Chapter 1
Introduction

Classification, which is the data mining task of assigning objects to predefined categories, is widely used in the process of intelligent decision making. Many classification techniques have been proposed by researchers in machine learning, statistics, and pattern recognition. Such techniques can be roughly divided according to the their level of comprehensibility. For instance, techniques that produce interpretable classification models are known as *white-box* approaches, whereas those that do not are known as *black-box* approaches. There are several advantages in employing white-box techniques for classification, such as increasing the user confidence in the prediction, providing new insight about the classification problem, and allowing the detection of errors either in the model or in the data [12]. Examples of white-box classification techniques are classification rules and decision trees. The latter is the main focus of this book.

A decision tree is a classifier represented by a flowchart-like tree structure that has been widely used to represent classification models, specially due to its comprehensible nature that resembles the human reasoning. In a recent poll from the *kdnuggets* website [13], decision trees figured as the most used data mining/analytic method by researchers and practitioners, reaffirming its importance in machine learning tasks. Decision-tree induction algorithms present several advantages over other learning algorithms, such as robustness to noise, low computational cost for generating the model, and ability to deal with redundant attributes [22].

Several attempts on optimising decision-tree algorithms have been made by researchers within the last decades, even though the most successful algorithms date back to the mid-80s [4] and early 90s [21]. Many strategies were employed for deriving accurate decision trees, such as bottom-up induction [1, 17], linear programming [3], hybrid induction [15], and ensemble of trees [5], just to name a few. Nevertheless, no strategy has been more successful in generating accurate and comprehensible decision trees with low computational effort than the greedy top-down induction strategy.

A greedy top-down decision-tree induction algorithm recursively analyses if a sample of data should be partitioned into subsets according to a given rule, or if no further partitioning is needed. This analysis takes into account a stopping criterion, for deciding when tree growth should halt, and a splitting criterion, which is responsible for choosing the "best" rule for partitioning a subset. Further improvements over this basic strategy include pruning tree nodes for enhancing the tree's capability of dealing with noisy data, and strategies for dealing with missing values, imbalanced classes, oblique splits, among others.

A very large number of approaches were proposed in the literature for each one of these *design components* of decision-tree induction algorithms. For instance, new measures for node-splitting tailored to a vast number of application domains were proposed, as well as many different strategies for selecting multiple attributes for composing the node rule (multivariate split). There are even studies in the literature that survey the numerous approaches for pruning a decision tree [6, 9]. It is clear that by improving these design components, more effective decision-tree induction algorithms can be obtained.

An approach that has been increasingly used in academia is the induction of decision trees through evolutionary algorithms (EAs). They are essentially algorithms inspired by the principle of natural selection and genetics. In nature, individuals are continuously evolving, adapting to their living environment. In EAs, each "individual" represents a candidate solution to the target problem. Each individual is evaluated by a fitness function, which measures the quality of its corresponding candidate solution. At each generation, the best individuals have a higher probability of being selected for reproduction. The selected individuals undergo operations inspired by genetics, such as crossover and mutation, producing new offspring which will replace the parents, creating a new generation of individuals. This process is iteratively repeated until a stopping criterion is satisfied [8, 11]. Instead of local search, EAs perform a robust global search in the space of candidate solutions. As a result, EAs tend to cope better with attribute interactions than greedy methods [10].

The number of EAs for decision-tree induction has grown in the past few years, mainly because they report good predictive performance whilst keeping the comprehensibility of decision trees [2]. In this approach, each individual of the EA is a decision tree, and the evolutionary process is responsible for searching the solution space for the "near-optimal" tree regarding a given data set. A disadvantage of this approach is that it generates a decision tree tailored to a single data set. In other words, an EA has to be executed every time we want to induce a tree for a giving data set. Since the computational effort of executing an EA is much higher than executing the traditional greedy approach, it may not be the best strategy for inducing decision trees in time-constrained scenarios.

Whether we choose to induce decision trees through the greedy strategy (top-down, bottom-up, hybrid induction), linear programming, EAs, ensembles, or any other available method, we are susceptible to the method's inductive bias. Since we know that certain inductive biases are more suitable to certain problems, and that no method is best for every single problem (i.e., the no free lunch theorem [26]), there is a growing interest in developing automatic methods for deciding which learner to use in each situation. A whole new research area named *meta-learning* has emerged for solving this problem [23]. Meta-learning is an attempt to understand data *a priori* of executing a learning algorithm. In a particular branch of meta-learning, *algorithm* *recommendation*, data that describe the characteristics of data sets and learning algorithms (i.e., meta-data) are collected, and a learning algorithm is employed to interpret these meta-data and suggest a particular learner (or ranking a few learners) in order to better solve the problem at hand. Meta-learning has a few limitations, though. For instance, it provides a limited number of algorithms to be selected from a list. In addition, it is not an easy task to define the set of meta-data that will hopefully contain useful information for identifying the best algorithm to be employed.

For avoiding the limitations of traditional meta-learning approaches, a promising idea is to automatically develop algorithms tailored to a given domain or to a specific set of data sets. This approach can be seen as a particular type of meta-learning, since we are learning the "optimal learner" for specific scenarios. One possible technique for implementing this idea is *genetic programming* (GP). It is a branch of EAs that arose as a paradigm for evolving computer programs in the beginning of the 90s [16]. The idea is that each individual in GP is a computer program that evolves during the evolutionary process of the EA. Hopefully, at the end of evolution, GP will have found the appropriate algorithm (best individual) for the problem we want to solve. Pappa and Freitas [20] cite two examples of EA applications in which the evolved individual outperformed the best human-designed solution for the problem. In the first application [14], the authors designed a simple satellite dish holder boom (connection between the satellite's body and the communication dish) using an EA. This automatically designed dish holder boom, albeit its bizarre appearance, was shown to be 20,000 % better than the human-designed shape. The second application [18] was concerning the automatic discovery of a new form of boron (chemical element). There are only four known forms of borons, and the last one was discovered by an EA.

A recent research area within the combinatorial optimisation field named "hyper-heuristics" (HHs) has emerged with a similar goal: searching in the heuristics space, or in other words, *heuristics to choose heuristics* [7]. HHs are related to metaheuristics, though with the difference that they operate on a search space of heuristics whereas metaheuristics operate on a search space of solutions to a given problem. Nevertheless, HHs usually employ metaheuristics (e.g., evolutionary algorithms) as the search methodology to look for suitable heuristics to a given problem [19]. Considering that an algorithm or its components can be seen as heuristics, one may say that HHs are also suitable tools to automatically design custom (tailor-made) algorithms.

Whether we name it "an EA for automatically designing algorithms" or "hyper-heuristics", in both cases there is a set of human designed components or heuristics, surveyed from the literature, which are chosen to be the starting point for the evolutionary process. The expected result is the automatic generation of new procedural components and heuristics during evolution, depending of course on which components are provided to the EA and the respective "freedom" it has for evolving the solutions.

The automatic design of complex algorithms is a much desired task by researchers. It was envisioned in the early days of artificial intelligence research, and more recently has been addressed by machine learning and evolutionary computation research groups [20, 24, 25]. Automatically designing machine learning algorithms can be seen as the task of teaching the computer how to create programs that learn from experience. By providing an EA with initial human-designed programs, the evolutionary process will be in charge of generating new (and possibly better) algorithms for the problem at hand. Having said that, we believe an EA for automatically discovering new decision-tree induction algorithms may be the solution to avoid the drawbacks of the current decision-tree approaches, and this is going to be the main topic of this book.

1.1 Book Outline

This book is structured in 7 chapters, as follows.

Chapter 2 [Decision-Tree Induction]. This chapter presents the origins, basic concepts, detailed components of top-down induction, and also other decision-tree induction strategies.

Chapter 3 [Evolutionary Algorithms and Hyper-Heuristics]. This chapter covers the origins, basic concepts, and techniques for both Evolutionary Algorithms and Hyper-Heuristics.

Chapter 4 [HEAD-DT: Automatic Design of Decision-Tree Induction Algorithms]. This chapter introduces and discusses the hyper-heuristic evolutionary algorithm that is capable of automatically designing decision-tree algorithms. Details such as the evolutionary scheme, building blocks, fitness evaluation, selection, genetic operators, and search space are covered in depth.

Chapter 5 [HEAD-DT: Experimental Analysis]. This chapter presents a thorough empirical analysis on the distinct scenarios in which HEAD-DT may be applied to. In addition, a discussion on the cost effectiveness of automatic design, as well as examples of automatically-designed algorithms and a baseline comparison between genetic and random search are also presented.

Chapter 6 [HEAD-DT: Fitness Function Analysis]. This chapter conducts an investigation of 15 distinct versions for HEAD-DT by varying its fitness function, and a new set of experiments with the best-performing strategies in balanced and imbalanced data sets is described.

Chapter 7 [Conclusions]. We finish this book by presenting the current limitations of the automatic design, as well as our view of several exciting opportunities for future work.

References

1. R.C. Barros et al., A bottom-up oblique decision tree induction algorithm, in *11th International Conference on Intelligent Systems Design and Applications*. pp. 450–456 (2011)
2. R.C. Barros et al., A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev. **42**(3), 291–312 (2012)
3. K. Bennett, O. Mangasarian, Multicategory discrimination via linear programming. Optim. Methods Softw. **2**, 29–39 (1994)
4. L. Breiman et al., *Classification and Regression Trees* (Wadsworth, Belmont, 1984)
5. L. Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
6. L. Breslow, D. Aha, Simplifying decision trees: a survey. Knowl. Eng. Rev. **12**(01), 1–40 (1997)
7. P. Cowling, G. Kendall, E. Soubeiga, A Hyperheuristic Approach to Scheduling a Sales Summit, in *Practice and Theory of Automated Timetabling III*, Vol. 2079. Lecture Notes in Computer Science, ed. by E. Burke, W. Erben (Springer, Berlin, 2001), pp. 176–190
8. A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing (Natural Computing Series)* (Springer, Berlin, 2008)
9. F. Esposito, D. Malerba, G. Semeraro, A comparative analysis of methods for pruning decision trees. IEEE Trans. Pattern Anal. Mach. Intell. **19**(5), 476–491 (1997)
10. A.A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms* (Springer, New York, 2002). ISBN: 3540433317
11. A.A. Freitas, A Review of evolutionary Algorithms for Data Mining, in *Soft Computing for Knowledge Discovery and Data Mining*, ed. by O. Maimon, L. Rokach (Springer, Berlin, 2008), pp. 79–111. ISBN: 978-0-387-69935-6
12. A.A. Freitas, D.C. Wieser, R. Apweiler, On the importance of comprehensible classification models for protein function prediction. IEEE/ACM Trans. Comput. Biol. Bioinform. **7**, 172–182 (2010). ISSN: 1545–5963
13. KDNuggets, *Poll: Data mining/analytic methods you used frequently in the past 12 months* (2007)
14. A. Keane, S. Brown, The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques, in *Conference on Adaptative Computing in Engineering Design and Control*. Plymouth, pp. 107–113 (1996)
15. B. Kim, D. Landgrebe, Hierarchical classifier design in high-dimensional numerous class cases. IEEE Trans. Geosci. Remote Sens. **29**(4), 518–528 (1991)
16. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992). ISBN: 0-262-11170-5
17. G. Landeweerd et al., Binary tree versus single level tree classification of white blood cells. Pattern Recognit. **16**(6), 571–577 (1983)
18. A.R. Oganov et al., Ionic high-pressure form of elemental boron. Nature **457**, 863–867 (2009)
19. G.L. Pappa et al., Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms, in *Genetic Programming and Evolvable Machines* (2013)
20. G.L. Pappa, A.A. Freitas, *Automating the Design of Data Mining Algorithms: An Evolutionary Computation Approach* (Springer Publishing Company Incorporated, New York, 2009)
21. J.R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Francisco, 1993). ISBN: 1-55860-238-0
22. L. Rokach, O. Maimon, Top-down induction of decision trees classifiers—a survey. IEEE Trans. Syst. Man, Cybern. Part C: Appl. Rev. **35**(4), 476–487 (2005)
23. K.A. Smith-Miles, Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**, 6:1–6:25 (2009)
24. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). ISSN: 1063–6560
25. A. Vella, D. Corne, C. Murphy, Hyper-heuristic decision tree induction, in *World Congress on Nature and Biologically Inspired Computing*, pp. 409–414 (2010)
26. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)