normal) then **Play is Yes**
#3: (Outlook is Overcast) then **Play is Yes**
#4: (Outlook is Rain) and (Wind is Weak) then **Play is Yes**
#5: (Outlook is Rain) and (Wind is Strong) then **Play is No**

(b) b

Figure 6.3: (a) A tree generated by J48 for the “play tennis” dataset. (b) Corresponding rule set extracted by our post-processing algorithm used by the CN2 rule-based aggregator.

```
function COMBINE_RULES($R, \mathbf{X}, Y$)
    $A \leftarrow (1 | i = 1, \dots, N)$
    $\hat{A} \leftarrow (1 | i = 1, \dots, N)$
    $\hat{R} \leftarrow \emptyset$
    $\hat{Q} \leftarrow \emptyset$
    while $\exists a \in A | a = 1$ do
        $\hat{q} \leftarrow -\infty$
        $\hat{r} \leftarrow \nexists$
        for $\{r | r \in R \land r \notin \hat{R}\}$ do
            $q \leftarrow$ quality($r, \mathbf{X}, Y, A$)
            if $q > \hat{q}$ then
                $\hat{q} \leftarrow q$
                $\hat{r} \leftarrow r$
        if $\hat{r} = \nexists$ then return $\hat{R}, \hat{Q}$
        $\hat{R} \leftarrow \hat{R} \cup \hat{r}$
        $\hat{Q} \leftarrow \hat{Q} \cup$ quality($\hat{r}, \mathbf{X}, Y, \hat{A}$)
        $A \leftarrow A \otimes \neg$(covers($\hat{r}, \mathbf{X}$) $\otimes$ (consequent($\hat{r}$) $= Y$))
    return $\hat{R}, \hat{Q}$
function QUALITY($r, \mathbf{X}, Y, A$)
    TP, FP, TN, FN $\leftarrow 0$
    for ($\mathbf{X}^{(i)} | A_i = 1, i = 1, \dots, N$) do
        if covers($r, \mathbf{X}^{(i)}$) then
            if consequent($r$) $= Y^{(i)}$ then
                TP $\leftarrow$ TP + 1
            else
                FP $\leftarrow$ FP + 1
        else if consequent($r$) $\neq Y^{(i)}$ then
            TN $\leftarrow$ TN + 1
        else
            FN $\leftarrow$ FN + 1
    $q \leftarrow \frac{TP}{TP+FP} \times \frac{TP}{TP+FN}$
    return $q$
```

Figure 6.4: CN2 rule-based aggregator: algorithm used for combining rules in a rule-based classifier. Adapted from [47, Section 3.2.1].

```
1: function ORDERED_RULES_QUALITY($R$, X, $Y$)
2:     $A \leftarrow (1 | i = 1, \dots, N)$
3:     while $\{\exists a \in A | a = 1\}$ do
4:         for $(r = R^{(i)} | i = 1, \dots, |R|)$ do
5:             $\hat{Q} \leftarrow \hat{Q} \cup \text{quality}(r, \mathrm{X}, Y, A)$
6:             $A \leftarrow A \otimes \neg(\text{covers}(r, \mathrm{X}) \otimes (\text{consequent}(r) = Y))$
7:     return $R, \hat{Q}$
```

Figure 6.5: Algorithm used for assessing quality of rules from ordered rule-list classifiers.

6.1.2 Probabilistic Graphical Model

The graphical model of EDNEL is a Dependency Network [119]. Dependency networks are different from Bayesian Networks in two main aspects: (i) the former allows cycles, whereas the latter does not; and (ii) because cycles are allowed, recently-sampled values are used right away, influencing the probability distribution of children variables, as shown in Figure 6.6.

```
1: function GIBBS_SAMPLER($D^{(i)}$)
2:     $D_1^{(i+1)} \sim P(D_1 = d_1 | D_2 = d_2^{(i)}, \dots, D_{|D|} = d_{|D|}^{(i)})$
3:     $D_2^{(i+1)} \sim P(D_2 = d_2 | D_1 = d_1^{(i+1)}, \dots, D_{|D|} = d_{|D|}^{(i)})$
4:     $\vdots$
5:     $D_{|D|}^{(i+1)} \sim P(D_{|D|} = d_{|D|} | D_1 = d_1^{(i+1)}, \dots, D_{|D|-1} = d_{|D|-1}^{(i+1)})$
6:     return $D^{(i+1)}$
```

Figure 6.6: Pseudo-code for a single Gibbs sampling execution. Adapted from [275].

In Figure 6.6, $D^{(i)}$ is the current assignment of values for each variable in the GM (i.e., the current point in the solution space), $|D|$ is the number of variables, D_j the j-th variable, and d_j its respective value, while $D^{(i+1)}$ is its next value.

The reasons for considering a Dependency network over a Bayesian network are as follows. First, since Dependency networks allow cycles, it is capable of capturing relationships that a Bayesian Network cannot. Consider the relationship $A \rightarrow B$, i.e., A influences the outcome of B, but not the opposite. It might be the case that the value of B is as influential on A as A is on B; however a Bayesian Network could never capture this relationship. With a Dependency Network, not only $A \rightarrow B$ is allowed, as $A \leftrightarrow B$ and $A \leftarrow B$ as well.

The second reason to prefer Dependency Networks is that they allow a better understanding of correlation relationships between variables, whereas the rigor of a causal relationship (expressed by Bayesian Networks) is a concept difficult to grasp to non-experts [119]. Hence, EDNEL provides interpretability in two levels: first, by generating small, interpretable ensembles of classifiers, and second by generating a graphical model that allows interpreting correlation relationships between variables in the problem.

Variables

Each variable encodes either a classifier's hyper-parameter (e.g., *J48_minNumObj*, the minimum number of instances allowed in a leaf node of a J48 tree), or a derivation of it (e.g., *J48_pruning*:

this variable will choose between three hyper-parameters, each of which defining a pruning policy). There are also five variables to decide whether each of the five base classifiers is present in that individual's ensemble, and a variable for deciding the aggregation policy.

All 41 GM variables are discrete. For the two variables that encode continuous hyper-parameters, namely *J48_confidenceFactorValue* and *PART_confidenceFactorValue*, we generate five values evenly distributed within the range of valid values (according to Weka) for that hyper-parameter. This approach is opposed to the one adopted for PUMA (Section 5.1.3), which keeps the variables continuous and samples new values from a Gaussian distribution. This adaptation is made to simplify the source code for computing the Adjusted Mutual Information (AMI) between pairs of variables, a step necessary for detecting correlation (described in Section 6.1.6). In our opinion, it is much easier to implement the code to compute AMI between two discrete variables, or two continuous variables, then to compute it between discrete-continuous pairs. We left the AMI computation between discrete-continuous pairs for future work.

We use EDA's ability of biasing probabilities to increase by 10% the probability to sample values that are default in Weka. We assume that some effort was made to define those hyper-parameters as the default choice for the base learners. For all other values, we set uniform probabilities. For instance, for *J48_numFolds*, the default value 3 has probability 20%, while values in $[2, 10] - \{3\}$ have probability 10%. All of this happens execpt for variable *DecisionTable_evaluationMeasure*, where the value *auc* has a 50% probability of being sampled. We do this to increase the chances that a base learner is using the same metric used as fitness function, which for EDNEL (similarly to PUMA) is the Area Under the ROC curve.

Sampling procedure

There are two ways to perform direct inference from a Dependency Network [119]: the first one converts the Dependency Network to a Markov network, deriving a decomposable graphical model, and then applies an algorithm for probabilistic inference (e.g., the junction-tree algorithm); the second one uses Gibbs sampling [275], which is the method we adopt in EDNEL.

We modify Gibbs Sampling to better suit our needs. Variables can manifest in two ways: as they appear on the graphical model, being denoted V; and on the individual's array of values, which is a dictionary of values, referred to as D. Variables in the GM are joint-probability distribution tables, whereas D contains pairs $(D_j^{(i)}, d_j^{(i)})$, where $D_j^{(i)}$ is a given variable and $d_j^{(i)}$ the value assumed by said variable in the i-th iteration of the Gibbs Sampler. We can refer to V as to-be-sampled variables, as opposed to D already-sampled variables. Additionally, D is allowed to have empty values for some keys (which, at implementation level, is a null value), but not V.

We initialize D with what we call a baseline individual: an ensemble with five base models (one for each base learner), generated by using default Weka hyper-parameters for each one of the base learners, with majority voting as aggregation policy. As mentioned earlier, we assume that some effort was made to define those hyper-parameter as the default ones, hence rendering them attractive for at least starting the search process in EDNEL.

Variables may have two types of parents: deterministic and probabilistic.

Definition 6.1. A deterministic parent is a variable that influences on the outcome of its children, and that can not have its relationship modified by EDNEL. In other words, deterministic relationship are defined prior to the start of EDNEL evolutionary procedure, and are never modified throughout this procedure.

Definition 6.2. Like a deterministic parent, a probabilistic parent also influences the outcome of its children. However, the relationship is not permanent, and can be modified by EDNEL. For example, in one generation a variable A can be parent of another variable B, but in the next generation both variables may be independent.

The value of a variable depends on the values assumed by its parents. Deterministic parents do not have their values included in the joint-probability table of their children, but nonetheless influence in the sampling process. Let us assume two examples for the sake of clarity. In the first, variable $J48$ (which encodes the probability of including a J48 model in the ensemble) is the deterministic parent of variable $J48\_pruning$. If $J48 =$ false, then there is no need to choose a pruning policy for J48 with $J48\_pruning$; we may skip that variable and set $J48\_pruning = null$ in D. On the second example, $J48\_pruning$ is a probabilistic parent of $JRip$. If $J48\_pruning = null$ in D, then $JRip$ may be sampled via an unconditional distribution; otherwise it will be conditioned to $J48_pruning \in \{$reducedErrorPruning, confidenceFactor, unpruned$\}$.

Deterministic relationships were created through a manual study of hyper-parameters of the base classifiers in the Weka toolkit. In the first generation, there are only deterministic relationships; all variables are probabilistically independent – which is to say that all distributions are unconditional. The graphical model for the first generation is shown in Figure 6.7.

We assign a sampling order for Gibbs sampling [118] once per generation, which will be used for sampling that generation's population. The sampling order is inferred by analyzing the topology of the GM at the current generation. At first, all variables that do not have any parent whatsoever are added to the list. Then, all variables that have all their parents (both deterministic and probabilistic) already in the sampling order list, and so on and so forth until all variables are added. If, for a given iteration of this process, none of the remaining variables have all its parents in the sampling list (both deterministic and probabilistic), a compromise is made. We perform a lexicographic sorting on the remaining variables, where the first sorting criterion is the number of missing deterministic parents; then the number of missing probabilistic parents; and lastly the number of children (both deterministic and probabilistic) that the candidate variable has. This procedure is repeated until all variables are added to the sampling list.

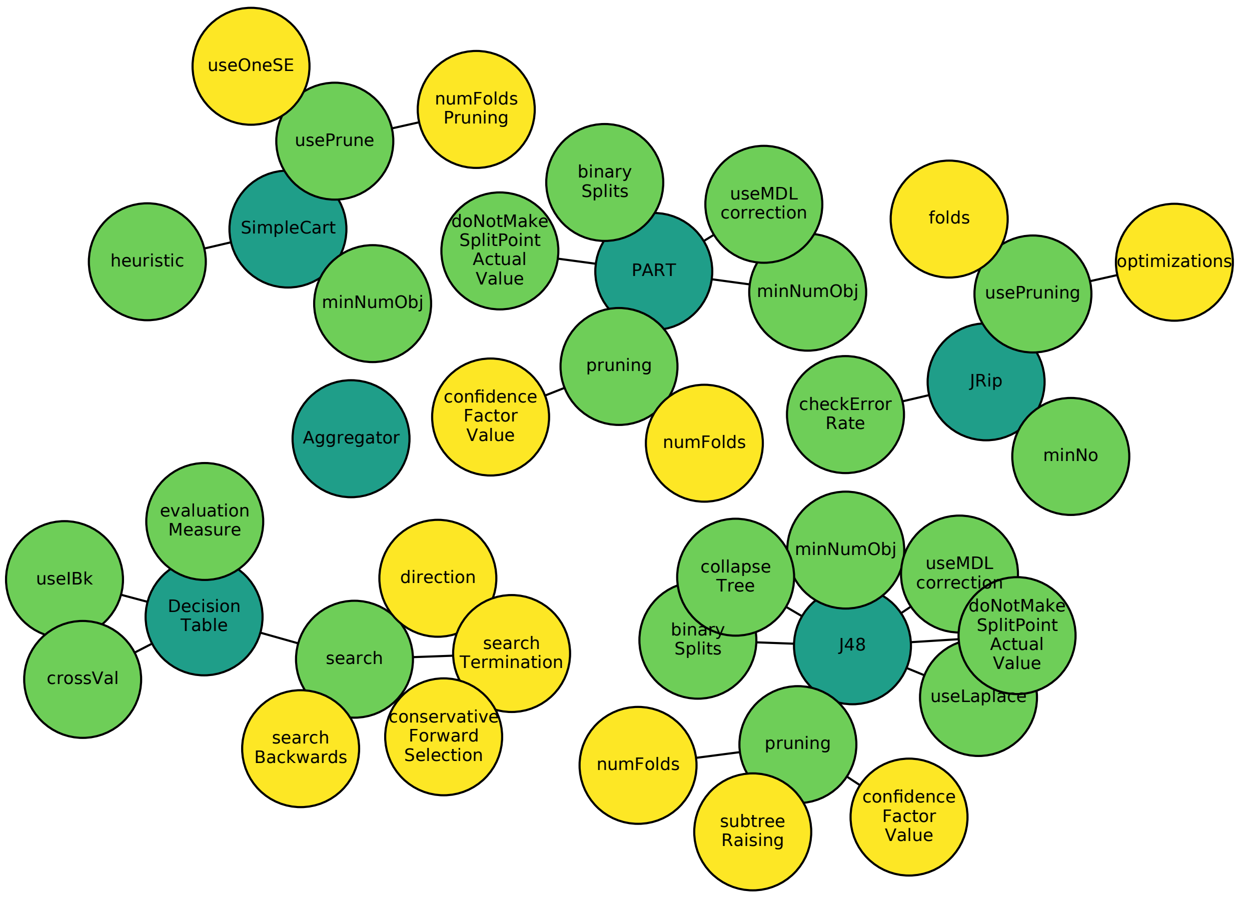


Figure 6.7: Initial structure of the graphical model used by EDNEL. All connections are deterministic (no probabilistic relationship is displayed). The structure is updated once per generation. The colors used denote which variables will be sampled: darker green-ish variables first; lighter yellow-ish variables last.

6.1.3 Sampling individuals

We choose to employ burn-in to help the Dependency Network achieve its stationary state. Burn-in is commonly used when one does not know a good starting point for sampling values in dependency networks, and thus wants to reduce the over-representation that uncommon states could have in a small sample size.

Definition 6.3. In a practical sense, burn-in refers to the process of sampling new individuals and immediately discarding them. Since Dependency Networks use newly sampled values as soon as they are available, performing burn-in helps a Dependency Network achieve its stationary state – in other words, a region of the solution space that does not contains outlier individuals.

There is a debate in the literature whether or not burn-in is effective in its purpose [27, p.19, c.1.11.4]. Since most of the related work do use burn-in, and in a sense burn-in does not *negatively* affects the outcome of our algorithm, we choose to use it in our experiments. The number of Gibbs Sampler executions to perform burn-in is a hyper-parameter.

We do not make any attempt to *thin* our Gibbs Sampler, since there are no clear indications that thinning produces more precise samples (in relation to the source distribution) than not thinning [168].

Definition 6.4. Thinning refers to the process of considering only samples that are divisible by a given number F, with F being a hyper-parameter. If e.g. $F = 2$, and 10 values are sampled from the Dependency Network, then only samples $\{2, 4, 6, 8, 10\}$ are deemed valid; odd number samples $\{1, 3, 5, 7, 9\}$ are discarded.

Moreover, it seems more reasonable to keep more (if not all) samples than to rely on only a few. However, it may be the case that some samples generate invalid individuals, due to (i) the nature of Gibbs sampling of using new values for variables as soon as they are available, as shown in Figure 6.6; and (ii) that some variables may have null values in D for that Gibbs sampling execution, if their deterministic parent was not sampled. We refer to this event as *passive thinning*, a by-product of our modified Gibbs Sampler.

The procedure of sampling new individuals is described as follows. We initialize D with the baseline individual, as described in the previous section. During burn-in, individuals (ensembles) are not trained, only values in D are changed. Since no ensemble is trained, no passive thinning occurs. The variable values are sampled based on the product of several bi-variate conditional distributions, one for each probabilistic parent, inspired by [107]:

$$D_i = \prod_{j \in \mathbf{Pa}^{(g)}_{\text{prob},i}} P(V_i | V_j) \tag{6.5}$$

where D_i is the sampled value for variable V_i, and $\mathbf{Pa}^{(g)}_{\text{prob},i}$ the probabilistic parents of V_i in the current generation.

After burn-in, any new sampled individual will be trained as soon as the Gibbs sampling finishes an iteration, and (provided that the individual is valid) added to the population of the current generation. The Gibbs sampling procedure finishes once it successfully sampled the requested number of individuals by the EDA.

There are two events worth noting that can occur during sampling: when a deterministic or a probabilistic parent for a child variable is non-defined (i.e., null) for a given execution. Recall that joint-probability tables in a GM do not allow null values, only D. In the former case, if a deterministic parent is non-defined, its children are also non-defined; on the other hand, if a probabilistic parent of a variable is non-defined, but all deterministic parents are defined, then the conditional joint-probability distribution $P(child|parent)$ is replaced by an unconditional distribution $P(child)$, computed based on the fittest population of the last generation; or the original distribution for the first generation of the EDA.

6.1.4 Fitness evaluation

Fitness assessment is done as soon as a new individual is sampled and trained. At the start of the evolutionary process, EDNEL receives a training set. This training set is split into two smaller sets: a learning set ($5/6$ of the training set), and a validation set ($1/6$). The learning set is used both for base classifiers to learn the data and to evaluate fitness, by means of an internal five-fold cross-validation. The learning set does not change throughout the evolutionary process, with all five folds remaining the same. This allows direct comparisons between individuals from different generations.

For fitness function, we use the area under the Receiving Operator Characteristic curve, or AUC [85]. AUC has a robust computation methodology when compared to, for example, the more popular accuracy score. For probabilistic classifiers, AUC does not impose a threshold for deciding whether an instance belongs to the positive or negative class; it is also more tolerant to changes in the distribution of classes, whereas accuracy is more sensitive [85].

AUC values are within $[0, 1]$, with the value 0.5 representing the predictive performance of random predictions in the case of binary-class problems. For EDNEL, regardless of the number of classes in the dataset, we calculate one AUC for each class, and then average the AUC among all classes. Hence, the fitness of an individual is actually a mean of means: first, the mean AUC among all classes for a given fold; then, the mean AUC among all five internal folds.

6.1.5 Population selection and Elitism

We use hyper-parameter $|\Phi|$ for selecting the fittest individuals from a generation's population ($|\Phi| < |S|$) for updating both GM's structure and probabilities. We carry on the fittest individual from the current generation to the next (i.e., elitism), while the rest of the population is resampled with the updated GM.

6.1.6 Updating the GM's structure

The first update performed is in the GM's structure. This procedure detects correlation between variables and link then within the Dependency Network. Thus, probabilistic relationships are updated once every G_{update} generations (with G_{update} as a hyper-parameter) for all variables. Deterministic relationships are built before the EDA starts its evolutionary process and are never changed. We do not allow a child variable to add a deterministic parent to its set of probabilistic parents. If $G_{update} = 1$, the structure of the GM will be updated every generation using the current fittest population for detecting correlation between variables. If $G_{update} > 1$, then the last G_{update} fittest subpopulations will be used.

For detecting correlation, we use the Adjusted Mutual Information (AMI), which detects both linear and non-linear correlations between variables [226]. Mutual information "is a symmetric measure that quantifies the mutual dependence between two random variables. It measures how much knowing one of these variables reduces our uncertainty about the other" [187].

The reason for deciding for the adjusted version of mutual information is twofold. First, variables in the GM have varying number of values (e.g., *J48_pruning* with 3 values; *J48_minNumObj* with 9; etc.). If not adjusted, variables with large number of values would be more likely to be linked (in the GM) with other variables with large number of values. Secondly, mutual information is an unbounded metric, with a lower bound at zero but no upper bound. By adjusting it, we are closer to limit its values to the range $[0, 1]$, where 0 is no correlation and 1 perfect correlation between variables. Computing AMI between discrete variable pairs is given by

$$AMI(V_i, V_j) = \frac{MI(V_i, V_j) - E(MI(V_i, V_j))}{\max(H(V_i), H(V_j)) - E(MI(V_i, V_j))} \tag{6.6}$$

where $MI(V_i, V_j)$ is the (unadjusted) mutual information, $E(MI(V_i, V_j))$ its expected value, and $H(V_i)$ is the entropy for V_i.

AMI can yield negative values when the actual mutual information is smaller than its expected value. Since a negative AMI is as bad as a zero-valued AMI for linking variables, we clip negative values to zero.

When computing these metrics, we only consider pairs of valid values, that is, for each individual in the fittest population, we collect values for a pair of variables where $V_i \neq null$ and $V_j \neq null$. We call individuals that have valid pairs of values for (V_i, V_j) *relevant elite* Φ^+. Thus, it is possible that the number of pairs for (V_i, V_j) is larger than, say, (V_i, V_k): another reason why AMI is preferred over Mutual Information, since coincidences are more likely to occur in small samples.

Having a small correlation does not mean that the EDA will link variables in the GM; the correlation must be strong enough to justify the linking. We use an heuristic proposed in [106] to decide which variables will parent other variables. The heuristic behaves as follows. Given variable V_i, a searching algorithm is employed to find another variable V_j to be a probabilistic parent of V_i. If the correlation (in this case, AMI) between V_i, V_j is greater than the average correlation between V_i and its probabilistic parents, and greater than $heuristic\_tolerance$ (an hyper-parameter), we add V_j to the parent set $\mathbf{Pa}_{\text{prob},i}$, removing it from candidate set cand_i; otherwise, we terminate the searching process for variable V_i. We now proceed to find another variable that is strongly correlated to V_i, but not $\mathbf{Pa}_{\text{prob},i}$:

$$AMI(V_i, V_j) - \frac{\sum_{V_k \in \mathbf{Pa}_{\text{prob},i}} AMI(V_k, V_j)}{|\mathbf{Pa}_{\text{prob},i}| + 1} \tag{6.7}$$

While searching for candidate parents, if a variable happens to have a heuristic equal or less than $heuristic\_tolerance$, we prematurely discard it from the candidate set. The reasoning behind the heuristic is that a new candidate parent must be more correlated to its candidate child than it is correlated to the rest of the parent set. In our experiments, we use $heuristic\_tolerance = 0.1$. The algorithm we use for searching parents is described in Figure 6.8, where τ is the maximum number of probabilistic parents a variable is allowed to have at any given time. This algorithm returns a a set of parents for each variable in the GM. Note that, if V_i is the parent of V_j and V_j is the parent of V_k, V_k is allowed to close the cycle – i.e., be the probabilistic parent of V_i even though it is in a deterministic chain of relationships.

Note that a variable cannot be a parent of its deterministic parents $\mathbf{Pa}_{\text{det},i}$, nor can it be a probabilistic parent of its deterministic children $\mathbf{Ch}_{\text{det},i}$. Once all probabilistic parents are identified, the next task is to update the structure of the probability tables. If probabilistic parents are the same from the previous generation, the structure of table is kept, otherwise all entries are removed and replaced by the combination of non-null values for each probabilistic parent and each value of child variable. For example, if child variable J48_pruning (3 values) is linked to parent variable J48_minNumObj (9 values), then J48_pruning's table will have $3 \times 8 = 24$ entries. The next task at hand is to update probabilities for each entry.

```
function SEARCH_PARENTS($V$, $\Phi^+$)
    for ($i = 1, \ldots, |V|$) do
        $\mathbf{Pa}_{\text{mut},i} \leftarrow \emptyset$
        $\text{cand}_i \leftarrow V - \{V_i\} \cup \mathbf{Pa}_{\text{det},i} \cup \mathbf{Ch}_{\text{det},i}$
        $\text{heuristic}_{\text{max}} \leftarrow 0$
        $\text{cand}_{\text{best}} \leftarrow \nexists$
        while $\text{cand}_i \neq \emptyset$ and $|\mathbf{Pa}_{\text{prob},i}| < \tau$ do
            for $V_j \in \text{cand}_i$ do
                $\text{heuristic}_j \leftarrow$ use $\Phi^+$ on Equation 6.7
                if $\text{heuristic}_j \leq heuristic\_tolerance$ then
                    $\text{cand}_i \leftarrow \text{cand}_i - \{V_j\}$
                else if $\text{heuristic}_j > \text{heuristic}_{\text{max}}$ then
                    $\text{heuristic}_{\text{max}} \leftarrow \text{heuristic}_j$
                    $\text{cand}_{\text{best}} \leftarrow V_j$
            if $\text{cand}_{\text{best}} \neq \nexists$ then
                $\mathbf{Pa}_{\text{prob},i} \leftarrow \mathbf{Pa}_{\text{prob},i} \cup \{\text{cand}_{\text{best}}\}$
            else
                $\text{cand}_i \leftarrow \emptyset$
    return Pa
```

Figure 6.8: Algorithm used for detecting parents of variables. Adapted from [106].

6.1.7 Updating the GM's probabilities

We apply the same procedure used to update the structure of the GM using only individuals with valid values (i.e., defined) for each pair of variables. We refer to these individuals as *relevant elite* Φ^+. We update the GM's probabilities every generation, as opposed to the structure update, which is learned every G_{update} generations. We use a learning rate to update probabilities, with small modifications to suit our specificities.

The update of probabilities is done in three steps. The first step is to collect all valid (parent, child) pairs. From this valid set, bivariate conditional distributions $P(child|parent)$ are computed. The second step is to update these bivariate statistics with a learning rate, much the same procedure used by PBIL-iUMDA [272, 277]. Using a learning rate requires that all probabilistic parents be kept the same between two generations, which might not be the case; if so, the unconditional probability distribution of the child variable in the current fittest population is used instead. The pseudocode for the second step of updating bivariate conditional distributions is shown in Equation 6.8:

$$p^{(g+1)}(V_i = v | V_j \in \mathbf{Pa}^{(g+1)}_{\text{prob},i}) = \begin{cases} (1-\alpha) \times p^{(g)}(V_i = v | V_j) + \\ \qquad \alpha \times p^{(g)}_{\Phi^+}(V_i = v | V_j) & \text{if } V_j \in \mathbf{Pa}^{(g)}_{\text{prob},i} \\ p^{(g)}_{\Phi^+}(V_i = v) & \text{otherwise} \end{cases} \tag{6.8}$$

where V_i is the i-th variable, V_j a probabilistic parent of V_i, $\mathbf{Pa}^{(g)}_{\text{prob},i}$, $\mathbf{Pa}^{(g+1)}_{\text{prob},i}$ are respectively the set of probabilistic parents of V_i in the current and next generations, α the learning rate, $p^{(g)}(V_i)$ the probability from the GM in the current generation, and $P^{(g)}_{\Phi^+}(V_i)$ the probability from the relevant elite individuals of the current generation. Equation 6.8 is iterated over all probabilistic parents of V_i, and all pairs of values v_i, v_j. Note that if parents change from one generation to another, only probabilities observed in the relevant elite are used (i.e., previous probabilities are discarded).

Once the bivariate statistics are updated, the product of all bivariate statistics can be computed as shown in Equation 6.5. All distributions – the many supportive bivariate distributions and the final product distribution – are normalized.

6.1.8 Early Stop, termination, and validation set

As discussed in Section 6.1.4, EDNEL splits the training set into two subsets: a learning set (which comprises $5/6$ of the training data), and a validation set ($1/6$). EDNEL individuals do not have access to the validation set; it is used solely by the early-stop algorithm to decide whether it is beneficial to prematurely stop the evolutionary process.

The early-stop policy is described next. Let ϕ_{learn} be the fitness of an individual $S^{(i)}$ in the learning set (given by an internal five-fold cross-validation procedure), and ϕ_{val} be the predictive performance of said individual when trained with the whole learning set whose inference is performed on the validation set. We take note of the best individual found so far given by ϕ_{learn} and evaluate its performance on the validation set recording ϕ_{val}. For the next generation, we take note if the fitness ϕ_{val} of the next generation's best individual (by ϕ_{learn}) decreases. If so we start a countdown of $counter$ generations (a hyper-parameter). This is the number of generations that EDNEL has to recover a fitness ϕ_{val} as good as the best found so far. If no improvement is detected, we prematurely terminate the evolutionary process, returning the individual with the best ϕ_{val}; otherwise, we annotate the new best ϕ_{val} and proceed until no generations remain to be executed or the countdown reaches $counter$ generations. In our experiments, $counter$ is also a hyper-parameter.

6.2 Experiments

We divide the experiments of this section in two parts: a nested cross-validation procedure, described in Section 6.2.1, and a holdout procedure, described in Section 6.2.3. The reason for dividing into two parts is also twofold. First, we informally detected that EDNEL does not present a set of hyper-parameters that work well for all cases, which means it must be optimized for different types of datasets in order to provide the best predictive performance. Thus, a nested cross-validation procedure coupled with a grid search for EDNEL hyper-parameters is best suited for evaluating EDNEL. Second, we could not – even with the best of our efforts – adapt the code of AUTOCVE [161] (the evolutionary algorithm chosen as baseline) for this framework. Hence, we use the authors' results in their last paper [162], which performs a holdout procedure, to compare AUTOCVE to EDNEL.

For the interested reader, we make available the source which we developed that tries to adapt AUTOCVE code to a nested cross-validation coupled with a grid search for AUTOCVE hyper-parameters. For such, we make available two Github repositories[1][2].

6.2.1 Nested Cross-validation experimental setup

We describe in this section the nested cross-validation procedure performed to EDNEL and baseline algorithms. The advantage of using a nested cross-validation procedure is that it allows an algorithm to optimize a set of hyper-parameters for each one of the external folds of the nested cross-validation, by means of an optimization strategy. We choose grid search for simplicity, aware that this is not the state-of-the-art for hyper-parameter optimization.

The nested cross-validation procedure is described as follows. The whole dataset is divided into ten folds, which we will refer to as *external folds*. Nine folds are used for training and one for testing. The nine external folds are split again into five folds, which we will refer to as *internal folds*. Different combinations of hyper-parameters for each classifier are tested via these five internal folds. The predictions for each internal test fold are stored in an array. At the end of the internal five-fold cross validation, the unweighted Area Under the ROC curve is computed, and the best combination of hyper-parameters is chosen. The best combination is then trained with the nine external folds, and its predictions are provided over the external test fold. This procedure is described in Figure 6.9.

[1]https://github.com/henryzord/AUTOCVE
[2]https://github.com/henryzord/AUTOCVE-star

```
1: function NESTED_CROSS_VALIDATION($H$, $\mathbf{X}$, $Y$)
2:     for $i = 1, 2, \ldots, 10$ do
3:         $\mathbf{X}^{(i)}_{\text{train}} = 9/10$ of $\mathbf{X}$
4:         $Y^{(i)}_{\text{train}} = 9/10$ of $Y$
5:         for each hyper-parameter configuration from set of all hyper-parameters configurations do
6:             for $j = 1, 2, \ldots, 5$ do
7:                 $\mathbf{X}^{(i,j)}_{\text{train}} = 4/5$ of $\mathbf{X}^{(i)}_{\text{train}}$
8:                 $Y^{(i,j)}_{\text{train}} = 4/5$ of $Y^{(i)}_{\text{train}}$
9:                 train H with hyper-parameter configuration on $\mathbf{X}^{(i,j)}_{\text{train}}$, $Y^{(i,j)}_{\text{train}}$
10:                test H on $\mathbf{X}^{(i,j)}_{\text{test}}$, annotate prediction scores
11:            compute unweighted Area Under ROC curve with annotated prediction scores
12:        choose hyper-parameter configuration with best unweighted AUC
13:        train H with best hyper-parameter configuration on $\mathbf{X}^{(i)}_{\text{test}}$, annotate prediction scores of $Y^{(i)}_{\text{test}}$
14:    compute unweighted AUC with annotated prediction scores
15:    return unweighted AUC
```

Figure 6.9: Nested cross-validation procedure. All subsets follow the same class distribution as the original dataset (in other words, both the external and internal cross-validations are stratified).

Datasets

We select 20 datasets from both KEEL [3] [3] and UCI Machine Learning repository[4] [165], which are shown in Table 6.1. We select these datasets to cover a wide range or characteristics. The datasets have from 1066 to 10992 instances, from 3 to 61 attributes, and from 2 to 13 classes.

Table 6.1: Datasets used in the experiments.

name	instances	attributes	classes
banana	5300	3	2
banknotes	1372	5	2
car	1728	7	4
contraceptive	1473	10	3
diabetic	1151	20	2
flare	1066	12	6
krvskp	3196	37	2
page_blocks	5472	11	5
penbased	10992	17	10
phoneme	5404	6	2
ring	7400	21	2
seismicbumps	2584	19	2
splice	3190	61	3
steelfaults	1941	34	2
texture	5500	41	11
thyroid	7200	22	3
titanic	2201	4	2
turkiye	5820	33	13
twonorm	7400	21	2
waveform	5000	41	3

Modifications to EDNEL

We had to adapt EDNEL to perform under this experimental setup. We refer to the original EDNEL code, as describe in the previous sections, as *production code*, whereas the modifications explained here are referred to as *evaluation code*.

[3]Available at https://sci2s.ugr.es/keel/datasets.php
[4]Available at https://archive.ics.uci.edu/ml/datasets

Both source-codes behave exactly the same, except for the way that the *evaluation code* computes the fitness function. Recall that, in *production code*, EDNEL receives a training set that is split into two subsets: learning set ($\frac{5}{6}$ of the training set) and validation set ($\frac{1}{6}$). An internal five-fold cross-validation is performed with the learning set to evaluate fitness of individuals (ensembles), whereas the validation set is used to check whether EDNEL is stuck in local optima. In the *evaluation code*, on the other hand, the training set is also split into learning and validation sets, but a holdout procedure is done instead using the learning set to evolve ensembles and the validation set to compute the fitness of individuals. We do this to avoid that datasets be split into many smaller subsets; remember that the *evaluation code* is used under a nested cross-validation procedure. If we were to use the *production code*, the set used to evolve ensembles would be $\frac{9}{10}\frac{4}{5}\frac{5}{6}\frac{4}{5} = 0.48$ of the entire dataset, which increases the chances that our algorithm overfits the data. By using holdout with 80% of the data for learning and 20% for validation, the data used to learn ensembles is improved to $\frac{9}{10}\frac{4}{5}\frac{8}{10} = 0.576$.

Baseline Algorithms and Hyper-Parameter Optimization

The comparison to EDNEL is separated into groups. All groups are summarized in Table 6.7, at the end of this section. The first group is comprised of evolutionary algorithms, containing two versions of EDNEL (one with and another without the CN2 rule extraction algorithm, described in Section 6.1.1), and PUMA [30]. PUMA was described in greater details in Chapter 5, and is similar to EDNEL, with the key difference that it does not capture relationships between variables. We modify PUMA to allow performing an internal holdout procedure, splitting the training set at a $80/20$ proportion for learning and validation sets, respectively (the same modification done to EDNEL and described in Section 6.2.1); and to allow two timeout hyper-parameters, both for general evolution and individual evolution. We denominate this modified PUMA version as PUMA-star. The hyper-parameters optimized by grid search, for both PUMA-star and EDNEL versions, are shown in Table 6.2.

For both EDNEL versions and PUMA-star, we do not optimize their population size because it is analogous to the Random Forests' number of trees: if we could hypothetically have an infinitely large population, then it would be guaranteed to have at least one individual with the best achievable performance, given each algorithm constraints (e.g., aggregation policies, algorithm behavior, etc). We use 50 individuals in all evolutionary algorithms.

In a similar sense, we do not optimize the number of generations nor the time available for each algorithm evolving their population, leaving these hyper-parameters at 100 generations and one hour, respectively. All algorithms also have a time limit for evolving individuals, which is implemented in the same way: if an individual exceeds 60 seconds to sample and train its ensemble members, then it is discarded and a new individual must be sampled.

The second group comprises base classifiers used by all evolutionary algorithms, namely J48, SimpleCart, PART, JRip, and Decision Table. If at least one of these algorithms outperforms either one of the evolutionary algorithms, then there is no justification to employ the EAs for evolving ensembles of interpretable classifiers: it would be easier to just use the single best base classifier. These classifiers

Table 6.2: Hyper-parameters used by both EDNEL versions and PUMA-star in the nested cross-validation procedure. Multiple values indicate that the hyper-parameter was optimized; single values were constant among all grid search iterations.

Algorithm	Hyper-parameter	Values
EDNEL	Learning rate	{0.13, 0.26, 0.52}
EDNEL	Selection share	0.5
EDNEL	Population size	50
EDNEL	Generations	100
EDNEL	Evolution timeout	one hour
EDNEL	Individual timeout	one minute
EDNEL	Burn-in	100
EDNEL	thinning factor	0
EDNEL	Early stop generations	{10, 20}
EDNEL	Maximum probabilistic parents per variable (same value for all variables)	{0, 1}
EDNEL	Delay structure learning	5
EDNEL	individual to report	best overall
PUMA-star	Learning rate	{0.13, 0.26, 0.52}
PUMA-star	Selection share	{0.3, 0.5}
PUMA-star	Population size	50
PUMA-star	Generations	100
PUMA-star	Evolution timeout	one hour
PUMA-star	Individual timeout	one minute
PUMA-star	individual to report	{best overall, best from last generation}

also have their hyper-parameters optimized by means of grid search, and no modifications were made to their source code. The list of optimized hyper-parameters is described in Table 6.3.

Table 6.3: List of base classifiers and hyper-parameters optimized by grid search. Hyper-parameters not shown in this table were kept constant, and the values used were the default ones from Weka.

Algorithm	Hyper-parameter	Values
J48	Minimum number of examples in leaf nodes	{2, 4, 6, 8}
J48	pruning policy	{unpruned, reduced error, confidence factor}
SimpleCart	Minimum number of examples in leaf nodes	{2, 5, 8}
SimpleCart	Apply pruning	{true, false}
SimpleCart	Use heuristic for binary splits of categorical attributes	{true, false}
JRip	Minimum number of examples to be covered by a rule	{2, 4, 6, 8}
JRip	use pruning	{true, false}
PART	Minimum number of examples to be covered by a rule	{2, 4, 6, 8}
PART	pruning policy	{unpruned, reduced error, confidence factor}
Decision Table	internal evaluation metric	{AUC, accuracy, Root Mean Squared Error, Mean Absolute Error}
Decision Table	Search Policy	{Greedy Stepwise, Best First}
Decision Table	Default rule assigns majority class or uses IBK	{majority class, IBK}

We call the algorithms in the third group as baseline ensembles. The first algorithm is an unoptimized baseline ensemble, and consists of the five base classifiers from EDNEL (J48, CART, JRip, PART, and Decision Table) with their default hyper-parameter configuration, and a simple majority voting scheme as aggregation policy. The reasoning for doing so is to check whether there is a difference between simply ensembling these five base classifiers and optimizing their hyper-parameter configurations with an evolutionary algorithm.

The second algorithm in this group is an optimized baseline ensemble. It has the same five base classifiers and also uses majority voting, but each base classifier had its hyper-parameters individually optimized by grid search. We compare the EAs to this optimized baseline ensemble in order to verify whether it is worth to apply an EA to perform global optimization or it is simply better to individually optimize base classifiers and ensemble them with a simple majority voting policy.

The fourth group, which contains a single algorithm, is the random classifier: a random search procedure with the same budget of the EAs. In the same sense, we are evaluating whether it is really justified to apply an EA or simply randomly guessing solutions over enough time solves the problem. Since we execute the EAs with a fixed population of 50 individuals for 100 generations, we use the source code of EDNEL to perform a random search in the space of solutions with a budget of $5,000$ evaluations, with no compromise to sample distinct solutions. This is achieved by running EDNEL with $5,000$ individuals and 1 generation. Note that the initial probabilities are the same as described in Section 6.1.2 – that is, they are slightly biased towards the default Weka values. Since we are re-using the source code of EDNEL, we have to set some hyper-parameters. The hyper-parameters used are listed in Table 6.4.

Table 6.4: Hyper-parameters used on EDNEL to perform random search in the space of solutions. Hyper-parameters marked with an asterisk (*) have no effect on the outcome of the algorithm run, but are listed here since EDNEL requires their values to be set.

Algorithm	Hyper-parameter	Values
Random Search	Learning rate	0.1*
Random Search	Selection share	0.1*
Random Search	Population size	5000
Random Search	Generations	1
Random Search	Evolution timeout	no limit
Random Search	Individual timeout	no limit
Random Search	Burn-in	100
Random Search	thinning factor	0
Random Search	Early stop generations	1*
Random Search	Maximum probabilistic parents per variable (same value for all variables)	0*
Random Search	Delay Structure Learning in N generations	5*
Random Search	evaluation procedure	holdout
Random Search	individual to report	best overall*

The fifth group has six ensembles, each one comprising the boosted versions (using Adaboost [100]) of each of the five aforementioned base classifiers, plus Adaboost itself with its default base classifier, decision stumps. We use the Weka toolkit version of Adaboost, which – translating to the original paper [100] – is the M1 version. We optimize these six ensembles with a grid search. It is important to note that none of the algorithms of this group is interpretable, nor their ensemble members, since the boosting process produces ensemble members with hypothesis that only make sense for that particular distribution of instances. We explain boosting in greater detail in Section 2.1.1.

In Weka, only three hyper-parameters are available for customization: weight threshold ($[0, 1]$) – how much of the training data should be used for training the classifier at each iteration;

number of iterations (i.e., number of classifiers in the ensemble), and type of classifier. We decide to leave the weight threshold at 1 (i.e., use all training data), and since the type of classifier is not really an optimization option (we will be using six different types of base classifiers), we optimize only the number of iterations, or conversely the number of classifiers in the final ensemble. The choice of ranges are defined in Table 6.5, and we base the choice of values on the work of Rijn and Hutter [258].

Table 6.5: Hyper-parameters of Adaboost optimized with grid search. Note that we consider all entries in this table as distinct algorithms.

Algorithm	Hyper-parameter	Values
Adaboost (decision stumps)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (J48)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (SimpleCart)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (JRip)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (PART)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}
Adaboost (DecisionTable)	number of iterations	{50, 91, 132, 173, 214, 255, 295, 336, 377, 418, 459, 500}

Finally, the sixth and last group has two classifiers, Random Forests [24] and Random Forests-star. Random Forests is a well-known ensemble algorithm, and is in general among the best classification methods regarding predictive performance [88]. We explain Random Forests in greater detail in Section 2.1.4.

Random Forests is also optimized by means of grid search. Based on [212], we consider the following hyper-parameters and range of values: percentage of the training set that will be used by each tree in the forest in $\{0.9, 1.0\}$; percentage of attributes to randomly sample for considering in a given internal node of a given tree, in relation to the total number of attributes in the dataset in $\{\frac{\sqrt{M}}{2}, \sqrt{M}, 2\sqrt{M}, \frac{\log_2 M}{2}, \log_2 M, 2\log_2 M\}$. As before, we do not optimize the number of trees in the forest, leaving it always at 1000. A summary of the hyper-parameters and their values is shown in Table 6.6. Random Forests-star, on the other hand, is not directly optimized by grid search; instead, it is the application of CN2 rule-based aggregator, described in Section 6.1.1, to the best forest found for a given external fold.

Table 6.6: Random Forests and the hyper-parameters used in nested cross-validation. Unmentioned hyper-parameters imply the use of Weka defaults.

Algorithm	Hyper-parameter	Values
Random Forest	Number of trees in forest	1000
Random Forest	Share of training set to be used at any given tree (randomly sampled)	{90%, 100%}
Random Forest	Number of features sampled to be used at any given split of trees	$\{\frac{\sqrt{M}}{2}, \sqrt{M}, 2\sqrt{M}, \frac{\log_2 M}{2}, \log_2 M, 2\log_2 M\}$

Table 6.7: Groups of algorithms in EDNEL nested-cross validation experiment.

Group	Algorithms
Evolutionary Algorithms	EDNEL (with CN2) EDNEL (without CN2) PUMA-star
Base classifiers	J48 SimpleCart JRip PART DecisionTable
Baseline Ensembles	Unoptimized ensemble Optimized ensemble
Random Search	Random Search
Boosted	Adaboost J48 SimpleCart JRip PART DecisionTable
Random Forest	Random Forest Random Forest-star

Hardware specifications and source code

EDNEL (both-versions) and PUMA-star were executed in two separate machines with the same hardware components. While there are minor differences between the machines (such as the operating system, and memory size), we do not expect any impact on the outcome of the experiments. The hardware specifications are described in Table 6.8. All other baseline algorithms were run in the G-Machine. The source code used for performing nested cross-validation in PUMA-star is available at https://github.com/henryzord/PBIL, while EDNEL source code is available at https://github.com/henryzord/ednel.

Table 6.8: Hardware components of machines on which code was run.

Specification	G-Machine	Z-Machine
Processor	AMD Ryzen Threadripper 1950X	AMD Ryzen Threadripper 1950X
Core speed	3.4GHz	3.4GHz
Cores	32	32
Architecture	x86_64	x86_64
RAM Memory	125GB	128GB
Operating System	Ubuntu 18.04.3 LTS	Ubuntu 20.04.2 LTS

6.2.2 Nested Cross-validation experimental results

As in PUMA's evaluation, We use the same assortment of statistical tests to assess EDNEL performance. All tests were conducted with a significance level of 0.05, which is standard for most machine learning experiments. We use the STAC site [221][5] for statistical tests, and the Orange

[5]Available at http://tec.citius.usc.es/stac/index.html. Accessed July 17 2021.

Python library [6] for generating critical difference graphics. The exact script used is available in our Github repository[7].

We discuss the results regarding the pre-established groups, starting with the EAs: EDNEL (both versions) and PUMA-star. For this group, the *p*-value of Friedman aligned ranks test is $8e-5$, indicating a significant difference within the group. The Nemenyi post-hoc test, whose critical difference graph can be visualized in Figure 6.10, confirms that no algorithm in this group is similar to any other. With PUMA-star being the best algorithm in this group, this frustrated our expectations, as we expected EDNEL (any version) to outperform the former algorithm considering its capability of capturing complex relationships between variables in the graphical model. With the data we have, we speculate on the reasons for this outcome as follows: (i) PUMA-star is simpler than EDNEL, hence allowing it to execute for more generations (as it can be viewed in Figure 6.11), and this has a positive impact on the predictive performance; (ii) the class of datasets tested in this thesis do not exploit the added benefits of inferring relationships between variables in the problem; (iii) the early stop policy of PUMA-star is more effective than EDNEL's; or (iv) EDNEL, which naturally requires more data than PUMA-star, due to its Dependency Network, is overfitting the learning data. Indeed, the EDNEL version with CN2 rules – which, in some sense, performs a *pruning* on the aggregated models – outperforms the EDNEL version without CN2 rules, providing some strong evidence regarding data overfitting. The unweighted AUCs for each algorithm are shown in Table 6.9.

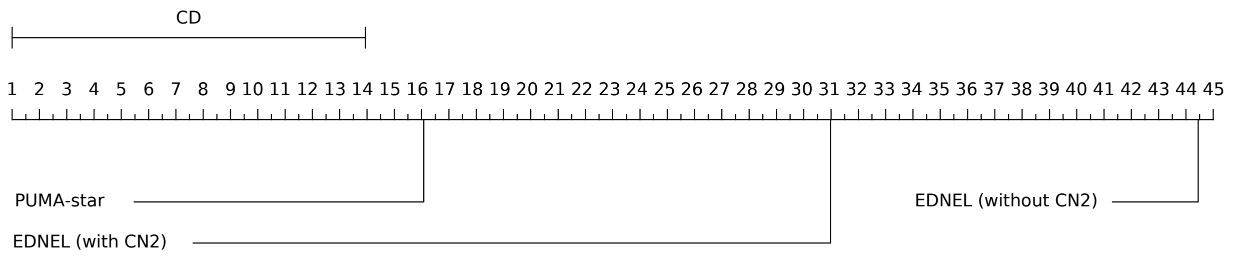


Figure 6.10: Critical difference graph for the EAs in the nested cross-validation experiment. Note that the ranks are aligned.

We proceed the analysis to the next group of algorithms, now comparing the two best evolutionary approaches (PUMA-star and EDNEL with CN2) to the group of base classifiers. All base classifiers were optimized with grid search. The *p*-value for the Friedman test is virtually 0. The Nemenyi post-hoc test (that can be seen in Figure 6.12) verifies that none of the EAs is statistically similar to the base classifiers. In other words, simply performing a grid search on the space of hyper-parameters is not as powerful as employing an EA for the same task. Table 6.10 shows the unweighted AUC for each algorithm.

The third group comprises the two baseline ensembles, the first with the default Weka's hyper-parameters values, the second a concatenation of individually-optimized base classifiers (the

[6]Available at https://orange3.readthedocs.io/projects/orange-data-mining-library/en/latest/index.html, accessed July 22 2021.

[7]https://github.com/henryzord/thesis_experiments

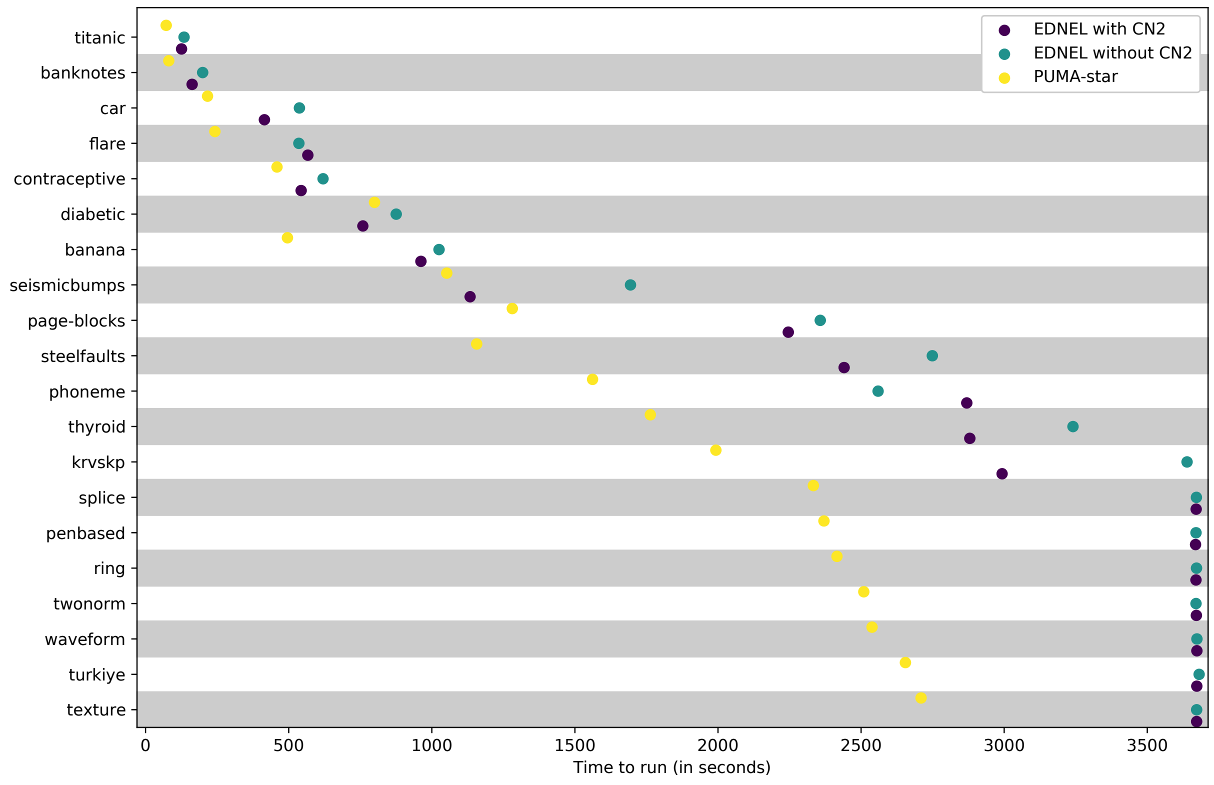


Figure 6.11: Average time to execute PUMA-star, EDNEL (with CN2 rule extractor), and EDNEL (without CN2 rule extractor). Datasets are ordered by average time among the three algorithms.

Table 6.9: Unweighted AUC for the EAs. Best algorithm for each dataset is shown in bold.

Dataset	EDNEL (with CN2)	EDNEL (without CN2)	PUMA-star
banana	0.9587	0.9531	**0.9588**
banknotes	0.9970	**0.9971**	0.9969
car	**0.9974**	0.9943	0.9973
contraceptive	0.7253	0.7167	**0.7341**
diabetic	0.7289	0.7280	**0.7335**
flare	0.9235	0.9223	**0.9255**
krvskp	0.9983	0.9983	**0.9991**
pageblocks	0.9885	0.9827	**0.9902**
penbased	**0.9991**	0.9988	0.9989
phoneme	0.9446	0.9323	**0.9468**
ring	0.9837	0.9814	**0.9854**
seismicbumps	0.7057	0.7463	**0.7739**
splice	**0.9902**	0.9857	0.9893
steelfaults	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	0.9974	**0.9981**
thyroid	**0.9997**	0.9986	**0.9997**
titanic	0.7582	0.7573	**0.7615**
turkiye	**0.8532**	0.8408	0.8495
twonorm	0.9677	0.9757	**0.9828**
waveform	0.9493	0.9456	**0.9518**
Average rank	1.950	2.675	**1.375**
Aligned rank	30.975	44.450	**16.075**

best versions from the last group). The *p*-value for the Friedman aligned ranks test is $3.45e-3$, which rejects the null hypothesis. Following a Nemenyi post-hoc test, we verify, according to Figure 6.13, that PUMA-star is the best algorithm, followed by the optimized baseline ensemble, then EDNEL with CN2 rules, and finally the unoptimized baseline ensemble. The fact that this test finds PUMA-star and the

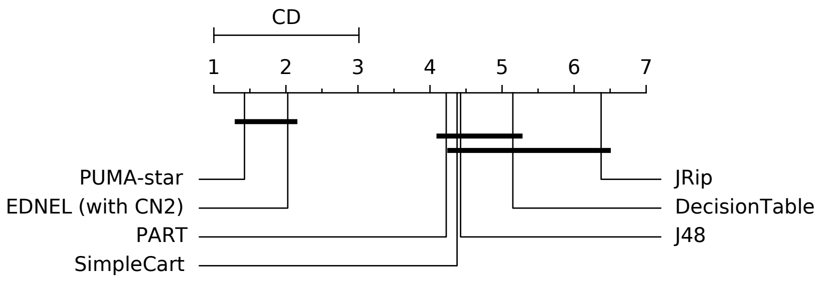


Figure 6.12: Critical difference graph for PUMA-star, EDNEL (with CN2), and optimized base classifiers in the nested cross-validation experiment.

Table 6.10: Unweighted AUCs for the base classifiers and the EAs (PUMA-star and EDNEL with CN2). Best algorithm for each dataset is shown in bold.

Dataset	EDNEL (with CN2)	PUMA-star	J48	SimpleCart	JRip	PART	DecisionTable
banana	0.9587	**0.9588**	0.9423	0.9559	0.8950	0.9453	0.8077
banknotes	**0.9970**	0.9969	0.9876	0.9891	0.9804	0.9912	0.9825
car	**0.9974**	0.9973	0.9724	0.9942	0.9533	0.9907	0.9742
contraceptive	0.7253	**0.7341**	0.7040	0.7015	0.6373	0.7014	0.7126
diabetic	0.7289	**0.7335**	0.7133	0.6810	0.6408	0.6955	0.7001
flare	0.9235	**0.9255**	0.9109	0.9076	0.8629	0.9089	0.9193
krvskp	0.9983	**0.9991**	0.9975	0.9979	0.9947	0.9960	0.9903
page-blocks	0.9885	**0.9902**	0.9576	0.9477	0.9057	0.9553	0.9773
penbased	**0.9991**	0.9989	0.9897	0.9889	0.9873	0.9883	0.9646
phoneme	0.9446	**0.9468**	0.9061	0.9032	0.8282	0.9037	0.8735
ring	0.9837	**0.9854**	0.9402	0.9334	0.9402	0.9614	0.8389
seismicbumps	0.7057	**0.7739**	0.7248	0.7464	0.5252	0.7207	0.7601
splice	0.9902	**0.9893**	0.9683	0.9811	0.9586	0.9756	0.9629
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	**0.9981**	0.9814	0.9808	0.9796	0.9819	0.9719
thyroid	**0.9997**	**0.9997**	0.9977	0.9964	0.9948	0.9962	0.9982
titanic	0.7582	**0.7615**	0.7100	0.7549	0.6755	0.7549	0.7389
turkiye	**0.8532**	0.8495	0.8182	0.8402	0.7384	0.8345	0.8325
twonorm	0.9677	**0.9828**	0.8895	0.8868	0.9216	0.9398	0.8538
waveform	0.9493	**0.9518**	0.8940	0.8994	0.8842	0.9117	0.8924
Average rank	2.025	**1.425**	4.425	4.375	6.375	4.225	5.150

optimized baseline ensemble statistically equivalent does not concern us; we perform a similar test in Section 5.3, Figure 5.7, and PUMA-star was found to be statistically different from both baseline ensembles. Also note that PUMA-star and the optimized baseline ensemble are on the edge of the critical difference distance. We believe the outcome from Figure 6.13 is due to the addition of EDNEL into the group, which diluted the analysis power of the Nemenyi test. Nonetheless, PUMA-star is the best algorithm in this group. Unfortunately, EDNEL is outperformed by the optimized baseline ensemble, at least when using aligned ranks, which takes into account the average performance of each method in each dataset. On another note, notice that both baseline ensembles present similar rank values; that is, the benefit from individually optimizing each base classifier does not significantly improve the predictive performance, at least not for this group of datasets. The unweighted AUCs for this group are shown in Table 6.11.

We already expected the results shown in Figure 6.14: both EAs outperform random search on the space of solutions. The *p*-value for the Friedman aligned ranks test is zero. Note that random search is so inferior to any one of the other two methods that makes the scale of the ranks very large,

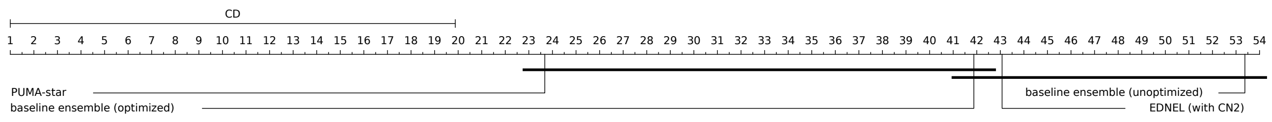


Figure 6.13: Critical difference graph for PUMA-star, EDNEL (with CN2), and both versions of baseline ensembles: one unoptimized (using base classifiers with their default hyper-parameters), and another that just ensembles the individually-optimized base classifiers by means of a grid search. Note that the ranks are aligned.

Table 6.11: Unweighted AUCs for PUMA-star, EDNEL with CN2, and two baseline ensembles: one with its default hyper-parameters from Weka, and another with its members individually optimized by means of a grid search. Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	baseline ensemble (unoptimized)	baseline ensemble (optimized)
banana	0.9587	**0.9588**	0.9529	0.9582
banknotes	0.9970	0.9969	**0.9980**	0.9967
car	**0.9974**	0.9973	0.9959	0.9939
contraceptive	0.7253	**0.7341**	0.7278	0.7280
diabetic	0.7289	**0.7335**	0.7188	0.7302
flare	0.9235	**0.9255**	0.9218	0.9236
krvskp	0.9983	**0.9991**	0.9986	0.9989
pageblocks	0.9885	0.9902	0.9885	**0.9910**
penbased	0.9991	0.9989	**0.9992**	0.9988
phoneme	0.9446	**0.9468**	0.9374	0.9349
ring	0.9837	**0.9854**	0.9824	0.9816
seismicbumps	0.7057	**0.7739**	0.6693	0.7667
splice	**0.9902**	0.9893	0.9876	0.9871
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	0.9981	**0.9984**	0.9979
thyroid	**0.9997**	**0.9997**	**0.9997**	0.9996
titanic	0.7582	**0.7615**	0.7566	0.7588
turkiye	**0.8532**	0.8495	0.8482	0.8521
twonorm	0.9677	**0.9828**	0.9825	0.9812
waveform	0.9493	**0.9518**	0.9508	0.9489
Average rank	2.600	**1.625**	2.850	2.925
Aligned rank	43.050	**23.625**	53.350	41.975

tricking Nemenyi into assessing that both EAs are statistically similar, even though we already verified that this is not the case. Predictive performance for random search is shown in Table 6.12.

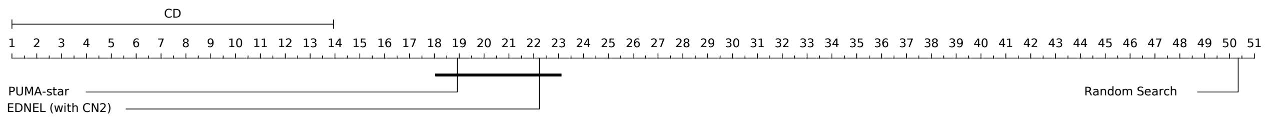


Figure 6.14: Critical difference graph for PUMA-star, EDNEL (with CN2), and random search. The random-search procedure has the same budget as both EAs (i.e., $50 \times 100 = 5,000$ random samples from the solution space). Note that the ranks are aligned.

Up to now, the presented algorithms were interpretable to some degree – be it base classifiers, baseline ensembles, or EAs. The next two groups comprise non-interpretable algorithms. Take boosted algorithms for example. For each iteration of Adaboost, the generated classifier is tailored towards drawing hypothesis to the the most difficult-to-classify instances. For this reason, the hypotheses do not fit the entire dataset, but a piece of it. It is not enough to understand a single classifier in this type of ensemble; one has to see the whole picture to understand the contribution of

Table 6.12: Unweighted AUCs for PUMA-star, EDNEL with CN2, and random search in the space of solutions. Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	Random Search
banana	0.9587	**0.9588**	0.8265
banknotes	**0.9970**	0.9969	0.9853
car	**0.9974**	0.9973	0.9459
contraceptive	0.7253	**0.7341**	0.6398
diabetic	0.7289	**0.7335**	0.6803
flare	0.9235	**0.9255**	0.9234
krvskp	0.9983	**0.9991**	0.9902
pageblocks	0.9885	**0.9902**	0.9367
penbased	**0.9991**	0.9989	0.9906
phoneme	0.9446	**0.9468**	0.8221
ring	0.9837	**0.9854**	0.9607
seismicbumps	0.7057	**0.7739**	0.6092
splice	**0.9902**	0.9893	0.9543
steelfaults	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	**0.9981**	0.9772
thyroid	**0.9997**	**0.9997**	0.9995
titanic	0.7582	**0.7615**	0.7248
turkiye	**0.8532**	0.8495	0.7342
twonorm	0.9677	**0.9828**	0.9514
waveform	0.9493	**0.9518**	0.8781
Average rank	1.725	**1.325**	2.950
Aligned rank	22.225	**18.925**	50.350

ensemble members, which adds complexity to the analysis. Besides – and because – of this effect, boosted algorithms present in general better predictive performance than their evolutionary counterparts (PUMA-star and EDNEL with CN2), on average, although no statistical difference was detected with Nemenyi (Figure 6.15). The Friedman test for this group yielded a *p*-value of $4.34e-3$. The best algorithm in this group, according to the average rank, is the Boosted (and grid-search optimized) version of SimpleCart, followed by PUMA-star. EDNEL (with CN2) is in fourth place, also behind boosted JRip. At a first glance, this seems to be bad news for the EAs. However, by looking at it from another perspective, this also means that prioritizing interpretability (which in general hinders predictive performance, as discussed in Section 2.3) does not significantly decreases predictive performance when using EAs. This can be due to the EA's capability of performing a global robust search on the space of solutions. Unweighted AUCs for methods in this group are shown in Table 6.13.

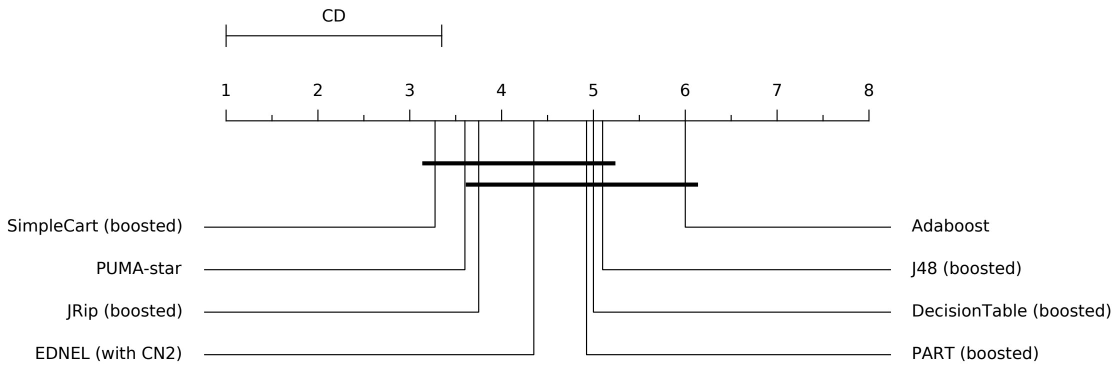


Figure 6.15: Critical difference graph for PUMA-star, EDNEL (with CN2), AdaBoost, and boosted base classifiers, all optimized with grid search.

Table 6.13: Unweighted AUCs for PUMA-star, EDNEL with CN2, Adaboost, and boosted base classifiers. Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	Adaboost	J48 (boosted)	SimpleCart (boosted)	JRip (boosted)	PART (boosted)	Decision Table (boosted)
banana	0.9587	0.9588	0.7842	0.9564	0.9385	**0.9598**	0.9589	0.8899
banknotes	0.9970	0.9969	**0.9997**	0.9990	**0.9997**	0.9996	0.9993	0.9996
car	0.9974	0.9973	0.8297	0.9967	**0.9997**	0.9948	0.9983	0.9937
contraceptive	0.7253	**0.7341**	0.5441	0.6866	0.6887	0.6348	0.6702	0.6942
diabetic	0.7289	0.7335	0.7537	0.7273	**0.7609**	0.7294	0.7231	0.7068
flare	0.9235	**0.9255**	0.6707	0.8992	0.9039	0.8814	0.8997	0.8981
krvskp	0.9983	**0.9991**	0.9953	0.9981	0.9975	0.9984	0.9972	0.9988
pageblocks	0.9885	0.9902	0.8778	0.9857	**0.9925**	0.9843	0.9913	0.9517
penbased	0.9991	0.9989	0.7085	0.9994	0.9993	**0.9996**	0.9993	0.9979
phoneme	0.9446	0.9468	0.8836	0.9511	**0.9544**	0.9374	0.9365	0.9044
ring	0.9837	0.9854	**0.9946**	0.9894	0.9915	0.9929	0.9900	0.9895
seismicbumps	0.7057	**0.7739**	0.7486	0.5865	0.6542	0.7066	0.6099	0.7295
splice	**0.9902**	0.9893	0.9681	0.9829	0.9884	0.9897	0.9846	0.9862
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	0.9981	0.7353	0.9994	**0.9996**	0.9995	0.9990	0.9989
thyroid	**0.9997**	**0.9997**	0.9895	0.9989	0.9980	0.9994	0.9978	0.9930
titanic	0.7582	**0.7615**	0.7466	0.7571	0.7605	0.7586	0.7578	0.7559
turkiye	**0.8532**	0.8495	0.7587	0.6625	0.7385	0.7316	0.6942	0.8333
twonorm	0.9677	0.9828	**0.9965**	0.9910	0.9936	0.9944	0.9906	0.9946
waveform	0.9493	0.9518	0.8795	0.9556	0.9599	0.9633	0.9556	**0.9651**
Average rank	4.350	3.600	6.000	5.100	**3.275**	3.750	4.925	5.000

Finally, we compare the EAs with Random Forests and a version of Random Forests that uses the same algorithm to extract rules from the trees in the ensemble. The *p*-value for Friedman aligned ranks test is zero, allowing us to proceed with a Nemenyi test. The only significant difference found was between Random Forests with rules and the other three algorithms; no statistical difference was detected between Random Forests, PUMA-star, and EDNEL (with CN2). By analyzing the average ranks of Figure 6.14, one can see that the best method was Random Forests, followed by PUMA-star, EDNEL (with CN2), and then Random Forests with rules. This indicates that simply applying the rule extractor algorithm as a post-hoc method over Random Forests significantly decreases its predictive performance. On the other hand, EDNEL can effectively accommodate a rule extract algorithm in its procedure, while not losing too much in terms of predictive performance – the lack of statistical difference is regarding the original version of Random Forests. The Unweighted AUCs for all methods are shown in Table 6.14.

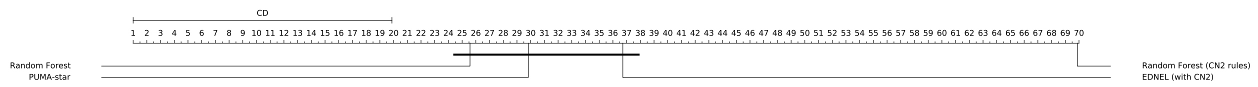


Figure 6.16: Critical difference graph for PUMA-star, EDNEL (with CN2), Random Forests, and rules extracted from Random Forests using the CN2 algorithm. Note that the ranks are aligned.

Table 6.14: Unweighted AUCs for PUMA-star, EDNEL with CN2, grid-search optimized Random Forests, and rules extracted with CN2 algorithm from the optimized Random Forests. Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	Random Forest	Random Forest (CN2 rules)
banana	0.9587	0.9588	**0.9637**	0.9308
banknotes	0.9970	0.9969	**0.9999**	0.9941
car	**0.9974**	0.9973	0.9921	0.9474
contraceptive	0.7253	**0.7341**	0.7045	0.6492
diabetic	0.7289	0.7335	**0.7715**	0.6929
flare	0.9235	**0.9255**	0.9090	0.8508
krvskp	0.9983	**0.9991**	0.9988	0.9963
pageblocks	0.9885	0.9902	**0.9930**	0.9488
penbased	0.9991	0.9989	**0.9997**	0.9916
phoneme	0.9446	0.9468	**0.9671**	0.9059
ring	0.9837	0.9854	**0.9940**	0.9712
seismicbumps	0.7057	**0.7739**	0.7543	0.6099
splice	0.9902	0.9893	**0.9957**	0.9504
steelfaults	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	0.9981	**0.9997**	0.9889
thyroid	0.9997	0.9997	**0.9999**	0.8558
titanic	0.7582	**0.7615**	0.7550	0.7234
turkiye	**0.8532**	0.8495	0.8305	0.7026
twonorm	0.9677	0.9828	**0.9965**	0.9666
waveform	0.9493	0.9518	**0.9684**	0.9101
Average rank	2.450	1.950	**1.675**	3.925
Aligned rank	36.725	29.825	**25.575**	69.875

Comparison to Random Forests

The previous section compared EDNEL and PUMA-star to the "default" configuration of Random Forests, with a sufficient number of trees in the ensemble (i.e., 1000). One argument that the reader could make is that if a more manageable number of trees was used (say, as many trees as there are models in the EAs' ensembles), then direct inspection of Random Forests trees would be humanly possible, while maintaining high predictive performance. We recall the reader that, while this is possible, Random Forests builds trees based on randomization; the choice of attributes to use in the inner nodes for each tree is randomized, and odd choices are often present in trees. Consider for example the *play tennis* dataset, introduced in Section 2.3.1, Table 2.2. The attributes that are the better predictors to whether play tennis or not are, in decreasing order of power: Outlook, Humidity, Temperature, and Wind. It is well within the realm of possibility that most trees in Random Forests still put Outlook as root node, but then opt more often than not for Wind, Humidity, and finally Temperature, effectively subverting the order of impurity decrease of attributes.

Still, even if the trees are not effectively explaining the underlying data, but only randomly guessing a path of tests that best fits them, a human user could grasp the dataset concept from a small-ish number of trees. If we use a sufficiently small number of trees, it could be possible to also keep the high predictive performance of Random Forests, right? To answer this question, we conducted a separated experiment, comparing EDNEL with CN2 rules and PUMA-star to two versions of Random Forests (one with and another without CN2 rules, as explained in the previous section), both with 5 trees in the ensemble (to match the ensemble size of the EAs). We follow the same protocol of the previous section, using Friedman aligned ranks, since four classifiers are being compared, followed by a Nemenyi test. Confidence interval for both tests is 0.05. The unweighted areas under the ROC curve of all methods are shown in Table 6.15.

Table 6.15: Unweighted AUCs for PUMA-star, EDNEL with CN2, grid-search optimized Random Forests (with only five trees in the ensemble), and rules extracted with CN2 algorithm from optimized Random Forests (again with five trees in the ensemble). Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	Random Forest (5 trees)	Random Forest (5 trees, CN2 rules)
banana	0.9587	**0.9588**	0.9389	0.9162
banknotes	0.9970	0.9969	**0.9982**	0.9961
car	**0.9974**	0.9973	0.9758	0.9246
contraceptive	0.7253	**0.7341**	0.6606	0.6410
diabetic	0.7289	**0.7335**	0.7073	0.6825
flare	0.9235	**0.9255**	0.8820	0.8173
krvskp	0.9983	**0.9991**	0.9974	0.9939
pageblocks	0.9885	**0.9902**	0.9600	0.9504
penbased	**0.9991**	0.9989	0.9984	0.9901
phoneme	0.9446	**0.9468**	0.9350	0.8912
ring	0.9837	**0.9854**	0.9759	0.9507
seismicbumps	0.7057	**0.7739**	0.6719	0.5700
splice	**0.9902**	0.9893	0.9802	0.9367
steelfaults	**1.0000**	**1.0000**	0.9999	0.9982
texture	0.9971	**0.9981**	0.9965	0.9824
thyroid	**0.9997**	**0.9997**	0.9996	0.8325
titanic	0.7582	**0.7615**	0.7590	0.7172
turkiye	**0.8532**	0.8495	0.7790	0.6454
twonorm	0.9677	**0.9828**	0.9720	0.9425
waveform	0.9493	**0.9518**	0.9305	0.9005
Average rank	1.850	**1.350**	2.800	4.000
Aligned rank	25.225	**21.675**	46.150	68.950

The Friedman Aligned ranks test yields a value of zero, which refuses the null hypothesis. Proceeding with a Nemenyi test (whose critical difference graph is shown in Figure 6.17), we can see that only EDNEL (with CN2) and PUMA-star are statistically equivalent; both methods far surpasses the average aligned rank of Random Forests (5 trees), and Random Forests with CN2 rules (5 trees). Thus, and confirming what the authors in [212] noted, it is evident that Random Forests requires a sufficient amount of trees in the forest to achieve good results and stability.

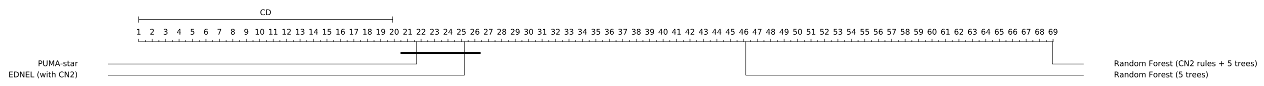


Figure 6.17: Critical difference graph for PUMA-star, EDNEL (with CN2), Random Forests (5 trees), and rules extracted from Random Forests using CN2 algorithm (5 trees). Note that ranks are aligned.

Comparison to Adaboost and boosted base classifiers

Likewise, the reader could make a point that, since the boosted SimpleCart classifier presented better predictive performance (according to average rank) than both EDNEL with CN2 rules and PUMA-star, it could be possible that if boosted SimpleCart had less classifiers, it could achieve good performance while also being interpretable – after all, one of the points we make for considering EDNEL and PUMA interpretable is the fact that produced ensembles never have more than five classifiers. We recall the reader that we do not consider boosted classifiers interpretable, due to the fact that ensemble members from these algorithms are trained to classify harder instances (which, in most cases, are outliers), instead of presenting a general hypothesis, which is the case for both EDNEL and PUMA ensemble members. Hence, to understand the whole dataset, one has to understand each and every ensemble member from boosted classifiers.

However, the reader could make the same argument than from previous section: if the ensemble is small enough (say, composed from five base classifiers), and composed solely from white-box classifiers, then it is still possible to interpret each end every one of them. To show that this approach is inadequate, we carry another experiment, this time comparing PUMA-star and EDNEL with CN2 rules to Adaboost with five ensemble members, as well as each one of the boosted base classifiers (namely J48, SimpleCart, JRip, PART, and DecisionTable) with exactly five classifiers in the final ensemble – which is the maximum amount of classifiers that either EDNEL and PUMA-star have in their ensembles. Since the only hyper-parameter optimized for boosted algorithms is the number of ensemble members, these algorithms are not grid-search optimized; only EDNEL and PUMA-star undergo this procedure, described in Section 6.2.1. We use a Friedman test (since eight algorithms are being compared), followed by a Nemenyi test. Confidence interval for both tests is 0.05. The unweighted areas under the ROC curve of all methods are shown in Table 6.16.

Table 6.16: Unweighted AUCs for PUMA-star, EDNEL with CN2, Adaboost, and boosted base classifiers: J48, SimpleCart, PART, JRip, and DecisionTable. While EDNEL and PUMA-star can have from one to five classifiers, all other algorithms have exactly five ensemble members. Best algorithm for each dataset is shown in bold.

dataset	EDNEL (with CN2)	PUMA-star	Adaboost (5)	J48 (B, 5)	SimpleCart (B, 5)	JRip (B, 5)	PART (B, 5)	Decision Table (B, 5)
banana	0.9587	**0.9588**	0.6995	0.9562	0.9458	0.9579	0.9586	0.8846
banknotes	0.9970	0.9969	0.9631	0.9994	0.9986	0.9989	**0.9996**	0.9977
car	0.9974	0.9973	0.8297	0.9891	0.9985	0.9801	**0.9991**	0.9925
contraceptive	0.7253	**0.7341**	0.5441	0.6922	0.6691	0.6358	0.6695	0.6798
diabetic	0.7289	**0.7335**	0.6618	0.7107	0.6660	0.7020	0.7013	0.6938
flare	0.9235	**0.9255**	0.6707	0.8964	0.9049	0.8779	0.8979	0.8957
krvskp	0.9983	0.9991	0.9150	0.9990	0.9980	**0.9994**	0.9993	0.9987
page-blocks	0.9885	**0.9902**	0.8775	0.9791	0.9817	0.9704	0.9840	0.9405
penbased	0.9991	0.9989	0.7085	0.9992	0.9992	0.9992	**0.9993**	0.9892
phoneme	0.9446	**0.9468**	0.8095	0.9315	0.9336	0.9238	0.9167	0.8971
ring	0.9837	0.9854	0.7222	0.9832	0.9761	0.9822	**0.9890**	0.9615
seismicbumps	0.7057	**0.7739**	0.7547	0.6580	0.6487	0.7184	0.6426	0.7191
splice	**0.9902**	0.9893	0.9525	0.9852	0.9852	0.9876	0.9829	0.9757
steelfaults	**1.0000**	**1.0000**	0.8112	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
texture	0.9971	0.9981	0.7353	0.9983	0.9976	**0.9988**	0.9987	**0.9888**
thyroid	0.9997	0.9997	0.9823	0.9997	0.9997	**0.9995**	0.9991	0.9965
titanic	0.7582	0.7615	0.7396	0.7594	0.7594	0.7597	**0.7637**	0.7570
turkiye	**0.8532**	0.8495	0.7591	0.6550	0.7298	0.7315	0.6995	0.8298
twonorm	0.9677	0.9828	0.8587	0.9733	0.9733	0.9840	**0.9886**	0.9555
waveform	0.9493	**0.9518**	0.8578	0.9285	0.9231	0.9343	0.9310	0.9386
Average rank	3.425	**2.475**	7.500	4.350	4.800	4.150	3.600	5.700

The best algorithm from this experiment is PUMA-star (with an average ranking of 2.475), followed by EDNEL with CN2 rules (3.425), then boosted base classifiers PART (3.600), JRip (4.150), J48 (4.350), SimpleCart (4.800), DecisionTable (5.700), and finally the original Adaboost (7.500). The Friedman test yields a *p*-value of zero, with following Nemenyi post-hoc tests being shown in the critical difference graph of Figure 6.18. It is important to note that, due to the high algorithms-to-datasets ratio, the graph is being compressed, thus showing some equivalences that would be unlikely to occur if the number of compared methods were lower.

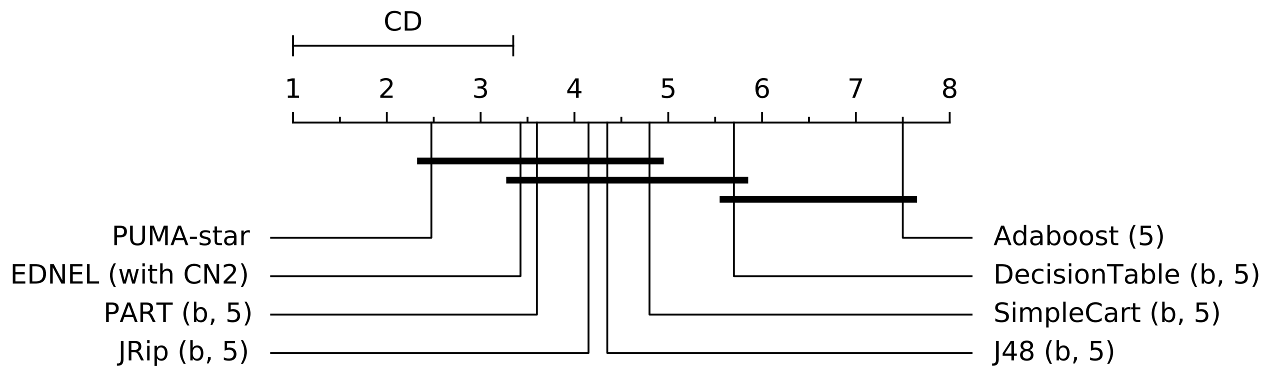


Figure 6.18: Critical difference graph for EDNEL with CN2 rules, PUMA-star, AdaBoost, and boosted base classifiers. While PUMA and EDNEL can have between one and five classifiers, the remaining algorithms all have five ensemble members.

With this experiment, we conclude that PUMA-star and EDNEL with CN2 are tailored to the task of inducing smallish, interpretable ensembles of white-box models; whereas the greedy technique of boosting is inadequate to this task. Indeed, the original Adaboost is the worst algorithm from this group. We speculate that this is because because it uses Decision Stumps as its default base classifier, which is not a strong classifier *per se*. On the other hand, base classifiers (J48, SimpleCart, JRip, PART, DecisionTable) are already well-performing classifiers on their on, being only benefited from boosting – although not benefited enough to surpass either EDNEL or PUMA regarding predictive performance.

6.2.3 Holdout experimental setup

This section describes the second experiment carried on EDNEL and PUMA, this time employing a holdout procedure. This experiment is aimed at comparing these algorithms to AUTOCVE [161, 162], a co-evolutionary algorithm. AUTOCVE uses two populations: one Genetic Programming (GP) population, for evolving ensemble members; and a Genetic Algorithm (GA) population, for selecting said members. AUTOCVE is non-interpretable in three fronts. First, it uses ensemble members that are already non-interpretable (e.g. Random Forests [24], XGBoost [40], among others). Second, the GP algorithm also performs transformations on the input data. Finally, there is no compromise in evolving small-ish ensembles: the number of ensemble members is bounded by the GA population size.

The experimental setup described in this section is the same proposed by AUTOCVE authors in their most recent work [162]: for each dataset, a holdout procedure is set, splitting datasets in a $70/30$ proportion, and the mean balanced accuracy of 10 runs is collected. We do not submit AUTOCVE to the nested cross-validation procedure, described in Section 6.2.1, because we could not execute it in any of our machines – segmentation faults, temporary memory folders being filled to the maximum, and CPUs being overflowed with a torrent of subprocesses plagued the experiment. E-mails were exchanged back and forth with the corresponding author, Celio Larcher, in hopes of solving these issues, but to no avail. Finally, we asked for the experimental metadata from their algorithm and baseline methods. We kindly thank the authors for attending our request, which allowed the experiments in this section. Hence, we compare EDNEL and PUMA to other two methods: the aforementioned AUTOCVE, and XGBoost [40], which is a gradient tree boosting algorithm. Both algorithms are executed with their default hyper-parameters.

Once we have the metadata for the baseline methods, we had only to run our EAs, EDNEL with CN2 and PUMA. We will refer to our algorithms as EDNEL-v2, and PUMA-v2, to make distinctions between the versions described in this section and the versions tested in the nested cross-validation experiment, described in Section 6.2.1. The new naming is necessary because new modifications were made to both algorithms. EDNEL-v2 and PUMA-v2 perform an internal 5-fold cross validation, which is the same procedure adopted by AUTOCVE. We use balanced accuracy as fitness function to match AUTOCVE's function. For equivalent hyper-parameters (e.g., population size, number of generations), we use AUTOCVE values. For other hyper-parameters, such as learning rate, selection share, we use the most frequently chosen hyper-parameters that yielded the best unweighted AUC when performing a grid search in the experiments of Section 6.2.1. While this might seem unfair, since neither AUTOCVE or XGBoost undergo a hyper-parameter optimization process, we should note that the datasets used in this section are different from the ones used in Section 6.2.1. Besides, there is no evidence to support that either PUMA or EDNEL benefits from a common set of hyper-parameters. Hyper-parameters used by three algorithms are shown in Table 6.17, while the datasets used in this experiment are shown in Table 6.18.

Hardware specifications and source code

For this experiment, both EDNEL-v2 and PUMA-v2 were executed in the same machine. As stated earlier, we do not run AUTOCVE or XGBoost, using the results provided by the authors in their work [162]. The hardware specifications for the T-Machine are described in Table 6.19. PUMA and EDNEL source codes are available respectively at https://github.com/henryzord/PBIL and https://github.com/henryzord/ednel, while AUTOCVE source code is available at https://github.com/celiolarcher/AUTOCVE.

6.2.4 Holdout experimental results

For this set of experiments, we repeat the same statistical tests used so far. We found that AUTOCVE is the best algorithm, with an average aligned rank of 10.679, followed by XGBoost (23.321), EDNEL-v2 (37.286), and finally PUMA-v2 (42.714). The Friedman aligned ranks test yields a *p*-value of $1e-5$, which discards the null hypothesis with high confidence. Pairwise comparisons find that AUTOCVE is statistically different from both EDNEL-v2 and PUMA-v2, but similar to XGBoost; XGBoost is also statistically similar to EDNEL-v2, but different from PUMA-v2. Finally, EDNEL-v2 and

Table 6.17: Hyper-parameters used in EDNEL-v2, PUMA-v2, and AUTOCVE.

Algorithm	Hyper-parameter	Values
EDNEL-v2	Learning rate	0.52
EDNEL-v2	Selection share	0.5
EDNEL-v2	Population size	50
EDNEL-v2	Generations	100
EDNEL-v2	Fitness function	balanced accuracy
EDNEL-v2	Fitness calculation method	internal 5-fold CV
EDNEL-v2	Evolution timeout	one and a half hour
EDNEL-v2	Individual timeout	one minute
EDNEL-v2	Burn-in	100
EDNEL-v2	thinning factor	0
EDNEL-v2	Early stop generations	20
EDNEL-v2	Maximum probabilistic parents per variable (same value for all variables)	1
EDNEL-v2	Delay structure learning	5
EDNEL-v2	individual to report	best overall
PUMA-v2	Learning rate	0.13
PUMA-v2	Selection share	0.3
PUMA-v2	Population size	50
PUMA-v2	Generations	100
PUMA-v2	Fitness function	balanced accuracy
PUMA-v2	Fitness calculation method	internal 5-fold CV
PUMA-v2	Evolution timeout	one and a half hour
PUMA-v2	Individual timeout	one minute
PUMA-v2	individual to report	best from last generation
AUTOCVE	Population: base classifiers (GP)	50
AUTOCVE	Population: ensembles (GA)	50
AUTOCVE	Generations	100
AUTOCVE	Mutation rate: base classifiers (GP)	0.9
AUTOCVE	Mutation rate: ensembles (GA)	0.1
AUTOCVE	Crossover rate: base classifiers (GP)	0.9
AUTOCVE	Crossover rate: ensembles (GA)	0.9
AUTOCVE	Evolution timeout	one hour and half
AUTOCVE	Individual timeout	one minute
AUTOCVE	Fitness function	balanced accuracy
AUTOCVE	Fitness calculation method	internal 5-fold CV

Table 6.18: Datasets used in this experiment. The ID column refers to the ID as it appears in the work of Balaji and Allen [12], which is same name system used in AUTOCVE [161, 162].

ID	name	instances	attributes	classes
15	BREAST-W	699	10	2
37	DIABETES	768	9	2
307	VOWEL	990	13	11
451	IRISH	500	6	2
458	ANALCATDATA AUTHORSHIP	841	71	4
469	ANALCATDATA DMFT	797	5	6
1476	GAS-DRIFT	13910	129	6
1485	MADELON	2600	501	2
1515	MICRO-MASS	571	1301	20
6332	CYLINDER-BANDS	540	38	2
23517	NUMERAI28.6	96320	22	2
40496	LED-DISPLAY-DOMAIN-7DIGIT	500	8	10
40499	TEXTURE	5500	41	11
40994	CLIMATE-MODEL-SIMULATION-CRASHES	540	19	2

Table 6.19: Hardware components of the machine used in the experiments.

Specification	T-Machine
Processor	Intel Core i7-5930K
Core speed	3.7GHz
Cores	12
Architecture	x86_64
RAM Memory	96GB
Operating System	Ubuntu 20.04.2 LTS

PUMA-v2 were found to be equivalent. Individual balanced accuracy for each method are shown in Table 6.20, with the critical difference graph shown in Figure 6.19.

These results are expected for a number of reasons. AUTOCVE has no limitation on (i) the number of classifiers in the ensemble (except for the GA population-size hyper-parameter), (ii) type of base classifiers (i.e., interpretable or not), and (iii) the type of transformations that can be performed on the input data. All these limitations are present in our algorithms EDNEL-v2 and PUMA-v2. Besides, AUTOCVE is an EA, which allows it to perform a global search on the space of solutions. This can help explain why it outperformed XGBoost, although they are still statistically equivalent. Another way of interpreting these results is that EAs are powerful optimization methods; when applied unconstrained to a problem, they can outperform well-established algorithms. Comparatively, when applied with constraints, they can still extract the best achievable outcome (e.g., XGBoost and EDNEL-v2). However, when two versions of EAs are compared, one unconstrained and another constrained, the unconstrained version is likely to win – and our experiments demonstrate just that (e.g., AUTOCVE and EDNEL-v2).

Finally, the fact that EDNEL-v2 outperforms PUMA-v2 in this experiment (even though both methods are statistically equivalent) is, to us, another indication that EDNEL could actually be overfitting the training set. From Figure 6.11 from the previous section, it is safe to assume that EDNEL takes longer to run than PUMA; hence, if for some datasets EDNEL was not able to sample a single solution, it defaults to the unoptimized baseline ensemble (this mechanism is described in Section 6.1.2). Therefore, it could be the case that EDNEL is actually benefiting from this mechanism, while PUMA has to perform yet another sampling procedure to sample at least one valid individual. Recall that EDNEL's baseline ensemble uses the Weka default hyper-parameters, which are likely to be good all-around values; and PUMA, with only one individual, is more likely to generate a worse result than EDNEL.

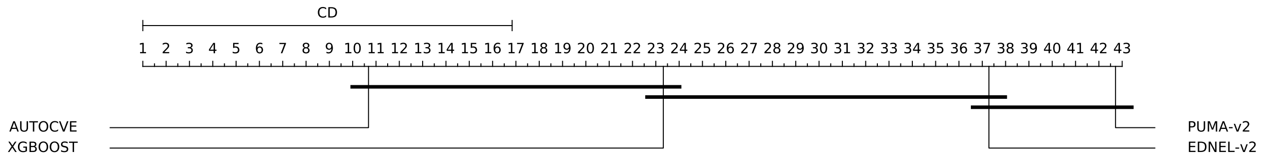


Figure 6.19: Critical difference graph for AUTOCVE, XGBoost, PUMA-v2, and EDNEL-v2. Note that the ranks are aligned.

Table 6.20: Average balanced accuracy for all methods from 10 holdout runs and the respective standard deviations. Best algorithm for each dataset is shown in bold.

dataset (id)	EDNEL-v2	PUMA-v2	AUTOCVE	XGBoost
15	0.9522 $\pm$ 0.01	0.9491 $\pm$ 0.01	**0.9589 $\pm$ 0.02**	0.9539 $\pm$ 0.01
37	0.7035 $\pm$ 0.03	0.7098 $\pm$ 0.02	0.7244 $\pm$ 0.03	**0.7291 $\pm$ 0.02**
307	0.8461 $\pm$ 0.03	0.7986 $\pm$ 0.04	**0.9816 $\pm$ 0.01**	0.8663 $\pm$ 0.03
451	0.9169 $\pm$ 0.17	0.9139 $\pm$ 0.17	**1.0000 $\pm$ 0.00**	**1.0000 $\pm$ 0.00**
458	0.9628 $\pm$ 0.02	0.9200 $\pm$ 0.02	**0.9938 $\pm$ 0.00**	0.9747 $\pm$ 0.01
469	0.1918 $\pm$ 0.02	0.1840 $\pm$ 0.02	**0.2181 $\pm$ 0.03**	0.2177 $\pm$ 0.03
1476	0.9886 $\pm$ 0.00	0.9862 $\pm$ 0.00	**0.9954 $\pm$ 0.00**	0.9881 $\pm$ 0.00
1485	0.7596 $\pm$ 0.02	0.7536 $\pm$ 0.02	**0.8709 $\pm$ 0.02**	0.7257 $\pm$ 0.02
1515	0.7115 $\pm$ 0.17	0.7149 $\pm$ 0.18	**0.9007 $\pm$ 0.02**	0.8564 $\pm$ 0.02
6332	0.7033 $\pm$ 0.05	0.7006 $\pm$ 0.05	**0.7877 $\pm$ 0.02**	0.7720 $\pm$ 0.03
23517	0.5179 $\pm$ 0.00	0.5098 $\pm$ 0.01	**0.5205 $\pm$ 0.00**	0.5196 $\pm$ 0.00
40496	0.7325 $\pm$ 0.03	0.7231 $\pm$ 0.03	**0.7354 $\pm$ 0.02**	0.7090 $\pm$ 0.03
40499	0.9620 $\pm$ 0.00	0.9552 $\pm$ 0.01	**0.9988 $\pm$ 0.00**	0.9756 $\pm$ 0.00
40994	0.6709 $\pm$ 0.13	0.6676 $\pm$ 0.06	**0.8458 $\pm$ 0.06**	0.7061 $\pm$ 0.05
Average rank	2.929	3.714	**1.107**	2.250
Aligned rank	37.286	42.714	10.679	23.321

Individual interpretability analysis

Statistically, EDNEL-v2 and XGBoost are equivalent, so why should we consider EDNEL-v2 as a usable classifier? We perform an analysis of EDNEL-v2 ensemble members to verify whether the gain in interpretability justifies the decrease in predictive performance.

We should recall that interpretability is difficult to be measured objectively, as discussed in Section 2.3. For this reason, we perform a subjective analysis of the ensemble members, without strong assumptions or conclusions.

We choose an ensemble produced by EDNEL-v2 on the LED-DISPLAY-DOMAIN-7DIGIT dataset (ID 40496), which has 500 instances, 8 attributes, and 10 classes. The ensemble generated by EDNEL-v2 uses all five base classifiers (namely J48, SimpleCart, PART, JRip, and DecisionTable), with majority voting as aggregation policy, and is the best ensemble found in the last generation (100-th). The ensemble members are shown in Tables 6.21, 6.22, 6.23, and Figures 6.20, 6.21.

We begin with the tree-based classifiers. Both J48 (Figure 6.20) and SimpleCart (Figure 6.21) produce smaller trees. J48 model has 21 nodes in total, with 10 internal (decision) nodes, and 11 leaf (outcome) nodes; while SimpleCart produces a tree with 19 nodes in total, from which 9 are internal nodes, and 10 leaf nodes. While the dataset *per se* does not help with regards to interpretability (the attributes are named VX, with X being the index of the attribute in the dataset), these trees are small enough for a human reader to grasp the overall concept in a few minutes.

We proceed the analysis with rule-based classifiers. We recall that both PART (Table 6.21) and JRip (Table 6.22) use an ordered list instead of a rule set. That is, the uppermost rule in the table is the first to be analyzed; if it is triggered by an instance, then the instance belongs to the class given by the rule. If this is not the case, then the next rule is analyzed, and so on until at least the last rule is triggered or the default rule (which usually is the majority class) is chosen. PART produces a system with 23 rules (including default clause), while JRip outputs a model with 11 rules. While these two algorithm have more tests than tree-based classifiers (because each rule has several conditions), we believe that 23 and 11 conditions would still qualify as simple models.

Next, we analyze the Decision Table model, in Table 6.23. With 29 rows, Decision Table is the largest model, however not by a lot. The decision table shown here will classify an instance that fulfills all criteria (i.e., conditions expressed in the columns). If an instance does not fulfill a single criterion, then the row is discarded and the instance must be "carried" over to the next row. An aspect of this model that might compromise interpretability is its default rule, which uses a nearest neighbor approach to assign instances that do not fit any other rule. While this is not as intuitive as assigning the majority class for all unmatched instances, there is still an interpretation to this practice: we assign the class of the instance that resembles the most the unknown example. Additionally, classification with decision tables can be quick, since a user can acquire intuition over some criteria. For example, if the instance in question for attribue V1 has a value greater than 0.5, than it is evident that it will not satisfy any of the multiple rows that have the $(-\infty–0.5]$ condition for the respective column in the table.

Finally, we can interpret the relationship between variables in EDNEL's Probabilistic Graphical Model (a Dependency Network), shown in Figure 6.22. As it was explained in Section 6.1.2, the graphical model starts only with deterministic relationships (Figure 6.22a). As the training process progresses, probabilistic relationships can be learned by EDNEL (Figure 6.22b). In the last generation, for example, we learn that the aggregation method depends, to some degree, on PART's pruning policy. Likewise, J48 pruning policy influences on the search algorithm for Decision Table.

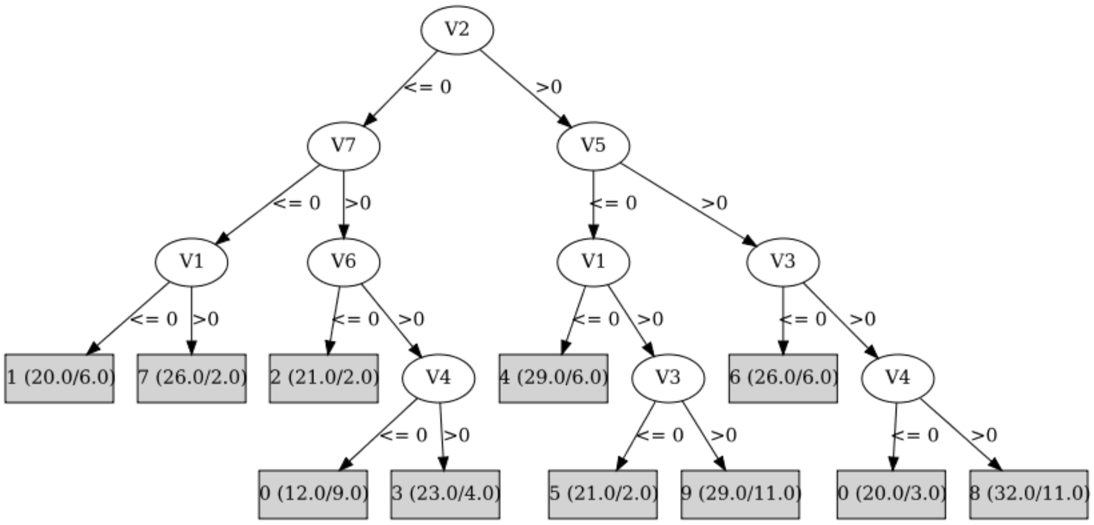


Figure 6.20: J48 classifier from the best ensemble found by EDNEL-v2 for the LED-DISPLAY-DOMAIN-7DIGIT dataset.

Table 6.21: PART classifier from the best ensemble found by EDNEL-v2 for the LED-DISPLAY-DOMAIN-7DIGIT dataset.

conditions	predicted class
$(V2 \leq 0.0) \wedge (V7 \leq 0.0) \wedge (V1 > 0.0) \wedge (V5 \leq 0.0)$	7 (27.0/2.0)
$(V5 > 0.0) \wedge (V6 \leq 0.0) \wedge (V2 \leq 0.0) \wedge (V4 > 0.0) \wedge (V7 > 0.0)$	2 (15.0)
$(V5 > 0.0) \wedge (V6 \leq 0.0) \wedge (V2 \leq 0.0) \wedge (V4 \leq 0.0)$	2 (6.0/2.0)
$(V5 > 0.0) \wedge (V3 \leq 0.0) \wedge (V4 > 0.0) \wedge (V1 > 0.0)$	6 (24.0/4.0)
$(V5 > 0.0) \wedge (V7 > 0.0) \wedge (V4 > 0.0) \wedge (V3 > 0.0) \wedge (V6 > 0.0)$	8 (30.0/8.0)
$(V2 \leq 0.0) \wedge (V4 > 0.0) \wedge (V5 \leq 0.0) \wedge (V6 > 0.0) \wedge (V1 > 0.0)$	3 (23.0/3.0)
$(V2 \leq 0.0) \wedge (V4 > 0.0) \wedge (V5 \leq 0.0) \wedge (V1 > 0.0)$	2 (8.0/4.0)
$(V2 \leq 0.0) \wedge (V4 > 0.0) \wedge (V5 \leq 0.0)$	3 (7.0/2.0)
$(V2 \leq 0.0) \wedge (V6 \leq 0.0) \wedge (V1 \leq 0.0)$	1 (5.0/3.0)
$(V1 \leq 0.0) \wedge (V2 \leq 0.0)$	1 (23.0/5.0)
$(V1 \leq 0.0) \wedge (V7 \leq 0.0) \wedge (V5 \leq 0.0) \wedge (V3 > 0.0)$	4 (22.0/1.0)
$(V5 > 0.0) \wedge (V6 \leq 0.0) \wedge (V7 > 0.0)$	8 (10.0/5.0)
$(V3 \leq 0.0) \wedge (V5 \leq 0.0) \wedge (V1 > 0.0) \wedge (V4 > 0.0)$	5 (25.0/3.0)
$(V5 > 0.0) \wedge (V6 > 0.0) \wedge (V4 \leq 0.0) \wedge (V3 > 0.0)$	0 (29.0/7.0)
$(V1 \leq 0.0) \wedge (V7 > 0.0) \wedge (V5 \leq 0.0)$	4 (11.0/6.0)
$(V1 \leq 0.0) \wedge (V7 \leq 0.0)$	4 (10.0)
$(V6 \leq 0.0)$	2 (9.0/3.0)
$(V5 > 0.0) \wedge (V4 > 0.0)$	8 (8.0/4.0)
$(V5 \leq 0.0) \wedge (V3 > 0.0) \wedge (V4 > 0.0)$	9 (27.0/10.0)
$(V3 \leq 0.0) \wedge (V5 \leq 0.0)$	5 (7.0/3.0)
$(V3 \leq 0.0)$	6 (7.0/3.0)
$(V2 > 0.0)$	9 (7.0/3.0)
default rule	3 (5.0/3.0)

Table 6.22: JRip classifier from the best ensemble found by EDNEL-v2 for the LED-DISPLAY-DOMAIN-7DIGIT dataset.

conditions	predicted class
$(V1 \leq 0) \wedge (V4 \leq 0) \wedge (V2 \leq 0)$	1 (28.0/8.0)
$(V3 \leq 0) \wedge (V5 \geq 1) \wedge (V6 \geq 1)$	6 (35.0/8.0)
$(V4 \leq 0) \wedge (V5 \geq 1) \wedge (V6 \geq 1)$	0 (29.0/7.0)
$(V2 \geq 1) \wedge (V3 \geq 1) \wedge (V5 \leq 0) \wedge (V1 \geq 1)$	9 (37.0/14.0)
$(V3 \leq 0) \wedge (V2 \geq 1) \wedge (V1 \geq 1) \wedge (V5 \leq 0)$	5 (29.0/4.0)
$(V4 \leq 0) \wedge (V7 \leq 0) \wedge (V1 \geq 1)$	7 (26.0/2.0)
$(V7 \leq 0) \wedge (V1 \geq 1) \wedge (V5 \leq 0)$	7 (4.0/1.0)
$(V2 \leq 0) \wedge (V5 \leq 0) \wedge (V3 \geq 1) \wedge (V4 \geq 1) \wedge (V7 \geq 1) \wedge (V6 \geq 1)$	3 (26.0/2.0)
$(V6 \leq 0)$	2 (46.0/15.0)
$(V1 \leq 0) \wedge (V7 \leq 0)$	4 (32.0/1.0)
default rule	8 (53.0/28.0)

Table 6.23: Decision Table classifier from the best ensemble found by EDNEL-v2 for the LED-DISPLAY-DOMAIN-7DIGIT dataset. Columns denote attributes in the dataset, with the last column being the class attribute; row values indicate the values, for each attribute, that an instance must meet to be considered from that class.

v1	v2	v3	v4	v5	class
(−∞–0.5]	(0.5–∞)	(0.5–∞)	(0.5–∞)	(0.5–∞)	4
(0.5–∞)	(0.5–∞)	(0.5–∞)	(0.5–∞)	(0.5–∞)	8
(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(0.5–∞)	(0.5–∞)	2
(0.5–∞)	(−∞–0.5]	(0.5–∞)	(0.5–∞)	(0.5–∞)	2
(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(0.5–∞)	(0.5–∞)	6
(0.5–∞)	(0.5–∞)	(−∞–0.5]	(0.5–∞)	(0.5–∞)	6
(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(0.5–∞)	2
(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(0.5–∞)	0
(−∞–0.5]	(0.5–∞)	(0.5–∞)	(−∞–0.5]	(0.5–∞)	0
(0.5–∞)	(0.5–∞)	(0.5–∞)	(−∞–0.5]	(0.5–∞)	0
(−∞–0.5]	(0.5–∞)	(0.5–∞)	(0.5–∞)	(−∞–0.5]	4
(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(0.5–∞)	1
(0.5–∞)	(0.5–∞)	(0.5–∞)	(0.5–∞)	(−∞–0.5]	9
(0.5–∞)	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(0.5–∞)	0
(0.5–∞)	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(0.5–∞)	6
(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(0.5–∞)	(−∞–0.5]	3
(0.5–∞)	(−∞–0.5]	(0.5–∞)	(0.5–∞)	(−∞–0.5]	3
(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	4
(0.5–∞)	(0.5–∞)	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	5
(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	(0.5–∞)	0
(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	6
(0.5–∞)	(0.5–∞)	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	9
(−∞–0.5]	(0.5–∞)	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	4
(−∞–0.5]	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	1
(0.5–∞)	(−∞–0.5]	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	7
(0.5–∞)	(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	5
(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	1
(0.5–∞)	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	(−∞–0.5]	7
Non-matches are assigned to the closest instance					

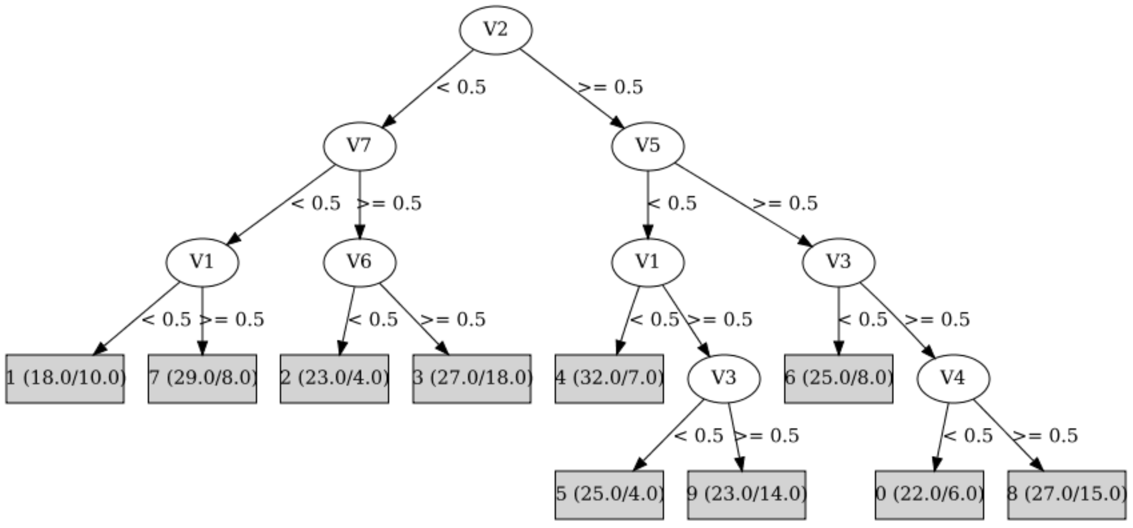


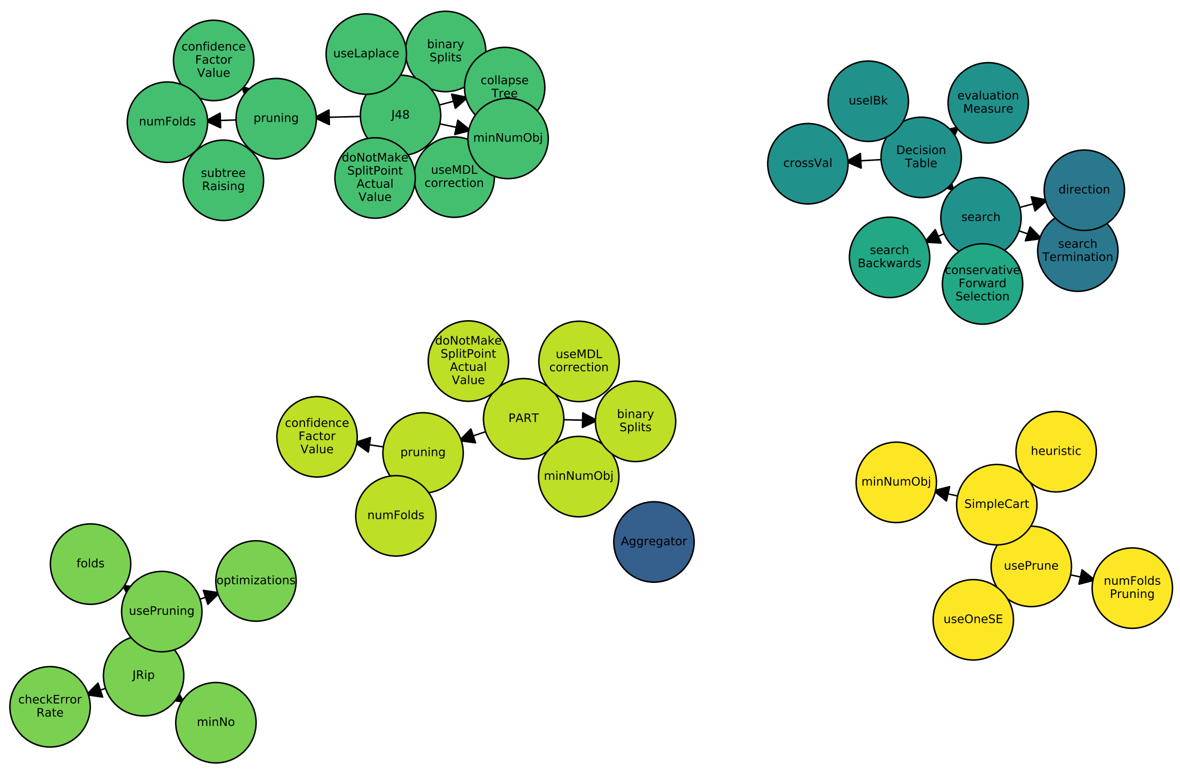
Figure 6.21: SimpleCart classifier from the best ensemble found by EDNEL-v2 for the LED-DISPLAY-DOMAIN-7DIGIT dataset.

6.3 Discussion and Final Remarks

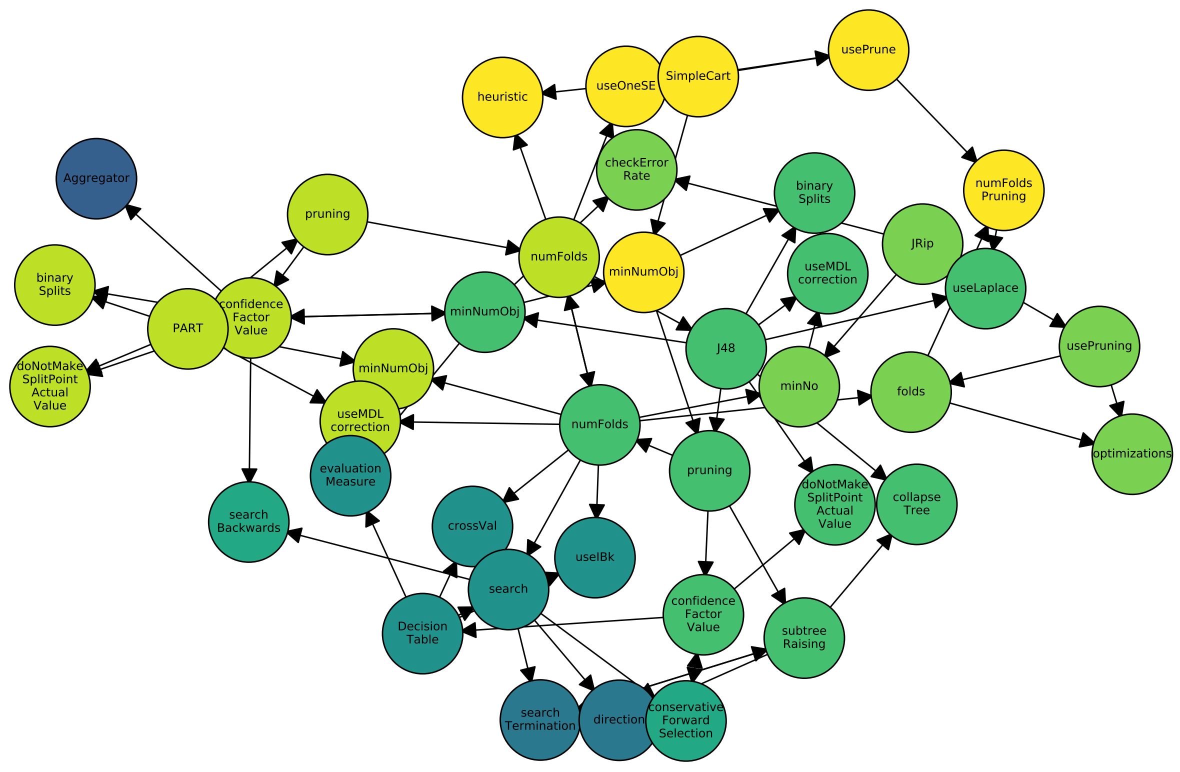
We presented in this Chapter EDNEL, an Estimation of Distribution Algorithm for evolving ensembles of interpretable classifiers. EDNEL, as opposed to PUMA (the algorithm presented in Chapter 5) uses a Dependency Network to infer relationship between variables (namely the hyper-parameters of base classifiers). This allows EDNEL to draw relationship between variables in the problem, which can be further interpreted by a user. EDNEL produces an ensemble with at most five classifiers, all from different learning algorithms (namely J48, SimpleCart, PART, JRip, and DecisionTable, from the Weka Toolkit [113]). EDNEL can also choose between using a simple majority voting, a weighted majority voting, or a rule extraction algorithm, that transforms J48, SimpleCart, and DecisionTable models into a single rule-based rule set model.

Despite these improvements, in the nested-cross validation experiments (Section 6.2.1), PUMA (from Chapter 5) was found to be superior than both versions of EDNEL, presenting a better average rank while also being statistically different to both versions. PUMA-star and EDNEL with CN2 rules outperform all base classifiers and random search. PUMA-star outperforms both baseline ensembles – be it with or without optimization – while EDNEL is statistically equivalent to the optimized version.

When comparing evolutionary algorithms to non-interpretable baselines, both PUMA-star and EDNEL with CN2 rules are statistically equivalent to all Adaboost versions, except for the original one, which is outperformed by every other algorithm within this group. PUMA-star and EDNEL with CN2 are also statistically equivalent to Random Forests. EDNEL with CN2 outperforms Random Forests with CN2 – a proof that simply applying post-hoc explaining methods to already-existing predictive models in the hope of achieving interpretability is not sufficient; it is necessary to develop an efficient strategy to accommodate both interpretability and predictive performance.



(a) First generation



(b) 100th generation

Figure 6.22: Relationship between variables in EDNEL's Dependency Network in the (a) first generation, and (b) last (100th) generation, during training on the LED-DISPLAY-DOMAIN-7DIGIT dataset (ID 40496). Arrows indicate the relationship between variables: heads point to children, whereas arrow bases point to parents.

In another set of experiments, when constraining Adaboost and Random Forests to have five base classifiers (the same constraint imposed on EDNEL and PUMA), EDNEL is capable of outperforming the two former methods, being only outperformed by PUMA. This confirms an observation made in Section 5.4, that it is inadequate to have small-ish ensembles learned from Adaboost and Random Forests; if that is the case – and having small-ish ensembles is a proxy measure of interpretability –, then an evolutionary algorithm (be it PUMA or EDNEL) is more appropriate for the task.

In a holdout experiment (Section 6.2.3), EDNEL presents better average rank than PUMA-star, although being deemed still statistically similar to the latter according to a Nemenyi post-hoc test.

With the data available at the moment, we are unable to pinpoint a reason for the results from both experiments; EDNEL was expected to outperform PUMA, since it is the next iteration of that algorithm with a more sophisticated behavior. However, the main suspicion is that EDNEL is actually meta-overfitting the training data, since it requires more data to correctly learn relationships between variables in its Dependency Network than PUMA, which does not make use of this structure. For the holdout results in particular, we believe that since EDNEL takes longer to run than PUMA, for most datasets the unoptimized baseline ensemble (i.e., the starting search point in the solution space) is being used, which is already a good all-around set of hyper-parameters for the selected base classifiers.

Overall, we argument that PUMA (either version) should be prioritized as a robust algorithm for inducing ensembles of interpretable classifiers. It is evident that using more powerful methods (such as Adaboost and Random Forests) could achieve better results, but at the cost of sacrificing interpretability. Moreover, PUMA-star and EDNEL use at most five base classifiers, while Random Forests used 1000 among all experiments, and Adaboost versions ranged between 50 and 500.

As future work, we intend to investigate the meta-overfitting suspicion by testing EDNEL in a new setup: using multiple datasets as input. EDNEL could find an overall set of hyper-parameters for base classifiers that works well for different training datasets. The performance would be measured as the average unweighted Area Under the ROC curve across multiple datasets. These hyper-parameters could then be re-used for new, unseen datasets that present similar properties to the training datasets, like number of instances, number of attributes, number of classes, class imbalance, or any other geometrical/topological measure.

7. CONCLUSIONS

This thesis proposed new evolutionary algorithms for learning ensembles of interpretable classifiers, namely PUMA – Probabilistic Univariate Estimation of Distribution Algorithm for Ensemble Learning (Chapter 5) and EDNEL – Estimation of Dependency Network Algorithm for Ensemble Learning (Chapter 6). While investigating the capabilities of this class of evolutionary algorithm, we also developed EEL (Chapter 4) and conducted a survey on evolutionary algorithms for ensemble learning (Chapter 3).

While PUMA and EDNEL are conceptually similar, EDNEL presents the advantage of also explaining the relationship between variable values when inducing ensembles. For example, EDNEL can show dependencies between the pruning policy of a decision tree, the branching factor of another decision tree, and the search policy of a Decision Table algorithm, all within the same ensemble. Nonetheless, if such insights are not valuable to the user, we recommend using PUMA since it presented a better average rank across all experiments while also not being statistically similar to EDNEL.

To assess the performance of these two algorithms, two sets of experiments were conducted. The first compared methods in a nested cross-validation framework, and compared EDNEL and PUMA to a set of baseline algorithms, e.g., Random Forests, Adaboost, C4.5 (or J48), RIPPER (or JRip), among others. While both EDNEL and PUMA produce ensembles of interpretable classifiers, none of the baseline algorithms that were white-box models could achieve statistical equivalence to either EDNEL or PUMA. EDNEL and PUMA are also statistically equivalent to powerful, black-box models (i.e., Random Forests and Adaboost).

In the second set of experiments (using a holdout dataset instead of cross-validation), EDNEL and PUMA were compared to AUTOCVE [161, 162], a co-evolutionary algorithm that employs data transformation and uses powerful black-box models as base classifiers (e.g., XGBoost [40], Random Forests [24]), and to XGBoost itself. For the evaluated datasets, AUTOCVE proved to be the best algorithm regarding predictive performance. AUTOCVE is not statistically equivalent to any other method, while XGBoost and EDNEL were statistically equivalent, tied in second. PUMA was the third best algorithm in this experiment in terms of predictive performance. When considering interpretability, however, only PUMA and EDNEL produce white-box models and do not transform the input space.

From all surveyed work, described in Chapter 3, no other algorithm operates in the three stages of ensemble learning: generation, selection, and integration. We produced two algorithms to this effect: PUMA and EDNEL. These algorithms also automatically adapt to the application domain at hand: they can choose the best combination of hyper-parameters, base classifiers, and aggregation policy to the dataset at hand. PUMA and EDNEL, by using already-interpretable white-box models, and not relying on explanation methods applied to black-box models (e.g., SHAP [176]), also steer towards producing interpretable ensembles – a shift in ensemble learning design that other proposed work, be it well-established in the literature (e.g., Random Forests [24]), or recently published (e.g., AUTOCVE [161, 162]) do not take into account. By using our algorithms, a user has only five models to interpret at most, which is much easier to analyze than, say, one hundred trees, as it is usually the case with Random Forests. Finally, while it is known that white-box models are outperformed by their black-box counterparts [35], we managed to achieve statistical equivalence to two well known black-box models that often fare among the best algorithms in general [88], i.e., Random Forest and Adaboost.

7.1 Limitations

There are two types of limitations that PUMA and EDNEL present. The first one is regarding execution time. This is derived from the framework our methods are built upon, evolutionary algorithms. Evolutionary algorithms will take longer to induce ensembles than greedy-search algorithms, while consuming more memory and processing power. This is due to the population-based strategy used. However, techniques can be employed to mitigate this effect [117]; we in fact made use of parallelization in all of our algorithms to decrease running time during the experiments. Furthermore, once an ensemble has been evolved, its prediction time will be exactly the same than an ensemble that has the same base classifiers but was evolved using a greedy-search strategy.

The second type of limitation this work presents is intrinsic to the type of base classifier our algorithms are evolving, white-box models. These models tend to present predictive performance inferior to black-box models (e.g., neural networks), however are attractive because no explanation model needs to be trained – the explanation model is the same as the predictive model. The experiments conducted in Section 6.2.4 confirm our hypothesis; AUTOCVE, an evolutionary algorithm not restricted to interpretability constraints (i.e., not restricted to using white-box models nor learning small ensembles), performs significantly better than EDNEL and PUMA. The answer to this shortcoming lies in the type of objective the user is willing to achieve. If it is adamant to have a predictive model that is also interpretable, such as when one has to decide the possible treatments for a disease, then our algorithms are a solid option; however, if one wishes to blindly trust the predictions of a black-box model (or, in other words, if the stakes of the problem are not that high), then a black-box model is more appropriate indeed.

Finally, a limitation that is specific to EDNEL is meta-overfitting, something that we suspect occurred in the experiments of Section 6.2. The meta-overfitting would occur since more data is necessary for EDNEL to learn its graphical model (a Dependency Network) structure, thus making EDNEL commit errors when selecting hyper-parameter settings and classifiers for composing a model.

7.2 Future work

We now discuss opportunities for future work: features that we want to implement for future iterations of our algorithms, or situations that we wish to investigate further. First, including more datasets is desirable: although we tried to cover as many types of datasets as possible (e.g., many instances, few attributes; many attributes, few instances; many classes, binary classification, etc.), there are certainly application domains not covered in this work. It might be the case that EDNEL (given the proper corrections) could outperform PUMA in some of these new datasets: for example, for gene expression problems, datasets usually have thousands of features and just tens or hundred of instances.

Next, it would be interesting to investigate whether EDNEL could benefit, in terms of predictive performance, from a Bayesian Network as its probabilistic graphical model as opposed to using a Dependency Network. Dependency Networks allow cycles and explain correlation relationships better than Bayesian Networks. While the use of Bayesian networks could confuse the user in terms of the meaning of the found relationships (causal or correlation-based), it would be nonetheless useful to learn how much the predictive performance of EDNEL could benefit from this new approach.

Finally, we did not test our algorithms in a multi-dataset setup during the evolutionary search. That is, instead of evolving hyper-parameters that are well-suited to a single dataset, we could ask PUMA and EDNEL to find the best set of hyper-parameter settings for a group of base classifiers that could work on multiple datasets. This would require adaptations to our algorithms; the fitness would be regarding a set of meta-training datasets. Datasets would also need to be grouped according to their geometrical/topological properties. Finally, a clustering strategy would need to be chosen to group the datasets based on their similarities. We believe this could address the meta-overfitting problem we suspect EDNEL is suffering, since the same was detected in the work of Barros [15] and the multi-dataset setup helped in achieving better predictive performance.

REFERENCES

[1] Adair, J.; Brownlee, A.; Daolio, F.; Ochoa, G. "Evolving Training sets for Improved Transfer Learning in Brain Computer Interfaces". In: International Workshop on Machine Learning, Optimization, and Big Data, 2017, pp. 186–197.

[2] Albukhanajer, W. A.; Jin, Y.; Briffa, J. A. "Classifier Ensembles for Image Identification using Multi-objective Pareto Features", *Neurocomputing*, vol. 238, May 2017, pp. 316–327.

[3] Alcalá-Fdez, J.; Fernández, A.; Luengo, J.; Derrac, J.; García, S.; Sánchez, L.; Herrera, F. "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework.", *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, April 2011, pp. 255–287.

[4] Ali, S.; Majid, A. "Can–Evo–Ens: Classifier Stacking based Evolutionary Ensemble System for Prediction of Human Breast Cancer using Amino Acid Sequences", *Journal of Biomedical Informatics*, vol. 54, April 2015, pp. 256–269.

[5] Aliakbarian, M. S.; Fanian, A. "Internet Traffic Classification Using MOEA and Online Refinement in Voting on Ensemble Methods". In: Iranian Conference on Electrical Engineering, 2013, pp. 1–6.

[6] Almeida, L. M.; Galvão, P. S. "Ensembles with Clustering-and-Selection Model Using Evolutionary Algorithms". In: Brazilian Conference on Intelligent Systems, 2016, pp. 444–449.

[7] Appel, K.; Haken, W. "Every planar map is four colorable", *Bulletin of the American mathematical Society*, vol. 82–5, September 1976, pp. 711–712.

[8] Asafuddoula, M.; Verma, B.; Zhang, M. "A Divide-and-Conquer Based Ensemble Classifier Learning by Means of Many-Objective Optimization", *IEEE Transactions on Evolutionary Computation*, vol. 22–5, December 2017.

[9] Athar, A.; Butt, W. H.; Anwar, M. W.; Latif, M.; Azam, F. "Exploring the Ensemble of Classifiers for Sentimental Analysis: A Systematic Literature Review". In: International Conference on Machine Learning and Computing, 2017, pp. 410–414.

[10] Augusto, D. A.; Barbosa, H. J. C.; Ebecken, N. F. F. "Coevolutionary Multi-population Genetic Programming for Data Classification". In: Conference on Genetic and Evolutionary Computation, 2010, pp. 933–940.

[11] Bagheri, M. A.; Gao, Q.; Escalera, S. "A Genetic-based Subspace Analysis method for Improving Error-correcting Output Coding", *Pattern Recognition*, vol. 46–10, October 2013, pp. 2830–2839.

[12] Balaji, A.; Allen, A., "Benchmarking automatic machine learning frameworks", 2018, unpublished manuscript.

[13] Baluja, S.; Caruana, R. "Removing the genetics from the standard genetic algorithm". In: International Conference on Machine Learning, 1995, pp. 38–46.

[14] Barros, R. C.; Basgalupp, M. P.; de Carvalho, A.; Freitas, A. A. "A Survey of Evolutionary Algorithms for Decision-Tree Induction", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42–3, June 2012, pp. 291–312.

[15] Barros, R. C.; de Carvalho, A. C.; Freitas, A. A. "Automatic Design of Decision-Tree Induction Algorithms". Springer, 2015, 176p.

[16] Basto-Fernandes, V.; Yevseyeva, I.; Méndez, J. R.; Zhao, J.; Fdez-Riverola, F.; Emmerich, M. T. "A Spam Filtering Multi-objective Optimization Study Covering Parsimony Maximization and Three-way Classification", *Applied Soft Computing*, vol. 48, November 2016, pp. 111–123.

[17] Basto-Fernandes, V.; Yevseyeva, I.; Ruano-Ordás, D.; Zhao, J.; Fdez-Riverola, F.; Méndez, J. R.; Emmerich, M. "Quadcriteria Optimization of Binary Classifiers: Error Rates, Coverage, and Complexity". In: *EVOLVE – A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation VI*, Tantar, A.-A.; Tantar, E.; Emmerich, M.; Legrand, P.; Alboaie, L.; Luchian, H. (Editors), Springer, 2018, pp. 37–49.

[18] Bautista, M. Á.; Pujol, O.; Baró, X.; Escalera, S. "Introducing the Separability Matrix for Error Correcting Output Codes Coding". In: International Workshop on Multiple Classifier Systems, 2011, pp. 227–236.

[19] Bazi, Y.; Alajlan, N.; Melgani, F.; AlHichri, H.; Yager, R. R. "Robust Estimation of Water Chlorophyll Concentrations with Gaussian Process Regression and IOWA Aggregation Operators", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7–7, June 2014, pp. 3019–3028.

[20] Belle, V.; Papantonis, I. "Principles and practice of explainable machine learning", *Frontiers in big Data*, vol. 4, July 2021, pp. 39.

[21] Bhowan, U.; Johnston, M.; Zhang, M. "Evolving Ensembles in Multi-objective Genetic Programming for Classification with Unbalanced Data". In: Conference on Genetic and Evolutionary Computation, 2011, pp. 1331–1338.

[22] Bhowan, U.; Johnston, M.; Zhang, M. "Comparing Ensemble Learning Approaches in Genetic Programming for Classification with Unbalanced Data". In: Conference on Genetic and Evolutionary Computation, 2013, pp. 135–136.

[23] Breiman, L. "Bagging predictors", *Machine learning*, vol. 24–2, August 1996, pp. 123–140.

[24] Breiman, L. “Random Forests”, *Machine learning*, vol. 45–1, October 2001, pp. 5–32.

[25] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. “Classification and regression trees”. Wadsworth International Group, 1984, 368p.

[26] Britto, A. S.; Sabourin, R.; Oliveira, L. E. “Dynamic Selection of Classifiers – A Comprehensive Review”, *Pattern Recognition*, vol. 47–11, November 2014, pp. 3665–3680.

[27] Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L. “Handbook of markov chain monte carlo”. CRC press, 2011, 592p.

[28] Cagnini, H.; Basgalupp, M.; Barros, R. “Increasing Boosting Effectiveness with Estimation of Distribution Algorithms”. In: Congress on Evolutionary Computation, 2018, pp. 1–8.

[29] Cagnini, H. E.; Barros, R. C.; Basgalupp, M. P. “Estimation of distribution algorithms for decision-tree induction”. In: Congress on Evolutionary Computation, 2017, pp. 2022–2029.

[30] Cagnini, H. E.; Freitas, A. A.; Barros, R. C. “An Evolutionary Algorithm for Learning Interpretable Ensembles of Classifiers”. In: Brazilian Conference on Intelligent Systems, 2020, pp. 18–33.

[31] Cao, J.-J.; Kwong, S.; Wang, R.; Li, K. “An Indicator-based Selection Multi-objective Evolutionary Algorithm with Preference for Multi-class Ensemble”. In: International Conference on Machine Learning and Cybernetics, 2014, pp. 147–152.

[32] Cao, P.; Li, B.; Zhao, D.; Zaiane, O. “A Novel Cost Sensitive Neural Network Ensemble for Multiclass Imbalance Data Learning”. In: International Joint Conference on Neural Networks, 2013, pp. 1–8.

[33] Cao, P.; Zhao, D.; Zaiane, O. “Measure Optimized Cost-sensitive Neural Network Ensemble for Multiclass Imbalance Data Learning”. In: International Conference on Hybrid Intelligent Systems, 2013, pp. 35–40.

[34] Caruana, R.; Lou, Y.; Gehrke, J.; Koch, P.; Sturm, M.; Elhadad, N. “Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission”. In: International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1721–1730.

[35] Carvalho, D. V.; Pereira, E. M.; Cardoso, J. S. “Machine Learning Interpretability: A Survey on Methods and Metrics”, *Electronics*, vol. 8–8, July 2019, pp. 832.

[36] Castro, P. A. D.; Von Zuben, F. J. “Learning Ensembles of Neural Networks by Means of a Bayesian Artificial Immune System”, *IEEE Transactions on Neural Networks*, vol. 22–2, February 2011, pp. 304–316.

[37] Chang, C.; Xu, D.; Quek, H. “Pareto-optimal set based Multiobjective Tuning of Fuzzy Automatic Train Operation for Mass Transit System”, *IEEE Proceedings-Electric Power Applications*, vol. 146–5, September 1999, pp. 577–583.

[38] Chaurasiya, R. K.; Londhe, N. D.; Ghosh, S. "Binary DE-based Channel Selection and Weighted Ensemble of SVM Classification for Novel Brain–computer Interface using Devanagari script-based P300 Speller Paradigm", *International Journal of Human–Computer Interaction*, vol. 32–11, July 2016, pp. 861–877.

[39] Chen, H.; Yao, X. "Evolutionary Multiobjective Ensemble Learning based on Bayesian Feature Selection". In: Congress on Evolutionary Computation, 2006, pp. 267–274.

[40] Chen, T.; Guestrin, C. "Xgboost: A scalable tree boosting system". In: International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[41] Chen, W.-C.; Tseng, L.-Y.; Wu, C.-S. "A Unified Evolutionary Training scheme for Single and Ensemble of Feedforward Neural Network", *Neurocomputing*, vol. 143, November 2014, pp. 347–361.

[42] Chen, Y.; Yang, B.; Abraham, A. "Flexible Neural Trees Ensemble for Stock Index Modeling", *Neurocomputing*, vol. 70–4, January 2007, pp. 697–703.

[43] Chen, Y.; Zhao, Y. "A novel Ensemble of Classifiers for Microarray Data Classification", *Applied Soft Computing*, vol. 8–4, September 2008, pp. 1664–1669.

[44] Chiu, C.-Y.; Verma, B. "Effect of Varying Hidden Neurons and Data Size on Clusters, Layers, Diversity and Accuracy in Neural Ensemble Classifier". In: International Conference on Computational Science and Engineering, 2013, pp. 455–459.

[45] Chiu, C.-Y.; Verma, B. "Multi-objective Evolutionary Algorithm Based Optimization of Neural Network Ensemble Classifier". In: International Conference on Signal Processing and Communication Systems, 2014, pp. 1–5.

[46] Chyzhyk, D.; Savio, A.; Graña, M. "Computer Aided Diagnosis of Schizophrenia on Resting State fMRI data by Ensembles of ELM", *Neural Networks*, vol. 68, August 2015, pp. 23–33.

[47] Clark, P.; Boswell, R. "Rule induction with CN2: Some recent improvements". In: European Working Session on Learning, 1991, pp. 151–163.

[48] Coelho, A. L.; Lima, C. A.; Von Zuben, F. "GA-based Selection of Components for Heterogeneous Ensembles of Support Vector Machines". In: Congress on Evolutionary Computation, 2003, pp. 2238–2245.

[49] Coello Coello, C. A. "MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization". In: Congress on Evolutionary Computation, 2002, pp. 1051–1056.

[50] Cohen, W. W. "Fast Effective Rule Induction". In: International Conference on Machine Learning, 1995, pp. 115–123.

[51] Connolly, J.-F.; Granger, E.; Sabourin, R. “Comparing Dynamic PSO Algorithms for Adapting Classifier Ensembles in Video-based Face Recognition”. In: Workshop on Computational Intelligence in Biometrics and Identity Management, 2011, pp. 1–8.

[52] Connolly, J.-F.; Granger, E.; Sabourin, R. “Evolution of Heterogeneous Ensembles through Dynamic Particle Swarm Optimization for Video-based Face Recognition”, *Pattern Recognition*, vol. 45–7, July 2012, pp. 2460–2477.

[53] Connolly, J.-F.; Granger, E.; Sabourin, R. “Dynamic Multi-objective Evolution of Classifier Ensembles for Video Face Recognition”, *Applied Soft Computing*, vol. 13–6, June 2013, pp. 3149–3166.

[54] Cordón, O.; Trawiski, K. “A Novel Framework to Design Fuzzy Rule-based Ensembles using Diversity Induction and Evolutionary Algorithms-based Classifier Selection and Fusion”. In: International Work-Conference on Artificial Neural Networks, 2013, pp. 36–58.

[55] Cruz, R. M.; Sabourin, R.; Cavalcanti, G. D. “Dynamic Classifier Selection: Recent Advances and Perspectives”, *Information Fusion*, vol. 41, May 2018, pp. 195–216.

[56] Das, A. K.; Das, S.; Ghosh, A. “Ensemble Feature Selection using Bi-objective Genetic Algorithm”, *Knowledge-Based Systems*, vol. 123, May 2017, pp. 116–127.

[57] Davidsen, S. A.; Padmavathamma, M. “Multi-modal Evolutionary Ensemble Classification in Medical Diagnosis Problems”. In: International Conference on Advances in Computing, Communications and Informatics, 2015, pp. 1366–1370.

[58] de Lima, T. P.; Ludermir, T. B. “Ensembles of Evolutionary Extreme Learning Machines through Differential Evolution and Fitness Sharing”. In: International Joint Conference on Neural Networks, 2014, pp. 2677–2682.

[59] De Lima, T. P. F.; Ludermir, T. B. “Optimizing Dynamic Ensemble Selection Procedure by Evolutionary Extreme Learning Machines and a Noise Reduction Filter”. In: International Conference on Tools with Artificial Intelligence, 2013, pp. 546–552.

[60] de Lima, T. P. F.; Sergio, A. T.; Ludermir, T. B. “Improving Classifiers and Regions of Competence in Dynamic Ensemble Selection”. In: Brazilian Conference on Intelligent Systems, 2014, pp. 13–18.

[61] de Oliveira Batista, J.; Rodrigues, R. B.; Varejão, F. M. “Soft Computing Classifier Ensemble for Fault Diagnosis”. In: International Symposium on Industrial Electronics, 2017, pp. 1348–1353.

[62] de Sa, A. G.; Pinto, W. J. G.; Oliveira, L. O. V.; Pappa, G. L. “RECIPE: a Grammar-based Framework for Automatically Evolving Classification Pipelines”. In: European Conference on Genetic Programming, 2017, pp. 246–261.

[63] De Stefano, C.; Cioppa, A. D.; Marcelli, A. “Evolutionary Approaches for Pooling Classifier Ensembles: Performance Evaluation”. In: International Conference of Soft Computing and Pattern Recognition, 2013, pp. 309–314.

[64] De Stefano, C.; Folino, G.; Fontanella, F.; Di Freca, A. S. “Using Bayesian Networks for Selecting Classifiers in GP Ensembles”, *Information Sciences*, vol. 258, February 2014, pp. 200–216.

[65] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II”, *IEEE Transactions on Evolutionary Computation*, vol. 6–2, April 2002, pp. 182–197.

[66] Debie, E.; Shafi, K.; Lokan, C.; Merrick, K. “Performance Analysis of Rough set Ensemble of Learning Classifier Systems with Differential Evolution based rule Discovery”, *Evolutionary Intelligence*, vol. 6–2, October 2013, pp. 109–126.

[67] Debie, E.; Shafi, K.; Merrick, K.; Lokan, C. “On Taxonomy and Evaluation of Feature Selection-Based Learning Classifier System Ensemble Approaches for Data Mining Problems”, *Computational Intelligence*, vol. 33–3, July 2016, pp. 554–578.

[68] Debie, E. S.; Shafi, K.; Lokan, C. “REUCS-CRG: Reduct based Ensemble of Supervised Classifier System with Combinatorial Rule Generation for Data Mining”. In: Conference on Genetic and Evolutionary Computation, 2013, pp. 1251–1258.

[69] Dehuri, S.; Jagadev, A. K.; Cho, S.-B. “Epileptic Seizure Identification from Electroencephalography signal using DE-RBFNs Ensemble”, *Procedia Computer Science*, vol. 23, November 2013, pp. 84–95.

[70] Demšar, J. “Statistical comparisons of classifiers over multiple data sets”, *Journal of Machine Learning Research*, vol. 7, June 2006, pp. 1–30.

[71] Dietterich, T. G. “Ensemble Methods in Machine Learning”. In: International Workshop on Multiple Classifier Systems, 2000, pp. 1–15.

[72] Dos Santos, E. M.; Oliveira, L. S.; Sabourin, R.; Maupin, P. “Overfitting in the Selection of Classifier Ensembles: a Comparative study Between PSO and GA”. In: Conference on Genetic and Evolutionary Computation, 2008, pp. 1423–1424.

[73] Dos Santos, E. M.; Sabourin, R.; Maupin, P. “Pareto Analysis for the Selection of Classifier Ensembles”. In: Conference on Genetic and Evolutionary Computation, 2008, pp. 681–688.

[74] Duell, P.; Fermin, I.; Yao, X. “Speciation Techniques in Evolved Ensembles with Negative Correlation Learning”. In: Congress on Evolutionary Computation, 2006, pp. 3317–3321.

[75] Dufourq, E.; Pillay, N. “Hybridizing Evolutionary Algorithms for Creating Classifier Ensembles”. In: World Congress on Nature and Biologically Inspired Computing, 2014, pp. 84–90.

[76] e Silva, E. J. d. R.; Ludermir, T. B.; Almeida, L. M. “Clustering and Selection using Grouping Genetic Algorithms for Blockmodeling to construct Neural Network Ensembles”. In: International Conference on Tools with Artificial Intelligence, 2013, pp. 420–425.

[77] Eberhart, R.; Kennedy, J. “A new Optimizer Using Particle Swarm Theory”. In: International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.

[78] Escalante, H. J.; Acosta-Mendoza, N.; Morales-Reyes, A.; Gago-Alonso, A. “Genetic Programming of Heterogeneous Ensembles for Classification”. In: Iberoamerican Congress on Pattern Recognition, 2013, pp. 9–16.

[79] Escovedo, T.; da Cruz, A.; Vellasco, M.; Koshiyama, A. “NEVE: A Neuro-evolutionary Ensemble for Adaptive Learning”. In: International Conference on Artificial Intelligence Applications and Innovations, 2013, pp. 636–645.

[80] Escovedo, T.; da Cruz, A. A.; Koshiyama, A.; Melo, R.; Vellasco, M. “NEVE++: A Neuro-evolutionary Unlimited Ensemble for Adaptive Learning”. In: International Joint Conference on Neural Networks, 2014, pp. 3331–3338.

[81] Escovedo, T.; da Cruz, A. V. A.; Vellasco, M.; Koshiyama, A. S. “Using Ensembles for Adaptive Learning: A Comparative Approach”. In: International Joint Conference on Neural Networks, 2013, pp. 1–7.

[82] Escovedo, T.; da Cruz, A. V. A.; Vellasco, M. M.; Koshiyama, A. S. “Learning Under Concept Drift Using a Neuro-evolutionary Ensemble”, *International Journal of Computational Intelligence and Applications*, vol. 12–4, December 2013, pp. 1340002.

[83] Espejo, P. G.; Ventura, S.; Herrera, F. “A survey on the application of genetic programming to classification”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40–2, November 2010, pp. 121–144.

[84] Fatima, I.; Fahim, M.; Lee, Y.-K.; Lee, S. “Classifier Ensemble Optimization for Human Activity Recognition in Smart Homes”. In: International Conference on Ubiquitous Information Management and Communication, 2013, pp. 1 – 7.

[85] Fawcett, T. “An introduction to ROC analysis”, *Pattern recognition letters*, vol. 27–8, June 2006, pp. 861–874.

[86] Fernández, A.; del Río, S.; Herrera, F. “A First Approach in Evolutionary Fuzzy Systems based on the Lateral Tuning of the Linguistic Labels for Big Data Classification”. In: International Conference on Fuzzy Systems, 2016, pp. 1437–1444.

[87] Fernández, J. C.; Cruz-Ramírez, M.; Hervás-Martínez, C. “Sensitivity versus Accuracy in Ensemble Models of Artificial Neural Networks from Multi-objective Evolutionary Algorithms”, *Neural Computing and Applications*, vol. 30–1, December 2016, pp. 289–305.

[88] Fernández-Delgado, M.; Cernadas, E.; Barro, S.; Amorim, D. "Do we need hundreds of classifiers to solve real world classification problems?", *Journal of Machine Learning Research*, vol. 15–1, October 2014, pp. 3133–3181.

[89] Ferri, C.; Hernández-Orallo, J.; Ramírez-Quintana, M. J. "From Ensemble Methods to Comprehensible Models". In: International Conference on Discovery Science, 2002, pp. 165–177.

[90] Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; Hutter, F. "Efficient and robust automated machine learning". In: Advances in Neural Information Processing Systems, 2015, pp. 2962–2970.

[91] Folino, G.; Pisani, F. S.; Sabatino, P. "An Incremental Ensemble Evolved by using Genetic Programming to Efficiently Detect drifts in Cyber Security Datasets". In: Conference on Genetic and Evolutionary Computation, 2016, pp. 1103–1110.

[92] Folino, G.; Pizzuti, C.; Spezzano, G. "Improving Cooperative GP Ensemble with Clustering and Pruning for Pattern Classification". In: Conference on Genetic and Evolutionary Computation, 2006, pp. 791–798.

[93] Folino, G.; Pizzuti, C.; Spezzano, G. "An Adaptive Distributed Ensemble Approach to Mine Concept-drifting Data Streams". In: International Conference on Tools with Artificial Intelligence, 2007, pp. 183–188.

[94] Folino, G.; Pizzuti, C.; Spezzano, G. "StreamGP: Tracking Evolving GP Ensembles in Distributed Data Streams using Fractal Dimension". In: Conference on Genetic and Evolutionary Computation, 2007, pp. 1751–1751.

[95] Folino, G.; Pizzuti, C.; Spezzano, G. "An Ensemble-based Evolutionary Framework for Coping with Distributed Intrusion Detection", *Genetic Programming and Evolvable Machines*, vol. 11–2, February 2010, pp. 131–146.

[96] Frank, E.; Witten, I. H. "Generating Accurate Rule Sets Without Global Optimization". In: International Conference on Machine Learning, 1998, pp. 144–151.

[97] Freitas, A. A. "A Critical Review of Multi-objective Optimization in Data Mining: A Position Paper", *ACM SIGKDD Explorations Newsletter*, vol. 6–2, December 2004, pp. 77–86.

[98] Freitas, A. A. "Comprehensible Classification Models: A Position Paper", *ACM SIGKDD Explorations Newsletter*, vol. 15–1, June 2014, pp. 1–10.

[99] Freund, Y.; Schapire, R. E. "A Desicion-theoretic Generalization of on-line Learning and an Application to Boosting". In: European Conference on Computational Learning Theory, 1995, pp. 23–37.

[100] Freund, Y.; Schapire, R. E. "Experiments with a new Boosting Algorithm". In: International Conference on Machine Learning, 1996, pp. 148–156.

[101] Friedman, M. "The use of ranks to avoid the assumption of normality implicit in the analysis of variance", *Journal of the American Statistical Association*, vol. 32–200, May 1937, pp. 675–701.

[102] Fuqiang, D.; Minqing, Z.; Jia, L. "Virus-Evolutionary Genetic Algorithm Based Selective Ensemble for Steganalysis". In: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014, pp. 553–558.

[103] Fürnkranz, J.; Kliegr, T.; Paulheim, H. "On cognitive preferences and the plausibility of rule-based models", *Machine Learning*, vol. 109–4, December 2020, pp. 853–898.

[104] Galar, M.; Fernández, A.; Barrenechea, E.; Herrera, F. "EUSBoost: Enhancing Ensembles for Highly Imbalanced Data-sets by Evolutionary Undersampling", *Pattern Recognition*, vol. 46–12, December 2013, pp. 3460–3471.

[105] Galea, M.; Shen, Q.; Levine, J. "Evolutionary Approaches to Fuzzy Modelling for Classification", *The Knowledge Engineering Review*, vol. 19–1, April 2004, pp. 27–59.

[106] Gámez, J. A.; Mateo, J. L.; Puerta, J. M. "EDNA: Estimation of Dependency Networks Algorithm". In: International Work-Conference on the Interplay Between Natural and Artificial Computation, 2007, pp. 427–436.

[107] Gámez, J. A.; Mateo, J. L.; Puerta, J. M. "Improved EDNA (estimation of dependency networks algorithm) using combining function with bivariate probability distributions". In: Conference on Genetic and Evolutionary Computation, 2008, pp. 407–414.

[108] García, S.; Fernández, A.; Luengo, J.; Herrera, F. "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power", *Information sciences*, vol. 180–10, May 2010, pp. 2044–2064.

[109] Garg, A.; Lam, J. S. L. "Improving Environmental Sustainability by Formulation of Generalized Power Consumption Models using an Ensemble based Multi-gene Genetic Programming Approach", *Journal of Cleaner Production*, vol. 102, September 2015, pp. 246–263.

[110] Gomes, H. M.; Barddal, J. P.; Enembreck, F.; Bifet, A. "A Survey on Ensemble Learning for Data Stream Classification", *ACM Computing Surveys*, vol. 50–2, March 2017, pp. 23.

[111] Goodfellow, I.; Bengio, Y.; Courville, A. "Deep learning". MIT press, 2016, 800p.

[112] Gu, S.; Jin, Y. "Generating Diverse and Accurate Classifier Ensembles using Multi-objective Optimization". In: Symposium on Computational Intelligence in Multi-Criteria Decision-Making, 2014, pp. 9–15.

[113] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H. "The WEKA data mining software: an update", *ACM SIGKDD explorations newsletter*, vol. 11–1, June 2009, pp. 10–18.

[114] Hansen, L. K.; Salamon, P. "Neural Network Ensembles", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12–10, October 1990, pp. 993–1001.

[115] Haque, M. N.; Noman, M. N.; Berretta, R.; Moscato, P. "Optimising Weights for Heterogeneous Ensemble of Classifiers with Differential Evolution". In: Congress on Evolutionary Computation, 2016, pp. 233–240.

[116] Hashem, S. "Optimal Linear Combinations of Neural Networks", *Neural Networks*, vol. 10–4, June 1997, pp. 599–614.

[117] Hauschild, M.; Pelikan, M. "An Introduction and Survey of Estimation of Distribution Algorithms", *Swarm and Evolutionary Computation*, vol. 1–3, September 2011, pp. 111–128.

[118] He, B. D.; De Sa, C. M.; Mitliagkas, I.; Ré, C. "Scan order in Gibbs sampling: Models in which it matters and bounds on how much". In: Advances in Neural Information Processing Systems, 2016, pp. 1–9.

[119] Heckerman, D.; Chickering, D. M.; Meek, C.; Rounthwaite, R.; Kadie, C. "Dependency networks for inference, collaborative filtering, and data visualization", *Journal of Machine Learning Research*, vol. 1, October 2000, pp. 49–75.

[120] Hernández, L. C.; Hernández, A. M.; Cardoso, G. M. C.; Jiménez, Y. M. "Genetic Algorithms with Diversity Measures to build Classifier Systems", *Investigación Operacional*, vol. 36–3, September 2015, pp. 206–225.

[121] Hodges, J.; Lehmann, E. "Rank Methods for Combination of Independent Experiments in Analysis of Variance", *The Annals of Mathematical Statistics*, vol. 33–2, November 1962, pp. 482–497.

[122] Holland, J. H. "Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence". MIT press, 1992, 232p.

[123] Huysmans, J.; Dejaeger, K.; Mues, C.; Vanthienen, J.; Baesens, B. "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models", *Decision Support Systems*, vol. 51–1, April 2011, pp. 141–154.

[124] Ishibuchi, H.; Yamamoto, T. "Evolutionary multiobjective optimization for generating an ensemble of fuzzy rule-based classifiers". In: Conference on Genetic and Evolutionary Computation, 2003, pp. 197–197.

[125] Jackowski, K. “Fixed-size Ensemble Classifier System Evolutionarily Adapted to a Recurring Context with an Unlimited Pool of Classifiers”, *Pattern Analysis and Applications*, vol. 17–4, November 2014, pp. 709–724.

[126] Jackowski, K. “Adaptive Splitting and Selection Algorithm for Regression”, *New Generation Computing*, vol. 33–4, October 2015, pp. 425–448.

[127] Jackowski, K.; Krawczyk, B.; Woniak, M. “Improved Adaptive Splitting and Selection: The Hybrid Training Method of a Classifier based on a Feature Space Partitioning”, *International Journal of Neural Systems*, vol. 24–3, January 2014, pp. 1430007.

[128] Joardar, S.; Chatterjee, A.; Bandyopadhyay, S.; Maulik, U. “Multi-size Patch based Collaborative Representation for Palm Dorsa Vein Pattern Recognition by Enhanced Ensemble Learning with Modified Interactive Artificial Bee Colony Algorithm”, *Engineering Applications of Artificial Intelligence*, vol. 60, April 2017, pp. 151–163.

[129] Kaiping, L.; Binglian, C.; Yan, D.; Ying, H. “A Genetic Neural Network Ensemble Prediction Model based on Locally Linear Embedding for Typhoon Intensity”. In: Conference on Industrial Electronics and Applications, 2013, pp. 137–142.

[130] Kapp, M. N.; Sabourin, R.; Maupin, P. “Adaptive Incremental Learning with an Ensemble of Support Vector Machines”. In: International Conference on Pattern Recognition, 2010, pp. 4048–4051.

[131] Kapp, M. N.; Sabourin, R.; Maupin, P. “A Dynamic Optimization Approach for Adaptive Incremental Learning”, *International Journal of Intelligent Systems*, vol. 26–11, July 2011, pp. 1101–1124.

[132] Karakati, S.; Heriko, M.; Podgorelec, V. “Weighting and Sampling data for Individual Classifiers and Bagging with Genetic Algorithms”. In: International Joint Conference on Computational Intelligence, 2015, pp. 180–187.

[133] Kennedy, J.; Eberhart, R. “Particle swarm optimization”. In: International Conference on Neural Networks, 1995, pp. 1942–1948.

[134] Khamis, A.; Xu, Y.; Dong, Z. Y.; Zhang, R. “Faster Detection of Microgrid Islanding Events using an Adaptive Ensemble Classifier”, *IEEE Transactions on Smart Grid*, vol. 9–3, August 2016, pp. 1889–1899.

[135] Kim, K.-J.; Cho, S.-B. “DNA Gene Expression Classification with Ensemble Classifiers Optimized by Speciated Genetic Algorithm”, *Pattern Recognition and Machine Intelligence*, vol. 3776, December 2005, pp. 649–653.

[136] Kim, K.-J.; Cho, S.-B. “An Evolutionary Algorithm Approach to Optimal Ensemble Classifiers for DNA Microarray Data Analysis”, *IEEE Transactions on Evolutionary Computation*, vol. 12–3, June 2008, pp. 377–388.

[137] Kim, K.-J.; Cho, S.-B. “Evolutionary Ensemble of Diverse Artificial Neural Networks using Speciation”, *Neurocomputing*, vol. 71–7, March 2008, pp. 1604–1618.

[138] Kim, K.-J.; Cho, S.-B. “Meta-classifiers for High-dimensional, Small Sample Classification for Gene Expression Analysis”, *Pattern Analysis and Applications*, vol. 18–3, May 2015, pp. 553–569.

[139] Kim, Y.; Street, W. N.; Menczer, F. “Meta-evolutionary Ensembles”. In: International Joint Conference on Neural Networks, 2002, pp. 2791–2796.

[140] Kiranyaz, S.; Ince, T.; Zabihi, M.; Ince, D. “Automated Patient-specific Classification of Long-term Electroencephalography”, *Journal of Biomedical Informatics*, vol. 49, June 2014, pp. 16–31.

[141] Kittler, J.; Hatef, M.; Duin, R. P.; Matas, J. “On combining classifiers”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20–3, March 1998, pp. 226–239.

[142] Ko, A. H.-R.; Sabourin, R.; Britto Jr, A. d. S. “Evolving Ensemble of Classifiers in Random Subspace”. In: Conference on Genetic and Evolutionary Computation, 2006, pp. 1473–1480.

[143] Kohavi, R. “The Power of Decision Tables”. In: European Conference on Machine Learning, 1995, pp. 174–189.

[144] Kordik, P.; Cerny, J.; Fryda, T. “Discovering Predictive Ensembles for Transfer Learning and Meta-learning”, *Machine Learning*, vol. 107, December 2018, pp. 177–207.

[145] Kotsiantis, S. B. “Bagging and Boosting Variants for Handling Classifications Problems: A Survey”, *The Knowledge Engineering Review*, vol. 29–1, August 2014, pp. 78–100.

[146] Krawczyk, B.; Galar, M.; Jele, Ł.; Herrera, F. “Evolutionary Undersampling Boosting for Imbalanced Classification of Breast Cancer Malignancy”, *Applied Soft Computing*, vol. 38, January 2016, pp. 714–726.

[147] Krawczyk, B.; Minku, L. L.; Gama, J.; Stefanowski, J.; Woniak, M. “Ensemble Learning for Data Stream Analysis: A Survey”, *Information Fusion*, vol. 37, September 2017, pp. 132–156.

[148] Krawczyk, B.; Schaefer, G. “Breast Thermogram Analysis using Classifier Ensembles and Image Symmetry Features”, *IEEE Systems Journal*, vol. 8–3, October 2014, pp. 921–928.

[149] Krawczyk, B.; Schaefer, G.; Woniak, M. “A Cost-sensitive Ensemble Classifier for Breast Cancer Classification”. In: International Symposium on Applied Computational Intelligence and Informatics, 2013, pp. 427–430.

[150] Krawczyk, B.; Schaefer, G.; Woniak, M. “A hybrid Cost-sensitive Ensemble for Imbalanced Breast Thermogram Classification”, *Artificial Intelligence in Medicine*, vol. 65–3, November 2015, pp. 219–227.

[151] Krawczyk, B.; Woniak, M. “Evolutionary Cost-sensitive Ensemble for Malware Detection”. In: SOCO/CISIS/ICEUTE, 2014, pp. 433–442.

[152] Krawczyk, B.; Woniak, M.; Schaefer, G. “Cost-sensitive Decision Tree Ensembles for Effective Imbalanced Classification”, *Applied Soft Computing*, vol. 14, January 2014, pp. 554–562.

[153] Krithikaa, M.; Mallipeddi, R. “Differential Evolution with an Ensemble of Low-quality Surrogates for Expensive Optimization Problems”. In: Congress on Evolutionary Computation, 2016, pp. 78–85.

[154] Krogh, A.; Vedelsby, J. “Neural Network Ensembles, Cross Validation, and Active Learning”, *Advances in Neural Information Processing Systems*, vol. 7, January 1995, pp. 231–238.

[155] Kumar, G.; Kumar, K. “Design of an Evolutionary Approach for Intrusion Detection”, *The Scientific World Journal*, vol. 2013, November 2013, pp. 1–14.

[156] Kuncheva, L. I. “Combining pattern classifiers: methods and algorithms”. John Wiley & Sons, 2004, 361p.

[157] Kuncheva, L. I.; Whitaker, C. J. “Measures of Diversity in Classifier Ensembles and their Relationship with the Ensemble Accuracy”, *Machine learning*, vol. 51–2, May 2003, pp. 181–207.

[158] Lacy, S. E.; Lones, M. A.; Smith, S. L. “A Comparison of Evolved Linear and Non-linear Ensemble Vote Aggregators”. In: Congress on Evolutionary Computation, 2015, pp. 758–763.

[159] Lacy, S. E.; Lones, M. A.; Smith, S. L. “Forming Classifier Ensembles with Multimodal Evolutionary Algorithms”. In: Congress on Evolutionary Computation, 2015, pp. 723–729.

[160] Lapuschkin, S.; Wäldchen, S.; Binder, A.; Montavon, G.; Samek, W.; Müller, K.-R. “Unmasking Clever Hans predictors and assessing what machines really learn”, *Nature Communications*, vol. 10–1, March 2019, pp. 1096.

[161] Larcher, C.; Barbosa, H. “Auto-CVE: a coevolutionary approach to evolve ensembles in automated machine learning”. In: Genetic and Evolutionary Computation Conference, 2019, pp. 392–400.

[162] Larcher, C. H.; Barbosa, H. J. “Evaluating Models with Dynamic Sampling Holdout”. In: International Conference on the Applications of Evolutionary Computation, 2021, pp. 729–744.

[163] Larrañaga, P.; Lozano, J. A. “Estimation of Distribution Algorithms: A new Tool for Evolutionary Computation”. Springer, 2001, 382p.

[164] Lévesque, J.-C.; Durand, A.; Gagné, C.; Sabourin, R. “Multi-objective Evolutionary Optimization for Generating Ensembles of Classifiers in the ROC Space”. In: Conference on Genetic and Evolutionary Computation, 2012, pp. 879–886.

[165] Lichman, M. "UCI Machine Learning Repository". Source: http://archive.ics.uci.edu/ml, March 28, 2017.

[166] Liew, W. S.; Loo, C. K.; Obo, T. "Optimizing FELM ensembles using GA-BIC". In: Joint World Congress of International Fuzzy Systems Association and International Conference on Soft Computing and Intelligent Systems, 2017, pp. 1–6.

[167] Lima, T. P.; Ludermir, T. B. "Differential Evolution and Meta-learning for Dynamic Ensemble of Neural Network Classifiers". In: International Joint Conference on Neural Networks, 2015, pp. 1–5.

[168] Link, W. A.; Eaton, M. J. "On thinning of chains in MCMC", *Methods in Ecology and Evolution*, vol. 3–1, June 2012, pp. 112–115.

[169] Liu, K.; Tong, M.; Xie, S.; Zeng, Z. "Fusing Decision Trees based on Genetic Programming for Classification of Microarray Datasets". In: International Conference on Intelligent Computing, 2014, pp. 126–134.

[170] Liu, K.-H.; Huang, D.-S.; Zhang, J. "Microarray Data Prediction by Evolutionary Classifier Ensemble System". In: Congress on Evolutionary Computation, 2007, pp. 634–637.

[171] Liu, K.-H.; Li, B.; Zhang, J.; Du, J.-X. "Ensemble Component Selection for Improving ICA based Microarray Data Prediction Models", *Pattern Recognition*, vol. 42–7, July 2009, pp. 1274–1283.

[172] Liu, K.-H.; Tong, M.; Xie, S.-T.; Yee Ng, V. T. "Genetic Programming based Ensemble System for Microarray Data Classification", *Computational and Mathematical Methods in Medicine*, vol. 2015, February 2015, pp. 1–11.

[173] Liu, N.; Cao, J.; Lin, Z.; Pek, P. P.; Koh, Z. X.; Ong, M. E. H. "Evolutionary Voting-based Extreme Learning Machines", *Mathematical Problems in Engineering*, vol. 2014, August 2014, pp. 1–7.

[174] Liu, Y.; Chen, W.; Hu, J.; Zheng, X.; Shi, Y. "Ensemble of Surrogates with an Evolutionary Multi-agent System". In: International Conference on Computer Supported Cooperative Work in Design, 2017, pp. 521–525.

[175] Lones, M. A.; Smith, S. L.; Alty, J. E.; Lacy, S. E.; Possin, K. L.; Jamieson, D. S.; Tyrrell, A. M. "Evolving Classifiers to Recognize the Movement Characteristics of Parkinson's Disease Patients", *IEEE Transactions on Evolutionary Computation*, vol. 18–4, August 2014, pp. 559–576.

[176] Lundberg, S. M.; Lee, S.-I. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*, Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; Garnett, R. (Editors), Curran Associates, Inc., 2017, pp. 4765–4774.

[177] Ma, N.; Fujita, H.; Zhai, Y.; Wang, S. "Ensembles of Fuzzy Cognitive Map Classifiers Based on Quantum Computation", *Acta Polytechnica Hungarica*, vol. 12–4, 2015, pp. 7–26.

[178] Mabu, S.; Obayashi, M.; Kuremoto, T. “Ensemble Learning of Rule-based Evolutionary Algorithm using Multi Layer Perceptron for Stock Trading Models”. In: Joint International Conference on Soft Computing and Intelligent Systems and International Symposium on Advanced Intelligent Systems, 2014, pp. 624–629.

[179] Mabu, S.; Obayashi, M.; Kuremoto, T. “Ensemble Learning of Rule-based Evolutionary Algorithm using Multi-layer Perceptron for Supporting Decisions in Stock Trading Problems”, *Applied Soft Computing*, vol. 36, November 2015, pp. 357–367.

[180] Mauša, G.; Grbac, T. G. “Co-evolutionary Multi-population Genetic Programming for Classification in Software Defect Prediction: An Empirical Case Study”, *Applied Soft Computing*, vol. 55, June 2017, pp. 331–351.

[181] Mehdiyev, N.; Krumeich, J.; Werth, D.; Loos, P. “Sensor Event Mining with Hybrid Ensemble Learning and Evolutionary Feature Subset Selection Model”. In: International Conference on Big Data, 2015, pp. 2159–2168.

[182] Mendes-Moreira, J.; Soares, C.; Jorge, A. M.; Sousa, J. F. D. “Ensemble Approaches for Regression: A Survey”, *ACM Computing Surveys*, vol. 45–1, November 2012, pp. 10:1–10:40.

[183] Mezura-Montes, E.; Reyes-Sierra, M.; Coello, C. “Multi-objective Optimization Using Differential Evolution: A Survey of the State-of-the-art”. In: *Advances in Differential Evolution*, Chakraborty, U. K. (Editor), Springer, 2008, pp. 173–196.

[184] Milliken, M.; Bi, Y.; Galway, L.; Hawe, G. “Multi-objective Optimization of Base Classifiers in StackingC by NSGA-II for Intrusion Detection”. In: Symposium Series on Computational Intelligence, 2016, pp. 1–8.

[185] Nemenyi, P. B. “Distribution-free multiple comparisons.” Princeton University, 1963, 24p.

[186] Neoh, S. C.; Zhang, L.; Mistry, K.; Hossain, M. A.; Lim, C. P.; Aslam, N.; Kinghorn, P. “Intelligent Facial Emotion Recognition using a Layered Encoding Cascade Optimization Model”, *Applied Soft Computing*, vol. 34, September 2015, pp. 72–93.

[187] Nguyen, V.; Epps, J.; Bailey, J. “Information theoretic measures for clusterings comparison: is a correction for chance necessary?” In: International Conference on Machine Learning, 2009, pp. 1073–1080.

[188] Obo, T.; Kubota, N.; Loo, C. K. “Evolutionary Ensemble Learning of Fuzzy Randomized Neural Network for Posture Recognition”. In: World Automation Congress, 2016, pp. 1–6.

[189] Oehmcke, S.; Heinermann, J.; Kramer, O. “Analysis of Diversity Methods for Evolutionary Multi-objective Ensemble Classifiers”. In: European Conference on the Applications of Evolutionary Computation, 2015, pp. 567–578.

[190] Ojha, V. K.; Abraham, A.; Snášel, V. “Ensemble of Heterogeneous Flexible Neural Trees using Multiobjective Genetic Programming”, *Applied Soft Computing*, vol. 52, March 2017, pp. 909–924.

[191] Ojha, V. K.; Jackowski, K.; Abraham, A.; Snášel, V. “Feature Selection and Ensemble of Regression Models for Predicting the Protein Macromolecule Dissolution Profile”. In: World Congress on Nature and Biologically Inspired Computing, 2014, pp. 121–126.

[192] Ojha, V. K.; Jackowski, K.; Abraham, A.; Snášel, V. “Dimensionality Reduction, and Function Approximation of Poly (Lactic-co-glycolic acid) Micro-and Nanoparticle Dissolution Rate”, *International Journal of Nanomedicine*, vol. 10, February 2015, pp. 1119.

[193] Olson, R. S.; Bartley, N.; Urbanowicz, R. J.; Moore, J. H. “Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science”. In: Conference on Genetic and Evolutionary Computation, 2016, pp. 485–492.

[194] Olson, R. S.; Urbanowicz, R. J.; Andrews, P. C.; Lavender, N. A.; Kidd, L. C.; Moore, J. H. “Automating biomedical data science through tree-based pipeline optimization”. In: European Conference on the Applications of Evolutionary Computation, 2016, pp. 123–137.

[195] Olvera-López, J. A.; Carrasco-Ochoa, J. A.; Martínez-Trinidad, J.; Kittler, J. “A Review of Instance Selection Methods”, *Artificial Intelligence Review*, vol. 34–2, May 2010, pp. 133–143.

[196] Onan, A.; Korukolu, S.; Bulut, H. “A Multiobjective Weighted Voting Ensemble Classifier based on Differential Evolution Algorithm for Text Sentiment Classification”, *Expert Systems with Applications*, vol. 62, November 2016, pp. 1–16.

[197] Opitz, D.; Maclin, R. “Popular Ensemble Methods: An Empirical Study”, *Journal of Artificial Intelligence Research*, vol. 11, August 1999, pp. 169–198.

[198] Opitz, D. W. “Feature Selection for Ensembles”. In: National Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence Conference, 1999, pp. 384.

[199] Oza, N. C.; Tumer, K. “Classifier Ensembles: Select Real-world Applications”, *Information Fusion*, vol. 9–1, January 2008, pp. 4–20.

[200] Pagano, C.; Granger, E.; Sabourin, R.; Gorodnichy, D. O. “Detector Ensembles for Face Recognition in Video Surveillance”. In: International Joint Conference on Neural Networks, 2012, pp. 1–8.

[201] Parhizkar, E.; Abadi, M. “BeeOWA: A Novel Approach based on ABC Algorithm and Induced OWA operators for Constructing One-class Classifier Ensembles”, *Neurocomputing*, vol. 166, October 2015, pp. 367–381.

[202] Parhizkar, E.; Abadi, M. “OC-WAD: A One-class Classifier Ensemble Approach for Anomaly Detection in Web Traffic”. In: Iranian Conference on Electrical Engineering, 2015, pp. 631–636.

[203] Park, C.; Cho, S.-B. “Evolutionary Computation for Optimal Ensemble Classifier in Lymphoma Cancer Classification”, *Foundations of Intelligent Systems*, vol. 2871, October 2003, pp. 521–530.

[204] Park, C.; Cho, S.-B. “Evolutionary Ensemble Classifier for Lymphoma and Colon Cancer Classification”. In: Congress on Evolutionary Computation, 2003, pp. 2378–2385.

[205] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al.. “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, October 2011, pp. 2825–2830.

[206] Peimankar, A.; Weddell, S. J.; Jalal, T.; Lapthorn, A. C. “Ensemble Classifier Selection using Multi-objective PSO for Fault Diagnosis of Power Transformers”. In: Congress on Evolutionary Computation, 2016, pp. 3622–3629.

[207] Peimankar, A.; Weddell, S. J.; Jalal, T.; Lapthorn, A. C. “Evolutionary Multi-Objective Fault Diagnosis of Power Transformers”, *Swarm and Evolutionary Computation*, vol. 36, October 2017, pp. 62–75.

[208] Pelikan, M.; Sastry, K.; Goldberg, D. E. “Multiobjective hBOA, clustering, and scalability”. In: Conference on Genetic and Evolutionary Computation, 2005, pp. 663–670.

[209] Piltaver, R.; Luštrek, M.; Gams, M.; Martini-Ipši, S. “What makes classification trees comprehensible?”, *Expert Systems with Applications*, vol. 62, November 2016, pp. 333–346.

[210] Pourtaheri, Z. K.; Zahiri, S. H. “Ensemble Classifiers with Improved Overfitting”. In: Conference on Swarm Intelligence and Evolutionary Computation, 2016, pp. 93–97.

[211] Priyam, A.; Abhijeeta, G.; Rathee, A.; Srivastava, S. “Comparative analysis of decision tree classification algorithms”, *International Journal of Current Engineering and Technology*, vol. 3–2, June 2013, pp. 334–337.

[212] Probst, P.; Wright, M. N.; Boulesteix, A.-L. “Hyperparameters and tuning strategies for random forest”, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9–3, January 2019, pp. e1301.

[213] Quinlan, J. R. “C4.5: programs for machine learning”. Morgan Kaufmann Publishers, 1993, 301p.

[214] Rahman, A.; Verma, B. “Cluster Based Ensemble Classifier Generation by Joint Optimization of Accuracy and Diversity”, *International Journal of Computational Intelligence and Applications*, vol. 12–4, December 2013, pp. 1340003.

[215] Rahman, A.; Verma, B. “Cluster Oriented Ensemble Classifiers using Multi-objective Evolutionary Algorithm”. In: International Joint Conference on Neural Networks, 2013, pp. 1–6.

[216] Rahman, A.; Verma, B. “Ensemble Classifier Generation using Non-uniform Layered Clustering and Genetic Algorithm”, *Knowledge-Based Systems*, vol. 43, May 2013, pp. 30–42.

[217] Rapakoulia, T.; Theofilatos, K.; Kleftogiannis, D.; Likothanasis, S.; Tsakalidis, A.; Mavroudi, S. “EnsembleGASVR: a Novel Ensemble Method for Classifying Missense Single Nucleotide Polymorphisms”, *Bioinformatics*, vol. 30–16, August 2014, pp. 2324–2333.

[218] Redel-Macías, M. D.; Fernández-Navarro, F.; Gutiérrez, P. A.; Cubero-Atienza, A. J.; Hervás-Martínez, C. “Ensembles of Evolutionary Product Unit or RBF Neural Networks for the Identification of Sound for Pass-by Noise Test in Vehicles”, *Neurocomputing*, vol. 109, June 2013, pp. 56–65.

[219] Ribeiro, M. T.; Singh, S.; Guestrin, C. “Why should I trust you? Explaining the predictions of any classifier”. In: International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.

[220] Rish, I. “An empirical study of the naive Bayes classifier”. In: Workshop on Empirical Methods in Artificial Intelligence, 2001, pp. 41–46.

[221] Rodríguez-Fdez, I.; Canosa, A.; Mucientes, M.; Bugarín, A. “STAC: a web platform for the comparison of algorithms using statistical tests”. In: International Conference on Fuzzy Systems, 2015, pp. 1–8.

[222] Roebber, P. J. “Adaptive Evolutionary Programming”, *Monthly Weather Review*, vol. 143–5, May 2015, pp. 1497–1505.

[223] Rokach, L. “Ensemble-based Classifiers”, *Artificial Intelligence Review*, vol. 33–1, November 2010, pp. 1–39.

[224] Rosales-Pérez, A.; García, S.; Gonzalez, J. A.; Coello, C. A. C.; Herrera, F. “An Evolutionary Multi-Objective Model and Instance Selection for Support Vector Machines with Pareto-based Ensembles”, *IEEE Transactions on Evolutionary Computation*, vol. 21–6, March 2017, pp. 863–877.

[225] Rosales-Pérez, A.; Gonzalez, J. A.; Coello, C. A. C.; Escalante, H. J.; Reyes-Garcia, C. A. “Multi-objective Model Type Selection”, *Neurocomputing*, vol. 146, December 2014, pp. 83–94.

[226] Ross, B. C. “Mutual information between discrete and continuous data sets”, *PloS One*, vol. 9–2, February 2014, pp. 1–5.

[227] Rudin, C. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”, *Nature Machine Intelligence*, vol. 1–5, May 2019, pp. 206–215.

[228] Sagi, O.; Rokach, L. "Ensemble Learning: A Survey", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8–4, January 2018.

[229] Saha, S.; Mitra, S.; Yadav, R. K. "A Multiobjective based Automatic Framework for Classifying Cancer-microRNA Biomarkers", *Gene Reports*, vol. 4, September 2016, pp. 91–103.

[230] Saleh, R.; Farsi, H.; Zahiri, S. H. "Ensemble Classification of PolSAR Data using Multi-objective Heuristic Combination Rule". In: Conference on Swarm Intelligence and Evolutionary Computation, 2016, pp. 88–92.

[231] Santu, S. K. K.; Rahman, M. M.; Islam, M. M.; Murase, K. "Towards better Generalization in Pittsburgh Learning Classifier Systems". In: Congress on Evolutionary Computation, 2014, pp. 1666–1673.

[232] Schaefer, G. "Evolutionary Optimisation of Classifiers and Classifier Ensembles for Cost-sensitive Pattern Recognition". In: International Symposium on Applied Computational Intelligence and Informatics, 2013, pp. 343–346.

[233] Schapire, R. E. "A Brief Introduction to Boosting". In: International Joint Conference on Artificial Intelligence, 1999, pp. 1401–1406.

[234] Schuman, C. D.; Birdwell, J. D.; Dean, M. E. "Spatiotemporal Classification using Neuroscience-inspired Dynamic Architectures", *Procedia Computer Science*, vol. 41, November 2014, pp. 89–97.

[235] Shunmugapriya, P.; Kanmani, S. "Optimization of Stacking Ensemble Configurations through Artificial Bee Colony Algorithm", *Swarm and Evolutionary Computation*, vol. 12, October 2013, pp. 24–32.

[236] Sikdar, U. K.; Ekbal, A.; Saha, S. "Differential Evolution based Feature Selection and Classifier Ensemble for Named Entity Recognition". In: International Conference on Computational Linguistics, 2012, pp. 2475–2490.

[237] Sikdar, U. K.; Ekbal, A.; Saha, S. "Differential Evolution based Mention Detection for Anaphora Resolution". In: India Conference, 2013, pp. 1–6.

[238] Sikdar, U. K.; Ekbal, A.; Saha, S. "Differential Evolution based Multiobjective Optimization for Biomedical Entity Extraction". In: International Conference on Advances in Computing, Communications and Informatics, 2014, pp. 1039–1044.

[239] Sikdar, U. K.; Ekbal, A.; Saha, S. "Entity Extraction in Biochemical Text using Multiobjective Optimization", *Computación y Sistemas*, vol. 18–3, February 2014, pp. 591–602.

[240] Sikdar, U. K.; Ekbal, A.; Saha, S. "MODE: Multiobjective Differential Evolution for Feature Selection and Classifier Ensemble", *Soft Computing*, vol. 19–12, January 2015, pp. 3529–3549.

[241] Sikdar, U. K.; Ekbal, A.; Saha, S. “A Generalized Framework for Anaphora Resolution in Indian Languages”, *Knowledge-Based Systems*, vol. 109, October 2016, pp. 147–159.

[242] Singh, I.; Sanwal, K.; Praveen, S. “Breast Cancer Detection using two-fold Genetic Evolution of Neural Network Ensembles”. In: International Conference on Data Science and Engineering, 2016, pp. 1–6.

[243] Stefano, C. D.; Fontanella, F.; Folino, G.; Freca, A. “A Bayesian approach for Combining Ensembles of GP Classifiers”. In: International Workshop on Multiple Classifier Systems, 2011, pp. 26–35.

[244] Storn, R.; Price, K. “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, *Journal of Global Optimization*, vol. 11–4, December 1997, pp. 341–359.

[245] Su, J.; Vargas, D. V.; Sakurai, K. “One pixel attack for fooling deep neural networks”, *IEEE Transactions on Evolutionary Computation*, vol. 23–5, October 2019, pp. 828–841.

[246] Tabassum, N.; Ahmed, T. “A Theoretical Study on Classifier Ensemble Methods and its Applications”. In: International Conference on Computing for Sustainable Global Development, 2016, pp. 374–378.

[247] Tan, C. J.; Lim, C. P.; Cheah, Y.-N. “A Multi-objective Evolutionary Algorithm-based Ensemble Optimizer for Feature Selection and Classification with Neural Network Models”, *Neurocomputing*, vol. 125, February 2014, pp. 217–228.

[248] Tang, H. L.; Goh, J.; Peto, T.; Ling, B. W.-K.; Al Turk, L. I.; Hu, Y.; Wang, S.; Saleh, G. M. “The Reading of Components of Diabetic Retinopathy: An Evolutionary approach for Filtering normal Digital Fundus Imaging in Screening and Population based Studies”, *PloS One*, vol. 8–7, July 2013, pp. e66730.

[249] Thornton, C.; Hutter, F.; Hoos, H. H.; Leyton-Brown, K. “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms”. In: International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.

[250] Tian, J.; Feng, N. “Adaptive Generalized Ensemble Construction with Feature Selection and its Application in Recommendation”, *International Journal of Computational Intelligence Systems*, vol. 7–sup2, July 2014, pp. 35–43.

[251] Trawinski, K.; Cordón, O.; Quirin, A. “Embedding Evolutionary Multiobjective Optimization into Fuzzy Linguistic Combination method for Fuzzy Rule-based Classifier Ensembles”. In: International Conference on Fuzzy Systems, 2014, pp. 1968–1975.

[252] Trawiski, K.; Cordón, O.; Quirin, A.; Sánchez, L. “Multiobjective Genetic Classifier Selection for Random Oracles Fuzzy Rule-based Classifier Ensembles: How Beneficial is the Additional Diversity?”, *Knowledge-Based Systems*, vol. 54, December 2013, pp. 3–21.

[253] Trivedi, S. K.; Dey, S. “A Study of Ensemble based Evolutionary Classifiers for Detecting Unsolicited Emails”. In: Conference on Research in Adaptive and Convergent Systems, 2014, pp. 46–51.

[254] Tsakonas, A. “An analysis of Accuracy-diversity Trade-off for Hybrid Combined System with Multiobjective Predictor Selection”, *Applied Intelligence*, vol. 40–4, January 2014, pp. 710–723.

[255] Tsakonas, A.; Gabrys, B. “A Fuzzy Evolutionary Framework for Combining Ensembles”, *Applied Soft Computing*, vol. 13–4, April 2013, pp. 1800–1812.

[256] Vaiciukynas, E.; Verikas, A.; Gelzinis, A.; Bacauskiene, M.; Kons, Z.; Satt, A.; Hoory, R. “Fusion of Voice Signal Information for Detection of Mild Laryngeal Pathology”, *Applied Soft Computing*, vol. 18, May 2014, pp. 91–103.

[257] Van Assche, A.; Blockeel, H. “Seeing the forest through the trees: Learning a comprehensible model from an ensemble”. In: European Conference on Machine Learning, 2007, pp. 418–429.

[258] Van Rijn, J. N.; Hutter, F. “Hyperparameter importance across datasets”. In: International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2367–2376.

[259] Veeramachaneni, K.; Derby, O.; Sherry, D.; O’Reilly, U.-M. “Learning Regression Ensembles with Genetic Programming at Scale”. In: Conference on Genetic and Evolutionary Computation, 2013, pp. 1117–1124.

[260] Vega-Pons, S.; Ruiz-Shulcloper, J. “A survey of Clustering Ensemble Algorithms”, *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 25–3, May 2011, pp. 337–372.

[261] Vluymans, S.; Triguero, I.; Cornelis, C.; Saeys, Y. “EPRENNID: An Evolutionary Prototype Reduction based Ensemble for Nearest Neighbor Classification of Imbalanced Data”, *Neurocomputing*, vol. 216, December 2016, pp. 596–610.

[262] Vukobratovi, B.; Struharik, R. “Hardware Acceleration of Nonincremental Algorithms for the Induction of Decision Trees”. In: Telecommunication Forum, 2017, pp. 1–8.

[263] Wang, D.; Alhamdoosh, M. “Evolutionary Extreme Learning Machine Ensembles with Size Control”, *Neurocomputing*, vol. 102, February 2013, pp. 98–110.

[264] Wen, Y.-W.; Ting, C.-K. “Learning Ensemble of Decision Trees through Multifactorial Genetic Programming”. In: Congress on Evolutionary Computation, 2016, pp. 5293–5300.

[265] Wilcoxon, F. “Individual comparisons by ranking methods”. In: *Breakthroughs in Statistics*, Kotz, S.; Johnson, N. L. (Editors), Springer, 1992, pp. 196–202.

[266] Winkler, S.; Schaller, S.; Dorfer, V.; Affenzeller, M.; Petz, G.; Karpowicz, M. “Data-based Prediction of Sentiments using Heterogeneous Model Ensembles”, *Soft Computing*, vol. 19–12, July 2015, pp. 3401–3412.

[267] Wolpert, D. H. “Stacked Generalization”, *Neural Networks*, vol. 5–2, July 1992, pp. 241–259.

[268] Wolpert, D. H.; Macready, W. G. “No Free Lunch Theorems for Optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 1–1, April 1997, pp. 67–82.

[269] Woon, W. L.; Kramer, O. “Enhanced SVR Ensembles for Wind Power Prediction”. In: International Joint Conference on Neural Networks, 2016, pp. 2743–2748.

[270] Wozniak, M. “Evolutionary Approach to Produce Classifier Ensemble based on Weighted Voting”. In: World Congress on Nature and Biologically Inspired Computing, 2009, pp. 648–653.

[271] Xavier-Júnior, J. a. C.; Freitas, A. A.; Feitosa-Neto, A.; Ludermir, T. B. “A Novel Evolutionary Algorithm for Automated Machine Learning Focusing on Classifier Ensembles”. In: Brazilian Conference on Intelligent Systems, 2018, pp. 1–6.

[272] Xavier-Júnior, J. a. C.; Freitas, A. A.; Feitosa-Neto, A.; Ludermir, T. B. “A Novel Evolutionary Algorithm for Automated Machine Learning Focusing on Classifier Ensembles”. In: Brazilian Conference on Intelligent Systems, 2018, pp. 1–6.

[273] Xu, H.; Caramanis, C.; Mannor, S. “Sparse Algorithms are not Stable: A no-free-lunch Theorem”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34–1, January 2012, pp. 187–193.

[274] Yao, X.; Islam, M. M. “Evolving Artificial Neural Network Ensembles”, *IEEE Computational Intelligence Magazine*, vol. 3–1, February 2008, pp. 31–42.

[275] Yildirim, I. “Bayesian inference: Gibbs sampling”, Technical Report, Department of Brain and Cognitive Sciences, University of Rochester, Rochester, USA, 2012, 6p.

[276] Zagorecki, A. “Feature Selection for Naive Bayesian Network Ensemble using Evolutionary Algorithms”. In: Federated Conference on Computer Science and Information Systems, 2014, pp. 381–385.

[277] Zangari, M.; Santana, R.; Mendiburu, A.; Pozo, A. T. R. “Not all PBILs are the same: Unveiling the different learning mechanisms of PBIL variants”, *Applied Soft Computing*, vol. 53, April 2017, pp. 88–96.

[278] Zhang, L.; Mistry, K.; Neoh, S. C.; Lim, C. P. “Intelligent Facial Emotion Recognition using Moth-firefly Optimization”, *Knowledge-Based Systems*, vol. 111, November 2016, pp. 248–267.

[279] Zhang, W.; Qu, Z.; Zhang, K.; Mao, W.; Ma, Y.; Fan, X. “A Combined Model based on CEEMDAN and Modified Flower Pollination Algorithm for Wind Speed Forecasting”, *Energy Conversion and Management*, vol. 136, March 2017, pp. 439–451.

[280] Zhang, Y.; Liu, B.; Cai, J.; Zhang, S. "Ensemble Weighted Extreme Learning Machine for Imbalanced Data Classification based on Differential Evolution", *Neural Computing and Applications*, vol. 28–1, May 2017, pp. 259–267.

[281] Zhang, Y.; Liu, B.; Yang, F. "Differential Evolution Based Selective Ensemble of Extreme Learning Machine". In: Trustcom/BigDataSE/ISPA, 2016, pp. 1327–1333.

[282] Zhang, Y.; Zhang, H.; Cai, J.; Yang, B. "A Weighted Voting Classifier based on Differential Evolution", *Abstract and Applied Analysis*, vol. 2014–1, May 2014, pp. 1–6.

MARISTA

PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br