

UNIVERSIDADE FEDERAL DE MINAS GERAIS  
Instituto de Ciências Exatas  
Programa de Pós-Graduação em Ciência da Computação

Alex Guimarães Cardoso de Sá

**Automated Multi-Label Classification: Methods, Issues and Prospects**

Belo Horizonte  
2019

Alex Guimarães Cardoso de Sá

**Automated Multi-Label Classification: Methods, Issues and Prospects**

**Final Version**

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Gisele Lobo Pappa

Belo Horizonte  
2019

Sá, Alex Guimarães Cardoso de.

S111a

Automated multi-label classification: [recurso eletrônico] methods, issues and prospects / Alex Guimarães Cardoso de Sá. 2019.

1 recurso online (210 f. il, color.): pdf.

Orientador: Gisele Lobo Pappa.

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.

Referências: f. 136-156.

1. Computação – Teses. 2. Aprendizado de máquina – Teses. 3 Classificação multirrótulo – Teses. 4. Mineração de dados – Teses. I. Pappa, Gisele Lobo. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6\*73(043)




UNIVERSIDADE FEDERAL DE MINAS GERAIS  
INSTITUTO DE CIÊNCIAS EXATAS  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO


## FOLHA DE APROVAÇÃO


Automated Multi-Label Classification: Methods, Issues and Prospects

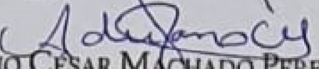
**ALEX GUIMARÃES CARDOSO DE SÁ**

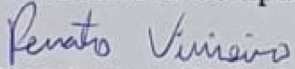
Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

  
PROFA. GISELE LOBO PAPPÁ - Orientadora  
Departamento de Ciência da Computação - UFMG

  
PROF. ANDRÉ CARLOS PONCE DE L. F. DE CARVALHO  
Instituto de Ciências Matemáticas e de Computação - USP

  
PROF. LUIZ HENRIQUE DE CAMPOS MERSCHMANN  
Departamento de Ciência da Computação - UFLA

  
PROF. ADRIANO CÉSAR MACHADO PEREIRA  
Departamento de Ciência da Computação - UFMG

  
PROF. RENATO VIMIEIRO  
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 17 de Dezembro de 2019.

*This work is dedicated to my family: Larissa, Meire, José Roberto and Lucas.*

# Acknowledgments

Eu penso que seria impossível começar esta seção sem agradecer aos meus pais, José Roberto e Meire, e ao meu irmão, Lucas. Sempre houve um apoio incondicional da parte deles, seja no nível educacional, profissional ou pessoal. Por isso, agradeço imensamente a vocês por tudo que vocês fizeram por mim.

De forma muito especial, também gostaria de agradecer profundamente à minha noiva, Larissa Natany, que me acompanhou pessoal e profissionalmente em todas as etapas do meu doutorado. Eu acredito que não conseguiria finalizar o meu doutorado sem o seu amor e companheirismo. Não tenho palavras suficientes para te agradecer.

À minha família e à família da Larissa (agora minha segunda família), que também contribuíram de forma significativa para este trabalho, principalmente em momentos em que nós nos reuníamos. Esses momentos foram aqueles em que eu tinha importantes períodos de descanso e relaxamento. Em especial, gostaria de citar os meus tios e tias (Andraiza e Sidney, Míriam, Mércia, Ediney, Ednaldo, Ruy e Eny), primos e primas (Letícia e Lesley, em particular), Ana Tércia, Wallace, Mateus e Ana, e dona Loya. Além disso, às minhas cadelas, Mel (*in memoriam*) e Amora, pela alegria e carinho.

Outros momentos que pude relaxar e descansar foram aqueles em que pude compartilhar com os meus amigos e colegas. Dentre eles estão: amigos do Véio do Rio (Murdock, Ph, Sérgio e William); amigos do LaIC VIP (Karen, Matheus, Iago e Cristiano); amigos do grupo de F1 (Arthur, Gustavo, Léo, Hendrik, Paulo e Danilo); amigos e colegas do LAIC (Samuel, Luiz Otávio, Tarsila, Pedro, Tiago Cunha, Renato Miranda, Marcelo, Luis Fernando, Juliana, João Francisco e Walter); alunos/amigos de IC do LaIC com quem trabalhei diretamente (Isadora, Felipe, Laura, Paulo e Ingrid); amigos e colegas do e-Speed (Carlos Mazza, Vinícius, Roberto Nalon, Gui Maluf, Gustavo, Walter, Elverton e Derick); sifu, amigos e colegas do Tai Chi (Sifu Vânio, Túlio, Henrique, Tulio e Gláucia, Cíntia e Fernando); amigos e colegas do LaPO (Vinicius, Vitor, Luis Henrique, Evelyn, José Maurício, Iago, Amadeu e Dilson); amigos e colegas do doutorado/qualificação (Bruno, Victor, Júlio, Filipe, Rodrigo e Paulo); amigos e colegas que fiz enquanto estive em Canterbury (Pablo, Fábio, Talita, Léo, Carol, Jak e An); colegas da residência em ciência de dados (Aracelli, Fernanda e Rúbia); amigos e colegas da UFMG e aqueles que estão espalhados pelo mundo (Afonso, Wagner e Sanye, Elerson, Michelle Brandão, David Saldaña, Sérgio Canuto, Rafael Aquino, Saulo, Thaís Tekka, Edoardo Fadda, Nathália e Borges). Obrigado a todos vocês.

Também gostaria de demonstrar o meu agradecimento às minhas professoras de

inglês Marcilla, Emily e Estefânia. Com certeza, pude crescer muito desde que comecei a ter aulas com vocês e tenho certeza que certas oportunidades não viriam sem isso. Em particular, à Marcilla, pelas ótimas conversas e pela amizade.

Aos meus psicólogos, Luciana Leão e Igor Gaigher, que me ajudaram de diferentes formas a conseguir lidar melhor com questões pessoais e profissionais. Meu muito obrigado a vocês dois por me ajudar tanto nessas questões.

Gostaria de dizer muito obrigado à minha orientadora, profa. Gisele Pappa, pela orientação contínua desde o meu mestrado e pelo trabalho em conjunto. Considero que evoluí muito como profissional, no mestrado e doutorado, sob a sua supervisão. O doutorado não é um dos momentos mais fáceis, mas com certeza foi excelente poder trabalhar com você para contribuir para a área de AutoML. Sem dúvidas, o nosso trabalho continua.

Também gostaria de destacar e reconhecer a supervisão e colaboração do prof. Alex Freitas, que me supervisionou enquanto estive na *University of Kent* no período de doutorado sanduíche. Embora tenham sido poucos meses no Reino Unido, acredito que o aprendizado foi muito extenso. Muito obrigado por isso e pela colaboração que se manteve em seguida. Nesse ponto, agradeço também à *School of Computing* da *University of Kent* por me aceitar como aluno de doutorado visitante. Essa foi uma das melhores fases do meu doutorado, visto que tive um número muito grande de ideias e vivências.

Aos funcionários e professores do DCC/UFMG, gostaria também de agradecer. Em especial, gostaria de dizer muito obrigado aos professores Wagner Meira Jr., Adriano César Pereira, Renato Assunção, Mário Alvim, Gabriel Coutinho e Jussara Almeida. Além disso, meu muito obrigado às funcionárias, Antônia, Ludmila, Ermelinda e Sônia, por toda ajuda e paciência enquanto estive no DCC. Adicionalmente, estendo o meu agradecimento aos professores membros da banca desta tese de doutorado, André de Carvalho, Luiz Merschmann, Adriano César Pereira e Renato Vimieiro, por aceitarem o convite.

Por fim, gostaria de agradecer e ressaltar o apoio financeiro, técnico e científico dos seguintes projetos, agências de pesquisa, empresas e instituições durante o meu doutorado: CAPES/PROEX (Código de Financiamento 001); CAPES/PDSE (número do processo: 88881.136011/2016-01); CNPq; FAPEMIG; projeto ATMOSPHERE; projeto MASWeb; Tenbu; Laboratório e-Speed; Universidade Federal de Minas Gerais (UFMG); e Departamento de Ciência da Computação (DCC-UFMG).

*“Autumn is a second spring when every leaf is a flower.”*  
(Albert Camus, 1942)



# Resumo

Aprendizado de Máquina Automatizado (AutoAM) surgiu para lidar com a tarefa de selecionar automaticamente algoritmos e seus hiper-parâmetros para resolver com sucesso um determinado problema de Aprendizado de Máquina (AM). Isto é feito principalmente para evitar abordagens *ad hoc* para essa finalidade. Com a crescente popularidade dos algoritmos de AM e seu uso indiscriminado por profissionais que não necessariamente conhecem as peculiaridades desses algoritmos, a área de AutoAM tornou-se mais relevante do que nunca. Esta tese, em particular, é centrada em AutoAM para problemas de Classificação Multirrótulo (CMR). Em CMR, cada exemplo no conjunto de dados pode estar associado simultaneamente a vários rótulos, tornando-o uma generalização de sua versão canônica ou monorrótulo (i.e., com a associação de um único rótulo de classe para cada exemplo). Essencialmente, CMR se preocupa em aprender um modelo que separa os rótulos de classe em relevantes e irrelevantes para cada exemplo da base de dados. Embora tenhamos experimentado a progressão do campo de AutoAM, que introduziu métodos eficazes para problemas de classificação tradicional (i.e., monorrótulo) e de regressão, ainda existem vários problemas na pesquisa de AutoAM que permanecem em aberto. Esta tese se concentra em três deles. Primeiro, investigamos se nossos quatro métodos AutoAM propostos podem funcionar tão bem para problemas de CMR, assim como funcionam para problemas de classificação tradicional e de regressão. Apesar dos desafios inerentes à CMR (e.g., a dificuldade de aprender com esse tipo de dados, o esforço para avaliar seus modelos e o custo computacional envolvido), nossos resultados mostraram que é possível desenvolver métodos AutoAM para problemas de CMR que executam tão bem quanto, ou melhor, do que métodos de busca conhecidos. Em segundo lugar, apresentamos uma análise relativa ao tamanho de três espaços de busca propostos e ao desempenho dos métodos AutoAM na recomendação de configurações de algoritmos de aprendizado. Ao aumentar e diminuir o tamanho do espaço de busca, mostramos que os métodos AutoAM propostos não balanceiam satisfatoriamente bem entre diversificação e intensificação, apesar de seus resultados. Nossa análise de convergência também indicou que ainda devemos melhorar os métodos AutoAM propostos para garantir esse balanceamento. Por fim, investigamos como limitações de tempo distintas podem influenciar e restringir o comportamento dos métodos de busca do AutoAM e seu desempenho preditivo geral.

**Palavras-chave:** Aprendizado de Máquina Automatizado, Classificação, Configuração, Métodos de Busca, Espaços de Busca.

# Abstract

Automated Machine Learning (AutoML) has emerged to deal with the task of automatically selecting learning algorithms and their hyper-parameters to successfully solve a given ML problem. This is mainly done to avoid *ad hoc* approaches to perform this task. With the outgrowing popularity of Machine Learning (ML) algorithms and their indiscriminate use by practitioners, who do not necessarily know the peculiarities of these algorithms, the field of AutoML has become more relevant than ever. This thesis, in particular, is centered on AutoML for Multi-Label Classification (MLC) problems. In MLC, each example in the dataset can be simultaneously associated with several class labels, making it a generalization of its canonical single-label version (i.e., with a single class label per example). Essentially, MLC is concerned with learning a model that separates each class label into relevant and irrelevant for each example in the dataset. Although we have experienced the progression of the field of AutoML, which introduced effective methods for Single-Label Classification (SLC) and regression problems, there are still several issues in AutoML research that remain open. This thesis focuses on three of them. First, we investigate if our four proposed AutoML methods can work for MLC problems as well as they work for SLC and regression problems. Apart from the inherent challenges in MLC (e.g., the hardness of learning from this type of data, the strain to evaluate its models, and the computational cost involved), our results showed that it is possible to develop AutoML methods for MLC problems that perform as good as or better than well-known global and local search methods. Second, we present an analysis relating to the size of three designed search spaces and the performance of the AutoML methods in recommending configured learning algorithms. By increasing and decreasing the search space size, we show that the proposed AutoML methods do not satisfactorily trade-off between exploration (novelty) and exploitation (locality) besides their results. Our convergence analysis also indicated that we must still improve the proposed AutoML methods (i.e., their internal mechanisms) to ensure this trade-off. Finally, we investigate how distinct time budgets (constraining the whole AutoML process) can influence and constrain the behavior of the AutoML search methods and their overall predictive performance.

**Keywords:** Automated Machine Learning (AutoML), Multi-Label Classification, Configuration, Search Methods, Search Spaces.

# List of Algorithms

2.1	A generic pseudo-code for evolutionary algorithms. . . . .	31
2.2	Example of a programming code. . . . .	33
2.3	A generic pseudo-code of Bayesian optimization. . . . .	38
5.1	General pseudo-code for evolutionary algorithms for MLC. . . . .	83

# List of Boxes

2.1	Grammar for generating the tree of Figure 2.1a. . . . .	36
5.1	Defined grammar – Part 1: General and SLC trees algorithms. . . . .	76
5.2	Defined grammar – Part 2: SLC rules and lazy algorithms . . . . .	77
5.3	Defined grammar – Part 3: SLC functions, Bayesian and other types of algorithms. . . . .	78
5.4	Defined grammar – Part 4: SLC meta-algorithms. . . . .	79
5.5	Defined grammar – Part 5: MLC problem transformation methods. . . . .	80
5.6	Defined grammar – Part 6: MLC algorithm adaptation methods. . . . .	81
5.7	Defined grammar – Part 7: MLC meta-algorithms. . . . .	81

# List of Figures

1.1	The possible hierarchy levels in MLC. . . . .	24
2.1	Examples of parse trees for Equation 2.1 and Algorithm 2.2. . . . .	34
4.1	A typical AutoML process. . . . .	56
5.1	The general AutoML framework to select and configure MLC algorithms. . . .	68
5.2	GA-Auto-MLC: The proposed genetic algorithm to select and configure MLC algorithms. . . . .	84
5.3	Possible phenotypes of GA-Auto-MLC's individuals (for the <i>search space</i> Small). . . . .	85
5.4	A possible genotype of an individual in the <i>search space</i> Small. . . . .	86
5.5	Evaluation process of one individual in GA-Auto-MLC. . . . .	87
5.6	Auto-MEKA <sub>GGP</sub> : The proposed GGP method to select and configure MLC algorithms. . . . .	88
5.7	Evaluation process of one individual in Auto-MEKA <sub>GGP</sub> . . . . .	88
5.8	Auto-MEKA <sub>spGGP</sub> : The proposed speciation-based GGP method to select and configure MLC algorithms. . . . .	91
5.9	Auto-MEKA <sub>BO</sub> : The proposed Bayesian optimization method to select and configure MLC algorithms. . . . .	93
5.10	Evaluation process of one individual in Auto-MEKA <sub>BO</sub> . . . . .	94
6.1	Bar plots for the algorithms' selection at the MLC level over all runs. . . . .	120
6.2	Bar plots for the algorithms' selection at the Meta-MLC level over all runs. . . .	121
6.3	Barplots for the algorithms' selection at the SLC level over all runs. . . . .	123
6.4	Bar plots for the algorithms' selection at the Meta-SLC level selection over all runs. . . . .	124
6.5	Convergence of fitness/quality values for the dataset <i>GPP</i> . . . . .	127
6.6	Convergence of fitness/quality values for the dataset <i>CAL</i> . . . . .	128

# List of Tables

3.1	Overview of the PT algorithms described in Section 3.1.1. . . . .	48
3.2	Overview of the AA algorithms described in Section 3.1.2. . . . .	49
3.3	Overview of the ensemble algorithms described in Section 3.1.3. . . . .	49
5.1	Multi-Label Classification (MLC) algorithms used in the MEKA data mining tool for the proposed AutoML methods*. . . . .	70
5.2	Single-Label Classification (SLC) algorithms used in the WEKA data mining tool for the proposed AutoML methods*. . . . .	71
5.3	Estimation of the number of possibilities for the Single-Label Classification (SLC) algorithms*. . . . .	74
5.4	Multi-Label Classification (MLC) algorithms used in the MEKA data mining tool for the proposed AutoML methods*. . . . .	75
5.5	Employed polynomial multi-fidelity approach in terms of the number of attributes of the dataset based on the specified time budget. . . . .	95
5.6	Employed exponential multi-fidelity approach in terms of the number of attributes of the dataset based on the specified time budget. . . . .	96
5.7	Employed polynomial multi-fidelity approach in terms of the number of instances of the dataset based on the specified time budget. . . . .	96
6.1	Datasets used in the experiments. . . . .	100
6.2	Results in terms of quality measure (defined in Equation 5.1) on the test set within one hour of training for the proposed AutoML <i>search methods</i> . . . . .	104
6.3	Results in terms of the quality measure (defined in Equation 5.1) on the test set within five hours of training for the proposed AutoML <i>search methods</i> . . . . .	104
6.4	Auto-MEKA <sub>spGGP</sub> 's results based on the quality measure (defined in Equation 5.1) on the test set with the presence and absence of resampling within one hour. . . . .	106
6.5	Auto-MEKA <sub>spGGP</sub> 's results based on the quality measure (defined in Equation 5.1) on the test set with the presence and absence of resampling within five hours. . . . .	106
6.6	Possible scenarios for the multi-fidelity approaches. . . . .	107
6.7	Auto-MEKA <sub>spGGP</sub> 's multi-fidelity tuning results based on the average quality measure (defined in Equation 5.1) on the test set within one hour. . . . .	108

6.8	Auto-MEKA <sub>spGGP</sub> 's multi-fidelity tuning results based on the average quality measure on the test set within five hours. . . . .	108
6.9	Auto-MEKA <sub>spGGP</sub> 's multi-fidelity average number of generations within one hour. . . . .	108
6.10	Auto-MEKA <sub>spGGP</sub> 's multi-fidelity average number of generations within five hours. . . . .	108
6.11	Auto-MEKA <sub>spGGP</sub> 's results based on the defined quality measure on the test set for varying the intra/inter-species crossover probabilities within one hour of training. . . . .	110
6.12	Auto-MEKA <sub>spGGP</sub> 's results based on the defined quality measure on the test set for varying the intra/inter-species crossover probabilities within five hours of training. . . . .	110
6.13	Comparison of the exact match (to be maximized) obtained by the proposed <i>search methods</i> and the baseline methods in the test set for the three designed <i>search spaces</i> within one and five hours of execution. . . . .	112
6.14	Comparison of the hamming loss (to be minimized) obtained by the proposed methods and the baseline methods in the test set for the three designed <i>search spaces</i> within one and five hours of execution. . . . .	113
6.15	Comparison of the $F_1$ macro-averaged by label (to be maximized) obtained by the proposed methods and the baseline methods in the test set for the three designed <i>search spaces</i> within one and five hours of execution. . . . .	114
6.16	Comparison of the ranking loss (to be minimized) obtained by the proposed methods and the baseline methods in the test set for the three designed <i>search spaces</i> within one and five hours of execution. . . . .	115
6.17	Comparison of the fitness (to be maximized) obtained by the proposed methods and the baseline methods in the test set for the three designed <i>search spaces</i> within one and five hours of execution. . . . .	116

# Contents

<b>1</b>	<b>Introduction</b>	<b>19</b>
1.1	Motivation . . . . .	22
1.2	Objectives . . . . .	26
1.3	Contributions . . . . .	27
1.4	Thesis Organization . . . . .	27
<b>2</b>	<b>Search and Optimization Methods</b>	<b>29</b>
2.1	Evolutionary Algorithms . . . . .	29
2.1.1	Genetic Algorithm . . . . .	32
2.1.2	Genetic Programming . . . . .	33
2.1.3	Grammar-Based Genetic Programming . . . . .	35
2.2	Bayesian Optimization Algorithms . . . . .	37
2.2.1	Tree-Structured Parzen Estimator . . . . .	39
2.2.2	Sequential Model-based Algorithm Configuration . . . . .	40
2.3	Final Remarks . . . . .	40
<b>3</b>	<b>Multi-Label Classification</b>	<b>42</b>
3.1	Categorization of MLC Methods . . . . .	43
3.1.1	Problem Transformation Methods . . . . .	44
3.1.2	Algorithm Adaptation Methods . . . . .	45
3.1.3	Ensembles Methods . . . . .	46
3.1.4	Overview of the MLC Algorithms . . . . .	48
3.2	Evaluation in MLC . . . . .	50
3.2.1	Bipartition Measures . . . . .	50
3.2.1.1	Example-based Bipartition Measures . . . . .	50
3.2.1.2	Label-based Bipartition Measures . . . . .	51
3.2.2	Ranking Measures . . . . .	53
3.3	Final Remarks . . . . .	53
<b>4</b>	<b>Automated Machine Learning</b>	<b>55</b>
4.1	A Categorization of the AutoML Methods . . . . .	57
4.2	Related Work on AutoML Methods for Selecting and Configuring ML Pipelines . . . . .	58
4.3	Related Work on Automated Multi-Label Classification . . . . .	64



4.4	Final Remarks . . . . .	66
<b>5</b>	<b>AutoML Methods for Multi-label Classification</b>	<b>67</b>
5.1	General Framework . . . . .	67
5.2	Search Spaces for Automated Multi-Label Classification . . . . .	68
5.2.1	Components of the Search Spaces . . . . .	69
5.2.1.1	Multi-label Classification Algorithms . . . . .	69
5.2.1.2	Single-label Classification Algorithms . . . . .	70
5.2.2	Search Space Structure and Size . . . . .	70
5.2.3	A Detailed Description of the MLC Search Space . . . . .	73
5.3	Search Methods for Automated Multi-Label Classification . . . . .	82
5.3.1	Evolutionary-based Methods . . . . .	82
5.3.1.1	MLC Fitness Evaluation Process . . . . .	84
5.3.1.2	Genetic Algorithm for Automated Multi-Label Classification	84
5.3.1.3	Automated Multi-Label Classification using Grammar-based Genetic Programming . . . . .	87
5.3.1.4	Automated Multi-Label Classification using Specialized Grammar-Based Genetic Programming . . . . .	89
5.3.2	Automated Multi-Label Classification using Bayesian Optimization	92
5.3.2.1	Quality Function . . . . .	94
5.4	Multi-fidelity Methods for MLC . . . . .	94
5.5	Final Remarks . . . . .	97
<b>6</b>	<b>Experimental Analysis</b>	<b>98</b>
6.1	Experimental Setup . . . . .	99
6.1.1	Datasets . . . . .	99
6.1.2	Parameter Setting . . . . .	100
6.1.3	Baseline Methods . . . . .	101
6.1.4	Statistical Comparisons . . . . .	102
6.2	Preliminary Comparison of the Proposed Methods . . . . .	103
6.3	Auto-MEKA <sub>spGGP</sub> 's Hyper-Parameter Tuning . . . . .	105
6.3.1	Resampling the Training Set . . . . .	106
6.3.2	The Multi-Fidelity Approach . . . . .	107
6.3.3	Tuning the Inter/Intra-Species Crossover Probability . . . . .	109
6.4	Experimental Results . . . . .	111
6.4.1	Final Remarks . . . . .	117
6.5	Analysis of the Diversity of the Selected Algorithms . . . . .	119
6.5.1	Final Remarks . . . . .	124
6.6	Analysis of Convergence . . . . .	125
6.6.1	Final Remarks . . . . .	129

<b>7</b>	<b>Conclusions and Future Work</b>	<b>130</b>
7.1	Issue 1: Proposing AutoML Methods for the Multi-label Classification Context . . . . .	130
7.2	Issue 2: Presence of an Exploration-Exploitation Trade-off in AutoML Methods . . . . .	131
7.3	Issue 3: The Impact of Constrained Time Budgets on the Performance of AutoML Methods . . . . .	133
7.4	Final Remarks . . . . .	133
7.5	Publications . . . . .	134
	<b>Bibliography</b>	<b>136</b>
	<b>Appendix A Multi-Label Classification Search Space in the MEKA Software</b>	<b>157</b>
A.1	Search Space – Algorithms from WEKA . . . . .	157
A.1.1	C4.5 . . . . .	157
A.1.2	Logistic Model Trees . . . . .	159
A.1.3	Decision Stump . . . . .	161
A.1.4	Random Forest . . . . .	161
A.1.5	Random Tree . . . . .	162
A.1.6	REPTree . . . . .	163
A.1.7	Decision Table . . . . .	163
A.1.8	JRip . . . . .	164
A.1.9	One Rule . . . . .	165
A.1.10	PART . . . . .	165
A.1.11	Zero Rule . . . . .	166
A.1.12	K-Nearest Neighbors . . . . .	166
A.1.13	K* . . . . .	167
A.1.14	Voted Perceptron . . . . .	168
A.1.15	Multi-Layer Perceptron . . . . .	169
A.1.16	Stochastic Gradient Descent . . . . .	170
A.1.17	Sequential Minimal Optimization . . . . .	171
A.1.18	Logistic Regression . . . . .	172
A.1.19	Simple Logistic . . . . .	173
A.1.20	Naïve Bayes . . . . .	174
A.1.21	Bayesian Network Classifier . . . . .	174
A.1.22	Naïve Bayes Multinomial . . . . .	175
A.2	Search Space – Meta Classification Algorithms from WEKA . . . . .	175
A.2.1	Locally Weighted Learning . . . . .	176
A.2.2	Random Subspace . . . . .	177

A.2.3	Bagging of Single-Label Classifiers . . . . .	177
A.2.4	Random Committee . . . . .	178
A.2.5	Ada Boost M1 . . . . .	179
A.3	Search Space – Preprocessing Algorithms from WEKA . . . . .	180
A.3.1	Attribute Selection Classifier . . . . .	180
A.4	Studying the Search Space of Multi-Label Classification Algorithms . . . . .	181
A.5	Search Space – Multi-Label Classification Algorithms . . . . .	182
A.5.1	Binary Relevance . . . . .	182
A.5.2	The ‘Quick’ Version of Binary Relevance . . . . .	182
A.5.3	Classifier Chain . . . . .	183
A.5.4	The ‘Quick’ Version of Classifier Chains . . . . .	183
A.5.5	Bayesian Classifier Chain . . . . .	184
A.5.6	(Bayes Optimal) Probabilistic Classifier Chain . . . . .	184
A.5.7	Monte-Carlo Classifier Chains . . . . .	185
A.5.8	Population of Monte-Carlo Classifier Chains . . . . .	186
A.5.9	Classifier Trellis . . . . .	187
A.5.10	Conditional Dependency Networks . . . . .	189
A.5.11	Conditional Dependency Trellis . . . . .	189
A.5.12	Four-Class Pairwise Classification . . . . .	191
A.5.13	Ranking and Threshold . . . . .	191
A.5.14	Label Combination . . . . .	191
A.5.15	Pruned Sets . . . . .	192
A.5.16	Pruned Sets with Threshold . . . . .	193
A.5.17	Random k-Label Pruned Sets . . . . .	193
A.5.18	Random k-Label Disjoint Pruned Sets . . . . .	194
A.5.19	Multi-Label Back-Propagation Neural Network . . . . .	195
A.6	Search Space – Multi-Label Meta Classification Algorithms . . . . .	196
A.6.1	Subset Mapper . . . . .	197
A.6.2	Bagging of Multi-Label Classifiers . . . . .	197
A.6.3	Bagging of Multi-Label Classifiers with Duplicates . . . . .	198
A.6.4	Ensemble of Multi-Label Classifiers . . . . .	199
A.6.5	Random Subspace Multi-Label . . . . .	200
A.6.6	Expectation Maximization . . . . .	201
A.6.7	Classification Maximization . . . . .	201

# Chapter 1

## Introduction

We are experiencing the era of data. With its extensive availability, people in general (e.g., practitioners, data scientists, enthusiasts, students, and researchers) are trying hard to extract useful information encoded on it [191]. This resulted in outgrowing popularity and the indiscriminate use of Machine Learning (ML) algorithms and data analytics techniques, which are the base approaches of data science. Consequently, the number of people self-entitled as data scientists has more than doubled between 2011 and 2015 [153]. And we continue to observe an inherent and logical expansion of the field of data science.

Accordingly, the increasing number of people in different institutions (e.g., companies, government, universities, hospitals, and others) trying to understand their data certainly brought more investments to the field of ML. But it also created a natural *conflicting issue*: it is unrealistic to think these people will have the same experience and knowledge regarding ML, programming languages, data analytics, and other common topics in data science. There is a great variety of algorithms, methods, models, and approaches used in data science. Hence, it is difficult for all these people to know all the details and inner peculiarities about them.

Furthermore, given the emerging problems that can use ML as a solution (at least, partially), the supply of data scientists capable of building such ML models to solve these problems has not kept up with the demand [184]. In this context, developing off-the-shelf solutions to assist different types of users became more relevant than ever.

The field of Automated Machine Learning (AutoML) [109] has emerged to deal with the aforementioned issues. This field aims to democratize ML in a way that can be used with fewer difficulties by general audiences [109]. In addition, AutoML also aims to assist experienced data scientists. In both scenarios, the field of AutoML has the scope of recommending learning algorithms (together with hyper-parameters or not) when people face a particular problem that might be (partially or totally) solved with ML.

AutoML consists of an end-to-end process where data is given as the input, and the model's predictions are given as outputs. With the great variety of proposed ML algorithms (and, consequently, the enormous variety and complexity of hyper-parameters to control each algorithm's behavior), this process seemed impractical to the users. AutoML methods tackled this problem by searching, executing, evaluating, and optimizing ML al-

---

gorithms (with a specific and/or default hyper-parameter setting) on the input dataset. In the end, the AutoML method returns the best ML algorithm (according to some predictive measure) on the input dataset. Tuning the hyper-parameter of these algorithms is an option, but several AutoML works dedicate their efforts only to choosing and recommending the learning algorithms [23, 109, 208], whereas others focus on selecting ML algorithms together with the configuration of their respective hyper-parameters [64, 73, 109]. Nevertheless, this whole process is transparent to the user, who does not need to worry about each decision (only if wanted) to make about each ML algorithm and its hyper-parameters to create an effective ML model.

Based on that, it is also worth mentioning that solving the tasks performed automatically by AutoML is considered hard even for experts, who usually follow *ad-hoc* approaches to choose and configure learning algorithms. In most cases, such decisions on these approaches are based on trial and error when testing different methods from the literature or on the recommendation of other experienced data scientists. Furthermore, the algorithm’s hyper-parameters are rarely deeply explored to achieve the best algorithm’s performance for the given problem, basically testing a few numbers of combinations or using their default settings. This scenario makes ML solutions overall biased, incomplete, and inefficient. Broadly speaking, the field AutoML proposes to deal with this user’s biases by customizing the solutions (in terms of algorithms and configurations) to ML problems following different approaches.

This work, in particular, is interested in AutoML methods for classification. Classification is one of the most essential tasks in ML [220]. In a traditional (single-label) classification problem, the goal is to learn a model that expresses the relationships between a set of predictive attributes (features describing the example) and a predefined set of class labels. A discrete value represents each class label. In a traditional Single-Label Classification (SLC) problem, each dataset example is associated with a single label.

However, an increasing number of applications require associating an example to more than one class label, including medical diagnosis, gene function prediction, image and video annotation, and tag suggestion for text mining [87]. For example, in the context of medical diagnosis, a patient can be associated with one or more diseases (e.g., diabetes, pancreatic cancer, and hypertension) simultaneously. In the case of gene function prediction, a gene in a protein can present several functions simultaneously. In an image (or video frame), we can identify several objects or types of landscape at once. Finally, a text can be concurrently associated with several subjects (e.g., soccer, politics, health, and other topics).

This classification scenario is known as Multi-Label Classification (MLC) [204] and is considered a more challenging problem for the ML community. The challenges in MLC are described as follows. First, the MLC algorithm needs to consider the label correlations (i.e., detecting if they exist or not) to learn a model that produces accurate classification

results [227]. Second, the limited number of examples for each class label in the dataset makes the generalization for MLC harder than SLC, as the MLC algorithm needs more samples to create a good model from such complex data [58]. Third, there is a strain to evaluate MLC classifiers as several metrics follow contrasting aspects to define a good MLC prediction [162]. Finally, the learning algorithms applied to solve MLC problems need more computational resources than the ones used to solve SLC [101]. This aspect is mainly because MLC is a generalization of SLC, meaning that the algorithms need to look at several labels instead of just one.

In the context of AutoML, most systems proposed to date focus on automating the generation of sequences of steps (i.e., ML pipelines<sup>1</sup>) to solve SLC problems [50, 75, 131, 144, 155, 200, 212]. Some of these works also concentrate on regression problems [75, 155, 200]. Nevertheless, to the best of our knowledge, only a few studies are associated with modern AutoML for MLC [179, 211].

Despite the challenges found in MLC, we justify that this step is needed to continue bringing progress to AutoML. Furthermore, we would like not only to create novel methods but also to make them ready to be used in different types of data. Therefore, this thesis investigates AutoML methods for MLC. Our central hypothesis is that if AutoML is successfully used for SLC and regression problems, it can also be appropriately translated and applied to MLC problems.

We conduct our investigation based on two well-known methods, i.e., Evolutionary Algorithms (EAs) [10, 62], and Bayesian Optimization (BO) [13, 108]. Algorithmically, both methods run a global procedure over the search space representing the solutions (in our case, the MLC configurations). EAs are inspired by the Darwinian evolutionary process and the survival of the fittest to search for or optimize solutions to specific problems. To do that, they use biological abstractions of heredity, natural selection, recombination, and mutation as an engine for the search itself. An EA works with a population of candidate solutions (individuals) to the problem at hand (in our case, MLC algorithms with a specific set of hyper-parameters) and employs an iterative process (namely evolution) to find approximate solutions to the problem.

BO, in contrast, is usually applied to problems where the evaluation functions are naturally expensive, which is an inherent aspect of MLC. BO methods approximate these functions with a cheaper surrogate to overcome this complexity. A traditional BO method relies on a so-called response surface (performance) model. Usually, a regression model takes place to predict the performance of configured algorithms and, consequently, to assist in their optimization. In essence, BO methods concentrate on iterating between constructing a performance model (with promising MLC configurations, in our case) and

---

<sup>1</sup>An ML pipeline is a sequence of tasks to be employed on data associated with an ML problem. It can include preprocessing steps (e.g., data cleaning, data discretization and feature selection [216]), a machine learning model (such as a classifier or a regressor [148]), and post-processing steps that may help to combine the results of several ML models (for instance, a voting method [216]).

selecting additional data to improve the model’s performance (in terms of better selection of MLC configurations).

## 1.1 Motivation

The AutoML research field has extensively used several types of methods to solve specific SLC and regression problems, including BO [75, 200], EA [50, 131, 155], random search [13, 14], multi-armed bandits [135], hierarchical planning [144, 212], and hybrid approaches [197]. As previously stated, this thesis pays particular attention to the first two types of methods (i.e., BO and EA) in MLC problems. These two types of methods have been chosen because they are considered state-of-the-art in several AutoML tasks.

In spite of that, there are a few open issues regarding AutoML methods and problems, in general, which we will investigate. From these investigated issues, we derive our research questions (see Section 1.2). Next, we describe each of the investigated issues and how they are linked to this thesis.

**Issue 1.** AutoML methods can work for MLC problems as well as for SLC and regression problems.

Most AutoML methods deal with hierarchical search, where the learning algorithm is first chosen and, thereafter, its hyper-parameters. Hence, the hierarchy is used to compose the AutoML configurations, i.e., which algorithm and hyper-parameter setting to use to achieve good predictive performance.

In the case of AutoML for SLC and regression problems, we have at most two algorithmic levels: the SLC (base) level and the meta/ensemble level. Whereas the former has algorithms that can be applied individually, the latter has algorithms that combine several base learning algorithms aiming to improve their individual results. During the AutoML search, the meta/ensemble level can be turned on or off in different configurations. After selecting the algorithms, their hyper-parameters are set in the subsequent levels.

In the case of AutoML for MLC, the problem is considered more complex as we can have more hierarchical levels. As detailed later (for more details, see Chapter 3), we can divide the MLC algorithms into three categories: problem transformation, algorithm adaptation, and meta/ensemble methods. It is worth noting that the first category (i.e., problem transformation) defines algorithms that transform an MLC problem into one or more SLC problems. Internally, this means this type of MLC algorithm will (somehow) use

the output of one or several SLC classifiers (that can be a traditional or a meta/ensemble SLC classifier) to create an MLC output.

As this first category of MLC must use SLC algorithms to accomplish its respective task, we would have, in this case at most, four algorithmic levels: the MLC (base) level, the MLC meta/ensemble level, the SLC (base) level, and SLC meta/ensemble level. For each level of the hierarchy, an AutoML method has to handle its essential components (i.e., the functional parts of the algorithm) and its respective hyper-parameter settings in their respective subsequent levels of the hierarchy. Besides, the AutoML method has to decide if the MLC and SLC meta/ensemble levels will take part of the MLC configuration. In the same way, the SLC and MLC meta/ensemble levels could be activated or not, depending on how the search proceeds. In contrast, we would have at most two algorithmic levels when using the algorithm adaptation category: the MLC (base) level and the MLC meta/ensemble level.

Figure 1.1 illustrates these possible levels of the hierarchy in one practical example. For this figure, suppose that among all possible algorithms, the AutoML method decides to select the classifier chain algorithm [175] at the MLC (base) level and the random forest algorithm [25] at the SLC (base) level. Whereas the former does not present hyper-parameters, the latter has hyper-parameters that should be configured by the AutoML method, including the number of features to subsample and the maximum depth of the trees. In addition, the AutoML method may opt for activating or not the MLC and SLC meta/ensemble levels. When the MLC meta/ensemble level is turned on, algorithms such as bagging of multi-label classifiers [169] and multi-label random subspace [25] have to be considered in the process. When the SLC meta/ensemble level is activated, algorithms such as Ada Boost M1 [82] and bagging [24] should be present in the final MLC configuration *iff.* they improve the performance of this configuration. For each algorithm at these levels, hyper-parameters such as the bag size percentage and the number of iterations (classifiers) should be tuned. For more details about the aforementioned algorithms and hyper-parameters, see Appendix A.

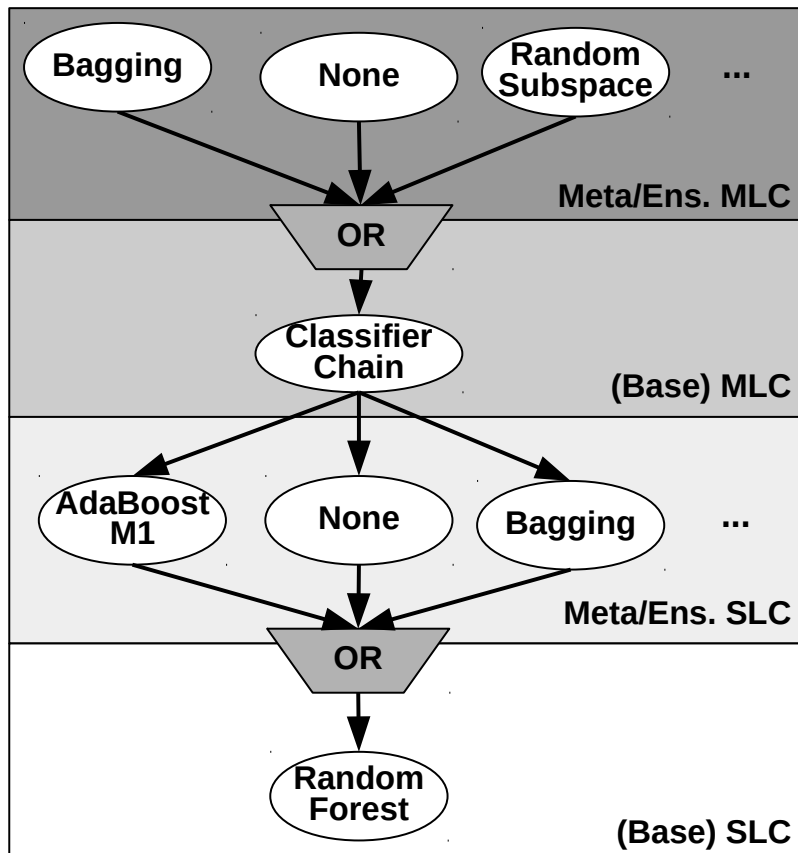
These different ways to compose the MLC hierarchy make the AutoML task more difficult. The method that performs this task must appropriately define more or fewer levels at different time frames. Besides, as MLC encompasses SLC in some categories of algorithms, the AutoML methods are looking for a more significant number of learning algorithms and, consequently, hyper-parameters.

**Issue 2.** AutoML methods do not trade-off well between exploration and exploitation.

Depending on the approach the AutoML method follows, the size and definition of its search space can highly influence the results obtained. This is because what is searched and/or optimized by an AutoML method represents how well this method performs its



Figure 1.1: The possible hierarchy levels in MLC.



task. In practice, the bigger the search space, the harder the search/optimization is. To properly search/optimize over bigger search spaces, the AutoML method should settle an appropriate trade-off between exploration (the search is focused on untested regions of the search space) and exploitation (the search concentrates on the neighborhood of known solutions) to select algorithms with good predictive results. Of course, this balance depends on the AutoML problem (i.e., the search space) and the AutoML method.

This investigation becomes particularly important as several methods in the AutoML literature presented results similar or not statistically different to the ones obtained by blind heuristic search methods (e.g., random search and beam search) [13, 14, 50, 155, 135]. This is more likely to happen in smaller search spaces. Nevertheless, if blind non-heuristic search methods can beat robust AutoML methods, this indicates that the AutoML methods do not produce a reasonable exploration-exploitation trade-off.

Hence, analyzing how good AutoML methods perform in search spaces with different sizes could indicate why and when blind non-heuristic search methods perform better than robust AutoML methods. After this analysis, the search or optimization mechanisms used by the AutoML methods could also be better understood and possibly modified to ensure the aforementioned trade-off satisfactorily.

Finally, we can associate this issue with two contrasting properties of genetic pro-

gramming [11], namely sufficiency and parsimony. Sufficiency means that the search space should be powerful enough to represent a candidate solution (in our case, an MLC algorithm) to the problem at hand. This power of expression is mandatory in AutoML, where we must find a suitably configured learning algorithm for a dataset of interest. However, as we already mentioned, if we enlarge too much the search space, this can make the search for an appropriate solution harder. Thus, in the case of AutoML, it implies that it is unnecessary to include all learning algorithms and hyper-parameters in the search space that may solve the problem, just the ones that matter (i.e., the ones that can perform satisfactorily well in accordance to a quality metric). This desirable property is called parsimony and means that the search space should contain only necessary solutions to solve the problem.

**Issue 3.** AutoML methods are budget-constrained, which can significantly influence their performances regarding the recommendation of learning algorithms and hyper-parameters.

ML is considered a very hard task. Looking at an important theoretical side of the problem, we can say, according to the No Free Lunch (NFL) theorem [217, 218]<sup>2</sup>, that the choice of which ML algorithm is the best for a given problem is still an open problem, even with the undeniable progression in the fields of AutoML (and meta-learning). The NFL theorem states that any two algorithms are equivalent when their performances are averaged based on all possible problems. In ML, each problem is associated with a dataset of interest. Given the NFL theorem and considering all possible datasets, it is not practical to find an ML algorithm (with its corresponding hyper-parameters) that could perform the best, on average, for all of them. Nevertheless, this will likely happen for specific ML problems (i.e., applications). Hence, this also justifies the use and study of AutoML because a practitioner is not interested in all ML problems but actually in particular applications.

Based on the diversity of the ML problems, we could have algorithms that take a short time to run and algorithms that take a long time to run. Nevertheless, it is not rare when the combination of algorithm and problem produces an impractical model to train (in terms of time and memory). For instance, when we have a dataset with thousands of attributes and millions of examples, and we decide to use an ML algorithm that depends on these two characteristics (e.g., Bayesian network classifier algorithms [15]), this is likely to happen. Hence, the AutoML method should avoid suggesting such algorithms in the aforementioned situations.

At this issue, we would like to investigate which time budget we should apply to each evaluated ML algorithm during the AutoML process to avoid that scenario and how

---

<sup>2</sup>This theorem was brought to the machine learning field into the law of conservation for generalization performance [90, 185].

this execution aspect can influence AutoML’s results in terms of selected algorithms and predictive accuracy. We believe that setting shorter time budgets would make the chosen algorithms to naturally create less effective models. In contrast, more extended time budgets may lead the AutoML methods to choose ML algorithms with a higher chance of producing models that overfit. For an AutoML method, the trade-off between these two scenarios is also relevant, making this study essential.

In addition, AutoML will work only if enough time budget is given to complete the AutoML process. In this context, this issue also relates to the minimal time budget to effectively proceed with a complete AutoML search/optimization process. Two aspects can influence this decision: (i) the input dataset, which represents the ML problem; (ii) the covered search space, which is associated with the AutoML problem itself. Therefore, we would like to examine how AutoML’s execution time budget can influence the respective method’s performance in terms of predictive accuracy and coverage of the search space. If the AutoML method is not stable enough, this budget could drastically affect its results, selecting algorithms with few conditions to produce models that generalize to the input data.

Although it is crucial to investigate these aspects, the literature in AutoML is more concerned with creating novel methods than analyzing why the already proposed ones presented good or bad behaviors in specific scenarios. This study of running the proposed methods using distinct time budgets (for the AutoML method and each algorithm’s configuration) will help us assess and understand the classification performance of these methods in other scenarios. Consequently, this can provide us with more reliable information regarding which method performs better.

## 1.2 Objectives

The overall objective of this thesis is to investigate the open issues in the AutoML field and to propose new methods to overcome their main associated challenges. More specifically, we would like to answer the following questions, corresponding to the issues presented in the previous section:

- **Research Question 1 (RQ1):** Can we propose novel AutoML methods to handle the complex hierarchical complexity of the MLC search space? (Relates to Issue 1)
- **Research Question 2 (RQ2):** How do the sizes of the search spaces affect the predictive performances of the AutoML methods? How can we enhance the search and

optimization mechanisms to promote an adequate exploration-exploitation trade-off? (Relates to Issue 2)

- **Research Question 3 (RQ3):** How do distinct time budgets (constraining the whole AutoML process) influence the behavior of AutoML methods and, consequently, the respective learning performances of the selected and configured learning algorithms? (Relates to Issue 3)

## 1.3 Contributions

Based on the defined issues and research questions, the main expected contributions of this thesis are:

1. Formalize AutoML for multi-label classification;
2. Propose novel AutoML methods for the context of multi-label classification;
3. Model MLC search spaces for the context of AutoML;
4. Evaluate whether the proposed AutoML methods can deal with the complex hierarchical nature of the MLC search spaces;
5. Understand how the size of the search space influences the performance of the AutoML search methods;
6. Comprehend how distinct time budgets applied to the AutoML methods can affect the final predictive performance of the selected and configured MLC algorithms.

## 1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapters 2 and 3 present the fundamental concepts regarding search and optimization methods and multi-label classification. Chapter 4 introduces the main AutoML methods and also the related work to AutoML for multi-label classification. Chapter 5, in turn, details the four proposed methods, showing that it is possible to create AutoML in the MLC context (RQ1). The

---

experimental results and analysis of the generated MLC algorithms are presented in Chapter 6, supporting the fulfilment of RQ1 and completely answering RQ2 and RQ3. Finally, Chapter 7 draws conclusions and discusses the steps necessary to completely cover the topics raised by RQ1, RQ2, and RQ3.

# Chapter 2

## Search and Optimization Methods

This chapter defines the fundamental concepts of search and optimization methods that are commonly and successfully used to solve AutoML problems, i.e., evolutionary and Bayesian optimization algorithms. It is worth mentioning that evolutionary and Bayesian optimization algorithms were chosen to solve the AutoML task because they are considered in the literature state-of-the-art methods to perform global search and optimization [62, 81, 109], respectively. Nevertheless, other methods could also be applied to this work without a lack of generalizability.

Section 2.1 overviews evolutionary search methods, covering their main framework (i.e., evolutionary algorithms) and specific methods, such as genetic algorithms, canonical genetic programming, and grammar-based genetic programming. Next, in Section 2.2, Bayesian optimization algorithms are introduced and have their main methods discussed, i.e., tree-structured Parzen estimator and sequential model-based algorithm configuration.

### 2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [10, 62] are a category of computational intelligence search methods that simulate the Darwinian evolutionary process. Basically, an EA aims to stochastically find approximate solutions in search or optimization problems. They are based on biological abstractions of heredity, natural selection, recombination, and mutation. This means that these biological abstractions will be used as an engine of evolutionary-based search methods.

Considering these biological aspects, Eiben and Smith [62] discuss the eight main components of an EA, which consequently define its main characteristics:

1. **Population:** In these algorithms, the concept of population is used to define the solution space to a given problem. The population is composed of a set of individuals, which represents the candidate's solutions to the problem at hand. The main

idea of an EA is to evolve these individuals through its iterations (also known as generations).

2. **Representation:** This component defines the individuals of a population. In an EA, the individuals can be represented in different ways, such as: trees (in the case of genetic programming algorithms); binary, integer, or real-coded arrays (in the case of genetic algorithms); real-valued arrays (in the case of evolutionary strategy algorithms); and others. The representation (and, consequently, the instantiation) of the individual in the EA will be hardly dependent on the problem to be modeled.
3. **Evaluation function:** The evaluation function (which is also known as fitness function) has the role of determining how good each individual of the population is at solving the input problem.
4. **Parent selection mechanism:** This mechanism is used to distinguish among the individuals in the population based on their fitness, i.e., their quality with respect to the problem. Particularly, this mechanism plays an important role in the EA, allowing the higher-ranked individuals (in terms of the employed fitness function) to become the parents of the individuals in the next generation. Parent selection is usually probabilistic in the sense that individuals with better fitness values have a higher probability of being chosen. Finally, it usually does not depend on the individual representation, but it inherently depends on the evaluation function.
5. **Variation operators:** Mutation and recombination are considered the two principal variation operators in EAs. These operators are also probabilistic and are responsible for creating new individuals from old ones. On the one hand, mutation, a unary operator, determines that an individual suffers modifications in specific regions of its genetic material. This brings novelty to the population of the next generation. On the other hand, recombination (or crossover), a binary operator, combines the genetic material of two individuals, generating one or two offspring.
6. **Survivor selection mechanism:** Also known as the replacement mechanism, it is called after producing the offspring of the selected parents. Replacement is used to decide which individuals of the population will be substituted by the produced offspring. Usually, this process is performed based on the fitness or age of the current individuals.
7. **Initialization:** Most EAs keep this step as simple as possible, generating the first population at random. However, heuristics can also be used to create a population with better average fitness or with a desirable feature.
8. **Termination condition:** It defines the criterion used to stop the evolutionary process to return the best-evolved individual to the problem. For example, if the

problem has an optimal solution, the EA can set a quality level to achieve. The number of generations and the number of evaluations are also known as instantiations of the termination condition. Finally, EAs can combine several termination conditions to help at providing good but also feasible solutions to hard problems.

Algorithm 2.1 denotes these main components [81]. It represents only the EA's basic steps, allowing several differences in methods proposed in the literature. For instance, crossover and mutation operators can be applied in parallel, instead of being performed in a sequenced fashion. It is also worth mentioning that the way the individual is represented is implicit in the algorithm, but crucially inherent to the problem at hand. Furthermore, an EA has several parameters to be defined, such as population size, crossover probability, mutation probability, parent selection approach (e.g., tournament or roulette wheel, and their respective parameters), stopping criteria (e.g., number of generations, quality level, and time budget), and others.

Initially, Algorithm 2.1 receives as inputs the population size (i.e.,  $S$ ), a parent selection mechanism (i.e.,  $M$ ), the crossover probability (i.e.,  $P_c$ ), the mutation probability (i.e.,  $P_m$ ), a fitness function (i.e.,  $f$ ), and a stopping criterion (i.e.,  $SC$ ). Given the individual representation, an initial population of  $S$  individuals is created (line 2). Whilst the stopping criteria  $SC$  is not reached, the evolutionary algorithm iterates by evaluating individuals  $i$  based on the fitness function  $f$  (line 4), selecting the evaluated individuals (line 5) with  $M$  (which is based on fitness), applying the variation operators (lines 6 and 7) to create new individuals, and updating the population with these created individuals (line 8). In the end, the EA will return the best individual with respect to the fitness function.

---

**Algorithm 2.1** A generic pseudo-code for evolutionary algorithms.

---

- 1: **Inputs:** Population size,  $S$ ; parent selection mechanism,  $M$ ; crossover probability,  $P_c$ ; mutation probability,  $P_m$ ; fitness function,  $f$ ; stopping criteria,  $SC$ .
  - 2: Create an initial population  $P$  of  $S$  individuals;
  - 3: **while** stopping criteria  $SC$  not reached **do**
  - 4:    $f(i)$ : Calculate the fitness of each individual  $i$  in  $P$ ;
  - 5:    $[SI]$ : Select individuals using  $M$  based on fitness;
  - 6:   Crossover( $[SI]$ ,  $P_c$ );
  - 7:   Mutation( $[SI]$ ,  $P_m$ );
  - 8:   Update the current population (new individuals replace old individuals);
  - 9: **end while**
  - 10: **return** the individual  $i$  with the best fitness value.
- 

EAs are usually well-known to present a balanced trade-off between exploration (i.e., emphasis on searching in not tested regions of the search space) and exploitation (i.e., emphasis on searching over the neighborhood of the known good solutions), although



this is not universally accepted by the EA community [61]. Whereas too much exploration would lead to an inefficient search as its behavior would be practically random, too much exploitation would lead to premature convergence, as the population would lose its diversity quickly.

### 2.1.1 Genetic Algorithm

The canonical Genetic Algorithm (GA) was proposed by Holland [105], who aimed to study adaptive behavior in natural and artificial systems. In Holland's proposed GA, the individual is represented by a binary string with fixed length [62]. Apart from its canonical version, other traditional binary representations have been proposed for GA. Given a new problem at hand that can be modeled by a GA, it is important to find an appropriate length for the individual genotype (in this case, the size of the array of bits) and how to decode it (from genotype to phenotype).

Integer or real-valued representations are also possible choices for GAs [62]. These two types of representations are used when the binary one is not suitable to define the problem search space. For instance, if the problem is to find the optimal values of categorical variables, which all take integer values, the integer GA is more appropriate.

Following Algorithm 2.1, the population (of individuals) is commonly initialized at random. With the population defined, its individuals can pass through evaluation and, thereafter, selection. For selection, mechanisms such as the roulette wheel and tournament are often employed. The roulette wheel mechanism proportionally selects individuals based on their fitness. Hence, given a population with  $N$  individuals and a particular individual  $i$  with fitness  $f_i$ , the roulette wheel selects this individual with probability  $p_i = \frac{f_i}{\sum_{i=1}^N f_j}$ . Differently, the tournament mechanism chooses  $k$  individuals from the population, ranks them by their respective fitness, and returns the individual with the best fitness value. For more details about these mechanisms, see [62].

It is worth noting that, in the binary GA, the variation operators are focused on recombination, consequently giving low emphasis (i.e., low probability) to mutating the genes of the individuals (i.e., flipping the bits of the array). The most common recombination approaches for this GA are the one-point crossover and the uniform crossover. Given two selected individuals from the population, the one-point crossover randomly chooses a position in the array, and the right and left parts (based on the chosen position) of the arrays of the two individuals are exchanged. The uniform crossover [199], in contrast, builds a uniformly distributed binary mask the size of the individual's genotype. A mask value of one (1) means that exchanges will occur in that gene position, while a value of

zero (0) does not modify the content of the corresponding gene.

In cases where the mutation is performed, the binary-coded GA uses the bit-wise approach. It considers each gene individually, allowing each bit to flip with a small probability. This means that the number of bits changed is not fixed and depends on the random numbers generated [62].

In the case of integer and real-coded GAs, we can still use the aforementioned crossover operators, i.e., one-point and uniform crossover operators. In the case of mutation, the one-point approach is commonly used. The one-point mutation is applied to one of the possible positions of the array (i.e., the genes of the individual). Each gene has the same probability of being selected, and the value of the selected gene is replaced by a value randomly chosen within its domain (which can vary with the problem).

## 2.1.2 Genetic Programming

Genetic Programming (GP) [11, 62, 130] is another EA-based algorithm initially created to evolve computer programs, i.e., executable codes. In GP, individuals are represented by parse trees, which map expressions (functions) into a formal syntax. Arithmetic functions and programming codes are examples of possible individual's phenotypes in GP. Equation 2.1 gives an example of an arithmetic formula, and Algorithm 2.2 specifies an example of a programming code. The possible parse trees of these two examples are illustrated in Figures 2.1a and 2.1b, respectively.

$$(7.0 \times x \div 80.0) \times (4.0 \times x \div 7.0) \quad (2.1)$$

---

**Algorithm 2.2** Example of a programming code.

---

```

i = -100;
while (i < 5) do
  i = i + 1
end while

```

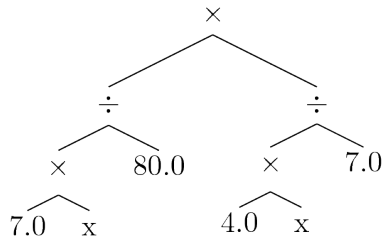
---

These parse trees are defined by two sets: the function set and the terminal set. The elements in the terminal set are allowed to be only at the leaves of the trees, whereas the elements of the function set are the internal nodes. Each one of the function set's elements (i.e., the function symbols) has an arity, which is the number of arguments these function symbols need to be properly specified. A function symbol can have as arguments terminal symbols or even expressions composed by other function and terminal symbols.

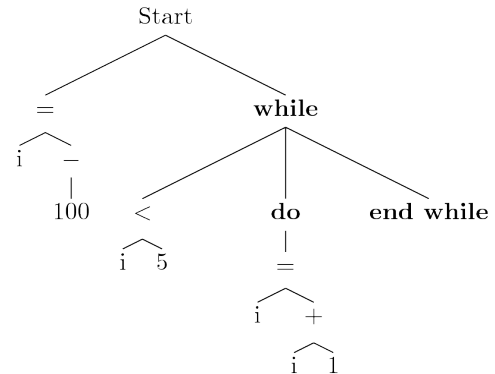
It is highly recommended that each function symbol accepts as input any terminal symbol or even any output produced by another function symbol. This is better known

Figure 2.1: Examples of parse trees for Equation 2.1 and Algorithm 2.2.

(a) Parse tree for Equation 2.1.



(b) Parse tree for Algorithm 2.2.



as the closure property. GP needs this property to allow operators, such as crossover and mutation, to be correctly performed.

After defining the representation and its properties, a typical EA initializes its respective population. In GP, initialization is typically performed by the three following methods: *grow*, *full*, and *ramped half-and-half*. All three methods consider a tree maximum depth  $D_{max}$  in different ways. The *grow* method constructs trees that have branches with different depths, up to the limit  $D_{max}$ . All tree's branches built by the *full* method have depth  $D_{max}$ . Finally, the *ramped half-and-half* method combines *grow* and *full* methods to promote diversity in the population. In *ramped half-and-half*, the depth of the individuals varies from two (2) to  $D_{max}$ . Given the length of this range,  $L_{range}$ , *ramped half-and-half* divides the population size by  $L_{range}$  and uses this value to create subsets of individuals per depth (with the same number of individuals). For each subset, half of the individuals are initialized using the *grow* method, and the other half using the *full* method.

After generating a population of individuals, they need to be evaluated and selected. Given that all individuals in the population have been associated with a fitness score, this evolutionary process goes into parent selection, where the individuals are selected based on their fitness score. For this reason, roulette wheel or tournament are mechanisms that can also be applied in GP.

Next, GP's mutation is commonly performed by randomly selecting an individual subtree (starting at a given internal node) and replacing it with a randomly generated tree. Crossover is usually implemented using subtree crossover, which means the operator interchanges the subtrees (starting at two randomly chosen internal nodes) of two parents. Both operators must respect the maximum tree depth  $D_{max}$ , which is an input parameter.

### 2.1.3 Grammar-Based Genetic Programming

The major issue of the canonical genetic programming algorithm, presented in the previous section, is that several problems present functions (non-terminals represented by internal nodes in the trees) with typed/specific arguments. I.e., the function symbols may not accept as inputs all terminal symbols and outputs of the remaining function symbols. For instance, in the division function, the denominator must not be zero (0), as this result is not valid. Therefore, this makes it difficult for the canonical GP to respect the closure property in several problems.

Whigham [214] discusses that this is simply overcome by creating a set of constraints in the syntactic structure of the individuals. Nevertheless, when operators are performed, they must respect the syntactic type of the trees to create only valid individuals. For instance, in the crossover operator, when the first internal node is selected in one individual, the second internal node of the other individual must syntactically match. One general way to handle these closure requirements for GP is to use a Context-Free Grammar (CFG) [192] to represent the search space and also to constrain the population initialization and the genetic operators.

Formally, a grammar  $G$  is represented by a four-tuple  $\langle N, T, P, S \rangle$ , where  $N$  represents a set of non-terminals,  $T$  a set of terminals,  $P$  a set of production rules and  $S$  (a member of  $N$ ) the start symbol. In this thesis, we will use the Backus Naur Form (BNF) to represent grammars. This means that each production rule has, for instance, the following form  $\langle Start \rangle ::= [\langle A \rangle] \langle B \rangle \mid \langle C \rangle \ (d \mid e)$ . Symbols wrapped in “ $\langle \rangle$ ” represent non-terminals, whereas terminals (such as  $d$  and  $e$ ) are not bounded by “ $\langle \rangle$ ”. The special symbols “[”, “[”, and “(” represent, respectively, a choice, an optional element, and a set of grouped elements that should be used together. Additionally, the symbol “#” represents a comment in the grammar, i.e., it is ignored by the grammar’s parser. The choice of one among all elements connected by “|” is made using a uniform probability distribution (i.e., all elements are equally likely to occur in an individual).

When using this formalism (i.e., the CFG), the GP algorithm is well-known as CFG-GP or Grammar-based GP (GGP) [141, 214]. Being a version of the canonical GP algorithm, each GGP individual is also represented by a tree, which is derived from the expansion of the production rules of the grammar. To create an executable program (phenotype) from the derivation tree (genotype), a mapping process is performed by taking the terminals from the tree and constructing a valid program from them as it is done in traditional GP. Box 2.1 shows a grammar used to create the tree of Figure 2.1a.

One of the benefits of using this formalism in the GP is that the CFG incorporates into the search space prior knowledge (from specialists) about the problem, properly guiding the search process. In addition, the CFG also gives flexibility in the definition of

Box 2.1: Grammar for generating the tree of Figure 2.1a.

```

<Start> ::= <equation>
<equation> ::= <argument> <operator> (<equation> | <argument>)
<operator> ::= × | ÷
<argument> ::= <float-number> | <variable>
<variable> ::= x | y | z
<float-number> ::= Random float number between 0.0 and 1000.0

```

the search space, as the grammar rules can be modified anytime. Finally, the grammar can introduce semantics along with its syntax, possibly allowing the evaluation of the complexity of the search space.

Taking the CFG into account, initialization, crossover, and mutation procedures for GGP are detailed next. The initialization methods from GP (i.e., *grow*, *full*, and *ramped half-and-half*) are also commonly used in GGP to create the individuals of the first population. The only practical difference is that the individual's genotypes (i.e., the derivation trees) are restricted to the input CFG.

After the population is created, the individuals are evaluated and selected. The selection mechanisms (i.e., roulette wheel and tournament) presented in the previous section can also be used in GGP, as they depend entirely on fitness values.

When compared to GP, the variation operators (i.e., crossover and mutation) need to be changed in GGP to ensure the creation of individuals that are valid according to the CFG. The crossover operator, for instance, depends on seven basic steps to be correctly used by GGP [214]: (i) the first individual represented by a derivation tree,  $t_1$ , is selected; (ii) the second individual,  $t_2$  is selected; (iii) an internal node (non-terminal) is randomly selected in  $t_1$ ; (iv) if no non-terminal from  $t_1$  matches in  $t_2$ , go to step (ii); (v) otherwise, a non-terminal from  $t_2$  is randomly selected; (vi) if this non-terminal does not match to the  $t_1$ 's selected non-terminal, go to step (ii); (vii) otherwise, a swapping operation is applied between the two subtrees below those selected non-terminals. It is worth mentioning that some crossover operations may be discarded as they may not respect the maximum depth  $D_{max}$  of the tree.

Mutation, in turn, is a simpler operator. As discussed by Whigham [214], this GGP operator has to first select a non-terminal (internal node) from a derivation tree. After that, the subtree below this non-terminal is removed, and a new subtree is created using the CFG to replace it. As it happens in the GGP's crossover, the maximum tree depth must be respected.

## 2.2 Bayesian Optimization Algorithms

Bayesian Optimization (BO)<sup>1</sup> – also known as Sequential Model-Based Optimization (SMBO)<sup>2</sup> – is a stochastic black-box technique for optimization. This technique and its algorithms are mainly used to globally configure algorithms in problems where acquiring the evaluation function,  $f : X \rightarrow \mathbb{R}$ , is really expensive [13, 108]. In practice, BO algorithms approximate complex and/or expensive functions with a cheaper surrogate function [13].

The idea of BO is to iterate between constructing a model and selecting additional data to improve the model’s performance [108]. Hence, a typical BO algorithm is based on a so-called response surface model, which is a regression model used to predict performance and, consequently, for optimization. This performance model helps to select the most promising configurations for the next iteration by employing an acquisition (utility) function. In other words, BO fits a surface response model to the data gathered (i.e., to the promising configurations) using an acquisition function.

Given these notions, Algorithm 2.3 defines a generic pseudo-code for BO [13, 200]. First, this algorithm receives as inputs the performance model (i.e.,  $M_L$ ), a time budget (i.e.,  $T$ ), an evaluation (quality) function (i.e.,  $f$ ) that returns a loss value (e.g., the classification error), and an acquisition function (i.e.,  $S$ ). In BO, the Gaussian process [167] is a common approach to model the configurations’ performance. The model is initialized in line 2 of the pseudo-code, by running the target algorithm with few initial parameter configurations [108]. In line 3, the set  $H$  is instantiated as having no elements. This set will track all the evaluated configurations. While there is time to be expended, the algorithm iterates by selecting the most suitable configuration of hyper-parameters  $\lambda^*$  with  $S$  given  $M_L$  (line 5), evaluating this configuration (line 6), adding the configuration and its evaluation performance to  $H$  (line 7), and updating  $M_L$  given  $H$  (line 8). At the end of its sequential process, the BO will return the configuration with the best evaluation performance value.

Algorithm 2.3 addresses the problem of finding promising configurations over an algorithm’s search space of configurations. According to Hutter et al. [108], promising configurations are the ones that can be found in regions where the model is still uncertain, and/or are predicted to perform well. To select these configurations, it is necessary to employ an appropriate acquisition function, which aims to trade-off between exploration and exploitation and, consequently, avoid issues such as premature convergence and lack

<sup>1</sup>Although the term “Bayesian optimization” is highly used in the machine learning community, it is considered a bad term because there is nothing particular Bayesian involved in the optimization process [107].

<sup>2</sup>This term is commonly used in the optimization community.

---

**Algorithm 2.3** A generic pseudo-code of Bayesian optimization.

---

- 1: **Inputs:** Performance model,  $M_L$ ; time budget,  $T$ ; evaluation function,  $f$ ; acquisition function,  $S$ .
  - 2: Initialise  $M_L$ ;
  - 3:  $H = \emptyset$ ; ▷ Set of evaluated configurations
  - 4: **while** optimization time budget  $T$  not exceeded **do**
  - 5:    $\lambda^* = \operatorname{argmax}_{\lambda} S(\lambda, M_L)$ ;
  - 6:   Evaluation:  $c(\lambda) = f(\lambda^*)$ ;
  - 7:    $H = H \cup \{(\lambda^*, c)\}$ ;
  - 8:   Update  $M_L$  given  $H$ ;
  - 9: **end while**
  - 10: **return**  $\lambda$  from  $H$  with the lowest  $c$ .
- 

of diversity of the configurations.

Apart from the existence of several acquisition functions in the literature [116, 118, 187, 195], the positive Expected Improvement (EI) is considered as the most prominent approach [200]. EI quantifies how much improvement we are expected to achieve if we sample at a given configuration [116]. BO simply maximizes this function over the set of all possible configurations  $\Lambda$  to find the most useful configuration  $\lambda^* \in \Lambda$  to evaluate next (line 5 in Algorithm 2.3).

Formally speaking, let  $c_{min}$  be the value of the minimal error loss when performing an evaluation, and let  $c(\lambda)$  be the error loss of the configuration  $\lambda$ . Further, following Thornton et al.’s definitions [200], the positive improvement  $I$  over  $c_{min}$  is formalized in Equation 2.2.

$$I_{c_{min}}(\lambda) = \max\{c_{min} - c(\lambda), 0\} \quad (2.2)$$

As it is impractical to know  $c(\lambda)$ , the expectation with respect to the current model  $M_L$ , computed in Equation 2.3, is a practical alternative.

$$\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)] = \int_{-\infty}^{c_{min}} \max\{c_{min} - c, 0\} \cdot p_{M_L}(c | \lambda) d_c \quad (2.3)$$

where  $p_{M_L}(c | \lambda)$  is the conditional probability distribution of the loss function  $c$  given the configuration  $\lambda$  based on the performance model  $M_L$ .

Bergstra et al. [13] and Thornton et al. [200] argue that one of the main differences among BO algorithms is how they choose the configurations given the model  $M_L$ , i.e., how they express  $\mathbb{E}_{M_L}$  to select the most promising configurations and improve the model’s performance. Next, we review two well-known BO algorithms to perform this task: Tree-structured Parzen Estimator (TPE) and Sequential Model-based Algorithm Configuration (SMAC). Both algorithms deal with a hierarchical structure of configurations, making them capable of handling different types of learning algorithms and hyper-parameters

(e.g., discrete, continuous, and their conditional dependence), which is the main focus of this thesis.

### 2.2.1 Tree-Structured Parzen Estimator

The Tree-structured Parzen Estimator (TPE) algorithm [13] represents the hierarchical nature of the hyper-parameters configurations by a tree structure. Each node of this tree has a 1-D Parzen estimator, which is used to model the node’s corresponding hyper-parameter. To simplify, TPE assumes the independence of the hyper-parameters that do not appear together in any path from the tree’s root to one of its leaves [200].

Formally, TPE models  $p_{M_L}(c | \lambda)$  from Equation 2.3 by using two separate models  $p_{M_L}(c)$  and  $p_{M_L}(\lambda | c)$ . Particularly, TPE expresses  $p_{M_L}(\lambda | c)$  by considering two density estimators, which are conditioned on the value of  $c$  and on a threshold  $t$ . Equation 2.4 shows this TPE expression.

$$p_{M_L}(\lambda | c) = \begin{cases} \ell(\lambda), & \text{if } c < t \\ g(\lambda), & \text{if } c \geq t \end{cases} \quad (2.4)$$

In Equation 2.4,  $\ell(\lambda)$  is a density estimator learned from all previous configurations  $\lambda$  with a loss smaller than the threshold  $t$ . Intuitively, this means  $\ell(\lambda)$  represents good configurations (with respect to the threshold). In the second part of this equation, we have  $g(\lambda)$ , which is a density estimator learned from all previous configurations  $\lambda$  with a loss greater than or equal to the threshold  $t$ . Hence,  $g(\lambda)$  characterizes configurations with poor performance values (with respect to the threshold). In Thornton et al. [200],  $t$  is chosen as the  $\gamma$ -quantile of the losses TPE obtained until the current time (where  $\gamma$  is a parameter with a default value of 0.15).

Bergstra et al. [13] derived  $\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)]$  of Equation 2.3, showing that the EI can be computed in terms of  $\gamma$ ,  $\ell_{M_L}(\lambda)$  and  $g_{M_L}(\lambda)$ . Equation 2.5 defines  $\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)]$  taking this TPE formulation into account.

$$\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)] \propto \left( \gamma + \frac{g(\lambda)}{\ell(\lambda)} \cdot (1 - \gamma) \right)^{-1} \quad (2.5)$$

In this case, TPE maximizes  $\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)]$  by generating several candidate configurations  $\lambda$  at random, and choosing the ones with the smallest value of the rate  $g(\lambda)/\ell(\lambda)$ .



## 2.2.2 Sequential Model-based Algorithm Configuration

Instead of applying separate models to define  $p_{M_L}(c | \lambda)$ , like TPE, the Sequential Model-based Algorithm Configuration (SMAC) algorithm [108] does it directly. In SMAC, the random forest regression model [25] (ML instantiation) is used to capture the dependence between the loss function  $c$  and the hyper-parameter configuration  $\lambda$  in  $p_{M_L}(c | \lambda)$ . Although other algorithms can be used, the authors mention that random forests were used as they tend to perform well with discrete and high-dimensional data.

Usually, random forest is not used as a probabilistic model. Hence, to determine  $p_{M_L}(c | \lambda)$ , SMAC uses a Gaussian  $\mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$ , where  $\mu_\lambda$  and  $\sigma_\lambda^2$  are the mean and the variance, respectively, obtained from frequentist estimates over the predictions of the individual’s trees for  $\lambda$  [200].

Given that  $c_{min}$  from Equation 2.3 is characterized as the error loss of the best hyper-parameter configuration found so far, we can derive this into Equation 2.6 [200]:

$$\mathbb{E}_{M_L}[I_{c_{min}}(\lambda)] = \sigma_\lambda \cdot [u \cdot \Phi(u) + \varphi(u)], \quad (2.6)$$

where  $u = \frac{c_{min} - \mu_\lambda}{\sigma_\lambda}$ , and  $\Phi$  and  $\varphi$  represent the cumulative distribution function and the probability density function of a Gaussian distribution, respectively.

Thornton et al. [200] emphasizes three key ideas used by SMAC. First, SMAC tries to progressively perform better estimates, balancing between accuracy and computational cost. Second, this algorithm implements what is so-called aggressive racing. This means that configurations that have poor performances are discarded as soon as they are evaluated and compared to the current best configuration. Finally, SMAC has a mechanism to promote diversification. That is, every second configuration is selected at random to ensure robustness even when the model is misleading.

## 2.3 Final Remarks

Evolutionary and Bayesian optimization algorithms are both types of methods that employ a global strategy to search or optimize – in this work, both are used for algorithm selection and configuration in machine learning (i.e., for AutoML). However, they are based on different frameworks. Whereas EA-based methods rely on biological abstractions of heredity, natural selection, and survival of the fittest individual, BO-based meth-

ods (such as sequential model-based algorithm configuration and tree-structured Parzen estimator) are based on a performance model and an acquisition function.

Similarly, both BO and EA follow an iterative-based process to perform their respective tasks. Although they have been presenting successful results in terms of selecting and configuring machine learning algorithms [12, 50, 75, 109, 153, 159, 200], which is the main focus of this thesis, it is not clear yet how they perform in search spaces of different sizes. Usually, there is little control in the experiments to evaluate the impact on the method's trade-off between exploration and exploitation based on the size of the search space. Besides, to the best of our knowledge, apart from this thesis, these methods have never been used to select and configure multi-label classification algorithms (the main topic of the next chapter).

Given the No Free Lunch (NFL) theorem for optimization [217, 218], we are aware that there is no single optimization/search method that can outperform all the others when taking into account all optimization/search problems. Based on the great variety of optimization/search methods in the literature (e.g., simulation annealing [121], ant colony optimization [59] and multi-armed bandits [215]), we have decided to proceed with the EA and BO frameworks because of their state-of-the-art AutoML results [109]. Basically, we believe these results can be transposed to AutoML for MLC problems. Furthermore, as AutoML is a specific field and encompasses particular problems, there are more chances for single optimization/search methods (such as EA and BO algorithms) to perform well. Therefore, this thesis investigates the performance of EA and BO methods in tailoring MLC algorithms for a given dataset of interest.

## Chapter 3

# Multi-Label Classification

Classification is one of the most important tasks in Machine Learning (ML) [18, 126, 216]. In a traditional classification problem, the goal is to learn a model that expresses the relationships between a set of predictive attributes (features describing the example) and a predefined set of class labels. Each class label is represented by a discrete value. In the traditional case, each example of the dataset is associated with a single label, which consequently entitles this type of problem to Single-Label Classification (SLC).

More formally, each example from the SLC domain is defined by a tuple  $(X, y)$ , where  $X = \{x_1, \dots, x_d\}$  is a  $d$ -dimensional vector representing the feature space (i.e., the categorical and/or numerical characteristics of that example) and  $y$  is the class value, where  $y \in L$ , a set of disjoint classes. Given that  $|L| > 1$ , if  $|L| = 2$ , the problem is categorized as binary classification. Otherwise, if  $|L| > 2$ , the problem is categorized as a multi-class classification.

There is a wide range of data mining applications that are concerned with solving SLC problems, e.g.: anomaly detection (e.g., classifying a credit card transaction into fraudulent or legitimate) [33], predicting cancer (e.g., classifying a patient into having or not having breast cancer) [46, 129], and text classification (e.g., classifying a text into a topic such as sports, politics, entertainment or commerce) [2].

Nevertheless, there is an increasing number of applications that require associating an example to more than one class label, including medical diagnosis [163, 190], tag suggestion for text classification [119], scene classification [21], and protein function prediction [209]. In the context of medical diagnosis, a patient can be associated with one or more diseases (e.g., diabetes, pancreatic cancer, and/or high blood pressure) at the same time. In tag suggestion, a text document can be associated with several different topics (e.g., medicine, sports, and/or economics). For scene classification, we may have photographs representing scenes that may contain different landscapes (e.g., urban, beach, mountain, and/or desert). Finally, a single protein may often have multiple functions (e.g., cell fate, cell type differentiation, and/or cell cycle and DNA processing).

This classification scenario is better known as Multi-Label Classification (MLC) [87, 136, 202, 204, 227]. According to Tsoumakas et al. [204], MLC is a task concerned with learning a model that returns a bipartition of the set of class labels. Given a query in-

stance, this bipartition separates the labels into relevant and irrelevant. More precisely, each example in MLC is also represented by a tuple  $(X, Y)$ , where  $X$  is the  $d$ -dimensional feature array, and  $Y \subseteq L$  is a set of non-disjoint class labels. Hence, we would like to find a model  $h: X \rightarrow 2^{|L|}$  such that  $h$  maximizes a quality criterion  $\lambda$ .

Besides, MLC is usually associated with the task of Label Ranking (LR) [136]. In LR, the objective is to learn a model that outputs an ordering of the class labels in accordance with their relevance to the query instance. In this case, we would like to find a model  $h: X \rightarrow R$  such that  $f$  maximizes a quality criterion  $\lambda$ , and returns a ranking  $R$  of the labels for a given example. Ideally, we would like to create methods that are capable of producing models that create both a bipartition (in the case of MLC) and an ordering (in the case of LR) of the labels. Such combined task is well-known as Multi-Label Ranking (MLR) [26] and is considered an interesting and useful generalization of both related tasks [204].

Given these characteristics, it is important to emphasize that MLC is considered a more general and challenging problem than SLC. First, the algorithm needs to consider the label correlations (i.e., detecting if they exist or not) in order to learn a model that produces accurate classification results [227]. Second, the limited number of examples for each class label in the dataset makes generalization harder, as the algorithm needs more examples to create a good model from such complex data [58]. Third, the imbalance is a problem that affects MLC even more than it does SLC, as each label may have different distributions and, consequently, locally imbalanced problems [34, 101]. Fourth and finally, there is usually a higher strain to evaluate MLC models as there exist several contrasting evaluation measures from different perspectives that try to express what is an optimal MLC model given the input data [162] (see Section 3.2 for more details).

## 3.1 Categorization of MLC Methods

The majority of works in the literature follow the taxonomy proposed by Tsoumakas and Katakis [202], which divides MLC methods into problem transformation and algorithm adaptation. More recently, some works [87, 136, 146] extended this taxonomy to include ensemble methods as a new category. We follow this extended taxonomy and review methods in these three categories.

### 3.1.1 Problem Transformation Methods

Basically, Problem Transformation (PT) methods transform the multi-label problem into one or more single-label classification problems. Using this concept, it is possible to apply traditional single-label classifiers in order to obtain the classification outputs. Different works discuss the main approaches to perform problem transformation(s) [21, 38, 87, 202, 204, 223].

The simplest PT approaches are *copy*, *copy-weight*, *select-max*, *select-min*, *select-random*, and *ignore*. Given a multi-label example  $(X_i, Y_i)$ , *copy* replaces the multi-label example  $(X_i, Y_i)$  with  $|Y_i|$  single-label examples  $(X_i, \phi_j)$ , for every  $\phi_j \in Y_i$ . Similarly to *copy*, *copy-weight* makes the same replacement, but also adds a weight of  $\frac{1}{|Y_i|}$  to each one of the copied examples. All select approaches, in contrast, replace  $Y_i$  with one of its members. For instance, *select-max* and *select-min* replace  $Y_i$  with the most and the least frequent members of  $Y_i$  among all examples, respectively. The *select-random* approach randomly selects a member of  $|Y_i|$  and makes the replacement. Finally, *ignore* just discards the multi-labeled examples.

Raising the level of complexity of the PT methods, we have the Label Power-set (LP) and Binary Relevance (BR). LP creates a single class for each unique set of labels that are associated with at least one example in a multi-label training set. This means that LP naturally considers all label correlations. However, if a novel labelset appears in the test set, LP will not be able to predict this labelset correctly, presenting just the most probable class labels from the training set. BR, in turn, learns  $|L|$  independent binary classifiers, one for each label in the labelset  $L$ . This implies that BR does not take into account the label correlations, as the binary classifiers are trained separately.

In addition, there are PT methods that only modify the previously cited PT approaches in order to improve their predictive performance or to reduce their learning complexity. For example, the Pruned Sets (PS) method [168, 169, 174] was created to use the power of LP's paradigm without its disadvantages. In order to do this, this algorithm has two important steps: a pruning step and a labelset subsampling step. The pruning step removes infrequently occurring labelsets from the training data. This removes unnecessary complexity from the LP-transformed data by reducing the number of labelsets. Nevertheless, PS does not simply discard the pruned examples. Instead, PS subsamples from these infrequent labelsets, evaluating those subsamples that occur more frequently in the training data. It then attaches these label subsets to these examples (those with infrequent labelsets), creating new modified examples and reintroducing them into the training set.

Following a different approach, Classifier Chain (CC) [175] changes the BR method to take into account label correlations. To do this, CC creates  $|L|$  binary classifiers, like

BR. However, unlike BR, the classifiers in CC are also linked along a chain, where each classifier deals with one BR problem. The attribute space of each link in the chain is increased with the classification outputs of all previous links.

Further, we have Bayesian Classifier Chain (BCC) [221], Probabilistic Classifier Chains (PCC) [52], Monte-Carlo Classifier Chains (MCC and M2CC) [170, 171], and Classifier Trellis (CT) [172], which are all variations of the CC method. The idea behind BCC is to create a maximum spanning tree based on marginal label dependencies, define a Bayesian network from it [15], and then employ a classifier chain using the order of the labels found by the Bayesian network model. The original paper used Naïve Bayes [115] as a base classifier, but other types of base classifiers can be used. PCC, in turn, acts exactly like CC at training time but explores all possible paths as inference at test time. For this reason, PCC is defined as a Bayes optimal method. The methods MCC and M2CC apply classifier chains with Monte-Carlo optimization [56, 71], using a maximum number of inference and chain-order trials. MCC has a tractable label prediction scheme only at test time, whereas M2CC performs an additional search for the optimal chain sequence at training time. Differently, CT builds classifier chains by considering a trellis structure rather than a cascaded chain. In CT, it is possible to set the width and type of connectivity of the trellis structure, and optionally change the payoff function that guides the placement of nodes (labels) within the trellis.

Finally, Conditional Dependency Networks (CDN) [95] build a fully connected undirected network, where each node (which defines the classification of one label) is connected to each other node (which defines the classification of each other label). Hence, CDN applies a binary classifier on each node  $j$  (where  $j \in \{1, \dots, |L|\}$ ), predicting the probability term  $p(y_j|x, y_1, \dots, y_{j-1}, \dots, y_{|L|})$ . Then, the inference is done using the iterative Gibbs Sampling method [89]. Additionally, a final number of iterations is used to collect the marginal probabilities, which become the label predictions.

### 3.1.2 Algorithm Adaptation Methods

Algorithm Adaptation (AA) methods simply extend single-label classification algorithms so they can directly handle multi-label data [87]. In other words, SLC algorithms such as (deep) artificial neural networks, decision trees, support vector machines, and  $k$ -nearest neighbors [216] can be internally modified to perform multi-label classification. Next, we discuss the main AA methods and how they deal with multi-label data.

Let us start with the Multi-Label Back-Propagation Neural Network (ML-BPNN) algorithm [225], which is a standard back-propagation neural network with multiple out-

puts corresponding to (a ranking of the) multiple labels. Each node in the output layer in the model created by ML-BPNN represents a different class label. ML-BPNN is trained with gradient descent with an error function that takes into account the multi-label data. The deep version of ML-BPNN is the Multi-Label Deep Back-Propagation Neural Network (ML-DBPNN) algorithm [173], which uses Restricted Boltzmann Machines (RBMs) or Deep-stacked Boltzmann Machines (DBMs) [103] to pre-train the network, learning a layer of hidden features in an unsupervised fashion. Basically, this layer is plugged into the ML-BPNN model.

Decision Tree (DTree) algorithms from the SLC context have also suffered modifications to perform MLC. E.g., Clare and King [40] carried out an adaptation of the entropy definition of the C4.5 algorithm [166], enabling multiple labels in the tree's leaves. Predictive Clustering Trees (PCT) [16] is another example of a DTree-based algorithm adapted to MLC. The idea of PCT is to build a decision tree by hierarchically clustering the data. The clusters separate tuples of variables to be predicted. In this case, each label is a member of this target tuple.

Finally, Ranking-based Support Vector Machines (Rank-SVM) [63] and Multi-Label  $K$ -Nearest Neighbors (ML-KNN) [224, 226] modify two other classical SLC algorithms for the MLC scenario, i.e., Support Vector Machines (SVM) [44] and  $K$ -Nearest Neighbors (KNN) [45], respectively. In a nutshell, Rank-SVM constructs a linear model that minimizes the ranking loss while trying to maintain a large margin, which is related to SVM's model representation. To do this, this SVM adaptation uses a cost function that is defined as the averaged fraction of incorrectly ordered pairs of labels, i.e., the ranking loss metric [136] (see Section 3.2 for more details about this metric). This makes the strategy of finding the maximum margin to consider the multiple labels.

Contrastingly, ML-KNN, like KNN, first identifies the  $k$  nearest neighbors for the unseen instance. However, unlike KNN, it applies the maximum *a posteriori* principle to define the labelset for this unseen instance, based on statistical frequency information of each label within the  $k$  nearest neighbors. For more details on algorithm adaptation methods, see [87, 101, 204, 227].

### 3.1.3 Ensembles Methods

The last category of multi-label methods concerns ensemble methods [87, 101, 146]. These methods use at their base level multi-label classifiers, such as the ones described in Sections 3.1.1 and 3.1.2. Hence, we have the ensemble method on top of the problem transformation and algorithm adaptation approaches, aiming to combine these multi-

label models to produce a more robust predictive performance. Next, we review the main ensemble methods.

Aiming to improve BR’s performance, we have Meta-BR (MBR, which is also known in the literature as stacked-BR or 2BR) [91] and Ensemble of BR (EBR) [169, 175]. Whereas MBR just stacks the BR method with previous predicted BR label outputs (i.e., the label predictions of a first BR method become the features for a second BR), EBR constructs a bagging [24] of BR classifiers, where each base classifier is trained on the part of the original training set. Although EBR brings more diversity over the MLC base classifiers, it can not deal with label correlations, while MBR does.

Ensemble of Label Powersets (ELP) [146], Ensemble of Pruning Sets (EPS) [174], and Ensemble of Classifier Chains (ECC) [169, 175] are three other examples of bagging of multi-label classifiers, which turn these methods broadly similar to EBR. ELP uses bagging of LPs to produce diversity in the set of LP classifiers, combining their predictions using majority voting. Given this process, ELP is able to predict a labelset that is not present in the training set, an advantage over LP. EPS, in turn, was mainly proposed to reduce the overfitting effects of the PS pruning strategy [146]. A voting process with a threshold (taking into account all PS classifiers in the ensemble) is used to determine the final labelset of a query instance. Finally, ECC tries to overcome the major issue of CC, which is to define the optimal label order for the data at hand. To do that, each CC in the ensemble is trained on a sample of the data with a different label order. As multiple orders are tested, ECC can use the average of the confidence values for each label to create the label bipartition for the test instance.

It is also worth mentioning other classical ensemble methods, such as Random  $k$ -Labelsets (RA $k$ EL) [204, 206] and Hierarchy of Multi-Label Classifiers (HOMER) [203]. RA $k$ EL trains several PS classifiers with a different, small, and fixed number of  $k$  labels, i.e., the taken labelsets for each classifier. The labelsets can be either disjoint (RA $k$ EL $_d$ ) or overlapping (RA $k$ EL $_o$ ), depending on the strategy to construct them. In the end, RA $k$ EL combines the label votes from the PS classifiers to get a label-vector prediction. HOMER also tries to reduce the complexity of performing MLC, like RA $k$ EL. However, it does not do that randomly. Instead, HOMER creates a cluster of labels by using an algorithm called balanced k-means, which is a variant of k-means. Further, given the created clusters, HOMER uses the divide-and-conquer paradigm to transform a large set of labels  $L$  into a tree-shaped hierarchy of simpler MLC tasks, which can be solved using a classical multi-label classifier, such as binary relevance. Given a new query instance to classify, HOMER starts at the root classifier (associated with all labels). Next, it goes to each child node (classifier) only if the parent predicts any of its labels. At the end of the process, HOMER returns the union of the predicted labels at the leaves for the given query instance [146].

Ensembles of algorithm adaptation classifiers are also possible options in MLC.



For example, the Random Forest of ML-C4.5 (RFML-C4.5) [136] and Random Forest of Predictive Clustering Trees (RF-PCT) [122, 123] are MLC ensembles that use the ML-C4.5 and PCT tree models, respectively, as base classifiers. Both methods use two diversification schemes. Whilst the first diversification scheme is based on bagging (i.e., random forest-based approach [24]), the second is based on randomly changing the feature space.

### 3.1.4 Overview of the MLC Algorithms

This subsection overviews the main aspects of the aforementioned MLC algorithms. Table 3.1 summarises the algorithms described in Section 3.1.1. Table 3.2, in turn, encompasses the details of the algorithms presented in Section 3.1.2. Finally, Table 3.3 shows the main aspects of the algorithms reviewed in Section 3.1.3. In these three tables, we present each MLC algorithm based on its name, acronym (if indicated), algorithm’s inspiration (if possible), and main idea.

Table 3.1: Overview of the PT algorithms described in Section 3.1.1.

MLC Algorithm	Acronym	Algorithm’s Inspiration	Main Idea of the Transformation
<i>copy</i>	-	-	For each label $\phi_j$ in the labelset $Y_i$ , it creates a copy of the single-label example $i$ , i.e., $(X_i, \phi_j)$ .
<i>copy-weight</i>	-	<i>copy</i>	For each label $\phi_j$ in the labelset $Y_i$ , it creates a copy of the single-label example $i$ , i.e., $(X_i, \phi_j)$ with a weight equals to $\frac{1}{ Y_i }$ .
<i>select-max</i>	-	-	It replaces $Y_i$ with its most frequent member among all examples.
<i>select-min</i>	-	-	It replaces $Y_i$ with its least frequent member of $Y_i$ among all examples.
<i>select-random</i>	-	-	It replaces $Y_i$ with a random member of its labelset.
<i>ignore</i>	-	-	It ignores the multi-label examples, discarding them.
Label Powerset	LP	-	It creates a single class for each unique labelset that occurs in the training set.
Binary Relevance	BR	-	It builds one independent SLC classifier for each label.
Classifier Chain	CC	BR	It builds one SLC classifier for each label. The classifiers are linked to each other into a chain structure. The output of one classifier serves as a new input for another classifier in the chain.
Pruned Sets	PS	LP	It employs LP but removes the infrequent labelsets that occur in the training set.
Bayesian CC	BCC	CC	It creates a Bayesian network to define the order of the labels, using a CC next.
Probabilistic CCs	PCC	CC	Like CC during training, but it explores all possible order paths during an inference phase at testing time.
Monte-Carlo CC (1st version)	MCC	CC	It is a CC with Monte Carlo optimization to define the optimal label-orders during training time.
Monte-Carlo CC (2nd version)	M2CC	CC and MCC	It is a CC with Monte Carlo optimization to define the optimal label-orders during training and testing times.
Classifier Trellis	CT	CC	Like CC, but it employs a trellis structure instead of just using a cascade chain.
Conditional Dependency Networks	CDN	CC	It builds a fully connected undirected network, where the nodes into the network are labels, making the prediction: $p(y_j   x, y_1, \dots, y_{j-1}, \dots, y_{ L })$ .

Table 3.2: Overview of the AA algorithms described in Section 3.1.2.

MLC Algorithm	Acronym	Algorithm's Inspiration	Main Idea of the Adaptation
Multi-Label Back-Propagation Neural Network	ML-BPNN	Neural Networks	It is trained with an error function that takes into account the multiple labels, where the output layer represents the predictions of them.
Multi-Label Deep Back-Propagation Neural Network	ML-DBPNN	Neural Networks	It uses restricted Boltzmann Machines to learn hidden features, plugging them into the ML-BPNN model.
Multi-Label C4.5	ML-C4.5	Decision Trees	It adapts the entropy definition of the C4.5 algorithm, enabling multiple labels in the tree's leaves.
Predictive Clustering Trees	PCT	Decision Trees	It builds a decision tree by performing hierarchical clustering on data, where the cluster represents tuples of labels.
Ranking-based Support Vector Machines	Rank-SVM	Support Vector Machines	It constructs a linear SVM model that minimizes a specific ranking-based multi-label measure, i.e., the ranking loss.
Multi-Label K-Nearest Neighbors	ML-KNN	Nearest Neighbors	Given the $k$ neighbors, it applies the maximum <i>a posteriori</i> principle to define the labelset for the unseen instance.

Table 3.3: Overview of the ensemble algorithms described in Section 3.1.3.

MLC Algorithm	Acronym	Algorithm's Inspiration	Main Idea of the Ensemble
Meta Binary Relevance, Stacked Binary Relevance or Two Binary Relevance	MBR, Stacked-BR or 2BR	Stacking	In this algorithm, the label predictions of a first BR method are used as the features for a second BR.
Ensemble of Binary Relevance	EBR	Bagging	It is a bagging of BR classifiers, where each classifier is trained with part of the training set.
Ensemble of Label Powerset	ELP	Bagging	It uses bagging of LPs to produce label diversity on this set of classifiers in the ensemble.
Ensemble of Classifier Chain	ECC	Bagging	It employs a bagging of CCs, where each CC is trained with part of the training set and also with a different label ordering.
Ensemble of Pruning Sets	EPS	Bagging	It tries to reduce the overfitting effects of the PS pruning strategy by creating a bagging of PS classifiers, where a voting process with a threshold is used to predict the target labelset.
Random $k$ -Labelsets	RA $k$ EL	Bagging	It trains several PS classifiers with a different, small and fixed number of $k$ labels, which models the labelsets for each classifier. These labelsets can be either disjoint (RA $k$ EL $_d$ ) or overlapping (RA $k$ EL $_o$ ) among the classifiers in the ensemble.
Hierarchy of Multi-Label Classifiers	HOMER	Clustering and tree-shaped hierarchy	It creates a cluster of labels and, given the members of the built clusters, it transforms a large set of labels into a tree-shaped hierarchy of simpler MLC tasks, which are solved by a common MLC model.
Random Forest of ML-C4.5	RFML-C4.5	Bagging	It employs a bagging of ML-C4.5, where a diversification scheme is used to randomly change the feature space for each ML-C4.5 classifier in the ensemble.
Random Forest of PCT	RFML-PCT	Bagging	It employs bagging of PCTs, where a diversification scheme is used to randomly change the feature space for each PCT classifier in the ensemble.

## 3.2 Evaluation in MLC

In multi-label classification, researchers and practitioners typically evaluate the MLC algorithm using multiple measures because of the additional degrees of freedom the MLC setting introduces [136]. Usually, these measures follow different perspectives to quantify how good a classification algorithm is to a given dataset [204]. This performance evaluation for MLC algorithms differs significantly from SLC, where researchers and practitioners are used to evaluate the whole classification system using a few SLC measures, such as  $F_1$ -measure ( $F_1$ ) and accuracy [216].

As already mentioned, MLC is also usually associated with the LR task. This means that we have measures to evaluate both a bipartition (in the case of MLC) and a ranking (in the case of LR) generated by a given multi-label model. Hence, let  $L = \{\phi_1, \dots, \phi_q\}$  be the total set of labels where  $q = |L|$ ;  $Y_i \subseteq L$  and  $Z_i \subseteq L$  be the true and the predicted subset of labels for the instance  $X_i$ , respectively. Let  $r(\phi_j)$  be the ranking for the predicted label  $\phi_j$ , for  $j = 1, \dots, q$  representing the order of the ranks, where the most and the least relevant ranks are represented by the highest (1) and lowest ( $q$ ) ranks, respectively. Given these definitions, we define the main measures in which group (bipartition or ranking) next. All measures are based on the description of the works of Tsoumakas et al. [204] and Pereira et al. [162].

### 3.2.1 Bipartition Measures

The bipartition measures are divided into two groups, i.e., example-based and label-based. Example-based measures are the ones that make an average of the differences between the true and the predicted set of labels over all examples for the input dataset. Label-based measures, on the other hand, break down the evaluation process by taking each label into account separately and, after that, combining their evaluations into an average over all labels. Sections 3.2.1.1 and 3.2.1.2 cover the most important measures in both groups.

#### 3.2.1.1 Example-based Bipartition Measures

Exact Match (EM, which is also known as classification accuracy and subset accuracy) is a very strict evaluation metric, as it only takes the value one when the predicted label set is an exact match to the true label set for a given example, and takes the value zero

otherwise. For this reason, EM is a really important measure in domains where the labels in the dataset are highly correlated. Given the generated model  $h$  and the dataset  $D$  with  $n$  examples, EM is defined in Equation 3.1:

$$EM(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n I(Z_i = Y_i), \quad (3.1)$$

where the function  $I(\cdot)$  takes the value 1 when  $Z_i = Y_i$ , and 0 otherwise.

Hamming Loss (HL), in turn, calculates how many times an example-label pair is misclassified. In other words, it counts when a label not belonging to the example is predicted or when a label belonging to the example is not predicted. This metric is basically the opposite measure of EM, as it disregards the label correlations. In other words, HL is actually the average binary classification error [204], as defined in Equation 3.2:

$$HL(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|Z_i \Delta Y_i|}{n}, \quad (3.2)$$

where  $\Delta$  represents the symmetric difference between the two labelsets.

Accuracy, precision, recall, and  $F_1$  are measures adapted from the SLC context to take into account partially correct predicted labelsets. They are formally defined in Equations 3.3, 3.4, 3.5, and 3.6, respectively. The accuracy measures the overall effectiveness of a classifier. Precision, in turn, measures how the actual labels agree with the positive labels predicted by the classifier. Recall, which is also known as sensitivity, evaluates the performance of the classifier in terms of retrieving positive labels. Finally,  $F_1$  is defined as the harmonic mean between precision and recall.

$$Accuracy(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|Z_i \cap Y_i|}{|Z_i \cup Y_i|} \quad (3.3)$$

$$Precision(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|Z_i \cap Y_i|}{|Z_i|} \quad (3.4)$$

$$Recall(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|Z_i \cap Y_i|}{|Y_i|} \quad (3.5)$$

$$F_1(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{2 \cdot |Z_i \cap Y_i|}{|Z_i| + |Y_i|} \quad (3.6)$$

### 3.2.1.2 Label-based Bipartition Measures

Any measure from the SLC context can be adapted for the label-based bipartition scenario, e.g., precision, recall, accuracy, and area under the Receiver Operating Characteristic (ROC) curve. As already mentioned, this happens because of the label decomposition of the evaluation process in label-based bipartition measures. First, the measures are

calculated individually for each label and then averaged over all labels. It is worth mentioning two averaging approaches, called micro-averaging and macro-averaging [87, 204]. The macro-averaging measures give equivalent weights to all class labels, regardless of their frequency (per-label averaging). This makes the macro-averaging measures more impacted by the performance of rare class labels. In contrast, the micro-averaging measures give equivalent weights to all examples (per-example averaging). This makes the micro-averaging measures more influenced by the performance of common class labels. Gibaja and Ventura [87] discuss that the macro-averaging approach is preferred when the multi-label method needs to perform well across all class labels, whereas the micro-averaging is more convenient when the density of the class labels is important.

Let TP, TN, FP, and FN be the number of true positives, true negatives, false positives, and false negatives used to evaluate single-label classifiers, respectively. Given an SLC classifier  $h$  and a dataset  $D$  with  $n$  examples, the traditional binary SLC measures, i.e., accuracy, precision, recall, and  $F_1$ , are defined as follows by Equations 3.7, 3.8, 3.9 and 3.10.

$$Accuracy_{SLC}(h, D) = \frac{TP + TN}{n} \quad (3.7)$$

$$Precision_{SLC}(h, D) = \frac{TP}{TP + FP} \quad (3.8)$$

$$Recall_{SLC}(h, D) = \frac{TP}{TP + FN} \quad (3.9)$$

$$F_1_{SLC}(h, D) = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (3.10)$$

Given these specific binary measures, let  $B(TP_\phi, TN_\phi, FP_\phi, FN_\phi)$  denote a general binary evaluation measure considering these countings for a given class label  $\phi$ . The macro-averaged and micro-averaged versions of any single-label binary measure  $B$  for a given MLC classifier  $h$  over a dataset  $D$  are defined in Equations 3.11 and 3.12, respectively.

$$B_{macro}(h, D) = \frac{1}{q} \cdot \sum_{\phi=1}^q B(TP_\phi, TN_\phi, FP_\phi, FN_\phi) \quad (3.11)$$

$$B_{micro}(h, D) = B\left(\sum_{\phi=1}^q TP_\phi, \sum_{\phi=1}^q TN_\phi, \sum_{\phi=1}^q FP_\phi, \sum_{\phi=1}^q FN_\phi\right) \quad (3.12)$$

### 3.2.2 Ranking Measures

In this section, we discuss the three main measures to evaluate multi-label ranking methods, which are ranking loss, coverage, and average precision. Starting with Ranking Loss (RL), which calculates the number of times that irrelevant labels are ranked higher than relevant labels. This means that RL penalizes the label pairs that are reversely ordered in the ranking for a given example. Equation 3.13 formalizes this measure.

$$RL(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{1}{|Y_i| \cdot |\bar{Y}_i|} \cdot |(\phi_a, \phi_b) : r_i(\phi_a) > r_i(\phi_b), (\phi_a, \phi_b) \in Y_i \times \bar{Y}_i| \quad (3.13)$$

Distinctly, Coverage (Cov) analyzes how far (in-depth) it is needed to go down in the label ranking list to cover all the relevant labels of a given example. This measure is defined in Equation 3.14.

$$Cov(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \max_{\phi \in Y_i} r_i(\phi) - 1 \quad (3.14)$$

Finally, we define the Average Precision (AvgPrec) measure in Equation 3.15. This measure checks the average fraction of labels that are ranked above a relevant label  $\phi$  in the true labelset  $Y_i$  of an example  $i$ .

$$AvgPrec(h, D) = \frac{1}{n} \cdot \sum_{i=1}^n \frac{1}{|Y_i|} \cdot \sum_{\phi \in Y_i} \frac{|\phi' \in Y_i : r_i(\phi') \leq r_i(\phi)|}{r_i(\phi)} \quad (3.15)$$

## 3.3 Final Remarks

This chapter showed there is a large variety of MLC algorithms, each one having its own assumptions and biases. For example, when the BR algorithm is chosen to deal with an MLC problem, the label correlations are disregarded. In the case of LP, RAKE, and CC, label correlations are considered in different ways.

Different algorithm assumptions can lead to different predictive performances, depending on the characteristics of the dataset and the algorithm. As Tsoumakas et al. [204] observed, the label cardinality (average number of labels of the examples in the dataset) and the label density (average number of labels of the examples divided by the total number of labels in the dataset) tend to influence the way the MLC algorithm performs and, consequently, lead to good or poor class label predictions for different algorithms.

Therefore, it is very difficult to choose the best MLC algorithm and its best hyper-parameter setting (i.e., the algorithm and the hyper-parameter setting that maximize the predictive accuracy) for a particular dataset provided by a user. In this context, this work proposes methods for automatically selecting and configuring the best MLC algorithm for a given input dataset, which is a type of AutoML task, as discussed in the next chapter.

# Chapter 4

## Automated Machine Learning

Although the term *Automated Machine Learning* (AutoML) was recently coined<sup>1</sup>, the field itself is not new. As far as we know, Rich [177] and Rich and Knight [178] were the first to point out the inherent/dual interaction between *methods for representing and using knowledge* (i.e., ML algorithms) and *methods for conducting heuristic search* (i.e., search and/or optimization methods). This interaction basically defines what AutoML is.

Because the ideas behind AutoML appeared in the fields of machine learning and optimization at different time frames and were developed mostly independently [160], there is still some level of confusion about naming this field. In fact, AutoML has already received names such as *algorithm selection* (for machine learning), *hyper-heuristics*, *hyper-parameter optimization*, and *selective or constructive meta-learning*.

Apart from this confusion, the main objective of AutoML is always to automatically recommend machine learning algorithms (together or not with their respective hyper-parameters) to learning tasks (associated with datasets) without much dependency on user knowledge [153]. Usually, the background knowledge required to learn the AutoML task is defined by a search space, which is embedded into a search mechanism that builds personalized solutions to the problem at hand.

This process is exemplified in the Figure 4.1. This figure represents a typical AutoML system receiving as input the dataset of interest and a search space composed of ML algorithms (together with their respective set of hyper-parameters). Besides, prior knowledge from the specialists can be included in the ML algorithms and hyper-parameters. For instance, the prior knowledge can come from a set of configuration files and/or from a context-free grammar. These define the search space of ML algorithms (and hyper-parameters) the AutoML system handles. This possibility is represented by the dotted line in the figure. In the end, the system will output a personalized ML solution to the input data, i.e., (a set of) learning algorithm(s) with the best hyper-parameters.

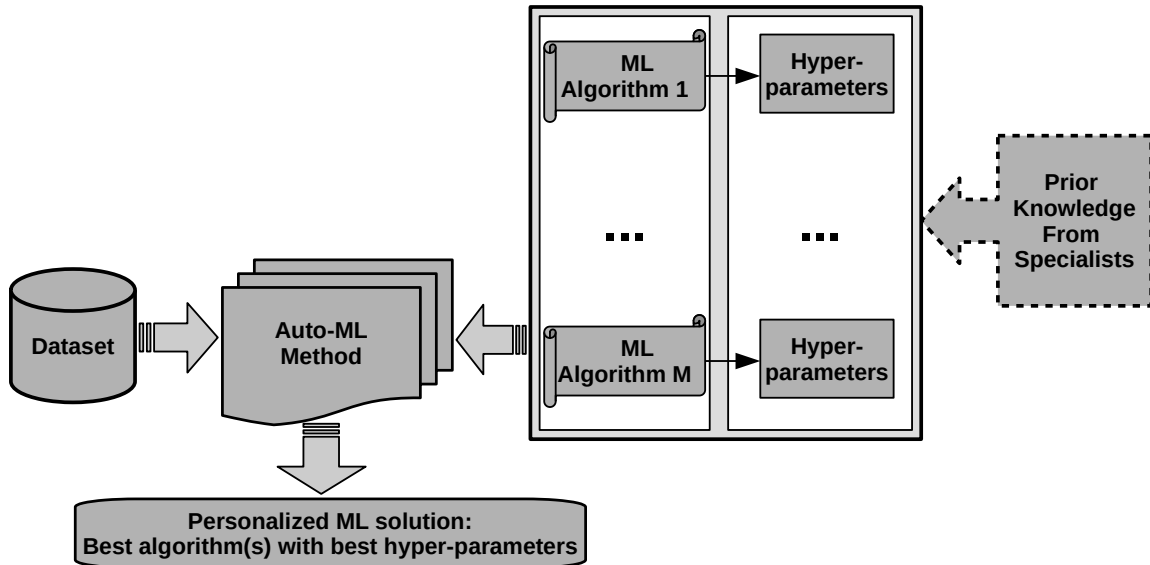
Given the constant growth of generated data and the need of interpreting, classifying, and contextualizing this data into useful information, the importance of AutoML systems is undeniable. This area has become even more important given the limited number of experts and an increasing number of enthusiastic practitioners that follow *ad hoc*

---

<sup>1</sup><https://www.automl.org/events/workshops/>



Figure 4.1: A typical AutoML process.



processes to deal with their data [50, 155].

Formally, an AutoML problem can be cast as the Combined Algorithm Selection and Hyper-parameter (CASH) optimization problem [17, 127, 200]. Therefore, let  $\mathbf{A} = \{A^{(1)}, \dots, A^{(M)}\}$  be a set of learning algorithms, where each algorithm  $A^{(i)} \in \mathbf{A}$  will have its own set of hyper-parameters,  $\Lambda^{(i)} = \{\lambda^{(1)}, \dots, \lambda^{(S)}\}$ , defined from the full set of algorithm's hyper-parameters  $\Omega$ . Further, let  $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$  be the training set containing  $n$  examples, which is divided into  $K$  cross-validation folds  $\{D_{learn}^{(1)}, \dots, D_{learn}^{(k)}\}$  and  $\{D_{valid}^{(1)}, \dots, D_{valid}^{(k)}\}$ , where  $D_{learn}^{(j)} = \{D_{train}/D_{valid}^{(j)}\}$  for each  $j = 1, \dots, K$ . Finally, let  $LF(A_{\Lambda^{(i)}}^{(i)}, D_{learn}^{(j)}, D_{valid}^{(j)})$  be the loss function, which measures the loss value of algorithm  $A^{(i)}$  on  $D_{valid}^{(j)}$ , having its model learned from  $D_{learn}^{(j)}$  with the hyper-parameters  $\Lambda^{(i)}$ . Based on these definitions, an AutoML problem deals with the task of selecting the learning algorithm and its respective hyper-parameters, which minimize the value achieved by the loss function, as defined in Equation 4.1:

$$A_{\Lambda^*}^* \in \operatorname{argmin}_{A^{(i)} \in \mathbf{A}, \Lambda^{(i)} \subseteq \Omega} \frac{1}{K} \cdot \sum_{j=1}^K LF(A_{\Lambda^{(i)}}^{(i)}, D_{learn}^{(j)}, D_{valid}^{(j)}) \quad (4.1)$$

In addition, we might extrapolate this problem to select and configure machine learning pipelines instead of focusing on algorithms and hyper-parameters. A pipeline may have one or several preprocessing techniques (e.g., feature selection, normalization, or imputation), must have at least one processing algorithm (e.g., single-label classification, multi-label classification, clustering, or regression), and may consider a post-processing approach (e.g., probability calibration or ensemble construction). So, given a search space of pipelines  $\mathbf{P} = \{\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(V)}\}$ , each pipeline is basically a sequence of learning algorithms,  $\mathbf{P}^{(g)} = \{\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(U)}\}$ , that are used to solve a learning task. Similarly to the previous definition, each pipeline will have its respective set of pipeline's hyper-

parameters,  $\Gamma^{(i)} = \{\Lambda^{(1)}, \dots, \Lambda^{(S)}\}$ , which defines the hyper-parameters for each process into the pipeline. In this case, Equation 4.1 would be generalized to Equation 4.2 to accept pipelines instead of only ML algorithms.

$$\mathbf{P}^*_{\Gamma^*} \in \underset{\mathbf{P}^{(i)} \subseteq \mathbf{P}, \Gamma^{(i)} \subseteq \Psi}{\operatorname{argmin}} \frac{1}{K} \cdot \sum_{j=1}^K LF(\mathbf{P}^{(i)}_{\Gamma^{(i)}}, D_{\text{learn}}^{(j)}, D_{\text{valid}}^{(j)}) \quad (4.2)$$

It is important to emphasize that the formulation in Equation 4.2 to include hierarchical ML pipelines into the CASH problem also makes part of the contributions of this thesis.

## 4.1 A Categorization of the AutoML Methods

In the literature, there are several ways to organize AutoML methods. In this section, we discuss the most common and novel ways to organize and categorize these methods. Hutter et al. [109], for instance, categorize the methods into hyper-parameter optimization [73], meta-learning [208] and neural architecture search [64, 65], separating the AutoML methods in terms of their main topics.

This is a simplistic way to look at AutoML, but there are other ways to organize it. We can categorize these methods by their purpose (general or specific), the type of search/optimization they use (such as Bayesian optimization, random search, evolutionary algorithms, multi-armed bandits, and hybrid methods), the learning task they are associated with (such as single-label classification, multi-label classification, regression, multi-target regression, and clustering), and their respective application domain (such as bioinformatics, fraud detection, and health informatics).

In this thesis, we define an alternative organization of the AutoML methods that respects their generality, i.e., the “level of complexity” they focus on. Some of them are more specific (i.e., taking into account one or few algorithms), and others are considered more general or complex (i.e., focusing on selecting entire pipelines). So far, we have identified three (3) categories of methods concerning different aspects of AutoML problems, as discussed below:

1. **Specific ML algorithms:** In this case, the proposed AutoML methods aim to select only components and/or hyper-parameters from ML algorithms. E.g., kernels and/or their hyper-parameters in support vector machines [57, 93], hyper-parameters in neural networks (such as learning rate and number of layers) [142], and distance functions for graph-based semi-supervised learning [143].

2. **Family of ML algorithms:** In this category, the proposed AutoML methods usually select or construct (new) ML algorithms based on a set of components describing them. Bayesian networks for classification [48, 49], decision trees [12], artificial neural networks [7, 219, 222] and rule induction [159] are examples of families of algorithms that search methods explore and recommend customized ML algorithms. E.g., given several algorithms in a family of tree-based algorithms – such as C4.5 [166], decision stump [216], random forest [25], logistic model trees [133, 196] and random tree [216]) – and their respective hyper-parameters, the AutoML method would have to choose one of them and configure its hyper-parameters to the input data.
3. **Complete ML pipelines:** Given the advances in the ML and AutoML fields, this category emerged to create methods that search for and configure pipelines over ML frameworks (such as WEKA [97] and Scikit-Learn [161]). Recall that ML pipelines are a sequence of tasks performed in order to accomplish an ML task associated with a dataset of interest [148]. The tasks included in the pipeline may represent different ways of transforming or preprocessing the dataset, as well as a classification or a regression algorithm and its associated hyper-parameters [50]. The works of Thornton et al. [200], Feurer et al. [75], Olson et al. [153], de Sá et al. [50] and Mohr et al. [144] are important examples of this category. It is worth mentioning that the methods of this category need to face a great number of preprocessing, classification/regression, and post-processing methods to return a pipeline that suits the input learning task. Thus, they must perform an outstanding search – i.e., presenting a suitable trade-off between exploration and exploitation – to generate appropriate pipelines.

Although all presented categories are important subfields in the context of AutoML, we will pay particular attention to subfields described in item 3, which are the ones we consider *contemporary* AutoML. For this reason, we review these AutoML methods next, grouping them by the type of method they use.

## 4.2 Related Work on AutoML Methods for Selecting and Configuring ML Pipelines

Let us start with the Bayesian Optimization (BO) algorithms for AutoML (see Section 2.2 for more details), which were primarily represented by Auto-Prognosis [5, 6],

Auto-WEKA [127, 200], and Auto-sklearn [74, 75, 76]. AutoPrognosis is an AutoML method specifically designed for clinical prognostic, using Scikit-Learn. AutoPrognosis searches for ensembles of pipelines using a novel batched BO algorithm. This new method uses the BO principles to predict the performance of different pipeline configurations in a scalable fashion by learning a structured kernel decomposition that identifies pipelines with correlated performance [113, 198]. AutoPrognosis consists of classification, temporal, and survival nodes, enabling distinct types of analysis of clinical data. Although presenting an important AutoML work, AutoPrognosis was not proposed for general purpose, meaning that this method can only be applied to clinical prognostic data. The next two BO-based methods are of general purpose, as they were developed to receive any classification or regression problem as input.

Auto-WEKA and Auto-sklearn are methods that currently rely on a BO algorithm called Sequential Model-based Algorithm Configuration (SMAC) [108]. A typical BO algorithm for AutoML optimizes the combination between complete ML pipelines and their respective hyper-parameters to a dataset at hand, using a hierarchical approach. In this case, the AutoML method first chooses the classification algorithm (or the preprocessing method) and, only after this step, the hyper-parameters of this algorithm are optimized. The use of this hierarchical optimization approach can be advantageous in the sense that it divides the search space into two, but it may also leave out algorithms that, with the right parameters, could generate better results than the selected ones.

Auto-WEKA automates the process of selecting the best ML pipeline in WEKA, whereas Auto-sklearn aims to optimize the pipelines in the Scikit-Learn library. The choice of these libraries by current AutoML methods is motivated by the great number of ML algorithms they already implement and their popularity. Both methods implemented the random forest-based version of SMAC that provides a better performance than the Tree-structured Parzen Estimator (TPE) BO-based algorithm [13, 200]. Nevertheless, we observed some improvements in Auto-sklearn when we compared it to Auto-WEKA. For instance, Auto-sklearn can be initialized via meta-learning [76] (to warm-start the SMAC process) and can also construct an ensemble in a post-processing stage by using an ensemble selection approach [30, 31] to combine the best pipelines found during the search [74].

Nevertheless, Fusi et al. [86] and Li et al. [135] observed that SMAC-based AutoML methods tend to suffer in high-dimensional hyper-parameter search spaces and usually present similar or worst performances than a random search, on average. This aspect was observed experimentally by both authors and occurred due to the necessity of sampling enough hyper-parameter configurations to achieve a suitable estimation of the predictive posterior probability over a high-dimensional space. This issue of BO methods is mainly due to the nature of the hyper-parameters, as the search space can have discrete, ordinal, categorical, continuous, and/or conditional hyper-parameters. This may

cause discontinuity of the function that models the performance of ML pipelines, and SMAC-based methods tend not to fit well in this case. As the search space enlarges, including different types of hyper-parameters, SMAC-based methods tend to suffer in the final predictive performance of their selected and configured pipelines.

In this context, Fusi et al. [86] try to overcome this issue by casting the AutoML task in Scikit-Learn as a collaborative filtering problem. In a nutshell, the main idea is if two datasets have similar results for a few ML pipelines, it is likely that the remaining pipelines will produce results that are similar as well. This resembles a collaborative filtering problem, which is solved by Fusi et al. using a probabilistic matrix factorization algorithm. Therefore, the authors propose the so-called Probabilistic Matrix Factorization for AutoML (PMF-AutoML) method. This method basically replaces the Gaussian process (or the random forest) model with a PMF model (more specifically, the one proposed by Salakhutdinov and Mnih [182]) to predict the performance of new ML pipelines and uses an acquisition function from Bayesian optimization methods (i.e., the expected improvement acquisition function) to decide which pipeline to evaluate next.

Aiming to overcome the same issue, Li et al. [135] proposed Hyperband and formulated the AutoML task as a pure-exploration non-stochastic infinite-armed bandit problem [27, 29, 67, 112]. This method relies on a principled early-stopping strategy to allocate resources (e.g., number of iterations, data samples, or number of features) to the randomly sampled configurations, allowing it to evaluate orders of magnitude more configurations than uniform allocation strategies. Basically, Hyperband dynamically allocates (in each round of its main procedure) more resources to the top  $K$  promising configurations<sup>2</sup>, after discarding the least promising ones<sup>3</sup>. Hyperband was evaluated using Scikit-Learn, PolyLearn [151] and Keras [39], and is considered a general-purpose approach that requires a low set of assumptions, unlike prior configuration evaluation approaches [135].

Although it is an important method in the literature, Hyperband does not stand for optimizing ML pipelines, only hyper-parameters of specific ML methods. Given that, das Dôres et al. [47] extended Hyperband and created AutoBand, which is an AutoML framework to deal with the task of recommending ML pipelines based on the WEKA tool. Noticing the importance of Hyperband to the context of AutoML, the authors of Auto-sklearn extended their AutoML method to consider success halving into Bayesian optimization, resulting in the hybrid method, namely Portfolio Successive Halving Auto-sklearn (or simply, PoSH Auto-sklearn) [72].

Auto-Tuned Models (ATM) [197] is a system that merges the aforementioned ideas by presenting a hybrid Bayesian and multi-armed bandit optimization method for Au-

---

<sup>2</sup>Hyperband takes into account a constant that defines the maximum amount of resource that can be allocated to a single configuration.

<sup>3</sup>At each round  $i$ , the number of configurations of that round,  $n_i$ , is reduced by a factor of  $\eta$ .

toML in the Scikit-Learn library. To proceed with this hybrid approach, ATM breaks the AutoML process into two phases. The first phase is used to select a hyper-partition, which is a set of hyper-parameters that defines a path through the conditional parameter tree. Basically, this tree expresses the hyper-parameter search space for a given learning algorithm. Given the selected hyper-partition, the second phase is then employed for hyper-parameter tuning. Whereas the first phase in ATM is solved by a multi-armed bandit method, the second phase uses a traditional BO method to tune the hyper-parameters. In addition, ATM comprehends a (meta) model recommender method that exploits previous learning methods' performances on a variety of datasets.

Evolutionary Algorithms (EAs) are also commonly adopted to perform AutoML (see Section 2.1 for more details about this type of method). For instance, the Tree-based Pipeline Optimization Tool (TPOT) [153, 154, 155] applies a canonical Genetic Programming (GP) algorithm to search for the most appropriate ML pipeline in the Scikit-Learn library. Each pipeline in TPOT is represented by a tree, representing a set of machine learning procedures (e.g., preprocessing steps, feature construction, applying an ML model, and/or constructing an ensemble of several ML models). One difference in the search space of TPOT when compared to the aforementioned methods is that it allows the use of many copies of the dataset, which are processed in parallel by different preprocessing methods and later combined. For example, a pipeline can have two or more feature selection methods, and then a combination method is used to verify the common and distinct features found by the techniques. This can turn the trees produced by TPOT very large, as unlimited ML components can be combined. Based on this, a multi-objective version of TPOT was developed considering the use of Pareto selection (with the Non-dominated Sorting Genetic Algorithm II, or simply NSGA-II) [51]. In this case, two separate objectives are considered: maximizing the final accuracy measure of the pipeline as well as minimizing the pipeline's overall complexity, given by the total number of pipeline components, in order to avoid overfitting.

Additionally, TPOT has two independent versions, which are TPOT with Multifactor Dimensionality Reduction (TPOT-MDR) [194] and Layered TPOT (LTPOT) [88]. TPOT-MDR is a version designed specifically for bioinformatics studies, implementing two new preprocessing operators that are used in genetic analysis of human diseases: Multifactor Dimensionality Reduction (MDR) and an Expert Knowledge Filter (EKF). On the other hand, LTPOT aims to create customized ML pipelines equally good as the original version of TPOT but in significantly less time. LTPOT does that by initially evaluating the pipelines on a small subset (of examples) of the data and only letting the most promising pipelines to be evaluated on the full dataset. This idea is similar to the one Hyperband uses.

Nevertheless, besides its popularity, it is important to mention that one of the major drawbacks of TPOT is that it can create ML pipelines that are arbitrary/invalid.

I.e., it can create an ML pipeline that fails to solve a classification problem, as there are no constraints in which type of components can be combined. For example, TPOT can create a pipeline without a classification algorithm [153, 154, 155]. This characteristic of TPOT leads to a waste of computational resources, as these individuals are identified as invalid and given a very low fitness value during the evaluation of the pipelines. Besides, even with its multi-objective version, there are also indications that TPOT can generate very complex pipelines<sup>4</sup> [144, 212], given the fact that it can create large trees with a great number of ML components during its iterative search.

Genetic Programming for Machine Learning (GP-ML) [131] is another EA-based AutoML method that overcomes this limitation by using a Strongly Typed Genetic Programming (STGP) algorithm [145]. The STGP algorithm restricts the Scikit-Learn pipelines in such a way they are always valid from the machine learning point of view. In addition, GP-ML applies an asynchronous evolutionary algorithm [188] instead of a generational one. Scott and De Jong [188] observed that asynchronous evolution is biased towards the evaluation of faster pipelines in some parts of the search space. However, Křen et al. [131] consider this bias an advantage to the AutoML task, because a faster pipeline is usually preferable to a slower one, when both present similar predictive accuracy values. GP-ML also has two multi-objective versions, named t-mGP-ML and s-mGP-ML [132], which use Pareto selection (i.e., with NSGA-II) to perform the search. Both try to maximize the pipeline’s classification performance (by using the quadratic-weighted kappa and the accuracy measures). The difference is that whereas the former minimizes the training time, the latter minimizes the size of the generated pipeline (i.e., the number of ML methods).

Resilient Classification Pipeline Evolution (RECIPE) [50] is an AutoML framework that follows the same basic principle of GP-ML by avoiding the generation of invalid pipelines during the evolutionary process. In order to guarantee that, RECIPE defines a grammar that encompasses the classification methods present in Scikit-Learn. Therefore, RECIPE makes use of a Grammar-based Genetic Programming (GGP) method to perform the search for the most suitable classification pipeline. The grammar prevents the generation of invalid/arbitrary pipelines and can also speed up the search process.

Autostacker [36] also follows an EA in the context of AutoML. Nevertheless, it presents a novel idea: given the dataset at hand, it composes a stacking layered-based architecture. In Autostacker, each layer can have several ML algorithms, which are nodes at the layer. A raw dataset is used as input for the first layer. In the subsequent layers, the classification results of the previous layers (considering each node) will be added to the raw dataset as new features. The main objective of Autostacker is to use an EA to search for the best instantiation of the ML algorithm and hyper-parameter(s) for each

---

<sup>4</sup>In fact, these pipelines are usually unfeasible to run on a dataset because they usually combine several base and ensemble methods together.

node in each layer.

Furthermore, novel and robust AutoML approaches are constantly being created as the field continues to evolve. ML-Plan [144, 212], Google Vizier [92], and Autotune [124] are examples of these approaches. ML-Plan is an AutoML method based on hierarchical planning. It relies on Hierarchical Task Networks (HTN) [66, 149], an established Artificial Intelligence (AI) planning technique. ML-Plan is implemented as a global best-first search over the graph induced by the planning problem at hand. To do that, ML-Plan guides the search by randomly completing partial pipelines, and its formulation of the HTN problem leads to an upper and a lower part of the search graph. The upper part is related to the decision of choosing the feature preprocessing algorithm (if any) and then the classification algorithm. The lower part refers to the hyper-parameter configurations of the classification algorithm and then the preprocessing algorithm (if any). ML-Plan can search for pipelines over the search spaces of both WEKA and Scikit-Learn. In addition, ML-Plan has a version that is capable of producing unlimited-length pipelines [212], like the ones generated by TPOT. In this version, it adds the capacity to resolve complex tasks, e.g., applying two or more feature selection methods in parallel and creating a feature union to combine the features selected by the methods.

On the other hand, Google Vizier is a general-purpose service for black-box optimization that can be used in the context of AutoML, i.e., for tuning models on the Google Cloud ML engine. It provides nine search methods so far: random search (traditional and  $2\times$ ) [14], grid search [14], multi-armed bandit technique using a Gaussian process regressor (or simply, Gaussian process bandit) [193], batched Gaussian process bandits [55], SMAC-based BO [108], covariance matrix adaptation evolutionary strategy [98], and probabilistic search [92]. Instead of making the user choose one of these search methods, Vizier dynamically selects which search method to use for a given input ML problem. It can change from one search method to another based on scalability or any other metric. For instance, it can start the search with a Gaussian Process Bandit, which usually provides an excellent result quality, and later change it to a more naïve method (such as grid search) that scales better with a bigger number of points. Thus, after collecting information about the configurations, Vizier may switch from a complex search method to a scalable one.

Finally, Autotune is a derivation-free AutoML framework within SAS<sup>®</sup> Visual Data Mining and Machine Learning [213], and also operates with multiple search methods, such as: random search, random Latin Hypercube Sample (LHS) [140], GA, Generating Set Search (GSS) [94], Bayesian optimization (with Gaussian process surrogate model) [116], Nelder-Mead variable shape simplex [150], and DIRECT (standard and hybrid versions) [117]. The idea of AutoTune is not just to run these search methods concurrently to perform a parallel hyper-parameter tuning but also to combine them to create new hybrid methods. For instance, the default search method in Autotune basi-



cally combines the elements of LHS, GA, and GSS methods, exploiting the strengths of each method. As there is a sharing of objective evaluations across the search methods, this feature could be implemented for other methods as well.

### 4.3 Related Work on Automated Multi-Label Classification

All AutoML methods discussed in the previous section were designed to solve the conventional single-label classification and regression tasks. In other words, they can not handle multi-label data. As far as we know, there are only a few works related to automated multi-label classification. We review them next.

Chekina et al. [35] were the first authors to note the importance of helping researchers in selecting the most appropriate multi-label classification algorithm for a dataset of interest. Particularly, they developed a meta-model for selecting one out of 11 multi-label classification algorithms, taking into account 30 characterizing measures and 36 meta-datasets. To induce the meta-model, the authors use a  $K$ -Nearest Neighbors (KNN) classifier [45], where the value of  $K$  was equal to 20. Nevertheless, as it is a preliminary work, it only selects the classification algorithm to use for a given multi-label problem, not setting the algorithm's hyper-parameters.

In a similar direction, Rivolli and de Carvalho [179] created a meta-learning system to recommend a suitable learning algorithm for each binary problem built by the BR multi-label classification algorithm. Therefore, given a dataset with  $|L|$  labels,  $|L|$  single-label algorithms will be recommended for the BR's base level, one for each label. The authors considered four classification algorithms in their system: (i) Support Vector Machines (SVM), (ii) Random Forest (RF), (iii) Naïve Bayes (NB), and (iv) KNN. However, they discarded NB in the end because of its poor performance. All these single-label classification algorithms had their hyper-parameters fixed before the aforementioned experiments. Considering these algorithms, a meta-dataset is generated for 13 datasets from Mulan repository<sup>5</sup> with 19 attributes describing each binary sub-problem. These attributes comprehend simple measures (e.g., number of attributes and number of instances), statistical measures (e.g., attribute correlation and skewness), and information theory measures (e.g., entropy and noise). They induced a meta-classification model – in this case, an SVM – with this meta-dataset, where the best of the three classification algorithms (e.g., SVM, RF, and KNN) for this meta-dataset is selected in terms of the  $F_1$

<sup>5</sup>Available at <http://mulan.sourceforge.net/datasets-mlc.html>.

metric. With the trained meta-model, they used it to recommend algorithms for other MLC datasets.

Differently from the previous authors, Cano et al. [28] proposed G3P-ML, which is a grammar-guided genetic programming algorithm for solving multi-label classification problems using IF-THEN classification rules. Thus, given a multi-label problem, G3P-ML constructs a customized set of rules, which are specified in the grammar. The major issue about G3P-ML is that this algorithm must be run several times to find the classification rules, where each run focuses on a particular label. Therefore, the minimum number of runs to cover all the labels is equal to the number of labels  $|L|$ . However, one rule is not usually sufficient to distinguish all the examples for that particular label.

Chen et al. [37], in turn, performed a study about kernel selection for multi-label classification based on a kernel alignment method [210]. Chen et al. describe that a kernel alignment method aims to calculate the degree of agreement (i.e., the alignment value) between a kernel and a learning task and find the one with the highest agreement. This alignment is for adapting a kernel to the target(s) before training the ML algorithm. In order to do that, they first set an optimal kernel in terms of the multiple labels, and later, they set the combined kernel as a linear combination of base kernels. After these definitions, their method aims to select the parameters of the combined kernel by maximizing the alignment value between the combined kernels and the one defined as ideal, which is inherently related to the learning task. To test their approach, a binary relevance multi-label classifier was augmented with a support vector machine classifier that incorporates the selected combined kernel. Although this is a very interesting study for the multi-label community, it is not general enough. Their kernel alignment method relies only on one problem transformation method, i.e., the binary relevance method. This means that correlations among the labels are never considered by the proposed method, resulting in poor performances on certain multi-label problems.

Furthermore, Pakrashi and Namee [158] have recently presented CascadeML, which is considered the first automatic neural network algorithm (i.e., a neural network architecture search method) for general multi-label classification problems. CascadeML is trained in a way that it can cascade neural networks. To do so, this method employs only the Multi-Label Back-Propagation Neural Network (ML-BPNN) algorithm [225] at the bottom level and uses the Cascade-correlation Neural Network (Cascade2) algorithm [68] at the top-level to cascade the neural network models. This AutoML method focuses only on finding the neural architecture, not caring about the hyper-parameters to train the learning algorithm (i.e., ML-BPNN). Therefore, this seems a very specific AutoML method as it considers only one learning algorithm in its search space. The difficulty is finding the best cascading of neural networks.

Evolutionary Multi-Label Ensemble (EME) [147] encompasses the problem of selecting MLC algorithms to compose MLC ensembles. Given a dataset of interest, EME

takes into account three main characteristics for the automatic generation of the final ensemble: (i) the relationships among the labels, (ii) the imbalance of the data, and (iii) the complexity of the output space. The main idea of EME stands on the simplicity of each multi-label classifier, which is focused on a small subset of the labels but still considers the relationships among them and avoids the high complexity of the output space. Although important to the AutoML field, EME takes into account only one type of model to compose the ensembles (i.e., the model produced by the label powerset algorithm). Therefore, it is still not general enough to deal with all types of MLC problems.

Finally, Wever et al. [211] extended ML-Plan in the context of multi-label classification, naming this novel method as ML<sup>2</sup>-Plan (Multi-Label ML-Plan). ML<sup>2</sup>-Plan searches over a huge MLC search space by also using a hierarchical planning approach. Basically, ML<sup>2</sup>-Plan follows the same principles as ML-Plan, although its search space of MLC algorithm configurations is bigger than those explored by ML-Plan, making it harder. Besides this method and the ones proposed in this thesis, we are not aware of other methods that perform *modern automated multi-label classification* in complex hierarchical search spaces.

## 4.4 Final Remarks

In this chapter, we observe the clear progress of the field of AutoML throughout the years. Guyon et al. [96] and Hutter et al. [109] basically showed the importance of the main AutoML methods, challenges, and workshops to this progress to happen.

Nevertheless, we still see AutoML in its early stages, where not all types of data were encompassed, and neither all-important analyses of the results of the already proposed methods were performed. Therefore, there are several issues in the field to be addressed. In other words, there is a need to bring more understanding regarding how and why AutoML works instead of just proposing novel methods. We claim that the science behind AutoML needs to be progressed fully.

This particular thesis aims to bring more understanding to AutoML in the context of multi-label classification, which is a field not very explored. Here, we propose four automated multi-label classification methods and provide analysis in terms of the size and difference of the search spaces, the trade-off between exploration and exploitation, and the convergence of these proposed methods. Thus, this thesis gives an overall assessment of automated multi-label classification.

## Chapter 5

# AutoML Methods for Multi-label Classification

This chapter unifies the concepts presented in the three previous chapters, i.e., proposing novel AutoML methods, based on evolutionary and Bayesian optimization methods, in the context of MLC. It addresses Issue 1 of Section 1.1 (and, consequently, RQ1 of Section 1.2), showing that it is possible to handle the complex hierarchical nature of the MLC search space.

To address the referred issue, Section 5.1 introduces the general framework followed by all proposed methods in this thesis to perform the MLC AutoML task, organizing them into two main components: the proposed *search methods* and the *search spaces* they explore. In Section 5.2, we present and compare the *search spaces* used by the proposed AutoML methods. In Section 5.3, we define the search methods. Furthermore, Section 5.4 presents multi-fidelity methods that were incorporated into the search methods in order to speed up their training procedures.

### 5.1 General Framework

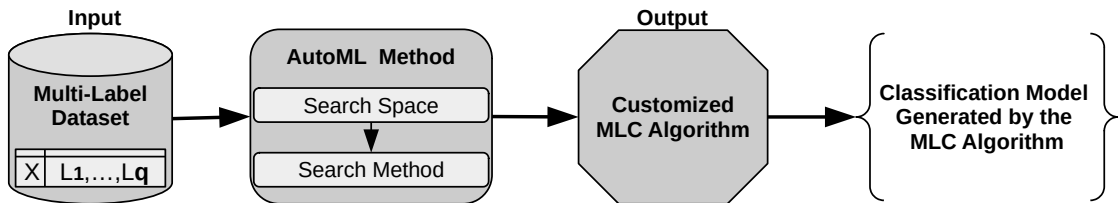
This section presents the generic framework followed by all AutoML methods proposed in this thesis. As illustrated in Figure 5.1, the AutoML method receives as input a specific multi-label dataset (with the attribute space  $X$  and the class labels  $L_1$  to  $L_q$ ). As aforementioned, the proposed AutoML methods have two main components: the *search space* and the *search method*.

The *search space* comprehends the main building blocks (e.g., the prediction threshold values, the hyper-parameters, and the algorithms at the MLC and SLC levels) from previously designed MLC algorithms. To explore this *search space*, the AutoML method uses a *search method*, which suggests appropriate MLC algorithms for the dataset at hand. However, the performance of the *search method* depends on what is specified in the *search*

*space*.

Following, the AutoML method outputs an MLC algorithm tailored to the input dataset based on that *search space*. This MLC algorithm is specifically selected and (hyper-)parameterized to this data, although it could be applied to any multi-label dataset. In the end, the customized MLC algorithm returns an MLC model and, consequently, its classification results.

Figure 5.1: The general AutoML framework to select and configure MLC algorithms.



We proposed three *search spaces*, namely Small, Medium, and Large. Section 5.2 introduces these *search spaces*, explaining how they differ. In summary, they differ from each other in terms of complexity, but all *search spaces* are based on the MLC and SLC algorithms described in Appendix A.

Furthermore, we have designed four *search methods*, namely GA-Auto-MLC, Auto-MEKA<sub>GGP</sub>, Auto-MEKA<sub>SpGGP</sub> and Auto-MEKA<sub>BO</sub>. Whereas Auto-MEKA<sub>BO</sub> works with BO, the other methods are EA-based, as detailed in Section 5.3.

## 5.2 Search Spaces for Automated Multi-Label Classification

The *search spaces* of the proposed AutoML methods for MLC data were specified using the MEKA framework [176]<sup>1</sup>, which is a multi-label extension to the WEKA software [97]. MEKA has a large variety of algorithms, focusing mainly on Problem Transformation (PT) methods. As it is mostly based on PT methods, MEKA intrinsically uses the algorithms from WEKA (see Chapter 3 for more details). MEKA also includes a variety of evaluation metrics from the literature, which are important to measure the performance of MLC algorithms from different classification perspectives.

To design the *search spaces* for the AutoML methods, we first performed a deep study about multi-label classification in the MEKA software, which resulted in Ap-

<sup>1</sup>Although we employed WEKA to develop our AutoML *search spaces* in the context of MLC, other software can be used with the same objective.

pendix A. We analyzed in detail all the algorithms and their hyper-parameters, the default value of each hyper-parameter, the constraints associated with different hyper-parameter settings, and the hierarchical nature of operations performed by problem transformation methods and meta-algorithms, among other issues.

Based on that, we decided to design three *search spaces*: Small, Medium, and Large. The reason behind this threefold modeling is basically because we want to test different levels of *search space* complexity. The components and organization of these *search spaces* are discussed next.

### 5.2.1 Components of the Search Spaces

Recall that the MLC algorithms were divided into three categories (see Section 3.1 and Appendix A for more details): Problem Transformation (PT), Algorithm Adaptation (AA), and Meta-MLC algorithms (Meta-MLC). PT algorithms call the SLC algorithms to solve an MLC problem, transforming the given problem into one or various SLC problems. On the other hand, AA methods do not need to transform the data in a preprocessing step, applying their learning process in a direct way. Finally, Meta-MLC algorithms have the aforementioned MLC algorithms (PT or AA) as base algorithms, using the base classifiers' outputs in different ways to try to improve MLC performance.

Because of that, the *search spaces* in this work have two main types of components: multi-label classification algorithms and single-label classification algorithms, together with their main hyper-parameters.

#### 5.2.1.1 Multi-label Classification Algorithms

Table 5.1 lists the 26 MLC algorithms present in the three different versions of the proposed search spaces, showing their names, acronyms, and their respective types. Table 5.1 also indicates if the MLC algorithm in the row belongs or not ('Y' for yes and 'N' for no) to the respective *search space* in the column (i.e., if the MLC *search space* comprehends that algorithm), and how many hyper-parameters (#HP) it encompasses (when it is used). With this information, we can compare the three defined *search spaces* in more detail.

Table 5.1: Multi-Label Classification (MLC) algorithms used in the MEKA data mining tool for the proposed AutoML methods\*.

id	Algorithm Name	Acronym	Category	Small		Medium		Large	
				Used?	#HP	Used?	#HP	Used?	#HP
1	Back Propagation Neural Network	ML-BPNN	AA	Y	4	Y	4	Y	4
2	Binary Relevance	BR	PT	Y	0	Y	0	Y	0
3	Classifier Chain	CC	PT	Y	0	Y	0	Y	0
4	Label Powerset	LP	PT	Y	0	Y	0	Y	0
5	Random $k$ -Label Pruned Sets	RA $k$ EL	PT	Y	4	Y	4	Y	4
6	Bayesian Classifier Chains	BCC	PT	N	-	Y	1	Y	1
7	Binary Relevance – Quick Version	BR <sub>q</sub>	PT	N	-	Y	1	Y	1
8	Classifier Chain – Quick Version	CC <sub>q</sub>	PT	N	-	Y	1	Y	1
9	Four-Class Pairwise Classification	FW	PT	N	-	Y	0	Y	0
10	Monte-Carlo Classifier Chains	MCC	PT	N	-	Y	3	Y	3
11	Probabilistic Classifier Chains	PCC	PT	N	-	Y	0	Y	0
12	Pruned Sets	PS	PT	N	-	Y	2	Y	2
13	Pruned Sets with Threshold	PSt	PT	N	-	Y	2	Y	2
14	Random $k$ -Label Disjoint Pruned Sets	RA $k$ EL <sub>d</sub>	PT	N	-	Y	3	Y	3
15	Ranking and Threshold	RT	PT	N	-	Y	0	Y	0
16	Classifier Trellis	CT	PT	N	-	N	-	Y	6
17	Conditional Dependency Networks	CDN	PT	N	-	N	-	Y	2
18	Conditional Dependency Trellis	CDT	PT	N	-	N	-	Y	5
19	Population of MCC	PMCC	PT	N	-	N	-	Y	6
20	Bagging of Multi-Label Classifiers	BaggingML	Meta-MLC	N	-	N	-	Y	1
21	Bagging of Multi-Label Classifiers (Duplicate)	BaggingMLDup	Meta-MLC	N	-	N	-	Y	2
22	Classification Maximization	CMax	Meta-MLC	N	-	N	-	Y	1
23	Ensemble of Multi-Label Classifiers	EnsembleML	Meta-MLC	N	-	N	-	Y	2
24	Expectation Maximization	EMax	Meta-MLC	N	-	N	-	Y	1
25	Random Subspace Multi-Label	RSML	Meta-MLC	N	-	N	-	Y	3
26	Subset Mapper	SM	Meta-MLC	N	-	N	-	Y	0

\*All algorithm names are clickable links to their respective description in Appendix A.

### 5.2.1.2 Single-label Classification Algorithms

As already mentioned, all PT methods in Table 5.1 use a single-label classifier (SLC)<sup>2</sup> in order to perform the transformed multi-label classification at a base level. Table 5.2 shows the 28 SLC algorithms used, i.e., the possible algorithms at the MLC base level and their hyper-parameters. Similar to Table 5.1, Table 5.2 defines the SLC algorithms in terms of their names, acronyms, and types (i.e., trees, rules, lazy, functions, Bayes, preprocessing, and meta-SLC). As before, we show if the SLC algorithm in the row is used or not ('Y' for yes and 'N' for no) by the respective *search space* in the column and how many hyper-parameters (#HP) it has (when it is used).

## 5.2.2 Search Space Structure and Size

Based on the information in Tables 5.1 and 5.2, note that for each *search space*, we have a different number of algorithms and hyper-parameters. For the *search space*

<sup>2</sup>For more details about the SLC algorithms and their hyper-parameters, see [216], and Sections A.1, A.2 and A.3 of Appendix A.

Table 5.2: Single-Label Classification (SLC) algorithms used in the WEKA data mining tool for the proposed AutoML methods\*.

id	Algorithm Name	Acronym	Category	Small		Medium		Large	
				Used?	#HP	Used?	#HP	Used?	#HP
1	JRip	JRip	Rules	Y	4	Y	4	Y	4
2	K-Nearest Neighbors	KNN	Lazy	Y	3	Y	3	Y	3
3	Logistic Regression	LR	Functions	Y	1	Y	1	Y	1
4	Naïve Bayes	NB	Bayes	Y	2	Y	2	Y	2
5	Random Forest	RF	Trees	Y	3	Y	3	Y	3
6	Bayesian Network Classifier	BNC	Bayes	N	-	Y	1	Y	1
7	C4.5	C4.5	Trees	N	-	Y	8	Y	8
8	Decision Table	DT	Rules	N	-	Y	4	Y	4
9	K Star	K*	Lazy	N	-	Y	3	Y	3
10	Logistic Model Trees	LMT	Trees	N	-	Y	7	Y	7
11	Multi-Layer Perceptron	MLP	Functions	N	-	Y	6	Y	6
12	PART	PART	Rules	N	-	Y	4	Y	4
13	REPTree	REPTree	Trees	N	-	Y	3	Y	3
14	Stochastic Gradient Descent	SGD	Functions	N	-	Y	5	Y	5
15	Sequential Minimal Optimization	SMO	Functions	N	-	Y	6	Y	6
16	Decision Stump	DS	Trees	N	-	N	-	Y	0
17	Naïve Bayes Multinomial	NBM	Bayes	N	-	N	-	Y	0
18	One Rule	OneR	Rules	N	-	N	-	Y	1
19	Random Tree	RTree	Trees	N	-	N	-	Y	4
20	Simple Logistic	SL	Functions	N	-	N	-	Y	3
21	Voted Perceptron	VP	Functions	N	-	N	-	Y	3
22	Zero Rules	ZeroR	Rules	N	-	N	-	Y	0
23	Attribute Selection Classifier	ASC	Preprocessing	N	-	N	-	Y	1
24	Ada boost M1	AdaM1	Meta-SLC	N	-	N	-	Y	3
25	Bagging of Single-Label Classifiers	Bagging	Meta-SLC	N	-	N	-	Y	3
26	Locally Weighted Learning	LWL	Meta-SLC	N	-	N	-	Y	2
27	Random Committee	RC	Meta-SLC	N	-	N	-	Y	1
28	Random Subspace	RSS	Meta-SLC	N	-	N	-	Y	3

\*All algorithm names are clickable links to their description in Appendix A.

Small, for instance, we have five (5) MLC algorithms, where four (4) of them can be combined with other five (5) SLC algorithms, as they are from the PT category. The only algorithm that can not be combined with SLC algorithms is ML-BPNN, which belongs to the AA category. Therefore, the *search space* Small comprehends ten (10) learning algorithms – five MLC algorithms and five SLC algorithms – which gives a combination of 21 MLC-SLC algorithm configurations.

In contrast, the *search space* Medium has 15 MLC algorithms, where again, only one is from the AA category, and the remaining is from the PT category. Considering the 14 PT algorithms in this *search space*, we can combine them with 15 SLC algorithms. Hence, this results in 30 learning algorithms for the *search space* Medium – 15 MLC algorithms and 15 SLC algorithms – which returns a combination of 211 MLC-SLC algorithm configurations.

Finally, we have the *search space* Large, which comprehends 26 MLC algorithms and 28 SLC algorithms. From the 26 MLC algorithms, one is from the AA category, 18 are from the PT category, and seven are from the Meta-MLC category. Considering the SLC algorithms, this totalizes 54 learning algorithms – 26 MLC algorithms and 28 SLC algorithms – which produces 16,568 possible combinations of MLC-SLC algorithm configurations. In this *search space*, it is important to emphasize that we also added



meta-algorithms for both MLC and SLC scenarios.

It is also important to note that those MLC and SLC algorithms were chosen to compose the *search spaces* considering their differences. For instance, the *search space* Small has five (5) MLC algorithms that differ in how they build the MLC model. We have tried to do the same for the SLC level. Considering that, the *search space* Small has one algorithm of each traditional algorithm category of WEKA (i.e., trees, rules, functions, lazy, and Bayesian).

We applied this idea of the selection of distinct algorithms to design the other instances of the *search spaces*. To understand that, we perform *search space* comparisons. By comparing the *search spaces* Small, Medium, and Large, we observe that there is an increase in the number of learning algorithms. The *search space* Small is contained by the Medium, which, in turn, is a subspace of Large.

Nevertheless, when comparing Medium to Large, besides the rise of complexity from one *search space* to another, we have a change in the structure of the *search space*. This happens because we added meta-MLC and meta-SLC algorithms into Large. Therefore, we have more levels in the multi-label hierarchy to consider. For example, when a *search method* is selecting a new algorithm in terms of this *search space*, it must decide to include or exclude algorithms at the meta-MLC and meta-SLC levels. As a result, this *search space* is hierarchically more complex than the other two (i.e., Small and Medium).

One aspect that is common to all search spaces – besides the ten (10) algorithms in the *search space* Small – is the definition of the strategy to optimize the prediction threshold in MLC. The prediction threshold dictates the final MLC performance, and it is real-valued and indicates if an instance belongs to one class or to another. The optimization of this threshold adjusts the bipartition result according to the probabilities scored given by the MLC models [180]. In general, this is better than simply using an arbitrary threshold of 0.5 [69, 175].

For the three *search spaces*, we include three strategies to optimize the threshold. First, we let the *search method* to globally optimize this threshold using a real value between zero (0.0) and one (1.0). This is a global strategy and establishes the same threshold for all labels. Second, we use the global strategy named Proportional Cut Method by Instance (PCut1), which minimizes the difference between the label cardinality<sup>3</sup> of the training set and the label cardinality obtained with a given set of predicted labels. Finally, we also consider the Proportional Cut Method by Label (PCutL) strategy. This strategy works like PCut1, but for each label individually. Therefore, it is a local strategy to set the prediction threshold. For more details about the strategies to optimize the prediction threshold, see Section A.4 of Appendix A.

Taking into account the number of learning algorithms and hyper-parameters, and the constraints in the choices of algorithms' components and (hyper)-parameters

---

<sup>3</sup>Label cardinality is the average number of labels of the examples in the dataset.

in MEKA, we estimated the size of the three *search spaces*<sup>4</sup>. In total, the *search space* Small has  $[(5.070 \times 10^7) + (8.078 \times 10^8 \times m) + (2.535 \times 10^{10} \times q)]$  possible MLC algorithm configurations, where  $m$  is the number of attributes and  $q$  is the number of labels of the dataset. The *search space* Medium, on the other hand, is estimated as having  $[(2.545 \times 10^{16}) + (8.078 \times 10^8 \times m) + (1.151 \times 10^{12} \times q)]$  possible MLC algorithm configurations. Finally, the *search space* Large is estimated to have approximately  $[(6.555 \times 10^{29}) + (1.443 \times 10^{14} \times m) + (3.042 \times 10^{22} \times q) + (1.291 \times 10^{27} \times \sqrt{q})]$  possible valid MLC algorithm configurations. For more details about the contribution of each MLC algorithm and also of each SLC algorithm (because of the PT methods), see Tables 5.3 and 5.4, respectively.

In Table 5.3, for instance, we have the number 2,000 defining the contribution of this algorithm (and its hyper-parameters) to the *search space* Small. As JRip has one real hyper-parameter (i.e., the *minimal total weight*) with 100 possibilities, two Boolean hyper-parameters (i.e., *check error rate* and *use pruning*) with two possibilities, and one integer hyper-parameter (i.e., the *number of optimizations*) with five possibilities, to achieve this contribution we have then  $100 \times 2 \times 2 \times 5$ , which returns 2,000.

In Table 5.4, in turn, we have the number 752,660,500 defining the contribution of BRq in the *search space* Medium. To reach this number, we use the total number of possibilities for the SLC algorithms in this *search space*, i.e., 7,526,605 (see the last line of Table 5.3). Apart from the base classifier, the BRq algorithm has only one real-valued hyper-parameter (i.e., the *down-sample ratio*). Thus, we reach 752,660,500 by multiplying the possible options of SLC classification algorithms by 100, which represents the contribution of BRq's hyper-parameter.

### 5.2.3 A Detailed Description of the MLC Search Space

This section describes the grammar used to formally specify the *search space* listed in Appendix A and, consequently, the MLC hierarchy. Although this formal definition is done by a grammar, it can be adapted to be organized by any other formalism or simple configuration files.

The proposed grammar<sup>5</sup> has 125 production rules, in a total of 124 non-terminals and 213 terminals. Boxes 5.1-5.7 present the produced grammar (in the Backus Naur Form) that encompasses the knowledge about multi-label classification in MEKA. This grammar models the *search space* Large, as this *search space* includes all learning algo-

<sup>4</sup>In these estimations, real-valued hyper-parameters have always taken 100 different discrete values.

<sup>5</sup>For a formal definition of grammar, see Section 2.1.3.

Table 5.3: Estimation of the number of possibilities for the Single-Label Classification (SLC) algorithms\*.

id	Algorithm Name	Acronym	Type	Small	Medium	Large
				#Possibilities	#Possibilities	#Possibilities
1	JRip	JRip	Rules	2,000	2,000	2,000
2	K-Nearest Neighbors	KNN	Lazy	384	384	384
3	Logistic Regression	LogR	Functions	100	100	100
4	Naïve Bayes	NB	Bayes	3	3	3
5	Random Forest	RF	Trees	163,200	163,200	163,200
6	Bayesian Network Classifier	BNC	Bayes	0	6	6
7	C4.5	C4.5	Trees	0	409,600	409,600
8	Decision Table	DT	Rules	0	80	80
9	K Star	K*	Lazy	0	800	800
10	Logistic Model Trees	LMT	Trees	0	206,848	206,848
11	Multi-Layer Perceptron	MLP	Functions	0	320,000	320,000
12	PART	PART	Rules	0	1,024	1,024
13	REP'Tree	REP'Tree	Trees	0	2,560	2,560
14	Stochastic Gradient Descent	SGD	Functions	0	120,000	120,000
15	Sequential Minimal Optimization	SMO	Functions	0	6,300,000	6,300,000
16	Decision Stump	DS	Trees	0	0	1
17	Naïve Bayes Multinomial	NBM	Bayes	0	0	1
18	One Rule	OneR	Rules	0	0	32
19	Random Tree	RTree	Trees	0	0	204,800
20	Simple Logistic	SL	Functions	0	0	404
21	Voted Perceptron	VP	Functions	0	0	45,000,000
22	Zero Rules	ZeroR	Rules	0	0	1
23	Attribute Selection Classifier	ASC	Preprocessing	0	0	105,463,626
24	Ada boost M1	AdaM1	Meta-SLC	0	0	171,139,846,696
25	Bagging of Single-Label Classifiers	Bagging	Meta-SLC	0	0	609,421,562,841
26	Locally Weighted Learning	LWL	Meta-SLC	0	0	222,124,920
27	Random Committee	RC	Meta-SLC	0	0	51,065,280
28	Random Subspace	RSS	Meta-SLC	0	0	332,210,421,900
<b>Total Number of Possibilities</b>				165,687	7,526,605	$1.113 \times 10^{12}$

\*All algorithm names are clickable links to their respective part in Appendix A.

rithms and hyper-parameters from the other *search spaces* (i.e., Small and Medium). To obtain the grammar for the other two *search spaces*, we would need just to exclude the respective production rules, non-terminals, and terminals – which represent the algorithms and hyper-parameters – that do not make part of them.

In Box 5.1, the first production rule (<Start>) is used to describe the multi-label classification (MLC) *search space*. In this grammar rule, <MLC-PT> denotes problem transformation, <MLC-AA> denotes algorithm adaptation, and <META-MLC-LEVEL> denotes the multi-label meta-algorithms. We designed the grammar in such a way that all the 26 MLC algorithms have the same probability of being chosen (i.e., each MLC algorithm has  $\approx 3.846\%$  of chance of being selected). Furthermore, the MLC algorithms must use a prediction threshold (<pred.tshd>), which defines the threshold to perform classification using the model's confidence outputs [4].

The grammar rule defining the problem transformation methods, i.e., <MLC-PT>, has two components on the right-hand side, namely the actual problem transformation algorithm <ALGS-PT> (defined in Box 5.5) and the single-label classification algorithm (SLC, which is represented by the rule <ALG-SLC> in the grammar) to perform the single-label classification task(s). This happens because the problem transformation

Table 5.4: Multi-Label Classification (MLC) algorithms used in the MEKA data mining tool for the proposed AutoML methods\*.

id	Algorithm/Strategy Name	Acronym	Small	Medium	Large
			#Possibilities	#Possibilities	#Possibilities
0	Prediction Threshold	pred_tshd	102	102	102
1	Back Propagation Neural Network	ML-BPNN	7,920,000m	7,920,000m	$7,920,000 \times m$
2	Binary Relevance	BR	165,687	7,526,605	$1.113 \times 10^{12}$
3	Classifier Chain	CC	165,687	7,526,605	$1.113 \times 10^{12}$
4	Label Powerset	LP	165,687	7,526,605	$1.113 \times 10^{12}$
5	Random K-Label Pruned Sets	RAkEL	248,530,500q	11,289,907,500q	$1,670 \times 10^{15} \times q$
6	Bayesian Classifier Chains	BCC	0	75,266,050	$1.113 \times 10^{13}$
7	Binary Relevance – Quick Version	BRq	0	752,660,500	$1.113 \times 10^{14}$
8	Classifier Chain – Quick Version	CCq	0	752,660,500	$1.113 \times 10^{14}$
9	Four-Class Pairwise Classification	FW	0	7,526,605	$1.113 \times 10^{12}$
10	Monte-Carlo Classifier Chains	MCC	0	$2,569 \times 10^{13}$	$3,800 \times 10^{18}$
11	Probabilistic Classifier Chains	PCC	0	$2,434 \times 10^{17}$	$1.113 \times 10^{12}$
12	Pruned Sets	PS	0	225,798,150	$3,340 \times 10^{13}$
13	Pruned Sets with Threshold	PSt	0	225,798,150	$3,340 \times 10^{13}$
14	Random K-Label Disjoint Pruned Sets	RAkELd	0	22,579,815,000	$3,340 \times 10^{15}$
15	Ranking and Threshold	RT	0	7,526,605	$1.113 \times 10^{12}$
16	Classifier Trellis	CT	0	0	$6.908 \times 10^{19} \times \sqrt{q}$
17	Conditional Dependency Networks	CDN	0	0	$1.001 \times 10^{17}$
18	Conditional Dependency Trellis	CDT	0	0	$1.801 \times 10^{18} \times \sqrt{q}$
19	Population of MCC	PMCC	0	0	$3.599 \times 10^{22}$
20	Bagging of Multi-Label Classifiers	BaggingML	0	0	$1.505 \times 10^{26} + 3.312 \times 10^{10} \times m + 6.937 \times 10^{18} \times q + 2.964 \times 10^{23} \times \sqrt{q}$
21	Bagging of Multi-Label Classifiers (Duplicate)	BaggingMLDup	0	0	$1.370 \times 10^{28} + 3.014 \times 10^{12} \times m + 6.312 \times 10^{20} \times q + 2.698 \times 10^{25} \times \sqrt{q}$
22	Classification Maximization	CMax	0	0	$1.632 \times 10^{22} + 3.312 \times 10^{10} \times m + 6.983 \times 10^{18} \times q + 2.964 \times 10^{23} \times \sqrt{q}$
23	Ensemble of Multi-Label classifiers	EnsembleML	0	0	$3.161 \times 10^{27} + 6.956 \times 10^{11} \times m + 1.466 \times 10^{20} \times q + 6.225 \times 10^{24} \times \sqrt{q}$
24	Expectation Maximization	EMax	0	0	$1.632 \times 10^{22} + 3.312 \times 10^{10} \times m + 6.983 \times 10^{18} \times q + 2.964 \times 10^{23} \times \sqrt{q}$
25	Random Subspace Multi-Label	RSML	0	0	$6.385 \times 10^{29} + 1.405 \times 10^{14} \times m + 2.962 \times 10^{22} \times q + 1.257 \times 10^{27} \times \sqrt{q}$
26	Subset Mapper	SM	0	0	$3.672 \times 10^{24} + 8.078 \times 10^8 \times m + 1.703 \times 10^{27} \times q + 7.230 \times 10^{21} \times \sqrt{q}$
<b>Total Number of Possibilities</b>			$5.070 \times 10^7 + 8.078 \times 10^8 \times m + 2.535 \times 10^{10} \times q$	$2.545 \times 10^{16} + 8.078 \times 10^8 \times m + 1.151 \times 10^{12} \times q$	$6.555 \times 10^{29} + 1.443 \times 10^{14} \times m + 3.042 \times 10^{22} \times q + 1.291 \times 10^{27} \times \sqrt{q}$

\*All algorithm names are clickable links to their respective part in Appendix A.

Box 5.1: Defined grammar – Part 1: General and SLC trees algorithms.

```

<Start> ::= (<MLC-PT> | <MLC-AA> | <META-MLC-LEVEL> ) <pred_tshd>

<pred_tshd> ::= PCut1 | PCutL | RANDOM-REAL(>0.0, <1.0)
                #pred_tshd='prediction threshold'
                #PCut1='P-Cut method',PCutL='P-Cut method by Label'

<MLC-PT> ::= <ALGS-PT> <ALGS-SLC>

<ALGS-SLC> ::= <ALG-TYPE> | <META1> <ALG-WEIGHTED-TYPE> | <META2> <ALG-RANDOM-TYPE> | <META3> <ALG-TYPE>

<ALG-TYPE> ::= [ASC <sm>] (<TREES> | <RULES> | <LAZY> | <FUNCTIONS> | <BAYES> | <OTHERS>)
                #ASC='Attribute Selection Classifier'

<sm> ::= GreedyStepwise | BestFirst
                #sm='search method'

<TREES> ::= <C4.5> | DecisionStump | ( ( (RandomForest <nt> | <RandomTree>) <nf> ) | <REPTree> ) <md>

<C4.5> ::= <C4.5-Basics> ( (<cf> [sr]) | u )
                #sr='subtree raising', u='unpruned'
<C4.5-Basics> ::= <mno> [ct] [bs] [umc] [ul]
                #ct='collapse tree', bs='binary splits'
                #umc='use MDL correction', ul='use Laplace'

<cf> ::= RANDOM-REAL(0.0, 1.0)
                #cf='confidence factor'
<mno> ::= RANDOM-INT(1, 64)
                #mno='minimum number of objects'

<nt> ::= RANDOM-INT(2, 256)
                #nt='number of trees'
<nf> ::= RANDOM-INT(2, 32)
                #nf='number of features'
<md> ::= RANDOM-INT(2, 20)
                #md='maximum depth'

<RandomTree> ::= <mw> <nfbgt>
<mw> ::= RANDOM-INT(1,64)
                #mw='minimum weight for instances in a leaf'
<nfbgt> ::= 2 | 3 | 4 | 5
                #nfbgt='number of folds for back-fitting and
                # for growing the tree'

<REPTree> ::= <mw> [up]
                #up='use pruning'
                #mw is not included in the same rule for Random Tree
                #and for REPTree because of the grammar's constraints

```

method transforms the multi-label task into one or more single-label tasks. We start discussing the rule defining `<ALG-SLC>`.

We divided the SLC algorithms into six (6) categories for the grammar following the WEKA software: Trees, Rules, Lazy, Functions, Bayes and Others. The last category was created to simplify the grammar (i.e., it is not an inherent WEKA's category). Box 5.1 shows the grammar rules for Tree algorithms. It also defines the Attribute Selection Classifier (ASC), a wrapper that can be used together with the SCL algorithms. In this case, a preprocessing method is used before the classification step is performed. Box 5.2 shows the grammar rules for Rules and Lazy algorithms. Box 5.3 shows the grammar rules for the other three types of SLC algorithms. For SLC algorithms, we employed the *search space* defined by Auto-WEKA [127, 200] to set their hyper-parameter values.

It is also important to mention that some methods at the single-label level, such as Decision Stump and ZeroR, do not have user-defined hyper-parameters. Others, such as the Bayesian Network Classifier algorithms, do not have user-defined hyper-parameters in the Auto-WEKA software, even though they have user-defined parameters in WEKA. That is, the developers of Auto-WEKA have chosen to use a fixed predefined number of parameter settings for some algorithms. As we are following Auto-WEKA to define the parameters at this level (because of its robustness to select SLC algorithms), the absence of user-defined parameters in some methods was maintained.

## Box 5.2: Defined grammar – Part 2: SLC rules and lazy algorithms .

```

<RULES> ::= <DT> | <JRip> | OneR <mbs> | <PART> | ZeroR

<DT> ::= <em> [uibk] <sm> <crv>
#uibk='use IBk'
#sm='search method -- defined earlier'
<em> ::= acc | rmse | mae | auc
#em='evaluation measure'
<crv> ::= 1 | 2 | 3 | 4
#crv='number of folds for cross-validation'

<JRip> ::= <mtw> [cer] [up] <o>
#cer='check error rate', up='use pruning'
<mtw> ::= RANDOM-REAL(1.0, 5.0)
#mtw='minimum total weight for instances
# covered by a rule'
<o> ::= RANDOM-INT(1,5)
#o='number of optimization runs'

<mbs> ::= RANDOM-INT(1,32)
#mbs='minimum bucket size'

<PART> ::= <PART-BASICS> (rep <nr> | ebp)
#rep='use reduced-error pruning'
#nr='number of folds for reduced-error pruning'
#ebp='use error-based pruning'

<PART-BASICS> ::= <mno> [bs]
#mno='minimum number of objects'
<nr> ::= RANDOM-INT(2,5)

<LAZY> ::= <KNN> | <K*>

<KNN> ::= <k_nn> [loo] [<dw>]
#loo='leave-one-out to set the k value given the range'
<k_nn> ::= RANDOM-INT(1,64)
#k_nn='number of neighbors'
<dw> ::= F | I
#dw='distance weighting'

<K*> ::= <gb> [eab] <mm>
#eab='entropic auto-blending'
<gb> ::= RANDOM-INT(1,100)
#gb='global blending'
<mm> ::= a | d | m | n
#mm='missing mode to deal with missing values'

```

At the single-label level, we also have meta-algorithms divided into three (3) types: <META1>, <META2> and <META3>. These three categories of meta-algorithms are firstly called in Box 5.1. As shown in Box 5.4, <META1> may take the two (2) meta-algorithms Ada Boost M1 (AdaM1) and Locally Weighted Learning (LWL), which need a base classifier at the SLC level that handles weighted instances. This is the reason the rule <ALG-WEIGHTED-TYPE> is defined. On the other hand, <META2> may take just one algorithm, i.e., Random Committee. The reason for that is because this SLC meta-algorithm can only be used with randomizable base classifiers. The rule <ALG-RANDOM-TYPE> expresses these randomizable classifiers. The least restricted meta-algorithms are Random Subspace and Bagging, which are specified by <META3>, being able to use any SLC base classifier (from <ALG-TYPE>).

It is important to emphasize that all 28 SLC methods (traditional, meta, and preprocessing) have the same chance of being chosen by a *search method* that follows this grammar. Therefore, each SLC algorithm has a probability of  $\approx 3.571\%$  of being selected.

The second component of problem transformation methods is the actual problem transformation algorithm to deal with the single-label classification. In other words, this step defines the choice of the MLC algorithm to handle the results created by the single-label classification models. For this component, we divided its respective algorithms into three categories, i.e., three production rules in the grammar (see Box 5.5): <ALGS-PT1>, <ALGS-PT2> and <ALGS-PT3>. The main reason for the creation of these (sub-)categories is related to the constraints of the multi-label meta-algorithms in the MEKA

Box 5.3: Defined grammar – Part 3: SLC functions, Bayesian and other types of algorithms.

```

<FUNCTIONS> ::= <VotedPerceptron> | <MultiLayerPerc> |
              (<StocGradDescent> | LogisticRegression) <r> | <SeqMinOptimization>

<VotedPerceptron> ::= <i> <mk> <e>
<i> ::= RANDOM-INT(1,10)                #i='number of iterations'
<mk> ::= RANDOM-INT(5000, 50000)        #mk='maximum number of alterations to the perceptrons'
<e> ::= RANDOM-REAL(0.2, 5.0)           #e='The exponent for the polynomial kernel'

<MultiLayerPerc> ::= <lr> <m> <nhn> [n2b] [r] [d]
#n2b='nominal to binary filter',
#r='use reset approach',
#d='decay in the learning rate'
<lr> ::= RANDOM-REAL(0.1, 1.0)          #lr='learning rate'
<m> ::= RANDOM-REAL(0.0, 1.0)          #m='momentum'
<nhn> ::= a | i | o | t                 #nhl='rules to define the number of hidden nodes'

<StocGradDescent> ::= <lf> <lr_sgd> [nn] [nrmv]
#nn='do not normalize',
#nrmv='do not replace missing values'
<lf> ::= 0 | 1 | 2                      #lf='loss function'
<lr_sgd> ::= RANDOM-REAL(0.00001, 1.0) #lr_sgd='learning rate for SGD'
<r> ::= RANDOM-REAL(0.000000000001,10.0) #r='ridge value in the log-likelihood'

<SeqMinOptimization> ::= <c> <ft> [bcm] <kernel>
#bcm='build calibration models'
<c> ::= RANDOM-REAL(0.5,1.5)            #c='the cost, i.e.,complexity parameter'
<ft> ::= 0 | 1 | 2                      #ft='filter type'
<kernel> ::= ( NormPolyKernel |
               PolyKernel
               ) <exp> [ulo] |
               Puk <om> <sig> | RBF <g>
#ulo='use lower order'
<exp> ::= RANDOM-REAL(0.2, 5.0)         #exp='the exponent'
<om> ::= RANDOM-REAL(0.1, 1.0)         #om='the omega value'
<sig> ::= RANDOM-REAL(0.1, 10.0)       #sig='the sigma value'
<g> ::= RANDOM-REAL(0.001, 1.0)       #g='the gamma value'

<BAYES> ::= NaiveBayes [<NB-Parameters>] | <BayesianNetworkClassifiers> | NaiveBayesMultinomial
<NB-Parameters> ::= uke | usd
#uke='use kernel estimator'
#usd='use supervised distribution'

<BayesianNetworkClassifiers> ::= TAN | K2 | HillClimber | LAGDHillClimber | SimulatedAnnealing | TabuSearch

<OTHERS> ::= (SimpleLogistic [ucv] |
              <LogisticModelTrees>
              ) [uaic] [<wtb>]
#ucv='use cross-validation'
#uaic='use AIC measure as stopping criteria'

<LogisticModelTrees> ::= <mno> [cn] [sor] [fr] [eop]
#cn='convert nominal to binary'
#sor='split on residuals'
#fr='fast regression', eop='error on probabilities'
<wtb> ::= RANDOM-REAL(0.0, 1.0)        #wtb='weight trim beta'

```

software. Although all MLC algorithms can be used in a standalone fashion, they can also be combined with multi-label meta-algorithms. In MEKA, some MLC algorithms work very well at the multi-label base level of meta-algorithms, whereas others do not. Thus, we had to create rules in the grammar to overcome the limitations of the used software. The next paragraphs will refer to the Boxes 5.5 and 5.7 to explain these links and constraints between problem transformation algorithms and multi-label meta-algorithms.

We referred to the first production rule to define problem transformation methods as <ALGS-PT1> in Box 5.5. This rule encompasses the traditional algorithms BR, CC, and LC (which is also known as LP). Besides, it includes the quick versions of BR and CC (i.e., BRq and CCq), all the complex classifier chains and trellis algorithms (which are defined by the rule <ComplexCC-Trellis>), Four-class pairWise (FW), Ranking and Threshold (RT), and all the label-powerset based algorithms (which are defined by the rule

Box 5.4: Defined grammar – Part 4: SLC meta-algorithms.

```

<META1> ::= <LWL> | <AdaM1>

<LWL> ::= <k_lwl> [<wk>]
<k_lwl> ::= -1 | 10 | 30 | 60 | 90 | 120
<wk> ::= 0 | 1 | 2 | 3 | 4
#LWL='Locally Weighted Learning'
#k_lwl='number of neighbors in LWL'
#wk='weighting kernel'

<AdaM1> ::= <wt> [<ur>] <ni_ada_and_bagging>
<wt> ::= RANDOM-INT(50, 100) | 100
<ni_ada_and_bagging> ::= RANDOM-INT(2, 128)
#ur='use resampling'
#wt='weight threshold'
#ni_ada_and_bagging='number of iterations for
# AdaM1 and Bagging'

<ALG-WEIGHTED-TYPE> ::= <TREES> | <RULES-PARTIAL> | <KNN> | <BAYES> | <FUNCTIONS-PARTIAL>
<RULES-PARTIAL> ::= <DT> | <JRip> | <PART> | ZeroR
<FUNCTIONS-PARTIAL> ::= <MultiLayerPerc> | <SeqMinOptimization> | <SimpleLogistic> <uaic> <wtb_activate>

<META2> ::= RandomCommittee <ni_random_methods>
<ni_random_methods> ::= RANDOM-INT(2, 64)
#ni_random_methods='number of iterations for
#random methods'
<ALG-RANDOM-TYPE> ::= ( ( (RandomForest <nt> | <RandomTree>) <nf> ) | <REPTree> ) <md> |
StocGradDescent <r> | <MultiLayerPerc>

<META3> ::= <Bagging> | <RandomSubspace>

<Bagging> ::= (<bsp> | 100 coob) <ni_ada_and_bagging>
#coob='calculate out-of-bag'
#when coob is true, bag percent size must be 100
#bsp='bag size percent'
<bsp> ::= RANDOM-INT(10, 100)
<RandomSubspace> ::= <sss> <ni_random_methods>
<sss> ::= RANDOM-REAL(0.1, 1.0)
#sss='subspace size'

```

<LP\_based>). The production rule <ALGS-PT1> is presented in the following rules in Box 5.7: <META-MLC1> (via <ALGS-PT>), <META-MLC2> and <META-MLC3>. This means that this category of PT methods describes the majority of the MLC algorithms in MEKA (84.21% of the cases, i.e., 16 of the 19 MLC algorithms) and, in addition, all these algorithms can be combined with all meta-algorithms in the MEKA software. Thus, <ALGS-PT1> can be considered the least restrictive of the PT method rules in the grammar.

<ALGS-PT2>, in Box 5.5, is the production rule to describe solely the Bayesian Classifier Chain (BCC) algorithm, one of the most constrained algorithms in the MEKA software. The BCC algorithm can only be executed in a standalone fashion or combined with the algorithms described by the production rules <META-MLC1> (via <ALGS-PT>) and <META-MLC3>. This means that BCC can be used with four (4) of the seven (7) meta-algorithms (in Box 5.7): Subset Mapper (SM), Random Subspace Multi-Label (RSML), Expectation Maximization (EMax) and Classification Maximization (CMax). In other cases of trying to use BCC, this will result in errors in MEKA's output and, therefore, these cases were not allowed in the grammar.

Similarly to <ALGS-PT2>, we have <ALGS-PT3>, a problem transformation rule that represents the Population of Monte-Carlo Classifier Chains (PMCC) algorithm. This algorithm can only be used by itself and at the multi-label base level of five (5) of the seven (7) meta-algorithms: Subset Mapper (SM), Random Subspace Multi-Label (RSML), Bagging of Multi-Label methods (BaggingML), Bagging of Multi-Label methods with Du-



## Box 5.5: Defined grammar – Part 5: MLC problem transformation methods.

```

<ALGS-PT> ::= <ALGS-PT1> | <ALGS-PT2> | <ALGS-PT3>

<ALGS-PT1> ::= BR | CC | LC | (BRq | CCq) <dsr> |
              <ComplexCC_Trellis> | FW | RT | <LP_based>
              #BR='Binary Relevance', CC='Classifier Chain'
              #LC='Label Combination'
              #BRq and CCq = 'quick versions for BR and CC'
              #FW='Four-class pairWise', RT='Ranking-Threshold'

<ALGS-PT2> ::= BCC <dp_complete>
              #BCC='Bayesian Classifier Chain'
<ALGS-PT3> ::= PMCC <B> <ts> <ii> <chi_PMCC> <ps> <pof>
              #PMCC='Population of Monte-Carlo Classifier Chains'

<dsr> ::= RANDOM-REAL(0.2, 0.8)
              #dsr='down-sample ratio'

<ComplexCC_Trellis> ::= PCC | (MCC <chi_MCC> | <CT>) <ii> <pof> |
                      (CDN | <CDT>) <i_cdn_cdt> <ci>
                      #PCC='Probabilistic Classifier Chains'
                      #MCC='Monte-Carlo Classifier Chains'
                      #CT='Classification Trellis'
                      #CDN='Conditional Dependency Networks'
                      #CDT='Conditional Dependency Trellis'

<chi_MCC> ::= <chi_CT> | 0
              #chi_MCC='number of chain iterations for MCC'
<ii> ::= RANDOM-INT(2, 100)
              #ii='number of inference interactions'
<pof> ::= Accuracy | Jaccard index | Hamming score | Exact match | Jaccard distance | Rank loss |
         Hamming loss | Zero One loss | Harmonic score | Log Loss lim:L | Micro Recall | One error |
         Log Loss lim:D | Micro Precision | Macro Precision | Macro Recall | F1 micro averaged |
         Avg precision | F1 macro averaged by example | F1 macro averaged by label | AUPRC macro averaged |
         AURUC macro averaged | Levenshtein distance
              #pof='Payoff function'

<CT> ::= <chi_CT> <w> <dp>
<dp> ::= C | I | Ib | Ibf | H | Hbf | X | F | None
              #dp='dependency type'
<chi_CT> ::= RANDOM-INT(2, 1500)
              #chi_CT='number of chain iterations for CT'
<w> ::= 0 | 1 | -1 <d>
              #w='width of the trellis'
<d> ::= RANDOM-INT(1, SQRT(L) + 1)
              #d='neighborhood density'
              #Where L is the number of labels
<CDT> ::= <w> <dp>
              #parameters defined earlier

<i_cdn_cdt> ::= RANDOM-INT(101, 1000)
              #i_cdn_cdt='total number of iterations'
<ci> ::= RANDOM-INT(1, 100)
              #ci='collection iterations'

<LP_based> ::= (PS | PSt | <RAkEL-based>) <sv> <pv>
              #PS='Pruned Sets'
              #PSt='Pruned Sets with Threshold'
<sv> ::= RANDOM-INT(0, 5)
              #sv='subsampling value'
<pv> ::= RANDOM-INT(1, 5)
              #pv='pruning value'

<RAkEL-based> ::= (RAkEL <sre> | RAkELd) <les>
              #RAkEL='Random k-labEL Pruned Sets'
              #RAkELd='Random k-labEL Disjoint Pruned Sets'
<sre> ::= RANDOM-INT(2, min(2L, 100))
              #sre='number of subsets to run in an ensemble'
<les> ::= RANDOM-INT(1, L/2)
              #les='number of labels in each label subset'
              #Where L is the number of labels
<dp_complete> ::= <dp> | LEAD
              #dp='complete dependency type for BCC'

<B> ::= RANDOM-REAL(0.01, 0.99)
              #B='Beta factor for decreasing the temperature'
<ts> ::= 0 | 1
              #ts='Temperature switch'
<ps> ::= RANDOM-INT(1, 50)
              #ps='population size'
<chi_PMCC> ::= RANDOM-INT(51, 1500)
              #chi_PMCC='number of chain iterations for PMCC'

```

plicates (BaggingMLDup) and Ensemble of Multi-Label methods (EnsembleML). These five multi-label meta-algorithms are defined by the production rules <META-MLC1> and <META-MLC2>. Therefore, the creation of <ALGS-PT2> is justified by the fact that the PMCC algorithm can only be combined with these meta-algorithms, i.e., a constraint that did not appear in the other rules of the grammar.

Besides the problem transformation methods, we also have a multi-label version of the back-propagation algorithm for training neural networks, called ML-BPNN. This algorithm is presented in Box 5.6 and is the only one (for now) representing the algorithm adaptation (AA) methods, defined by the production rule <MLC-AA>. ML-BPNN can

also be associated with meta-algorithms. As we can see in Box 5.7, this MLC algorithm can be linked to the meta-algorithms defined by the production rules <META-MLC1>, <META-MLC2> and <META-MLC3>.

Box 5.6: Defined grammar – Part 6: MLC algorithm adaptation methods.

```

<MLC-AA> ::= <ML-BPNN>

<ML-BPNN> ::= <ne> <nhu_bpnn> <lr_bpnn> <m_bpnn>
#ML-BPNN='Multi-Label Back Propagation
# Neural Network'
#ne='number of epochs'
#nhu_bpnn='number of hidden units, that
#is a parameter that depends on the
#number of attributes of the dataset'

<ne> ::= RANDOM-INT(10, 1000)
<nhu_bpnn> ::= RANDOM-REAL(0.2, 1.0) * n_attributes

<lr_bpnn> ::= RANDOM-REAL(0.001, 0.1)
#lr_bpnn='learning rate for BPNN/DBPNN'
<m_bpnn> ::= RANDOM-REAL(0.2, 0.8)
#m_bpnn='momentum for BPNN and DBPNN'

```

Finally, Box 5.7 covers all the multi-label meta-algorithms, which are defined by the production rule <META-MLC-LEVEL>. As previously explained, we created the production rules <META-MLC1>, <META-MLC2> and <META-MLC3> in order to expand these five rules into <META-MLC-LEVEL> to control the limitations, constraints, and dependencies of the MEKA software between meta-algorithms and multi-label algorithms (problem transformation and algorithm adaptation methods).

Box 5.7: Defined grammar – Part 7: MLC meta-algorithms.

```

<META-MLC-LEVEL> ::= <META-MLC1> | <META-MLC2> | <META-MLC3>
#META-MLC 1-3='meta MLC algorithms
# with different constraints'

<META-MLC1> ::= (SM | <RSML>) (<ALGS-PT> <ALGS-SLC> | <ML-BPNN>)
#SM='Subset Mapper -- MLC method as parameter'

<RSML> ::= <bsp> <i_metamlc> <ap>
#RSML='Random Subspace Multi-Label'
#bsp='bag size percent'
#i_metamlc='number of iterations for
#meta MLC methods'
#ap='attribute percent'

<ap> ::= RANDOM-INT(10, 100)

<META-MLC2> ::= <alg-meta-mlc2> (<ALGS-PT1> | <ALGS-PT3>) <ALGS-SLC> | <ML-BPNN>

<alg-meta-mlc2> ::= ((BaggingML | BaggingMLDup <bsp>) | EnsembleML <bsp_ensembleML>) <i_metamlc>
#BaggingML='Bagging of Multi-Label methods'
#BaggingMLDup='BaggingML with duplicates'
#EnsembleML='Ensemble of Multi-Label methods'
#bsp='bag size percent -- defined earlier'

<bsp_ensembleML> ::= RANDOM-INT(52, 72)
#bsp_ensembleML='specific bsp for EnsembleML'

<META-MLC3> ::= ( (EMax | CMax ) <i_metamlc> ) (<ALGS-PT1> | <ALGS-PT2>) <ALGS-SLC> | <ML-BPNN>
#EMax='Expectation Maximization'
#CMax='Classification Maximization'

```

## 5.3 Search Methods for Automated Multi-Label Classification

In this section, we instantiate the *search method* component of our proposed methods. We have proposed four different methods: the first three, namely GA-Auto-MLC, Auto-MEKA<sub>GGP</sub>, and Auto-MEKA<sub>spGGP</sub>, are based on evolutionary algorithms. The fourth, Auto-MEKA<sub>BO</sub>, follows a BO approach. All proposed methods are open-source and are freely available for download on Github<sup>6</sup>.

### 5.3.1 Evolutionary-based Methods

This section details the three methods developed following the evolutionary algorithm framework. The main difference among these methods is in their representation (and hence genotype-phenotype mapping, and crossover and mutation operations), which has a direct impact on how the method searches the space of solutions.

However, all methods follow the main framework proposed by evolutionary methods, described in Algorithm 5.1 and already discussed in Section 2.1. Note that here, an individual represents an MLC algorithm generated from a combination of the available components in the *search space* (described in Section 5.2).

Given a *search space*  $Sp$ , we want to generate a customized MLC algorithm for a dataset  $Ds$ . In addition, we need to specify the basic parameters of the evolutionary framework (i.e., the population size, crossover and mutation probabilities, a selection method, and a fitness function). Considering these specifications, the evolutionary-based methods work as follows.

First, the dataset is divided into three different sets: the learning and validation sets – which will be used for training, and the test set. For training, 80% of the examples are in the learning set ( $Lr$ ) and 20% in the validation set ( $Val$ ). For the test ( $Ts$ ), 20% of the examples are used to evaluate the final predictive performance of the produced model.

To perform this partitioning, we use an iterative stratified sampling method [189]. In a nutshell, this sampling procedure adds the examples to the subsets by considering the rarest label at first. As there are multiple labels on the examples, it also takes into account the non-rare labels simultaneously. In this part, it just updates the counts for the non-rare labels as well. This process continues – by adding examples to the subsets

<sup>6</sup>Code and documentation are available at <https://github.com/alexgcsa/automlc/>

relying on the label rarity – until the least rare label is completely sampled.

Having the data division completed, the individuals in the initial population are generated at random, regardless of the representation used, according to the components of  $Sp$ . Then, while a maximum time budget  $MTB$  is not reached, a mapping process generates an MLC algorithm  $A$  from each individual of the population  $i$ . Next,  $A$  produces a classification model  $M$  using  $Lr$ , and the fitness of the individual is calculated using the validation set  $Val$ , as discussed in Section 5.3.1.1.

Following, the best individual of the population is saved to be added to the new population (an elitist process), and a probabilistic selection process starts to choose the individuals that will suffer mutation and crossover. In the end, the current population is updated. In order to avoid overfitting, during the evolutionary process, the learning and validation sets are resampled every  $R$  generations. Besides, to control premature convergence, we check if the best individual is kept the same for a predefined number of generations,  $GPC$ , and if the search has reached at least a predefined number of generations, which is based on  $CG$ . To do so, we maintain a list of the best individuals found so far. If this convergence criteria is reached, we restart the population by creating new  $S$  individuals.

---

**Algorithm 5.1** General pseudo-code for evolutionary algorithms for MLC.

---

- 1: **Inputs:** Search space,  $Sp$ ; Dataset,  $Ds$ ; Population size,  $S$ ; Tournament parent selection mechanism,  $TS$ ; Crossover probability,  $P_c$ ; Mutation probability,  $P_m$ ; Fitness function,  $f$ ; Maximum Time Budget,  $MTB$ ; Number of generations on resampling,  $R$ ; Convergence Criteria Function,  $CT$ ; Current number of generations,  $CG$ ; Number of generations to population's convergence,  $GPC$ .
  - 2: Divide  $Ds$  into three partitions,  $Lr$ ,  $Val$ ,  $Ts$ ;
  - 3:  $P =$  Randomly generate an initial population of  $S$  individuals using definitions in  $Sp$ ;
  - 4: **while** maximum time budget  $MTB$  not reached **do**
  - 5:   Map each individual  $i$  in  $P$  to a MLC algorithm  $A_i$
  - 6:    $M(i)$ : Train each algorithm  $A_i$  using  $Lr$ ;
  - 7:    $f(i)$ : Calculate the fitness of each individual  $A_i$  in  $Val$ ;
  - 8:    $best = A_i$  with best  $f(i)$ , saved to the new population;
  - 9:    $[History]$ : Best ▷ Keep the history of the best individuals.
  - 10:    $[SI]$ : Select individuals via  $TS$ ;
  - 11:   Crossover( $[SI]$ ,  $P_c$ );
  - 12:    $P' =$  Mutation( $[SI]$ ,  $P_m$ );
  - 13:    $P = [P', best]$ ;
  - 14:   Resample  $Lr$  and  $Val$  every  $R$  generation;
  - 15:   Restart  $P$  with new  $S$  individuals if  $CT([History], CG, GPC)$  is reached
  - 16: **end while**
  - 17: **return**  $A_i$  with the best fitness value;
  - 18: Run  $A_i$  in  $Ts$ .
- 

Next, in Sections 5.3.1.2, 5.3.1.3 and 5.3.1.4, we present the peculiarities for the three proposed algorithms, focusing on individual representation, genotype-phenotype

mapping, and genetic operators.

### 5.3.1.1 MLC Fitness Evaluation Process

In multi-label classification, an MLC algorithm is usually evaluated using multiple measures because of the additional degrees of freedom the MLC setting introduces, as already explained in Section 3.2.

After a MLC model is induced by  $A$ , the fitness function is calculated using the validation set, using the average of four MLC measures, described in Section 3.2: Exact Match (EM), Hamming Loss (HL),  $F_1$  Macro averaged by label (FM) and Ranking Loss (RL), as indicated in Equation 5.1:

$$Fitness = \frac{EM + (1 - HL) + FM + (1 - RL)}{4} \quad (5.1)$$

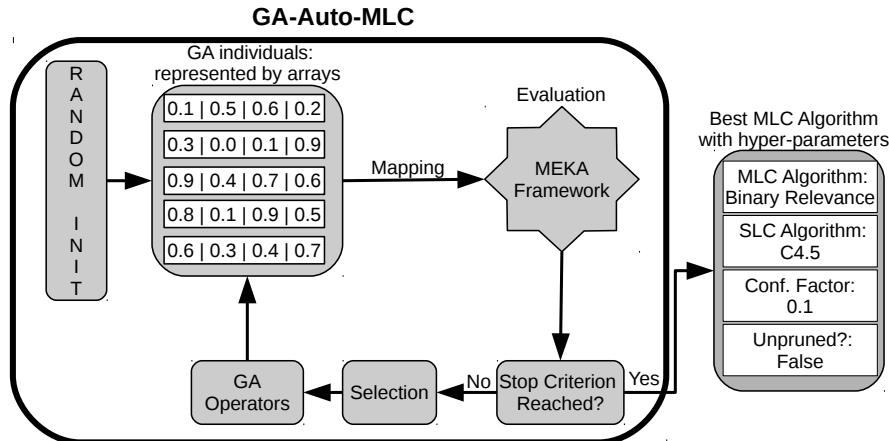
All four metrics in Equation 5.1 are within the  $[0, 1]$  interval. However, EM and FM are measures that should be maximized, whereas HL and RL should be minimized. In order for the fitness function to be maximized, in Equation 5.1, HL and RL are subtracted from one (1).

It is important to highlight that these four measures were chosen because they usually present low correlation among each other, when comparing their results across different algorithms within different MLC problems [162].

### 5.3.1.2 Genetic Algorithm for Automated Multi-Label Classification

Following the principles described in the previous section, the first method we propose is GA-Auto-MLC (Genetic Algorithm for Automated Multi-Label Classification). It uses a real-valued GA as its search engine, and its main steps are detailed in Figure 5.2. They reflect the pseudo-code already discussed in Algorithm 5.1.

Figure 5.2: GA-Auto-MLC: The proposed genetic algorithm to select and configure MLC algorithms.



This section discusses the individual representation and the genotype/phenotype mapping, which are the particularities of this method. Apart from that, the algorithm implements a roulette wheel individual selection, followed by uniform crossover and one-point mutation operations (details of these operations are given in Chapter 2).

**Individual Representation:** GA-Auto-MLC is based on the individual representation proposed in de Sá and Pappa [49] to search and explore the space of MLC algorithms. GA-Auto-MLC is guided by the hierarchy of algorithm choices defined by the *search space*. It only generates valid individuals at the phenotypic level. A set of configuration files wraps the MLC hierarchy and, consequently, the whole *search space*. These configuration files have the same role as the grammar and describe the algorithms, hyper-parameters, and constraints found in the search space. Nevertheless, they are represented with a markup language (in our work, a set of Extensible Markup Language [XML] files are used to encompass the *search space*).

Given the algorithms and their respective components, we followed the hierarchy to create a suitable representation for them. An individual genotype is a real-valued array with values within the  $[0, 1]$  interval.

Considering that each learning algorithm has a different number of hyper-parameters (see Tables 5.1 and 5.2, and Appendix A to check this information), the GA uses a dynamic representation, but only at the phenotypic level. The individual genotype representation is static, always having 12, 16, or 26 positions. Therefore, a position in the real-valued array can be ignored during the mapping process. Crossover and mutation operations are then applied to the individual genotype (real-coded array) to avoid problems of individuals with different sizes. Figure 5.3 gives examples of possible phenotypes<sup>7</sup>. In this figure, a gray box means that this array position refers to an empty component.

Figure 5.3: Possible phenotypes of GA-Auto-MLC’s individuals (for the *search space* Small).

threshold	0.70	MLC: BR	SLC: LogR	r: 0.1	-	-	-	-	-	-	-
threshold	PCut1	MLC: BPNN	ne: 100	nhu: 12	lrt: 0.02	m: 0.5	-	-	-	-	-
threshold	0.09	MLC: RAKEL	pv: 4	sv: 2	les: 3	sre: 100	SLC: NB	uke: false	usd: true	-	-
threshold	PCut1	MLC: RAKEL	pv: 1	sv: 0	les: 1	sre: 10	SLC: JRip	mtw: 2.5	cer: false	up: true	o: 3

As the size of the individual depends on the search space, individuals will have different sizes for the three *search spaces* proposed in Section 5.2. For the *search space*

<sup>7</sup>Linked to the acronyms of algorithms and hyper-parameters in the Appendix A.

Small, we have a genome size of 12. I.e., the maximum number of options an MLC algorithm can have at this *search space* is 12: the choice of the threshold approach (which was modeled using two positions in the genome<sup>8</sup>, choosing JRip at the single-label base level (four hyper-parameters and the choice of the method itself) and RAKEL at the multi-label base level (four hyper-parameters and the choice of the method itself).

In contrast, the *search space* Medium has a genome size of 16. In this case, the maximum number of options would be the selection of the threshold approach in two positions, choosing the C4.5 algorithm at the single-label base level (eight hyper-parameters and the choice of the method itself) and RAKEL at the multi-label base level (five positions in the genome).

The *search space* Large is the most complex and, consequently, has a bigger genome size. At this *search space*, we could have a maximum of 26 options to build a MLC algorithm. This would happen if we consider the threshold approach (two positions in the genome), the C4.5 algorithm at the single-label base level (nine positions in the genome), AdaM1, bagging or RSS at the meta-SLC level (three hyper-parameters and the choice of the method itself), CT or PMCC at the multi-label base level (six hyper-parameters and the choice of the method itself) and RSML as the meta MLC algorithm (three hyper-parameters and the choice of the method itself). Figure 5.4 gives an example of a possible genotype for the *search space* Small.

Figure 5.4: A possible genotype of an individual in the *search space* Small.

.95	.26	.42	.54	.41	.59	.40	.51	.07	.01	.48	.59
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

**Genotype-Phenotype Mapping:** Figure 5.5 illustrates the whole process of mapping followed by individual evaluation. To get the phenotype from this real-valued array, we convert it into an intermediate representation, which is an integer-coded chromosome of the same length as the original individual, based on the configuration files.

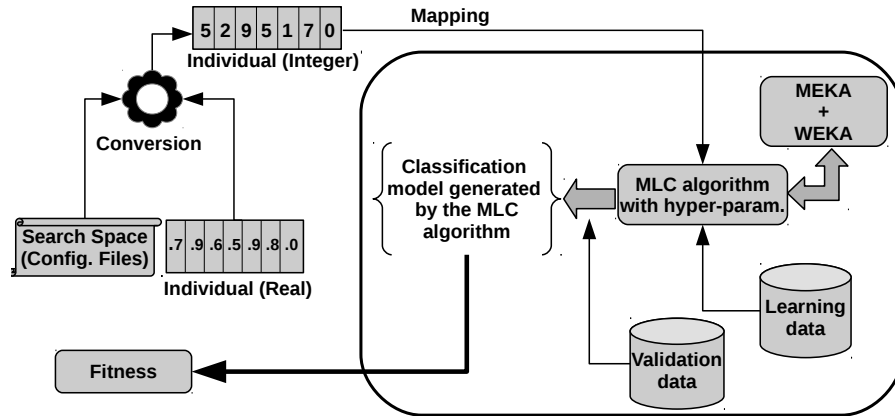
In this conversion, the real number of a gene is multiplied by the maximum number of choices associated with that component and the resulting value is rounded into an integer number that indicates a component option in the configuration file. For instance, suppose that the real value of a gene representing the single-label classification algorithm for a given PT method is equal to 0.20, and the total number of components for SLC algorithms is equal to 10. In this case, the rounded integer value will be 2, which means the second SLC algorithm – presume that the second SLC algorithm would be a Random Forest – in this configuration file will be selected. When the mapping process finishes,

<sup>8</sup>This was done because of a drawback in the GA representation. To define a hyper-parameter standalone, we need a tuple <gene, allele>. Therefore, we need two positions in the genome to represent an isolated hyper-parameter. This does not happen when we have an algorithm together with the hyper-parameter.

unused genes in the integer-coded chromosome receive the value -1, representing empty components.

Given the integer-coded individual, its chromosome is mapped into an MLC algorithm based on the configuration files. After that, we used the MEKA and WEKA frameworks to determine the MLC algorithm and calculate the fitness value based on the process described in Section 5.3.1.1.

Figure 5.5: Evaluation process of one individual in GA-Auto-MLC.



### 5.3.1.3 Automated Multi-Label Classification using Grammar-based Genetic Programming

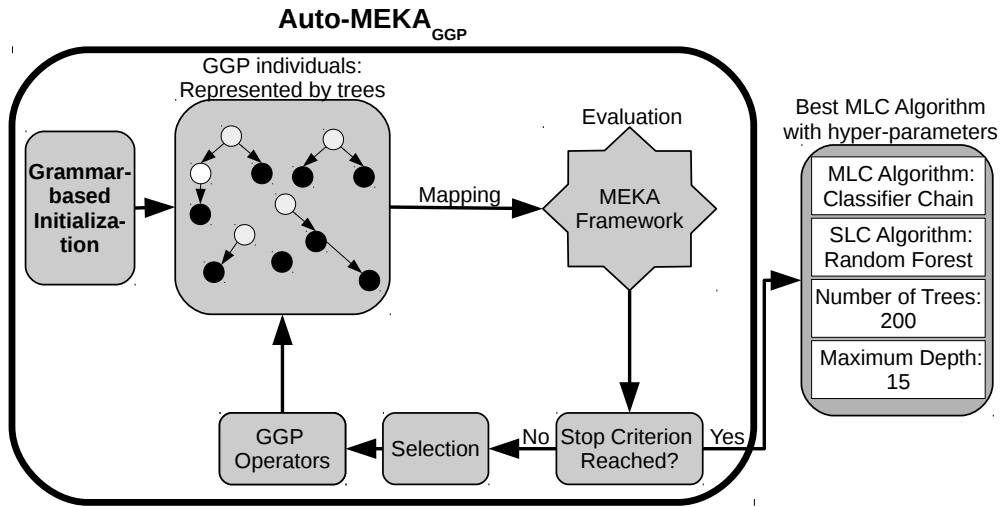
This section introduces Auto-MEKA<sub>GPP</sub>, a method that relies on a Grammar-based Genetic Programming (GPP) search to automatically select and configure MLC algorithms. The GPP search has the advantage of naturally exploring the hierarchical nature of the problem, a missing feature of GA-Auto-MLC.

In order to do that, Auto-MEKA<sub>GPP</sub> relies on a grammar (see Section 5.2.3), which encompasses the (hierarchical) *search space* of MLC algorithms and their hyper-parameters. Besides, the grammar directly influences the search, as each individual created by the GPP is based on its production rules, guaranteeing that all individuals are valid. In other words, the MLC grammar defines the *search space* and how the individuals are created and modified. Figure 5.6 illustrates the main steps of Auto-MEKA<sub>GPP</sub>, again in accordance with Algorithm 5.1.

In Auto-MEKA<sub>GPP</sub>, each individual is represented by a derivation tree generated from the grammar that represents the search space. Individuals are first generated by choosing valid production rules from the grammar at random. By choosing the production rules, the method also derives the respective parse trees for the individuals. Auto-MEKA<sub>GPP</sub> uses the *ramped half-and-half* method to create the initial population of individuals (see Section 2.1.3 for more details). The mapping process here is much simpler than in GA-Auto-MLC, as described later in this section.



Figure 5.6: Auto-MEKA<sub>GPP</sub>: The proposed GGP method to select and configure MLC algorithms.

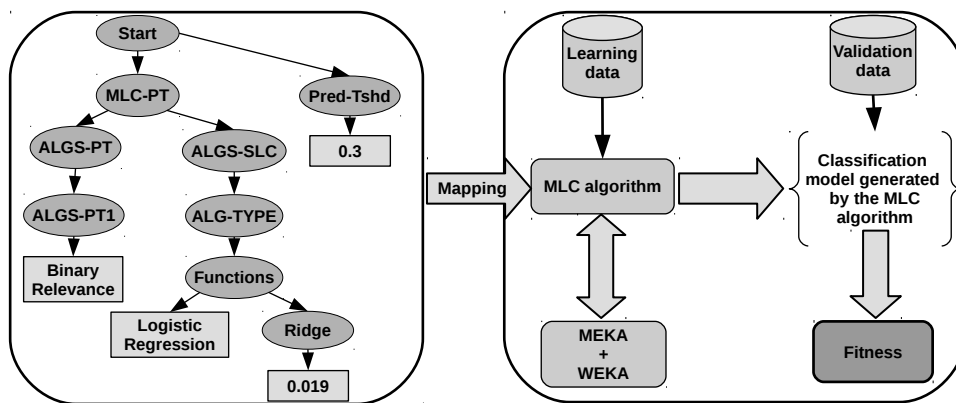


Auto-MEKA<sub>GPP</sub> selects individuals by using tournament selection. Next, the GGP operators (i.e., Whigham’s crossover and mutation [214]) are applied to the selected individuals to create a new population. These operators also respect the grammar constraints, ensuring that the produced individuals represent valid solutions.

It is worth noting that Auto-MEKA<sub>GPP</sub> was implemented using a modified version of EpochX [157], an open-source genetic programming framework.

**Genotype-Phenotype Mapping:** Figure 5.7 details the whole mapping process followed by the evaluation process for each GGP individual. Recall that each individual is represented by a tree, which is derived from the expansion of the production rules of the MLC grammar. In the example of Figure 5.7, ellipsoid nodes are the non-terminals, whereas the rectangles are the terminals.

Figure 5.7: Evaluation process of one individual in Auto-MEKA<sub>GPP</sub>.



The mapping process takes the terminals from the tree and constructs a valid MLC algorithm. The mapping in the figure will produce the following MLC algorithm:

a Binary Relevance method combined with a Logistic Regression algorithm (with the hyper-parameter ridge set to 0.019), using a threshold level of 0.3 to classify the MLC data. Given the mapped MLC algorithm in MEKA (and WEKA), the fitness function follows the same process described in Section 5.3.1.1.

#### 5.3.1.4 Automated Multi-Label Classification using Specialized Grammar-Based Genetic Programming

This section introduces a variation of Auto-MEKA<sub>GPP</sub> that adds a speciation process to the search method [9], namely Auto-MEKA<sub>spGPP</sub>. The general idea is to use speciation to improve the trade-off between exploration and exploitation of the search for MLC algorithms and hyper-parameters. Because the proposed *search spaces* have an exponential size and a complex hierarchical nature, it may be crucial to use this approach to deal with these aspects.

A species is basically a set of individuals that resemble each other more inherently than the individuals in another species [9]. Back et al. [9] also state that speciation (in evolutionary computation) has the idea of restricting mating to that among like individuals from the population. In this case, likeness among individuals (i.e., solutions to the problem at hand) is identified if they have similar genotypes or phenotypes. On the other hand, speciation tries to discourage mating among individuals with distinct genotypes or phenotypes.

There are a few different ways to introduce species to the problem. One of them is by separating the MLC (and SLC) algorithms into groups – e.g., we could have a species that groups the algorithms based on label powerset (including label powerset itself, pruned sets, and RAKEL), where different groups of learning algorithms would correspond to distinct species. Another approach is trying a traditional niching scheme (e.g., fitness sharing [9, 137]).

We decided to define a set of species for Auto-MEKA<sub>spGPP</sub>, but each species is based on the types of hyper-parameters (i.e., categorical, discrete, or continuous) and their interactions. Therefore, our speciation-based method focuses not exclusively on the choice of the learning algorithms but primarily on the different types of hyper-parameters, where the choice of the learning algorithms is set as a special case of the categorical hyper-parameter.

In general lines, we would like to understand if there is a dependence between the final AutoML predictive performance and the types (and the interactions) of hyper-parameters for a given dataset of interest. For instance, if we would like to recommend MLC algorithms for two datasets with different characteristics, understanding the categorical hyper-parameters for the first dataset may be more beneficial than approaching discrete and/or continuous hyper-parameters. This could be the opposite for the second

dataset, where the interaction between discrete and continuous hyper-parameters may be more relevant.

Based on this scenario, we design eight (8) species. All species may have instances of all learning algorithms at both MLC and SLC levels, but they vary on the types of hyper-parameters that are left with their default values during evolution and cannot be updated. The eight species are:

**Species 1 – Learning algorithms:** Only the categorical hyper-parameters referring to the names of the (traditional and meta) learning algorithms at the MLC and SLC levels can be combined and evolved. Categorical (not referring to the names of the learning algorithms), continuous, and discrete hyper-parameters are set to their default values<sup>9</sup>.

**Species 2 – Learning algorithms and common categorical hyper-parameters:** Together with the categorical hyper-parameters indicating the names of the learning algorithms (Species 1), this species also allows the combination and evolution of common categorical hyper-parameters (e.g., the names of a metric). In addition, it is important to emphasize that this species also encompasses Boolean hyper-parameters. Discrete and continuous hyper-parameters, in turn, are set to their default values.

**Species 3 – Learning algorithms and discrete hyper-parameters:** This species considers, alongside the categorical hyper-parameters indicating the names of the learning algorithms, the discrete (integer) hyper-parameters. The remaining types of hyper-parameters are set to their default values.

**Species 4 – Learning algorithms and continuous hyper-parameters:** During the evolution of the individuals at this species, we allow the modification and combination of the continuous hyper-parameters of Species 1. Common categorical and discrete hyper-parameters are not explored and remain with their default values.

**Species 5 – Learning algorithms and the combination of common categorical and discrete hyper-parameters:** At this species, we evolve the individuals considering the learning algorithms themselves (Species 1) together with common categorical and discrete hyper-parameters, which are also allowed to be modified and combined. This would model the interaction of categorical and discrete hyper-parameters. Continuous hyper-parameters have their values set to their default values.

---

<sup>9</sup>The default values are defined in Appendix A.

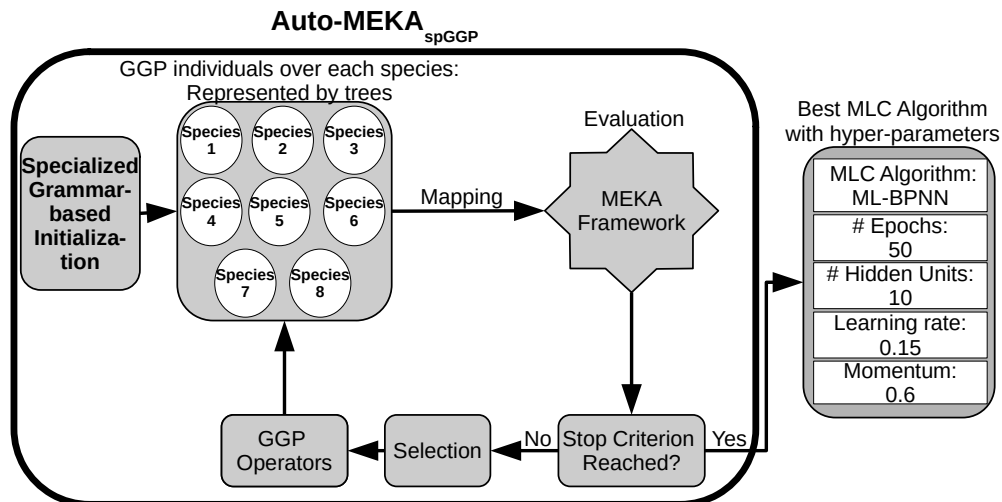
**Species 6 – Learning algorithms and the combination of common categorical and continuous hyper-parameters:** In this case, we make the evolutionary process to consider the hyper-parameters representing the learning algorithms, the common categorical hyper-parameters and the continuous hyper-parameters. Therefore, the interaction between categorical and continuous hyper-parameters is modeled. Discrete (i.e., integer) hyper-parameters are not explored in this species.

**Species 7 – Learning algorithms and the combination of discrete and continuous hyper-parameters:** This species allows the combination of the names of the learning algorithms with discrete and continuous hyper-parameters. In other words, this species tries to understand the effects of the interaction of discrete and continuous hyper-parameters on the selection of learning algorithms. However, this species does not take into consideration the common categorical hyper-parameters, which take their default values.

**Species 8 – All types of hyper-parameters:** This species is more general and recalls the grammar defined in Section 5.2.3, where all types of hyper-parameters (categorical referring to the names of the learning algorithms, common categorical, discrete, continuous hyper-parameters) are modeled to be explored/exploited.

Figure 5.8 gives an overview of the main steps of  $\text{Auto-MEKA}_{\text{spGPP}}$ . Broadly speaking, these steps are quite similar to  $\text{Auto-MEKA}_{\text{GPP}}$ 's evolutionary process. Basically, each species takes the steps defined in Figure 5.6. Here, we are going to underline the differences between  $\text{Auto-MEKA}_{\text{spGPP}}$  and  $\text{Auto-MEKA}_{\text{GPP}}$ .

Figure 5.8:  $\text{Auto-MEKA}_{\text{spGPP}}$ : The proposed speciation-based GPP method to select and configure MLC algorithms.



The first step of Auto-MEKA<sub>spGGP</sub>'s evolutionary process is the initialization procedure. Here, we have for each species a population of individuals, which are represented by trees and built based on a specific grammar for that species.

Besides initialization, Auto-MEKA<sub>spGGP</sub> differs from Auto-MEKA<sub>GGP</sub> in the cross-over operator, which can be performed for both intra-species and inter-species individuals. By interchangeably using both types of crossover operations, we have more chances to test unknown regions of the search space (exploration) when using the inter-species crossover, while a more straightforward local search over the different types of hyper-parameters is performed in each species by the intra-species.

It is worth noting that we decided to design the mutation operator as being local to each species. By doing that, Whigham's mutation uses the *grow* method on the individual's derivation tree but ensures that the MLC grammar of that respective species is applied over the *grow* method.

As the operators are based on speciation-based grammars, they also respect their respective production and constraints, ensuring that the produced individuals represent valid solutions (i.e., valid MLC algorithms).

### 5.3.2 Automated Multi-Label Classification using Bayesian Optimization

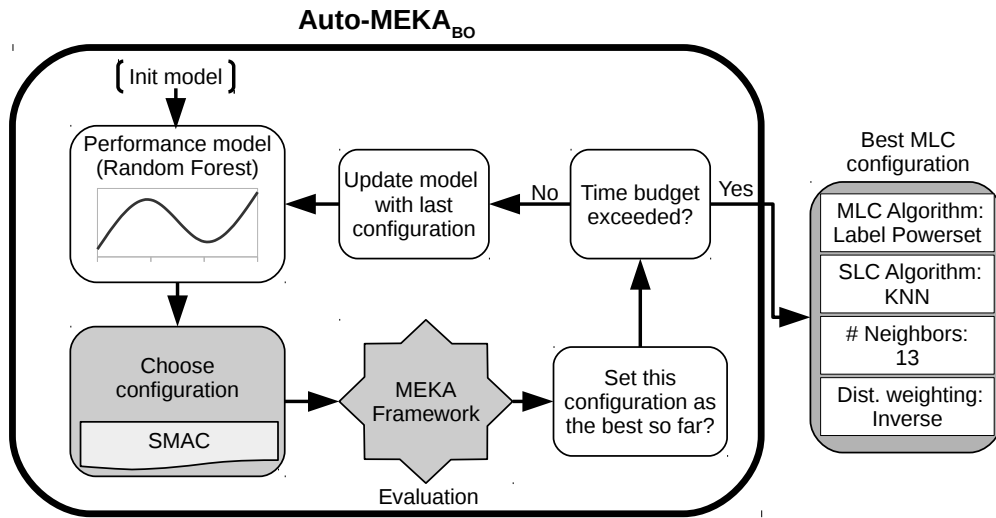
Besides the evolutionary-based *search methods*, we also developed a Bayesian Optimization (BO) *search method* to perform automated multi-label classification, namely Auto-MEKA<sub>BO</sub>. For more details about BO methods, see Section 2.2. This method is an extension of the Auto-WEKA tool [127, 200] for MLC domains.

Figure 5.9 illustrates the main aspects of Auto-MEKA<sub>BO</sub>. First, given an MLC dataset of interest and the defined *search space* (see Figure 5.1), a BO method uses a performance model (in our case, a Random Forest) to help the selection of MLC configurations (MLC algorithms with hyper-parameters).

In Figure 5.9, this model is initialized with a default MLC algorithm with default hyper-parameters. In the case of Auto-MEKA<sub>BO</sub>, we initialize the model with different algorithms as the *search spaces* allow different types of learning algorithms. For the *search space* Small, we run and include into the model the results of the Classifier Chain (CC) algorithm using the Naïve Bayes (NB) classification algorithm at the single-label base level.

As the *search space* Medium has a similarity in terms of the hierarchical levels, we decided to keep the CC algorithm at the multi-label level. However, we have tried to

Figure 5.9: Auto-MEKA<sub>BO</sub>: The proposed Bayesian optimization method to select and configure MLC algorithms.



improve the single-label classification level by using a more robust algorithm. I.e., we use a more sophisticated Bayesian Network Classifier (BNC) algorithm instead of a simple NB classification algorithm. Hence, at this level, the K2 algorithm is employed.

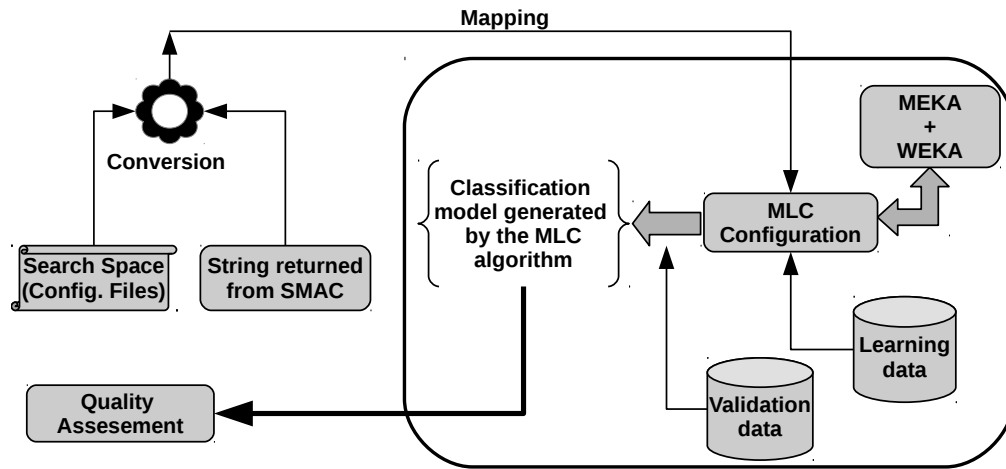
Finally, for the *search space* Large, we define as the initial configuration to the model the Random Subspace Meta-Learning algorithm for multi-label classification (RSML), using the Bayesian Classifier Chain (BCC) algorithm at the multi-label base level, the Locally Weighted Learning (LWL) algorithm at the single-label meta-level, and the BNC K2 algorithm at the single-label base level. Except for RSML, which is a robust meta-algorithm, the other levels were chosen in an arbitrary fashion, although they are also considered strong algorithms in the machine learning literature.

After this initialization step, we choose the next configuration from the MLC *search space* in the configuration files, relying on this performance model. To do that, the Sequential Model-based Algorithm Configuration (SMAC) method is used to select a better configuration (see Section 2.2.2). Next, this MLC configuration is evaluated in the MEKA framework and then compared with the best MLC configuration found so far (see Section 5.3.2.1 for more details). If the current configuration has a better score than the current best configuration, it is saved and set as the new best configuration. Otherwise, the process continues by verifying if the time budget (i.e., the instantiation of the stopping criteria) was reached. If this stopping criteria is met, Auto-MEKA<sub>BO</sub> returns the best configuration found until the current time to that input dataset. Otherwise, the last evaluated MLC configuration is added to the performance model with its corresponding quality value, updating it. The process continues following these same steps until the time budget expires.

### 5.3.2.1 Quality Function

Broadly speaking, the evaluation process in Auto-MEKA<sub>BO</sub> is quite similar to the ones presented in Section 5.3.1.1. Figure 5.10 illustrates this process. In this figure, we can observe that the main difference is that Auto-MEKA<sub>BO</sub> uses the string (representing the configuration, i.e., the MLC algorithm with its hyper-parameters) returned from the SMAC process, instead of the real-valued array or the parse tree. Auto-MEKA<sub>BO</sub> simply maps this string into an MLC algorithm by employing the configuration files, which describe the *search space*, and a set of conversion rules.

Figure 5.10: Evaluation process of one individual in Auto-MEKA<sub>BO</sub>.



Next, MEKA and WEKA frameworks are used to evaluate the respective MLC configuration in the same way as described in Section 5.3.1.1.

## 5.4 Multi-fidelity Methods for MLC

The idea of multi-fidelity is to reduce the computational cost involved when training the AutoML *search methods* on specific *search spaces*. In this thesis, the computational cost is related to train an MLC algorithm with a specific hyper-parameter setting. Depending on the type of the dataset of interest, it can be unfeasible to find good solutions to its associated ML problem. Using multiple types of fidelities to represent this data is one way to overcome its difficulty. In spite of that, this must be performed carefully in order to preserve the characteristics of the dataset of interest.

We propose different ways to reduce data complexity when training the AutoML methods. In order to perform it, we evaluate distinct multi-fidelity approaches by focusing

on two factors regarding the input dataset: the number of attributes and the number of instances. While the AutoML method is iterating, we keep evaluating and changing data multi-fidelity at specific checkpoints.

Our idea is to begin with a few instances and/or a few attributes. As the methods iterate, we augment data, increasing its complexity until it reaches the original number of instances and attributes.

To reduce data complexity on the number of attributes, we propose two procedures. The first employs a polynomial function (in our case, a linear function) to perform attribute selection. At the beginning of this procedure, we define a maximum number of steps on the multi-fidelity approach to follow,  $\#steps$ , and divide the number of attributes of the dataset by  $\#steps$ , resulting in a dataset with a lower number of attributes. In the next step, we decrease the value of  $\#steps$  by one and perform the division again. This is done until we reach the final *search method's* checkpoint, where  $\#steps$  finally turns into the value one, and we use all attributes of the dataset. By doing that, we change/increase the fidelity/complexity of data over time.

Table 5.5 shows how we increase polynomially the actual number of attributes ( $\#attributes$ ) as a factor of  $\#steps$  (which took the initial value of five) based on the employed *search method's* time budget.

Table 5.5: Employed polynomial multi-fidelity approach in terms of the number of attributes of the dataset based on the specified time budget.

Time budget range	Actual Number of Attributes
From 0 hour to 1 hour	$\#attributes / 5$
From 1 hour to 2 hours	$\#attributes / 4$
From 2 hour to 3 hours	$\#attributes / 3$
From 3 hour to 4 hours	$\#attributes / 2$
From 4 hour to 5 hours	$\#attributes / 1$

The second procedure is similar to the first, but it employs an exponential function to change the number of attributes over time. Therefore, we set the number of steps ( $\#steps$ ) as the exponent of a predefined base,  $b$ . In our case, we start dividing the number of attributes by  $b^{\#steps}$ . We keep decreasing the value of  $\#steps$  (at specific checkpoints) until its value reaches zero, where data transforms in your original form (i.e., with all attributes). Table 5.6 shows how we increase exponentially the actual number of attributes ( $\#attributes$ ) as a factor of  $\#steps$  (which took the initial value of four) based on the employed time budget.

For both procedures that vary the number of attributes, we select the attributes considering a ranking-based method, where the scores at its ranking are calculated considering the gain ratio of the label powerset attribute evaluator [201]. The decision about the attribute evaluator was taken to allow the attribute selection to consider the correlation



Table 5.6: Employed exponential multi-fidelity approach in terms of the number of attributes of the dataset based on the specified time budget.

Time budget range	Actual Number of Attributes
From 0 hour to 1 hour	$\#attributes / 2^4$
From 1 hour to 2 hours	$\#attributes / 2^3$
From 2 hour to 3 hours	$\#attributes / 2^2$
From 3 hour to 4 hours	$\#attributes / 2^1$
From 4 hour to 5 hours	$\#attributes / 2^0$

among the labels. For some MLC problems, this is inherently important and, therefore, must be taken care of.

Regarding the second dimension of multi-fidelity (i.e., number of examples), we test only one procedure for selecting instances on the datasets of interest. Basically, this procedure employs a polynomial function (in our case, a linear function) to divide data into learning and validation sets during *search method*'s training. It is worth noting that we always use a stratified cross-validation method to divide data [189]. In this scenario, we define a total number of folds to split the training data,  $\#folds$ , where we start learning the MLC models with one fold and validating with  $\#folds$  minus one. In the next step, we increase by one the number of folds to learn the MLC models and decrease by one the respective number of folds to validate these models. We perform these operations until we invert the number of folds from learning to validation. Table 5.7 describes how we actually change the number of folds (and, consequently, the number of instances) to train and validate the MLC models.

Table 5.7: Employed polynomial multi-fidelity approach in terms of the number of instances of the dataset based on the specified time budget.

Time budget range	# Folds to Learn	# Folds to Valid
From 0 hour to 1 hour	1	5
From 1 hour to 2 hours	2	4
From 2 hours to 3 hours	3	3
From 3 hours to 4 hours	4	2
From 4 hours to 5 hours	5	1

## 5.5 Final Remarks

Within these three proposed *search spaces* (Small, Medium and Large), four proposed *search methods* (GA-Auto-MLC, Auto-MEKA<sub>GGP</sub>, Auto-MEKA<sub>SpGGP</sub> and Auto-MEKA<sub>BO</sub>) and five different multi-fidelity approaches, we believe we gave a starting seed to the AutoML field for multi-label classification problems.

In general lines, the proposed *search spaces* differ from each other in terms of (hierarchical and extended) complexity. From the *search space* Small to Medium, we change just the number of possible learning algorithms (and, consequently, the number of considered hyper-parameters) the AutoML methods can explore/exploit. However, from the *search space* Medium to Large, we also include more hierarchical levels, which makes the search more challenging than on the two other *search spaces*.

Besides, recall that each *search method* has an inherent way to deal with the *search spaces*. GA-Auto-MLC, for example, encompasses the *search spaces* into an array. We observe that this form to represent individuals has traces to the Grammatical Evolution (GE) method [156] because of the dynamic phenotype generation for the different individuals. This gives GA-Auto-MLC an exploration nature as a simple modification on the genotype can produce a very distinct individual (MLC algorithm).

On the other hand, Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>SpGGP</sub> handles this hierarchical aspect of the *search space* naturally. This is due to the tree-based representation followed by genetic programming methods. Besides, both methods respect a context-free grammar to build MLC algorithms, which derives a formal validation of the *search space*. Nevertheless, Auto-MEKA<sub>SpGGP</sub> differs from Auto-MEKA<sub>GGP</sub> in the way it deals with the different types of hyper-parameters.

Furthermore, Auto-MEKA<sub>BO</sub> approaches the AutoML problem for MLC data by applying a *search method* that follows the Bayesian Optimization (BO) principles. Employing such a hierarchical method is necessary to evaluate if there are no issues in the search employed by the evolutionary methods. Therefore, in addition to the evolutionary *search methods*, we decide to test a state-of-the-art *search method* (i.e., a BO method) for the MLC AutoML scenario.

Finally, we define different ways to reduce the cost of training the AutoML methods, introducing four multi-fidelity approaches. Basically, they aim to change the instance and/or attribute spaces of the MLC problems in order to provide a simpler training dataset. Based on that, the MLC algorithms can be run faster, but as data loses fidelity in comparison to the original problem, the fitness, representing how good this algorithm is, also loses fidelity and can produce misleading results. Therefore, multi-fidelity approaches must be used carefully.

## Chapter 6

# Experimental Analysis

This chapter presents the experimental analysis regarding the proposed AutoML methods in the context of MLC. Experiments were divided into three phases. An initial set of experiments was performed with the four proposed methods in a set of tuning datasets, defining the first phase of the experiments. From there, we selected the most promising method – which was Auto-MEKA<sub>spGGP</sub> – and dedicated a new phase to tune the parameters of this single method. This decision was made based on the high cost involved in the experiments. In a third phase, we selected a new set of datasets and compared the results of Auto-MEKA<sub>spGGP</sub> to two out of three remaining proposed methods – i.e., Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BO</sub> –, and two baseline methods<sup>1</sup> – Auto-MEKA<sub>RS</sub>, which employs a random *search method* [14], and Auto-MEKA<sub>BS</sub>, which uses a beam *search method* [152] to seek for and configure MLC algorithms.

It is worth noting that the results of the third phase are also categorized into three main parts: (i) an analysis of the quality metrics used to evaluate the performance of the customized MLC algorithms generated by the four proposed AutoML methods and by the baseline methods; (ii) a discussion of the selected and configured MLC algorithms (and, consequently, the selected and configured SLC algorithms) tailored by the AutoML methods; (iii) an analysis of convergence of the AutoML methods.

Recall that the proposed AutoML methods are divided into two main components in this thesis, i.e., (i) a *search space* and (ii) a *search method*. We refer to them considering these two components. For the first part of the experimental analysis into the third phase, we use five MLC quality measures and 14 datasets to evaluate the performance of the proposed *search methods* against the methods we use to compare with (i.e., Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>).

In the second part, we contrast the methods in terms of the generated MLC algorithms. This can help us to gain insights into the general performance presented by the methods in our first analysis. Analyzing the generated MLC algorithms also helps us understand the bias and the possible issues of the search performed by an AutoML method. For instance, these results could show us if the selected and configured MLC

---

<sup>1</sup>Both baseline methods were developed for the purpose of comparison within the context of this thesis.

algorithms are indeed good multi-label classification algorithms and how similar they are to the state-of-the-art MLC algorithms.

Another way to check the quality of an AutoML method is to verify whether its search or optimization process is converging accordingly. We analyze this aspect in the third part. Looking at the convergence curves regarding a given quality measure can help us to implicitly verify whether the search or optimization is stuck in a local optima. This analysis can change the way the methods operate in future work to improve their search or optimization processes.

## 6.1 Experimental Setup

This section discusses the experimental setup. All experiments were run using a stratified five-fold cross-validation procedure [189]. In addition, five metrics were considered when evaluating the results in terms of classification quality: Exact Match (EM, see Equation 3.1), Hamming Loss (HL, see Equation 3.2),  $F_1$  Macro averaged by label (FM, see Equations 3.10 and 3.11), Ranking Loss (RL, see Equation 3.13), and the general metric we defined as the fitness/quality criteria in Chapter 5 (see Equation 5.1).

### 6.1.1 Datasets

The experiments reported in this chapter involve a set of 19 datasets selected from the KDIS (Knowledge and Discovery Systems) repository<sup>2</sup> and described in Table 6.1. These datasets were chosen based on their differences in application domain, the number of instances ( $n$ ), the number of attributes ( $m$ ), the number of labels ( $q$ ), the label cardinality – the average number of labels associated with each example in the dataset (Card.), the label density – the average number of labels associated with each example divided by the number of labels (i.e., the cardinality divided by the number of labels.) (Dens.), and the label diversity – the percentage of labelsets present in the dataset divided by the number of possible labelsets (Div.). [204] states that these different characteristics can influence the performance of the MLC methods in different applications.

From these 19 datasets, five (5) were selected to be used for tuning the parameters

---

<sup>2</sup>The datasets are also available at <http://www.uco.es/kdis/mlresources/>.

of the most promising method, and the other 14 were used in the remaining experiments<sup>3</sup>. From now on, we refer to these datasets using their acronyms, which are also specified in Table 6.1.

Table 6.1: Datasets used in the experiments.

Datasets	Acronym	Domain	n	m	q	Card.	Dens.	Div.
<b>Tuning datasets</b>								
3sources_reuters1000	3SR	Text	294	1000	6	1.126	0.188	0.219
Emotions	EMT	Music	593	72	6	1.868	0.311	0.422
GpositiveGO	GPG	Biology	519	912	4	1.008	0.252	0.438
HumanPseAAC	HPA	Biology	3106	440	14	1.185	0.085	0.027
Langlog	LGL	Text	1460	1004	75	1.180	0.016	0.208
<b>Experimental datasets</b>								
Bibtex	BTX	Text	7395	1836	159	2.402	0.015	0.386
Birds	BRD	Audio	645	260	19	1.014	0.053	0.206
CAL500	CAL	Music	502	68	174	26.044	0.150	1.000
CHD_49	CHD	Medicine	555	49	6	2.580	0.430	0.531
Enron	ENR	Text	1702	1001	53	3.378	0.064	0.442
Flags	FLG	Image	194	19	7	3.392	0.485	0.422
Genbase	GBS	Biology	662	1186	27	1.252	0.046	0.048
GpositivePseAAC	GPP	Biology	519	440	4	1.008	0.252	0.438
Medical	MED	Text	978	1449	45	1.245	0.028	0.096
PlantPseAAC	PPA	Biology	978	440	12	1.079	0.090	0.033
Scene	SCN	Image	2407	294	6	1.074	0.179	0.234
VirusPseAAC	VPA	Biology	207	440	6	1.217	0.203	0.266
Water-quality	WQT	Chemistry	1060	16	14	5.073	0.362	0.778
Yeast	YST	Biology	2417	103	14	4.237	0.303	0.082

## 6.1.2 Parameter Setting

This section describes the default values used in the proposed methods after preliminary experiments and considers the values used by other methods in the literature [50, 62, 109, 144]. For the evolutionary-based *search methods*, we follow Eiben and Smith [62] to set the evolutionary parameters (e.g., mutation and crossover rates).

Unless otherwise stated, the three evolutionary methods (i.e., GA-Auto-MLC, Auto-MEKA<sub>GPP</sub> and Auto-MEKA<sub>spGPP</sub>) were run with the following parameters: 80 individuals evolved considering a time budget of five hours, tournament selection of size two (2), elitism of one (1) individual, and crossover and mutation probabilities of 0.8 and 0.2, respectively. For these methods, the learning and validation sets are also resampled from the training set every five generations in order to avoid overfitting. Additionally, we use time and memory budgets for each MLC algorithm (generated by the proposed

<sup>3</sup>As aforementioned, these datasets structurally differ from each other. Therefore, to perform a fair selection, we have randomly chosen the datasets composing tuning and comparison experiments.

methods) of 180 seconds (three minutes) and 2GB of RAM, respectively. If the MLC algorithms reach these budgets, they are associated with the lowest fitness, i.e., a fitness of zero (0.0). Furthermore, the following convergence criterion is considered: at each iteration, we check if the best individual has remained the same for over five (5) generations and the *search method* has run for at least 20 generations. If this happens, we restart the evolutionary process with another pseudo-random seed.

In the case of Auto-MEKA<sub>spGGP</sub>, as we have eight (8) species, we specify ten (10) individuals for each species. We define Auto-MEKA<sub>spGGP</sub>'s convergence criteria for each species individually. Furthermore, we set the intra-species and inter-species crossover probabilities for Auto-MEKA<sub>spGGP</sub> as 0.3 and 0.7, respectively.

Finally, based on the aforementioned parameters, Auto-MEKA<sub>BO</sub> has kept only the time and memory budgets – i.e., five hours of run for its respective *search method*, and three minutes and 2GB of time and memory budgets for each produced MLC algorithms, respectively. As in the EA-based methods, the MLC algorithms that reach time and memory budgets are set to a quality score of 0.0. Furthermore, one intrinsic parameter is the employed acquisition function, which has taken the expected improvement (EI) function [110, 127, 200] for Auto-MEKA<sub>BO</sub> in this thesis.

### 6.1.3 Baseline Methods

The proposed methods are compared to two well-known methods<sup>4</sup>: (i) a random *search method* [14]; and (ii) a beam *search method* [152].

Basically, the Random Search (RS) iterates over the predefined *search space* at random. First, it creates  $p$  MLC algorithm configurations (by using a pseudo-random seed), evaluates them, and saves the best configuration in terms of the proposed quality measure (see Equation 5.1) into a list. Next, it creates another  $p$  new MLC algorithm configurations, evaluates these configurations, and saves the best at this iteration into the same list. RS keeps doing this procedure until the total time budget is reached. In the end, we return the best MLC algorithm configuration from the list based on the quality measure.

On the other hand, the Beam Search (BS) iterates locally over the given *search space*. Based on that, BS works as follows. Taking the *search space* into account, an initial random solution (i.e., an MLC algorithm configuration) is generated. We evaluate

<sup>4</sup>We still have not compared our proposed methods to ML<sup>2</sup>-Plan [211] due to technical issues in the experiments. For instance, ML<sup>2</sup>-Plan does not allow the change of the *search space* of MLC algorithms in a straightforward way. Thus, we will keep this comparison for future work.

this random solution and keep it as the current best (with a score based on the quality measure). From this solution, we generate  $p$  others by performing local changes into its representation<sup>5</sup>. We evaluate these solutions and check if one of them has a better quality score than the current best MLC configuration. If so, we update the best configuration with the best score. Otherwise, we maintain the best MLC configuration. Next, we continue looking at its neighbors from the current best configuration to create, evaluate, and possibly find better solutions. This search process remains until the final time budget is reached. In the end, the best-found MLC algorithm configuration is returned based on the proposed quality measure.

In order to be fair in the comparisons with the evolutionary methods – which deal with a population of 80 individuals – we set the value of  $p$  equal to 80 for both random and beam *search methods*.

#### 6.1.4 Statistical Comparisons

Given the average of the runs for each method, results are evaluated using the well-known statistical approach proposed by Demšar [54] to compare these methods, using an adapted Friedman test followed by Nemenyi’s *post hoc* test with a significance level of 0.05. This approach is used for comparing several methods in several datasets and determines the non-randomness of the obtained results. More specifically, the Friedman test [84] is considered a non-parametric counterpart of ANOVA [77] because it does not rely on a probabilistic distribution to perform its corresponding statistic.

Going into details on Demšar’s approach, let  $R_j^i$  be the rank of the  $j$ -th of the  $k$  methods on the  $i$ -th of the  $n$  datasets. With this information, the Friedman test compares the  $k$  methods by assigning a rank for each algorithm  $j$  in each dataset  $i$ . Next, the test uses the average of the ranks  $R_j$  over all datasets, defined in Equation 6.1.

$$R_j = \frac{1}{n} \cdot \sum_{i=1}^n R_j^i \quad (6.1)$$

Under the null hypothesis, which states that the methods are equivalent in these conditions (i.e., on those datasets), the Friedman statistic is presented in Equation 6.2, being distributed according to  $\chi_F^2$  with  $k - 1$  degrees of freedom, when  $n$  and  $k$  are big enough.

---

<sup>5</sup>In this work, we use the aforementioned grammar-based representation for both random and beam searches. Thus, we generate a derivation tree from the grammar and employ Whigham’s mutation to perform local operations in this respective tree.

$$\chi_F^2 = \frac{12 \cdot n}{k \cdot (k + 1)} \cdot \left[ \sum_{j=1}^k R_j^2 - \frac{k \cdot (k + 1)^2}{4} \right], \quad (6.2)$$

Iman and Davenport [111] showed that  $\chi_F^2$  is undesirably conservative and derived an adjustment for a better statistic, which is formalized in Equation 6.3. This adapted statistic is distributed according to Snedecor’s F-distribution with  $k - 1$  and  $(k - 1)(n - 1)$  degrees of freedom.

$$F_F = \frac{(n - 1) \cdot \chi_F^2}{n \cdot (k - 1) - \chi_F^2}, \quad (6.3)$$

If the null hypothesis is rejected, we can proceed with Nemenyi’s *post hoc* test for pairwise comparisons. The results of the two methods are significantly different if their respective average ranks differ by at least the Critical Difference (CD). The CD is defined in Equation 6.4, where the critical values  $q_\alpha$  are based on the Student’s statistic divided by  $\sqrt{2}$ .

$$CD = q_\alpha \cdot \sqrt{\frac{k \cdot (k + 1)}{6 \cdot n}} \quad (6.4)$$

## 6.2 Preliminary Comparison of the Proposed Methods

This section presents an initial and general comparison of the proposed methods – i.e., GA-Auto-MLC, Auto-MEKA<sub>GGP</sub>, Auto-MEKA<sub>spGGP</sub> and Auto-MEKA<sub>BO</sub> – in the five tuning datasets described in Table 6.1. Given the high computational cost involved in running the MLC algorithms, tuning procedures will be performed only for the most promising method and again considering the *search space* Large.

Tables 6.2 and 6.3 show their results according to the metric defined in Equation 5.1 – the proposed fitness/quality measure – for time budgets of one and five hours, respectively. For these two tables (and the following ones), we present the average results for each dataset across the cross-validation runs together with their respective standard deviations, which are shown between parenthesis. In addition, results in bold in these tables indicate the best average values or the ones we take as the most appropriate, and shaded cells illustrate cases where overfitting was identified when comparing the results from one hour to five hours of running.

Overall, we observe in Tables 6.2 and 6.3 that the best average value of the proposed quality measure (based on the five employed datasets) was achieved by Auto-MEKA<sub>GGP</sub>



and Auto-MEKA<sub>spGGP</sub> for both one and five hours of time budget. Nevertheless, Auto-MEKA<sub>spGGP</sub> was the only *search method* that improved its average results from one hour to five hours and also did not suffer from overfitting.

In the case of AutoML, we define overfitting (or meta-overfitting) when the results at the beginning of the training process are better than those results achieved at the end of this process. For instance, this happened on Auto-MEKA<sub>GGP</sub>, where its results on the datasets 3SR and GPG within one hour of training are higher than the respective results within five hours of training. The same occurred for Auto-MEKA<sub>BO</sub>, which had this issue for both datasets. Therefore, we can conclude that Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BO</sub> suffered from overfitting on these two datasets.

Table 6.2: Results in terms of quality measure (defined in Equation 5.1) on the test set within one hour of training for the proposed AutoML *search methods*.

Datasets	Auto-MEKA <sub>GGP</sub>	Auto-MEKA <sub>spGGP</sub>	GA-Auto-MLC	Auto-MEKA <sub>BO</sub>
3SR	0.394 (0.042)	0.392 (0.041)	0.327 (0.003)	0.389 (0.033)
EMT	0.660 (0.017)	0.669 (0.015)	0.645 (0.005)	0.665 (0.017)
GPG	0.930 (0.023)	0.926 (0.048)	0.916 (0.029)	0.930 (0.024)
HPA	0.547 (0.016)	0.549 (0.014)	0.371 (0.003)	0.531 (0.017)
LGL	0.555 (0.017)	0.552 (0.019)	0.427 (0.005)	0.528 (0.024)
<b>Avg. Val.</b>	0.617	<b>0.618</b>	0.537	0.609

Table 6.3: Results in terms of the quality measure (defined in Equation 5.1) on the test set within five hours of training for the proposed AutoML *search methods*.

Datasets	Auto-MEKA <sub>GGP</sub>	Auto-MEKA <sub>spGGP</sub>	GA-Auto-MLC	Auto-MEKA <sub>BO</sub>
3SR	0.387 (0.050)	0.395 (0.048)	0.327 (0.003)	0.376 (0.045)
EMT	0.660 (0.013)	0.671 (0.015)	0.645 (0.005)	0.666 (0.015)
GPG	0.926 (0.042)	0.926 (0.022)	0.916 (0.029)	0.921 (0.029)
HPA	0.551 (0.013)	0.550 (0.013)	0.371 (0.003)	0.539 (0.015)
LGL	0.559 (0.013)	0.558 (0.018)	0.427 (0.005)	0.535 (0.023)
<b>Avg. Val.</b>	0.617	<b>0.620</b>	0.537	0.607

The case of GA-Auto-MLC’s results being the same from the first time budget to the other has a reason. The issue is related to its selected MLC algorithms, which were too complex to run at a feasible time during testing (we set a time budget of 15 minutes to run each MLC algorithm). As we save the best MLC algorithms found in each iteration of GA-Auto-MLC (elitism), we keep trying to evaluate the second-best, third-best, and so on. If no selected MLC algorithm reaches the expected final result, we then run a default MLC algorithm, which is a classifier chain algorithm augmented with a random forest (with hyper-parameters also set to their default values).

Within this scenario on the general comparison among the proposed AutoML *search methods*, we will keep trying to tune the parameters only for the Auto-MEKA<sub>spGGP</sub> *search method*. By looking at and analyzing the overall results of Tables 6.2 and 6.3, we

believe this is the best decision to take as we consider this *search method* the most promising one.

### 6.3 Auto-MEKA<sub>spGGP</sub>'s Hyper-Parameter Tuning

This section analyzes the effects of the three hyper-parameters we consider most relevant to the proposed method in Auto-MEKA<sub>spGGP</sub>. These hyper-parameters are considered more important than others because they affect both the search methods and the execution time. They are: (i) whether to use a resampling approach from time to time to avoid overfitting; (ii) the use of multi-fidelity approaches; (iii) the hyper-parameter that controls the inter/intra-species crossover on Auto-MEKA<sub>spGGP</sub>.

In the resampling, the computational cost increases as we need to re-train and re-evaluate individuals as the learning and validation sets are resampled. Eliminating this approach would increase the number of evaluated MLC algorithms and hyper-parameters – and, as a consequence, enhance the coverage of the *search space*.

In the case of the multi-fidelity approaches, we would like to evaluate how we can change the attributes and examples in the learning and validation datasets during the iterations of the AutoML method without compromising the predictive results. In a nutshell, these approaches evaluate and validate the MLC algorithms on data with few instances and few attributes (when compared to the original data) at the first iteration of the AutoML methods and keep increasing the complexity of the input data (growing the number of instances and attributes of the learning and validation sets) until we reach the original data size.

The third parameter we investigate in depth is the inter/intra-species crossover. Given the probability of crossover on Auto-MEKA<sub>spGGP</sub>, we split this probability for happening crossover intra-species (i.e., this crossover is able to change only the types of hyper-parameters that a species encompasses by mixing the individuals on that species) and inter-crossover (i.e., this crossover turns possible to mix individuals from different species, allowing different types of hyper-parameters to be explored/exploited during the communications between two species). We believe that understanding this parameter would give Auto-MEKA<sub>spGGP</sub> a better chance to settle a suitable trade-off between exploration and exploitation to this specific *search method*.

Note that we focus on the *search space* Large in the experiments reported in this section because of the computational cost involved in training the AutoML methods. This *search space* was chosen because it encompasses the others and is the most complex. Therefore, we believe this setting is a good way to test the method's capabilities.

### 6.3.1 Resampling the Training Set

This subsection investigates whether it is possible to exclude the resampling procedure from Auto-MEKA<sub>spGGP</sub>'s evolutionary process. Considering this experimental scenario, we ran the experiments without resampling in the evolutionary process and compared to Auto-MEKA<sub>spGGP</sub>'s results of Section 6.2. Tables 6.4 and 6.5 present these results within one and five hours of training for the five previously defined datasets.

Table 6.4: Auto-MEKA<sub>spGGP</sub>'s results based on the quality measure (defined in Equation 5.1) on the test set with the presence and absence of resampling within one hour.

Datasets	With Resampling	Without Resampling
3SR	0.392 (0.041)	0.389 (0.044)
EMT	0.669 (0.015)	0.661 (0.016)
GPG	0.926 (0.048)	0.925 (0.020)
HPA	0.549 (0.014)	0.537 (0.020)
LGL	0.552 (0.019)	0.546 (0.020)
<b>Avg. Val.</b>	<b>0.618</b>	0.612

Table 6.5: Auto-MEKA<sub>spGGP</sub>'s results based on the quality measure (defined in Equation 5.1) on the test set with the presence and absence of resampling within five hours.

Datasets	With Resampling	Without Resampling
3SR	0.395 (0.048)	0.395 (0.049)
EMT	0.671 (0.015)	0.660 (0.019)
GPG	0.926 (0.022)	0.917 (0.046)
HPA	0.550 (0.013)	0.544 (0.015)
LGL	0.558 (0.018)	0.552 (0.018)
<b>Avg. Val.</b>	<b>0.620</b>	0.614

We can observe from these tables that the average results for the quality measure are slightly higher/better in the presence of resampling for both time budgets. Besides, by using resampling on data, Auto-MEKA<sub>spGGP</sub> continued, on average, to improve its results for all evaluated datasets.

This is not always true when we remove the resampling approach every five generations from Auto-MEKA<sub>spGGP</sub>'s evolutionary process. The average results for the datasets EMT and GPG were slightly lower/worse at the end of the evolutionary process (within five hours of training) when we compare them to the beginning of this process (i.e., within one hour of training). Although this issue is not very significant, as the differences between five hours and one hour are into the standard deviation, we would like to avoid these situations as much as possible.

Accordingly, as we expect to train the AutoML methods in different datasets with distinct characteristics, we decide to keep the resampling approach into the evolutionary

process of Auto-MEKA<sub>spGGP</sub> for every five generations, mostly because we are afraid to face overfitting in sensible datasets in a more serious form.

### 6.3.2 The Multi-Fidelity Approach

In our experiments, we set  $\#steps$  equal to five for the number of attributes, and we change data at each hour of Auto-MEKA<sub>spGGP</sub>'s evolutionary process. Nevertheless, this could be done for all proposed AutoML *search methods*.

For the number of examples, we set  $b$  to two (2) and the initial value of  $\#steps$  to four (4). This procedure was performed only in Auto-MEKA<sub>spGGP</sub>. However, it can also be generalized to other *search methods*.

Given these procedures to perform multi-fidelity in terms of the number of attributes and instances (see Section 5.4), we can use the respective procedures in isolation or in conjunction. The tuning procedure for the multi-fidelity approaches refers to this analysis, i.e., which multi-fidelity procedure to choose for the attribute selection and whether the attribute and instance procedures should be used together. Table 6.6 defines these possible multi-fidelity scenarios and associates them with an acronym.

Table 6.6: Possible scenarios for the multi-fidelity approaches.

Acronym	Multi-Fidelity Procedures
NAS-NIS	No Attribute Selection, No Instance Selection
EAS-PIS	Exponential Attribute Selection, Polynomial Instance Selection
PAS-PIS	Polynomial Attribute Selection, Polynomial Instance Selection
NAS-PIS	No Attribute Selection, Polynomial Instance Selection
EAS-NIS	Exponential Attribute Selection, No Instance Selection
PAS-NIS	Polynomial Attribute Selection, No Instance Selection

So far, we have not used any multi-fidelity approach. Thus, the experiments until Section 6.3.2 use no multi-fidelity procedure, i.e., NAS-NIS. In this subsection, we vary the possible multi-fidelity approaches and evaluate their results in terms of the proposed quality/fitness measure and the achieved number of generations. The results considering the variation of the multi-fidelity approach can be found in Tables 6.7, 6.8, 6.9 and 6.10 for one and five hours. The results of Tables 6.7 and 6.8 are in accordance with average fitness values on the test set achieved by the selected and configured MLC algorithms. The results of Tables 6.9 and 6.10, in turn, correspond to the average number of generations reached by Auto-MEKA<sub>spGGP</sub> within one and five hours of training, respectively.

By analyzing the multi-fidelity results in terms of the proposed quality measure, we

Table 6.7: Auto-MEKA<sub>spGGP</sub>'s multi-fidelity tuning results based on the average quality measure (defined in Equation 5.1) on the test set within one hour.

Datasets	NAS-NIS	EAS-PIS	PAS-PIS	NAS-PIS	EAS-NIS	PAS-NIS
3SR	0.392 (0.041)	0.424 (0.050)	0.424 (0.048)	0.408 (0.042)	0.378 (0.047)	0.415 (0.047)
EMT	0.669 (0.015)	0.657 (0.018)	0.668 (0.015)	0.674 (0.013)	0.649 (0.032)	0.668 (0.017)
GPG	0.926 (0.048)	0.924 (0.032)	0.923 (0.028)	0.909 (0.101)	0.912 (0.042)	0.927 (0.023)
HPA	0.549 (0.014)	0.526 (0.030)	0.542 (0.024)	0.537 (0.020)	0.534 (0.023)	0.545 (0.015)
LGL	0.552 (0.019)	0.509 (0.024)	0.525 (0.023)	0.546 (0.019)	0.506 (0.032)	0.529 (0.018)
<b>Avg. Val.</b>	0.618	0.608	<b>0.616</b>	0.615	0.596	0.617

Table 6.8: Auto-MEKA<sub>spGGP</sub>'s multi-fidelity tuning results based on the average quality measure on the test set within five hours.

Datasets	NAS-NIS	EAS-PIS	PAS-PIS	NAS-PIS	EAS-NIS	PAS-NIS
3SR	0.395 (0.048)	0.412 (0.042)	0.436 (0.040)	0.405 (0.038)	0.404 (0.045)	0.396 (0.044)
EMT	0.671 (0.015)	0.668 (0.013)	0.671 (0.013)	0.674 (0.011)	0.667 (0.018)	0.671 (0.015)
GPG	0.926 (0.022)	0.929 (0.023)	0.923 (0.022)	0.912 (0.067)	0.930 (0.020)	0.931 (0.020)
HPA	0.550 (0.013)	0.541 (0.025)	0.540 (0.036)	0.546 (0.016)	0.543 (0.022)	0.545 (0.019)
LGL	0.558 (0.018)	0.543 (0.018)	0.548 (0.021)	0.553 (0.020)	0.551 (0.018)	0.551 (0.018)
<b>Avg. Val.</b>	0.620	0.619	<b>0.624</b>	0.618	0.619	0.618

Table 6.9: Auto-MEKA<sub>spGGP</sub>'s multi-fidelity average number of generations within one hour.

Datasets	NAS-NIS	EAS-PIS	PAS-PIS	NAS-PIS	EAS-NIS	PAS-NIS
3SR	9.5 (2.6)	28.2 (13.7)	20.9 (5.4)	13.4 (2.2)	19.1 (3.8)	15.2 (3.4)
EMT	11.6 (2.3)	21.6 (8.0)	17.3 (5.5)	15.7 (5.4)	17.5 (4.2)	13.8 (3.0)
GPG	12.1 (2.1)	27.8 (10.8)	21.4 (5.5)	15.2 (3.3)	21.1 (6.0)	15.8 (3.2)
HPA	4.0 (0.7)	15.2 (4.6)	9.0 (2.7)	7.1 (1.3)	8.5 (1.3)	6.8 (1.5)
LGL	3.4 (0.9)	11.5 (2.1)	9.1 (1.4)	4.9 (0.7)	8.7 (1.9)	6.2 (1.3)
<b>Avg. Val.</b>	8.1	20.9	<b>15.5</b>	11.3	15.0	11.6

Table 6.10: Auto-MEKA<sub>spGGP</sub>'s multi-fidelity average number of generations within five hours.

Datasets	NAS-NIS	EAS-PIS	PAS-PIS	NAS-PIS	EAS-NIS	PAS-NIS
3SR	68.3 (13.8)	104.4 (25.1)	96.6 (24.8)	76.7 (12.5)	85.2 (19.3)	80.2 (20.0)
EMT	73.8 (13.9)	72.9 (20.1)	69.1 (22.2)	72.0 (18.2)	75.2 (18.1)	72.0 (14.3)
GPG	80.3 (14.4)	93.6 (21.8)	92.2 (17.1)	79.5 (18.5)	92.1 (22.7)	87.9 (16.0)
HPA	42.7 (5.4)	53.2 (15.0)	41.2 (9.4)	39.9 (5.7)	39.6 (5.5)	43.3 (10.1)
LGL	47.8 (5.3)	51.6 (6.8)	47.7 (10.1)	49.1 (8.0)	46.8 (11.0)	42.4 (6.8)
<b>Avg. Val.</b>	62.6	75.2	<b>69.3</b>	63.5	67.8	65.1

can conclude that the multi-fidelity scenarios EAS-PIS and EAS-NIS improved the average quality measure for almost all datasets (except for EAS-PIS on the dataset 3SR) when comparing its results for one hour to the results for five hours. However, in these scenarios, we believe that the reduction of the original dataset was too aggressive. Observing the results of the dataset 3SR, for example, which has 1000 attributes and only 294 instances. Using EAS-PIS or EAS-NIS for this particular type of dataset would significantly change the data characteristics (e.g., average number of attributes per the number of instances). Although both multi-fidelity scenarios better explore the *search space* (see the average number of generations of Tables 6.9 and 6.10), this is not appropriate.

On the other hand, the multi-fidelity results on the scenarios NAS-PIS and PAS-NIS are quite conservative. In these scenarios, Auto-MEKA<sub>spGGP</sub> improved or stabilized the predictive performance (see Tables 6.7 and 6.8), but the search itself was not improved as the coverage of the *search space* was broadly the same (see the average number of generations for both scenarios). As the number of generations was almost the same for both scenarios compared to NAS-NIS (i.e., the default scenario), we believe employing such multi-fidelity approaches would be a waste of time.

In our perspective, the best multi-fidelity scenario is PAS-PIS, which applies linear functions on the number of attributes and instances to reduce computational time while keeping or improving the final predictive performance. For PAS-PIS, the results of Auto-MEKA<sub>spGGP</sub> kept improving or stayed (about) the same from one hour to five hours. Besides, the number of generations (i.e., iterations) increased by (about) seven in both one and five hours of training.

Hence, given this analysis, we conclude that the most appropriate multi-fidelity scenario is PAS-PIS. As a consequence of its results, we will continue using the multi-fidelity approach based on this scenario for the next Auto-MEKA<sub>spGGP</sub>'s experiments.

### 6.3.3 Tuning the Inter/Intra-Species Crossover Probability

Despite the fact that we have already changed parameters regarding how we sample and resample data, we would also like to check the parameters that influence the behavior of the *search methods per se*. As we have maintained the tuning only Auto-MEKA<sub>spGGP</sub> after Section 6.2 as we believe this is the most promising method, we focus here on one of its essential parameters, the probability of applying the crossover operator inter-species and intra-species.

Before going into tuning the inter/intra-species crossover probability itself, recall that we have a general probability of 0.8 of performing crossover. In addition, we define

the intra-species and inter-species crossover probabilities for Auto-MEKA<sub>spGGP</sub> as 0.3 and 0.7 for the previous experiments, respectively. Furthermore, we have ten (10) individuals per species, resulting in a total of 80 evaluated individuals per generation.

In order to tune these crossover probabilities, we vary their values in a complementary form. We start with even intra-species and inter-species probabilities (i.e., 0.5 and 0.5, respectively). Next, we increase the value of the intra-species probability by 0.1 while we decrease the value of the inter-species probability by one. We do that until we have 1.0 for intra-species crossover probability and 0.0 for inter-species probability. Tables 6.11 and 6.12 present Auto-MEKA<sub>spGGP</sub>'s results regarding the tuning of these crossover probabilities. The results are in terms of the defined quality/fitness measure for one and five hours, respectively.

Table 6.11: Auto-MEKA<sub>spGGP</sub>'s results based on the defined quality measure on the test set for varying the intra/inter-species crossover probabilities within one hour of training.

Datasets	0.5/0.5	0.6/0.4	0.7/0.3	0.8/0.2	0.9/0.1	1.0/0.0
3SR	0.423 (0.086)	0.381 (0.049)	0.446 (0.024)	0.416 (0.062)	0.405 (0.061)	0.407 (0.062)
EMT	0.667 (0.020)	0.674 (0.020)	0.663 (0.015)	0.656 (0.025)	0.656 (0.022)	0.676 (0.015)
GPG	0.924 (0.031)	0.917 (0.036)	0.921 (0.017)	0.922 (0.013)	0.914 (0.025)	0.932 (0.014)
HPA	0.545 (0.022)	0.530 (0.022)	0.533 (0.017)	0.537 (0.020)	0.538 (0.023)	0.516 (0.013)
LGL	0.532 (0.019)	0.531 (0.024)	0.525 (0.011)	0.526 (0.010)	0.532 (0.016)	0.528 (0.012)
<b>Avg. Val.</b>	<b>0.618</b>	0.608	0.618	0.611	0.609	0.612

Table 6.12: Auto-MEKA<sub>spGGP</sub>'s results based on the defined quality measure on the test set for varying the intra/inter-species crossover probabilities within five hours of training.

Datasets	0.5/0.5	0.6/0.4	0.7/0.3	0.8/0.2	0.9/0.1	1.0/0.0
3SR	0.434 (0.063)	0.426 (0.039)	0.370 (0.034)	0.366 (0.038)	0.421 (0.050)	0.374 (0.052)
EMT	0.667 (0.013)	0.667 (0.011)	0.658 (0.007)	0.641 (0.028)	0.661 (0.022)	0.670 (0.018)
GPG	0.917 (0.030)	0.923 (0.027)	0.930 (0.035)	0.947 (0.013)	0.922 (0.030)	0.917 (0.032)
HPA	0.543 (0.022)	0.545 (0.032)	0.550 (0.017)	0.536 (0.022)	0.520 (0.022)	0.536 (0.015)
LGL	0.557 (0.012)	0.539 (0.016)	0.547 (0.016)	0.559 (0.009)	0.556 (0.017)	0.551 (0.022)
<b>Avg. Val.</b>	<b>0.624</b>	0.620	0.611	0.610	0.616	0.610

Results were better when we considered 0.5 and 0.5 of intra-species and inter-species crossover probabilities, respectively. The even values for the probabilities made Auto-MEKA<sub>spGGP</sub> achieve the best average predictive performance so far (0.618 within one hour and 0.624 within five hours of training). Although these results are quite close to the best ones found in Section 6.3.2 and possibly the same when we look at the standard deviation, one small improvement in AutoML is usually difficult in such enormous *search spaces*<sup>6</sup>. For this reason, in the final experiments of Auto-MEKA<sub>spGGP</sub> we use resampling, employ the PAS-PIS multi-fidelity approach, and operate over the *search space* by working with 0.5/0.5 of intra/inter-species crossover probabilities.

<sup>6</sup>We are aware that the datasets GPG and HPA are sensitive to overfitting and we would like to avoid it. Nevertheless, considering the standard deviations, the results are actually the same for both datasets. This is also one reason that motivates us to use such intra-species and inter-species crossover rates in our final experiments.

It is worth noting that even with using the multi-fidelity approach, our predictive results were similar to those obtained with the original dataset, without affecting the multi-label classifier’s predictive performance.

## 6.4 Experimental Results

This section further analyzes the results obtained by three proposed *search methods* – Auto-MEKA<sub>BO</sub>, Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>spGGP</sub> – in the 14 datasets, selected for a more detailed experimental analysis. We let GA-Auto-MLC out of the final experiments because of its poor performance in recommending MLC algorithms with a specific hyper-parameter setting. These results are compared with those obtained by the baseline methods: Random Search (RS) and Beam Search (BS), here referred to as Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>, respectively.

Tables 6.13, 6.14, 6.15, 6.16 and 6.17 show the average values, the average ranks, and the final statistical analysis – based on the 14 datasets – of Exact Match (EM), Hamming Loss (HL),  $F_1$  Macro-averaged by label (FM), Ranking Loss (RL) and fitness for the three proposed methods (Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BO</sub>) and the two baseline methods (Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>) in the test set, respectively. We perform this comparison for the three designed *search spaces* within one and five hours of running. We take the subscript names for these tables to define the *search methods*.

Regarding the results of the EM measure, which is the most conservative, in Table 6.13, we observe that the results do not present statistical differences in all three *search spaces* in one hour of running. For the *search space* Small, the best average value and the best average rank are achieved by Auto-MEKA<sub>RS</sub>. On the other hand, the best average value in terms of the 14 datasets was found by Auto-MEKA<sub>BS</sub> and the best average rank by Auto-MEKA<sub>spGGP</sub> for the *search space* Medium. Both averages (value and rank) for the *search space* Large were found by Auto-MEKA<sub>BS</sub>.

When we change the time budget to evaluate the results of the EM measure, our analysis has also become different. For the *search space* Small, the best average and the rank value are found by Auto-MEKA<sub>GGP</sub>, which is statistically superior to Auto-MEKA<sub>RS</sub> on this particular *search space*. This statistical difference is indicated in Table 6.13 and in the following tables by the symbol  $\succ$ . For the *search space* Medium, we have not found evidence of statistical differences, but whereas Auto-MEKA<sub>BS</sub> presented the best average value, Auto-MEKA<sub>GGP</sub> showed again the best average rank. In respect of the *search space* Large, we identify that the best *search methods* were Auto-MEKA<sub>BO</sub> and Auto-



Table 6.13: Comparison of the exact match (to be maximized) obtained by the proposed *search methods* and the baseline methods in the test set for the three designed *search spaces* within one and five hours of execution.

Time Budget	Search Space	Evaluated Result	spGGP	GGP	BO	RS	BS
1 hour	Small	Avg. Val.	0.344	0.346	0.343	<b>0.352</b>	0.348
		Avg. Rank.	3.500	2.679	3.357	<b>2.536</b>	2.928
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	0.389	0.371	0.346	0.376	<b>0.394</b>
		Avg. Rank.	<b>2.750</b>	2.821	3.571	2.929	2.929
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.323	0.351	0.327	0.354	<b>0.358</b>
		Avg. Rank.	2.857	2.821	3.714	3.179	<b>2.429</b>
		Stat. Comp.	no differences among all methods				
5 hours	Small	Avg. Val.	0.343	<b>0.349</b>	0.334	0.339	0.343
		Avg. Rank.	2.929	<b>1.964</b>	3.500	3.679	2.929
		Stat. Comp.	{GGP} > {RS}, no differences among the others				
	Medium	Avg. Val.	0.390	0.379	0.337	0.361	<b>0.397</b>
		Avg. Rank.	3.571	<b>2.429</b>	3.464	2.821	2.714
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.349	0.335	<b>0.355</b>	0.348	0.347
		Avg. Rank.	2.607	<b>2.429</b>	2.571	4.286	3.107
		Stat. Comp.	{spGGP, GGP, BO} > {RS}, no differences among the others				

MEKA<sub>GGP</sub> based on the average value and rank, respectively. In addition, on this *search space*, we have statistical evidence that Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BO</sub> are better methods than a pure random search (i.e., than Auto-MEKA<sub>RS</sub>).

These results for the EM measure indicate indeed what we expected, i.e., the size of the *search space* affects the performance of the *search methods*. This is not so clear when we give a small amount of time for the *search methods* (e.g., one hour) to cover the *search spaces*, but it is more evident when we give an appropriate time budget (e.g., five hours) for them. Note in Table 6.13 that from five hours of time budget, all proposed *search methods* could deal better with the trade-off between exploration and exploitation, whilst the *search space* increases exponentially, beating the random *search method*. In other words, by giving enough time budget to the *search methods*, those with a more robust AutoML approach can better handle the trade-off between exploration and exploitation. As a consequence, their results become better in the comparison.

It is worth noting that we identified some cases of overfitting on the proposed and comparison *search methods*. In order to perform this analysis, we only looked at the average values of the EM measure, which partially encompasses the results of 14 datasets (see Table 6.1). In special, we observe overfitting on: Auto-MEKA<sub>GGP</sub> in the *search space* Large; Auto-MEKA<sub>BO</sub> in the *search space* Small and Medium; Auto-MEKA<sub>RS</sub> in all *search spaces*; and Auto-MEKA<sub>BS</sub> in the *search spaces* Large. We believe that Auto-MEKA<sub>spGGP</sub> was not so affected by overfitting due to our tuning procedure on its more important parameters.

For the results of the HL measure in Table 6.14, Auto-MEKA<sub>GGP</sub> showed the best average rank and also the best average value in all *search spaces* within one hour of run-

ning. Although there was no statistical significance among the *search methods* for the *search spaces* Small and Medium, we detected statistical difference in the *search space* Large, where Auto-MEKA<sub>GPP</sub> showed to have a better score than Auto-MEKA<sub>spGPP</sub>. This result was slightly expected in some cases as we employed a specific multi-fidelity approach on Auto-MEKA<sub>spGPP</sub>. Even with tuning, Auto-MEKA<sub>spGPP</sub> might lose predictive performance in some scenarios, such as the one described here.

Table 6.14: Comparison of the hamming loss (to be minimized) obtained by the proposed methods and the baseline methods in the test set for the three designed *search spaces* within one and five hours of execution.

Time Budget	Search Space	Evaluated Result	spGPP	GPP	BO	RS	BS
1 hour	Small	Avg. Val.	0.139	<b>0.134</b>	0.140	0.135	<b>0.134</b>
		Avg. Rank.	3.500	<b>2.214</b>	3.821	2.607	2.857
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	0.162	<b>0.135</b>	0.139	0.139	0.140
		Avg. Rank.	3.464	<b>2.500</b>	3.107	2.890	3.036
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.148	<b>0.134</b>	0.208	0.139	0.137
		Avg. Rank.	4.036	<b>2.286</b>	3.107	3.000	2.571
		Stat. Comp.	{GPP} > {spGPP}, no differences among the others				
5 hours	Small	Avg. Val.	0.135	<b>0.134</b>	0.208	0.137	0.135
		Avg. Rank.	3.143	<b>2.250</b>	3.714	3.107	2.786
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	0.157	0.136	<b>0.134</b>	0.139	0.142
		Avg. Rank.	4.036	<b>2.251</b>	2.607	3.00	3.107
		Stat. Comp.	{GPP} > {spGPP}, no differences among the others				
	Large	Avg. Val.	0.137	0.134	<b>0.130</b>	0.143	0.139
		Avg. Rank.	3.214	2.571	<b>2.07</b>	4.00	3.142
		Stat. Comp.	{BO} > {RS}, no differences among the others				

The results for the HL measure within five hours, in Table 6.14, differs from those for the time budget of one hour. In that case, Auto-MEKA<sub>GPP</sub> continued having the best average values and ranks in the *search space* Small, but it had only the best average rank in the *search space* Medium. Furthermore, Auto-MEKA<sub>GPP</sub> indicated to be statistically better than Auto-MEKA<sub>spGPP</sub> in the *search space* Medium. Nonetheless, there is no indication of statistical difference for the other cases of *search space* Medium, neither for all cases of the *search space* Small.

Auto-MEKA<sub>BO</sub>, in turn, had the best average result in the *search space* Medium and the best average result and rank in the *search space* Large for five hours of time budget. In fact, Auto-MEKA<sub>BO</sub> was the only proposed method to present statistically better results when compared to Auto-MEKA<sub>RS</sub>. This was quite a surprise based on the tuning experiments, where Auto-MEKA<sub>BO</sub> resulted in one of the methods with the lowest predictive performance in its selection and configuration of MLC algorithms. On behalf of overfitting, Auto-MEKA<sub>BO</sub> was the only case to suffer from it in the *search space* Small. We suppose this was why this method was not the best one in all *search spaces*.

Furthermore, we evaluated the results of the *search methods* in the *search spaces* on a measure that is commonly used in the single-label scenario and adapted for the multi-

label context, i.e., the FM measure. Table 6.15 shows the results for FM, indicating that the best average value for the *search space* Small – within one hour – was produced by Auto-MEKA<sub>RS</sub>, whereas the best average rank was processed by Auto-MEKA<sub>GGP</sub> within five hours of execution, the best average rank and value was achieved by Auto-MEKA<sub>GGP</sub> for the same *search space*, instead.

Table 6.15: Comparison of the  $F_1$  macro-averaged by label (to be maximized) obtained by the proposed methods and the baseline methods in the test set for the three designed *search spaces* within one and five hours of execution.

Time Budget	Search Space	Evaluated Result	spGGP	GGP	BO	RS	BS
1 hour	Small	Avg. Val.	0.444	0.457	0.444	<b>0.460</b>	0.457
		Avg. Rank.	3.571	<b>2.571</b>	3.571	2.714	2.571
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	0.441	<b>0.471</b>	0.454	0.448	0.467
		Avg. Rank.	3.571	<b>2.464</b>	3.393	2.750	2.821
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.444	<b>0.467</b>	0.437	0.450	0.452
		Avg. Rank.	3.643	<b>2.607</b>	2.750	3.286	2.714
		Stat. Comp.	no differences among all methods				
5 hours	Small	Avg. Val.	0.451	<b>0.468</b>	0.442	0.461	0.462
		Avg. Rank.	2.929	<b>2.250</b>	3.571	3.250	3.000
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	0.449	0.464	0.453	0.452	<b>0.472</b>
		Avg. Rank.	3.179	3.143	2.571	3.571	<b>2.536</b>
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.461	<b>0.476</b>	0.463	0.456	0.459
		Avg. Rank.	2.964	<b>2.464</b>	2.571	3.643	3.357
		Stat. Comp.	no differences among all methods				

In the case of the *search space* Medium Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BS</sub> presented the best average values and ranks in terms of the FM measure for one and five hours of searching, respectively. In this *search space*, we found signs of overfitting for Auto-MEKA<sub>GGP</sub>. This caused this particular method not to present a good final predictive performance. The *search space* Large, on the other hand, had the best average found by Auto-MEKA<sub>GGP</sub>.

Besides the results regarding the average values and the average ranks, we also statistically evaluated the *search methods*. Nevertheless, we have not found any evidence of statistical differences (based on the 14 datasets) among the methods for both time budgets and three *search spaces*. One possible explanation for this statistical behavior is that this measure was not specifically built for multi-label classification – as it is adapted from the single-label classification perspective. Hence, finding flatter results for this measure is comprehensible.

Considering another evaluation context, the ranking produced by multi-label methods for each example was also used in the comparisons. The RL measure summarizes this distinct evaluation criteria, and the results regarding this measure are presented in Table 6.16.

Table 6.16: Comparison of the ranking loss (to be minimized) obtained by the proposed methods and the baseline methods in the test set for the three designed *search spaces* within one and five hours of execution.

Time Budget	Search Space	Evaluated Result	spGGP	GGP	BO	RS	BS
1 hour	Small	Avg. Val.	<b>0.157</b>	0.164	0.178	0.166	0.167
		Avg. Rank.	3.000	<b>2.393</b>	3.500	3.107	3.000
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	<b>0.140</b>	0.155	0.179	0.162	0.150
		Avg. Rank.	<b>2.071</b>	3.107	3.786	3.179	2.857
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.158	0.144	<b>0.140</b>	0.155	0.148
		Avg. Rank.	3.321	<b>2.392</b>	2.536	3.929	2.821
		Stat. Comp.	no differences among all methods				
5 hours	Small	Avg. Val.	<b>0.153</b>	0.161	0.210	0.167	0.167
		Avg. Rank.	2.321	<b>2.214</b>	3.750	3.429	3.286
		Stat. Comp.	no differences among all methods				
	Medium	Avg. Val.	<b>0.146</b>	0.150	0.152	0.156	0.148
		Avg. Rank.	<b>2.679</b>	2.821	3.250	3.321	2.929
		Stat. Comp.	no differences among all methods				
	Large	Avg. Val.	0.147	<b>0.135</b>	0.149	0.157	0.140
		Avg. Rank.	2.821	<b>2.536</b>	2.786	4.071	2.786
		Stat. Comp.	no differences among all methods				

Whilst  $\text{Auto-MEKA}_{spGGP}$  was the proposed method with the best average value within one hour in the *search space* Small,  $\text{Auto-MEKA}_{GGP}$  achieved the best average rank in this scenario. Distinctly,  $\text{Auto-MEKA}_{spGGP}$  selected and configured MLC algorithms in such a way they produced the best average value and rank in the *search space* Medium considering the same time budget and measure. For the *search space* Large,  $\text{Auto-MEKA}_{BO}$  and  $\text{Auto-MEKA}_{GGP}$  turned into the *search methods* with the best average value and the best average rank, respectively. The quality behavior of the proposed and baseline methods was basically the same within five hours, except for *search space* Large, where  $\text{Auto-MEKA}_{GGP}$  achieved the best average value and rank.

Similarly to the results of FM, the results of RL did not present any evidence of statistical significance. Therefore, based on the average RL for the 14 datasets, one and five hours of execution, and three *search spaces*, the proposed and comparison *search methods* did not differ from each other. As this metric comes from another context, we would like to understand why it presented such a flat result for all *search spaces* and *search methods*. Our claim stands for the RL measure being also way conservative and not showing the sensibility of different multi-label classifiers. As it only penalizes reversed pairs of labels into the ranking and does not take into account the label-pair depth in the ranking to penalize, this can make this measure not good enough to be used in isolation to evaluate MLC algorithms. In future work, it might be interesting to evaluate whether this measure is appropriate to be part of our study and/or whether we should consider another rank-based measure (e.g., coverage and average precision).

Apart from the flat results, we note three signs of overfitting in the RL results, which are important to finally decide which method to use.  $\text{Auto-MEKA}_{spGGP}$ , for in-

stance, showed a small increase of RL in the *search space* Medium. This is not a particular case of overfitting, as this difference is in the standard deviation of these results. The case of Auto-MEKA<sub>BO</sub> is more complicated for the *search space* Small. As this happened in the tuning experiments as well for the *search space* Large, we believe this method should be augmented with overfitting avoidance approaches before we run it again.

From Table 6.13 to Table 6.16, we ultimately analyzed the results of the measures that compose the fitness/quality function (see Equation 5.1 for more details), which is a combination of the previously analyzed measures. We should then examine the results corresponding to the fitness measure for a general assessment of the proposed and baseline methods. In respect of the *search space* Small, the *search method* with the overall best result is Auto-MEKA<sub>GGP</sub>, which presented the best average value rank for one hour of running and the best average value and rank within five hours. Auto-MEKA<sub>RS</sub> completed this *search space* by showing the best average value within one hour. Besides, for one and five hours of running, we found statistical evidence that Auto-MEKA<sub>GGP</sub> has better results than Auto-MEKA<sub>BO</sub> in this *search space*.

Table 6.17: Comparison of the fitness (to be maximized) obtained by the proposed methods and the baseline methods in the test set for the three designed *search spaces* within one and five hours of execution.

Time Budget	Search Space	Evaluated Result	spGGP	GGP	BO	RS	BS
1h	Small	Avg. Val.	0.623	0.626	0.617	<b>0.628</b>	0.626
		Avg. Rank.	3.321	<b>2.25</b>	4.071	2.536	2.821
		Stat. Comp.	{GGP} > {BO}, no differences among the others				
	Medium	Avg. Val.	0.632	0.638	0.621	0.631	<b>0.643</b>
		Avg. Rank.	2.75	<b>2.607</b>	4.143	2.643	2.857
		Stat. Comp.	{GGP} > {BO}, no differences among the others				
	Large	Avg. Val.	0.616	<b>0.635</b>	0.586	0.628	0.632
		Avg. Rank.	3.607	<b>2.214</b>	3.357	3.286	2.536
		Stat. Comp.	no differences among all methods				
5hs	Small	Avg. Val.	0.626	<b>0.631</b>	0.590	0.624	0.626
		Avg. Rank.	2.679	<b>1.964</b>	3.857	3.535	2.964
		Stat. Comp.	{GGP} > {BO}, no differences among the others				
	Medium	Avg. Val.	0.634	0.639	0.626	0.629	<b>0.645</b>
		Avg. Rank.	3.107	<b>2.643</b>	2.929	3.429	2.893
		Stat. Comp.	no differences for all methods				
	Large	Avg. Val.	0.632	<b>0.635</b>	<b>0.635</b>	0.626	0.632
		Avg. Rank.	2.607	<b>2.429</b>	2.571	4.286	3.107
		Stat. Comp.	{spGGP, GGP, BO} > {RS}, no differences among the others				

Next, we analyze the results of the *search space* Medium. We observe similar results for one hour of running in this particular *search space*. I.e., Auto-MEKA<sub>GGP</sub> produced the best average ranking, one of the baseline methods showed the best average value (in this case, Auto-MEKA<sub>BS</sub>), and we found statistical evidence that Auto-MEKA<sub>GGP</sub> is better than Auto-MEKA<sub>BO</sub> based on this particular measure. Nevertheless, the results of Auto-MEKA<sub>BO</sub> become better within five hours of run. This is proved by the statistical results of Table 6.17, where we have not found evidence of statistical differences among all the proposed and baseline methods.

The results regarding the *search space* Large for the fitness measure are similar to those found for the EM measure. Within one hour of the time budget, we have not found evidence of statistical differences among the methods. However, after continuing the search for five hours, the results of the proposed methods improved. In this case, Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BO</sub> achieved better results than Auto-MEKA<sub>RS</sub>, showing their capabilities to handle enormous *search spaces* – when giving enough time for them to proceed with their searches. Apart from the statistical results, we can observe in Table 6.17 that Auto-MEKA<sub>GGP</sub> also reached the best average value and rank within one and five hours. Furthermore, Auto-MEKA<sub>BO</sub> had even results to Auto-MEKA<sub>GGP</sub> in terms of the average value.

Signs of overfitting were also identified for the fitness measure for one particular case: Auto-MEKA<sub>BO</sub> in the *search space* Medium. We have already pointed out that this *search method* needs to be improved with overfitting avoidance approaches because its results have shown signs of overfitting on several occasions. However, we believe that this method is capable of improving through time in larger *search spaces*. We can see that in the *search space* Large, in particular, where the results of Auto-MEKA<sub>BO</sub> improved in several metrics from one hour to five hours.

### 6.4.1 Final Remarks

We can now provide overall conclusions with these results. First, the competitiveness of Auto-MEKA<sub>GGP</sub> is outstanding, given the differences across the evaluation measures and the increasing complexity of the analyzed *search spaces*

Second, we need to highlight the competitive results achieved by Auto-MEKA<sub>GGP</sub>, which is a *search method* that relied on a multi-fidelity optimization approach [109] in our final experiments. When we reduce the fidelity of the input dataset during training time, we expect to lose performance. However, Auto-MEKA<sub>spGGP</sub> showed to be competent in terms of the selection and configuration of MLC algorithms, showing results that are mostly equal to or better than the baseline methods.

Third, the Research Question 1 (RQ1) of Chapter 1, which is associated with Issue 1, wonders whether AutoML methods can properly work for MLC problems as well as for SLC and regression problems. This is mainly due to the complex hierarchical complexity of MLC *search spaces*. The competitive results achieved by the proposed AutoML *search methods* in different evaluation measures (against the baseline methods) indicate that it is possible to perform automated multi-label classification. Therefore, these achieved results in this section answer RQ1.

Fourth, recall that the Research Question 2 (RQ2) of Chapter 1, which is related to Issue 2, concerns about the influence of *search spaces* of different sizes on the performance of AutoML methods and the inherent hardness of searching over enormous *search spaces*. By looking at the results of this section, we believe that the size of the *search spaces* explored/exploited by the proposed *search methods* influenced in the multi-label classification. This was more evident for three of the five evaluated measures (i.e., EM, HL, and fitness). Given the results for these three measures, we understand that for smaller *search spaces* (e.g., Small and Medium), the proposed *search methods* have more facilities to proceed with their searches and, as a result of that, their results become broadly similar among each other and among one of the baseline methods (i.e., random search, which is a pure exploration *search method*). When we increase the *search space* to Large, only the proposed *search methods* were robust enough to beat Auto-MEKA<sub>RS</sub>, the evaluated pure exploration baseline method. With this analysis, we partially answer RQ2.

However, based on the results, we believe we can still improve the proposed *search methods*. Even with their improvement (on three out of five evaluated measures), when we relax the constraint of time, the proposed *search methods* could not beat the beam *search method*, a pure-exploitation method. This happened only against the random *search method* in some cases. Thus, the proposed *search methods* could not satisfactorily balance between exploration and exploitation. In our perspective, this would occur if they could beat both baseline methods. For the *search spaces* Small and Medium, this result is more understandable. The smaller the *search space*, the easier it is to perform the search on them. This yields better results for the comparison *search methods* in such a way the proposed *search methods* could not be statistically different from them.

Finally, with respect the Research Question 3 (RQ3) of Chapter 1, we believe with measure and overfitting results, we are able to partially answer this research question. Overall, the improvement of three measures (EM, HL, and fitness) from one to five hours for the *search space* Large – making them beat the random *search method* – was noted, and it is one proof that the time budget applied to the search and optimization of the AutoML methods. Nevertheless, we also identify a few cases (especially for Auto-MEKA<sub>BO</sub>) where giving more time led to poorer predictive performances. Basically, this is a clear sign of overfitting and is likely to happen in smaller *search spaces*. One possible way to avoid overfitting is to always look at the convergence of the *search method* and stop the search process when its search process stabilizes. This would make the *search method* not expending time in a “dead” process. Another way is to continue with the search process but in an unexplored area of the *search space*. Furthermore, it is also possible to try other data (re)sampling techniques [144].

With this conclusion, we partially answer RQ3 as we demonstrated how the budget constraints affect the predictive performance of the selected MLC algorithm. We complete this research question in Section 6.6, where the convergence of the methods is analyzed

through time.

## 6.5 Analysis of the Diversity of the Selected Algorithms

This section analyzes the diversity of the MLC algorithms and meta-algorithms selected by the five AutoML *search methods* – i.e., Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>GGP</sub>, Auto-MEKA<sub>BO</sub>, Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>. Besides, we also examine the selected SLC algorithms and meta-algorithms by these five evaluated *search methods*. We focus only on the selected MLC and SLC algorithms and meta-algorithms (which are the “macro-components”), and not on their selected hyper-parameter settings (the “micro-components”), to simplify the analysis.

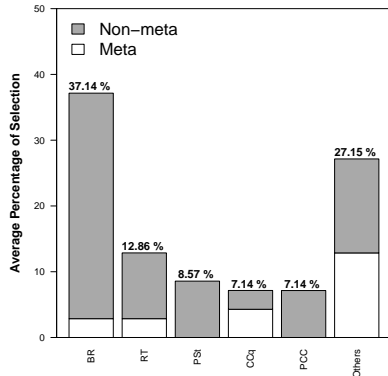
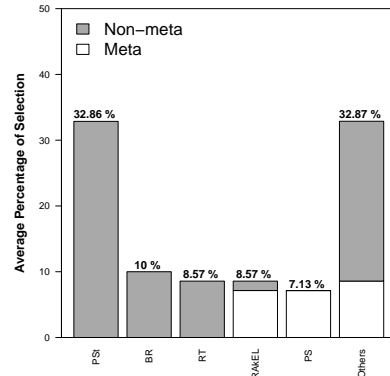
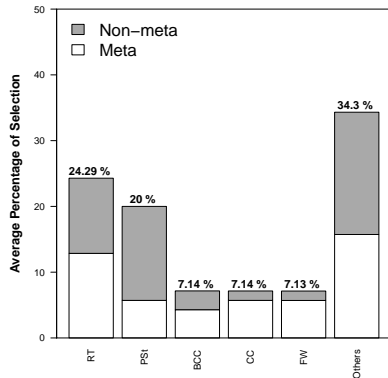
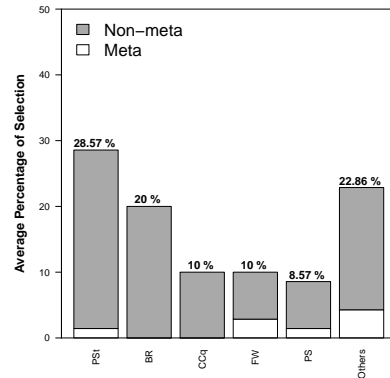
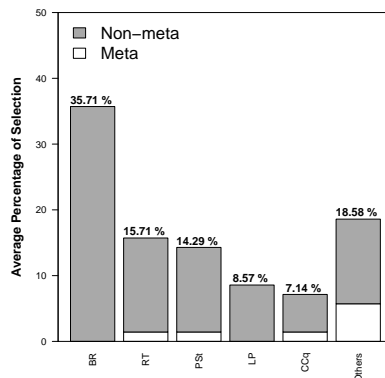
It is important to emphasize that, by analyzing the MLC and SLC algorithms and meta-algorithms selected by all *search methods* in each *search space*, we can better understand the results from Table 6.13 to Table 6.17. This would give an idea of how the choice of an MLC algorithm and an SLC algorithm influences the performance of the proposed and the baseline methods. However, for the sake of simplicity, we perform this analysis only for the *search space* Large within five hours of the time budget. This choice is purely related to the fact that, after five hours, the proposed *search methods* presented results that were statistically better than the random *search method* in this *search method*. We would like to understand why this happened against random search but not against beam search.

We present from Figure 6.1a to Figure 6.1e the bar plots to analyze the percentage of selection of MLC algorithms for the proposed and AutoML *search methods*. For more details about each MLC algorithm, see Appendix A. In these figures, we have, for each MLC algorithm, a (gray/white) bar representing the average percentage of selection of an algorithm type over all runs. These percentages rely on two cases: (i) when the traditional MLC algorithm is solely selected; and (ii) when the traditional MLC algorithm is selected together with an MLC meta-algorithm. To emphasize these two cases, the bar for each traditional MLC algorithm is divided into two parts, with sizes proportional to the percentage of selection as a standalone algorithm (in gray color) and the percentage of selection as part of a meta-algorithm (in white color).

Considering this information, BR, PSt, and RT were the traditional MLC algorithms most frequently selected by all AutoML *search methods* in the *search space* Large. BR was chosen, on average, in 21.43% of all runs for all methods. PSt and RT, in turn,



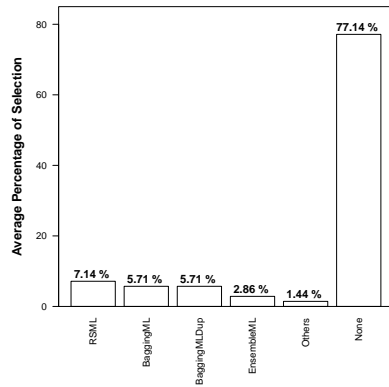
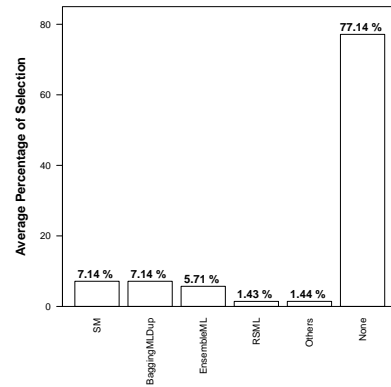
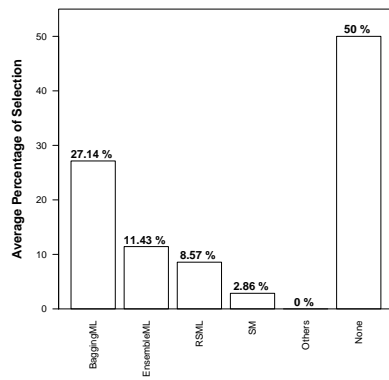
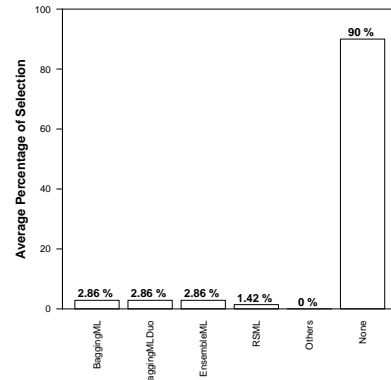
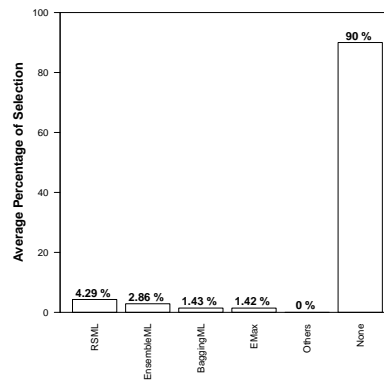
Figure 6.1: Bar plots for the algorithms' selection at the MLC level over all runs.

(a) Auto-MEKA<sub>GGP</sub>.(b) Auto-MEKA<sub>spGGP</sub>.(c) Auto-MEKA<sub>BO</sub>.(d) Auto-MEKA<sub>RS</sub>.(e) Auto-MEKA<sub>BS</sub>.

appeared, on average, in 20.66% and 13.43% of all runs for the five evaluated *search methods*, respectively. Nevertheless, some of these MLC algorithms were not so present in the selections performed by the *search methods*. For instance, BR and RT were not frequently chosen by Auto-MEKA<sub>BO</sub> and Auto-MEKA<sub>RS</sub>. This partially shows the differences in the selection and configuration of the AutoML *search methods*, although most of them had similar algorithms at the top five regarding the ranking of selection.

Together with Figures 6.2a, 6.2b, 6.2c, 6.2d, and 6.2e, we can also justify the performance of the proposed and baseline methods in accordance with their selection

Figure 6.2: Bar plots for the algorithms' selection at the Meta-MLC level over all runs.

(a) Auto-MEKA<sub>GGP</sub>.(b) Auto-MEKA<sub>spGGP</sub>.(c) Auto-MEKA<sub>BO</sub>.(d) Auto-MEKA<sub>RS</sub>.(e) Auto-MEKA<sub>BS</sub>.

at the MLC level and at the MLC meta-level. For example, Auto-MEKA<sub>GGP</sub> achieved the best results for the *search space* Large in terms of the average value and rank on the fitness, which are the measures we use to decide (in all methods) what algorithm is the most appropriate. We can understand why this happened by looking at Auto-MEKA<sub>GGP</sub>'s selection at the MLC meta-level. Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>spGGP</sub> were the proposed *search methods* with the lowest percentage of selection of MLC meta-algorithms. Therefore, the complexity of the final solution made them turn into better options for AutoML in the MLC context when contrasted to Auto-MEKA<sub>BO</sub>. However,

their level of selection of MLC meta-algorithms is still high. This might be a reason why Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub> have close results to both proposed methods.

One test that might be interesting is to remove the MLC meta-algorithms from the *search space* and re-execute the proposed *search methods*. This could show us whether, when we select very complex combinations of base and meta-algorithms, the produced model can have or not more chances to overfit on the test set. We did that in the *search spaces* Small and Medium, but they do not comprehend all traditional MLC algorithms as the *search space* Large does.

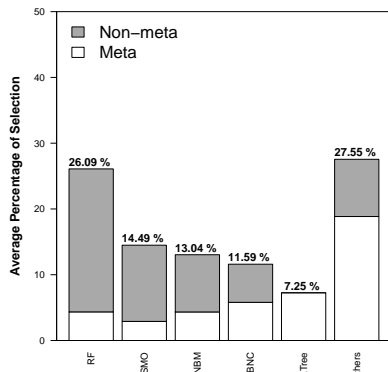
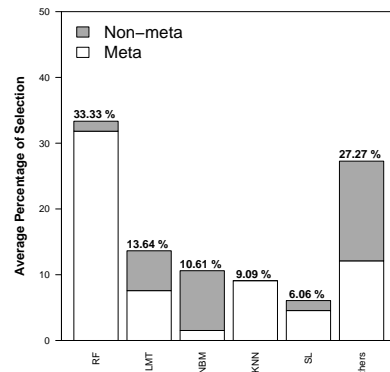
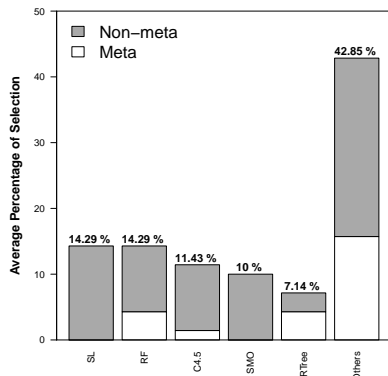
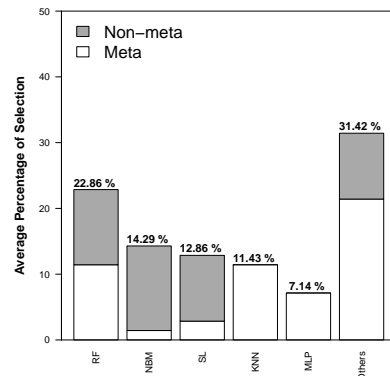
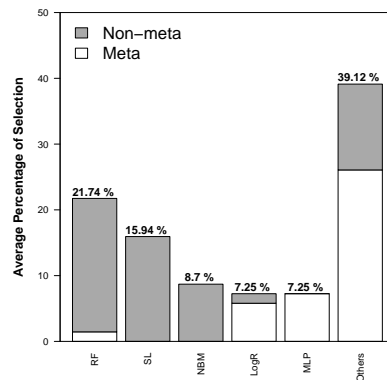
Now, we look at the algorithms selected at the SLC level (see Appendix A for more details). Observing Figures 6.3a, 6.3b, 6.3c, 6.3d and 6.3e, it is clear that the most selected algorithm was the Random Forest (RF), which appeared, on average, in 23.66% of the cases for all methods. Simple Logistic (SL), in turn, became evident in the selection of Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>BO</sub>, Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>. Actually, even though it is not evident in Auto-MEKA<sub>GGP</sub> at the top five of the selection ranking, SL was chosen on an average of 10.7% of the cases by the five methods. Differently, Naïve Bayes Multinomial (NBM) was regularly selected by Auto-MEKA<sub>GGP</sub>, Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub>. In fact, NBM was selected on an average of 11.66% by these four methods. On the other hand, Auto-MEKA<sub>BO</sub> focused on other SLC algorithms when performing its search, such as C4.5 and support vector machine (i.e., SMO).

Concerning the meta and preprocessing algorithms selected at the SLC level, the first thing we observe when looking at Figures 6.3a, 6.3b, 6.3c, 6.3d, and 6.3e is that these methods differ on how they choose the meta-algorithms. Figures 6.4a, 6.4b, 6.4c, 6.4d, and 6.4e complement this information by showing that the percentage of selection of meta or preprocessing algorithms. Taking it into account, we can observe that Auto-MEKA<sub>spGGP</sub>, Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>GGP</sub> selected and configured more meta and preprocessing algorithms at the SLC level (percentages of 62.86%, 55.71% and 42.86%, respectively), whilst Auto-MEKA<sub>BS</sub> and Auto-MEKA<sub>BO</sub> had a smaller percentage of selection of such meta-learning algorithms (40% and 25.71%, respectively).

An interesting observation is that the preprocessing algorithm, namely Attribute Selection Classifier (ASC), was the most selected algorithm for the five analyzed methods, appearing on an average of 20.28% of the cases. The other meta-algorithms were not frequently selected by any of the proposed or baseline methods. For the other meta-algorithms, the selection percentages varied very much in such a way that it is difficult to identify an algorithm regularly chosen by the *search methods*.

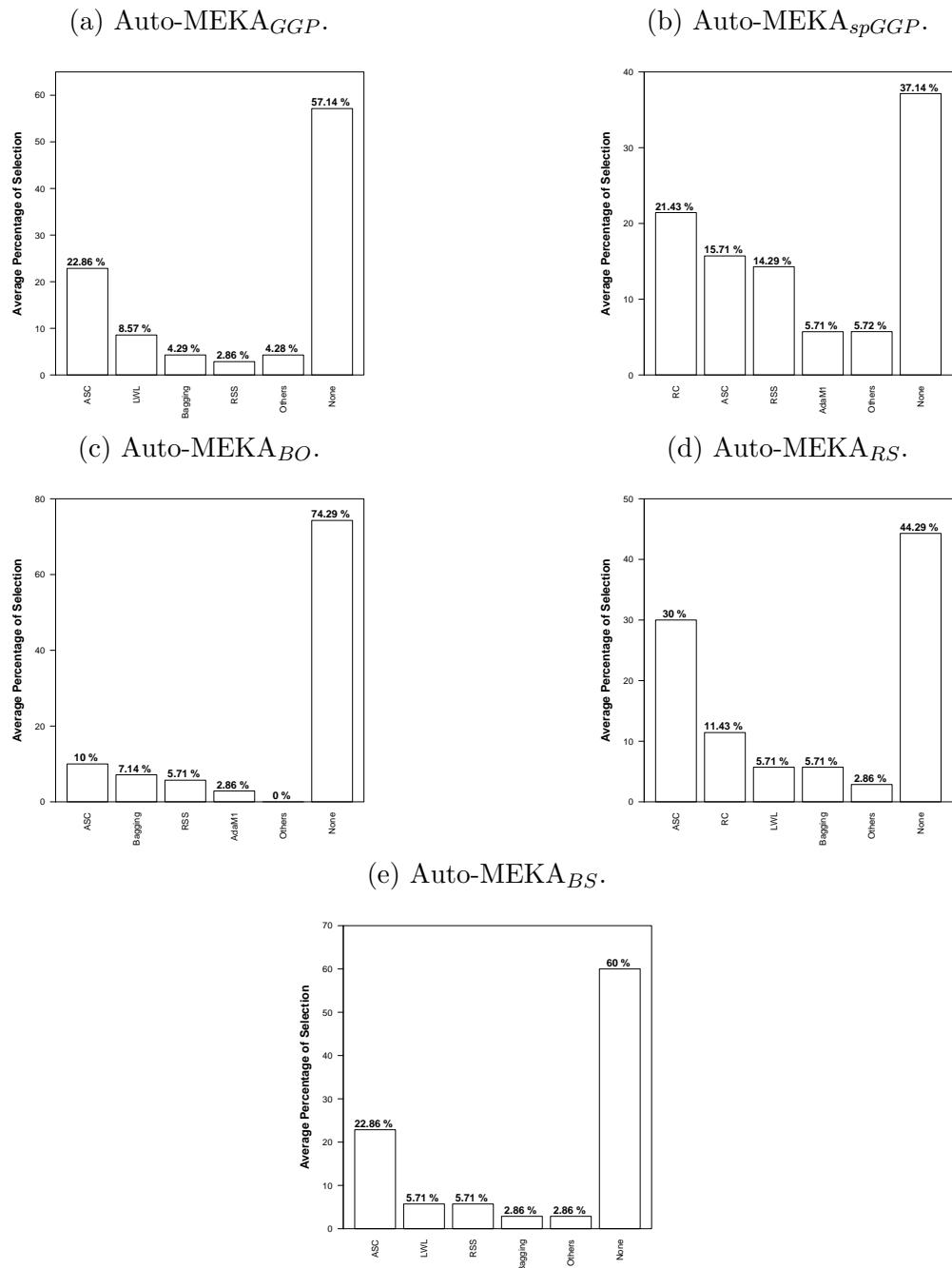
For instance, we analyze that Locally Weighted Learning (LWL) was frequently chosen by Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>BS</sub>. Random Committee (RC), in turn, appeared in the top two of the selections of Auto-MEKA<sub>spGGP</sub> and Auto-MEKA<sub>RS</sub>. Distinctly, Bagging is more present in Auto-MEKA<sub>BO</sub>'s choices. Finally, Random Subspace

Figure 6.3: Barplots for the algorithms' selection at the SLC level over all runs.

(a) Auto-MEKA<sub>GPP</sub>.(b) Auto-MEKA<sub>spGPP</sub>.(c) Auto-MEKA<sub>BO</sub>.(d) Auto-MEKA<sub>RS</sub>.(e) Auto-MEKA<sub>BS</sub>.

(RSS) was also one of the most selected methods of Auto-MEKA<sub>spGPP</sub>, Auto-MEKA<sub>BO</sub>, and Auto-MEKA<sub>BS</sub>.

Figure 6.4: Bar plots for the algorithms' selection at the Meta-SLC level selection over all runs.



### 6.5.1 Final Remarks

Based on these results, we can conclude that the *search methods* have different biases on the selection of learning algorithms at the SLC meta-level. One possible way to better understand this case is to isolate these types of algorithms to run the experiments and check if they are really needed to improve the classification performance of the *search*

*methods*. By comparing to the results of the *search spaces* Small and Medium, it seems they do not influence so much on the final predictive performance of the selected MLC algorithms. This is associated with the complexity of the search because, when we include these meta-algorithms into the *search space*, we actually make the search more and more difficult. This occurs as a consequence of adding other hierarchical levels (e.g., the meta SLC level) to the *search space* (see Chapter 5 for more details). This is out of the scope of this thesis. We intend to deal with it in future work.

By considering and noting the biases of the proposed *search methods* on the final selection of SLC and MLC algorithms and meta-algorithms, we can actually have a better comprehension of the performances of these methods and complement the answers of RQ1. Besides, this analysis of the diversity assists us in improving the way the *search methods* behave in future work.

## 6.6 Analysis of Convergence

This section compares the convergence behaviors of the proposed AutoML methods, aiming to understand (even more) how they work. We did not include the baseline methods in this analysis because we primarily would like to understand the behavior of the proposed methods. Besides, random search (i.e., Auto-MEKA<sub>RS</sub>) does not have convergence because it is a pure-exploration *search method*. In contrast, beam search (i.e., Auto-MEKA<sub>BS</sub>) is too simple as it only changes the best solution found so far if and only if it discovers a solution in the neighborhood that has a higher fitness score than the current best solution. This makes Auto-MEKA<sub>BS</sub> to converge quickly – due to its pure-exploitation way to look at the *search space*. Finally, to understand the convergence behavior of Auto-MEKA<sub>spGPP</sub>, we examine only Species 7 to be fair in the comparisons to the other *search methods*. This species fully encompasses all the features (components and hyper-parameters) of the *search spaces* (for more details, see Chapter 5).

To analyze the convergence of the proposed *search methods*, we explore the *search spaces* Small and Large. This decision was made due to the clear difference in the size of these *search spaces*. In addition, we choose only two (2) out of the 14 datasets, i.e., *GPP* and *CAL*. This was done to simplify the assessment of the methods. Because of this simplification, we selected the datasets based on their differences in the basic features, such as domain, number of attributes, number of labels, and number of labels.

Figures 6.5a to 6.5d illustrate the fitness evolution of the best individuals of the population and the average fitness of the population of individuals of the evolutionary AutoML methods for the dataset *GPP* (in just one run) for the *search spaces* Small and

Large. Figure 6.5e and 6.5f, in contrast, show the convergence of the configurations' quality for the Bayesian optimization method through time for the same dataset for the *search space* Small and Large, respectively.

Observe in these figures that, as the *search space* Small has less complex learning algorithms, it evaluates way more MLC hyper-parameterized algorithms than the *search space* Large. For this reason, as well, we can analyze that all methods suffer from premature convergence, but the methods that perform in the *search space* Small seem to be more stable and, as a consequence, converge faster in all cases. On the other side (i.e., in the *search space* Large), Auto-MEKA<sub>GPP</sub> and Auto-MEKA<sub>GPP</sub> seem to struggle during the convergence process. Although there is a faster convergence after a few minutes (less than 30 minutes in most cases) of time budget (see the best fitness curves), the fitness average curve for these evolutionary methods does not stabilize. Mainly, this happens due to data resampling and reinitialization of the population after convergence. Nevertheless, we expected flatter curves instead. Perhaps, the proposed *search methods* are exploring/exploiting so many complex algorithms that, when they reach the algorithm time limit (of three minutes), having fitness values equal to zero, they push the average curve to lower values. As we identified in the last section, this can happen in a great amount of time, and it is something to be studied in future work (i.e., to remove the meta-algorithms from the *search space* and run the experiments again).

Regarding Auto-MEKA<sub>BO</sub>, we can say that this proposed Bayesian optimization method also converges much earlier than expected for both *search spaces*. The difference is the stability of the configuration's quality curve. In this case, we see a flatter curve. The change of the *search space* just added more points being evaluated. One possible test is to check the performance of the selected and configured MLC algorithms after 30 minutes of running and compare them to one and five hours of time budget. Given Auto-MEKA<sub>BO</sub>'s quality curve, we have a hypothesis that the final predictive result would be the same within 30 minutes, saving computational time. We can also conclude the same for the other *search methods*.

We also draw the same curves for the dataset *CAL*, which we consider harder than the previous one (i.e., *GPP*). These curves are presented in Figures 6.6a to 6.6f. Even though it is computationally harder to be evaluated (based on its characteristics), the results are broadly similar to the ones reached for the dataset *GPP*. For example, after about 15 minutes, Auto-MEKA<sub>BO</sub> could not find any solution better than the current one. In accordance with these results, the evolutionary-based methods needed only a few generations to stabilize the best fitness curve.

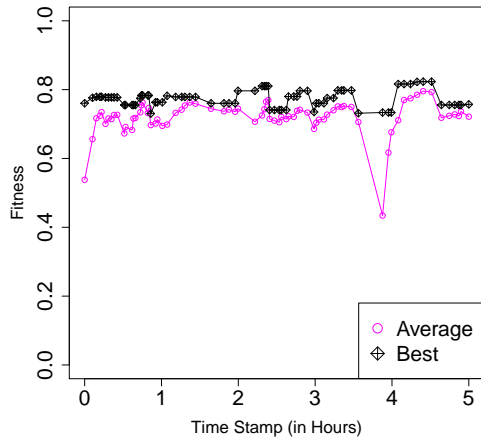
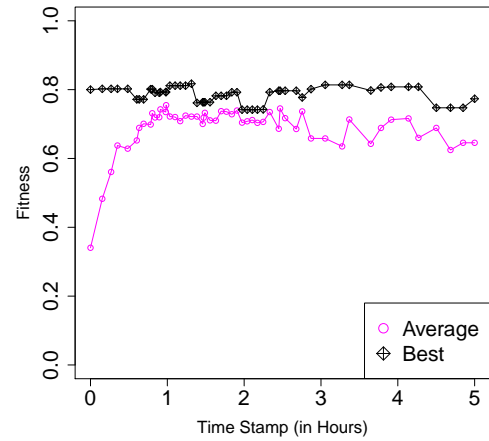
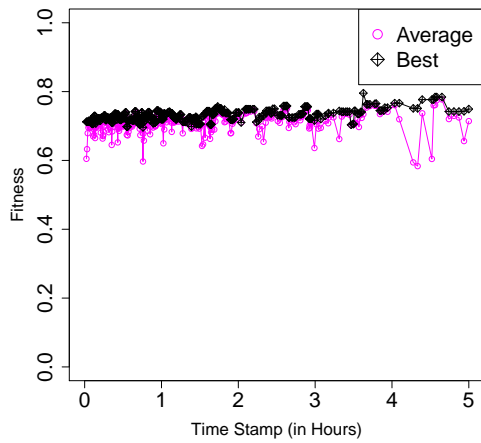
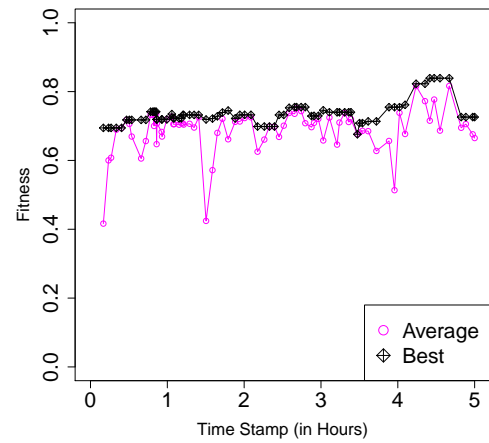
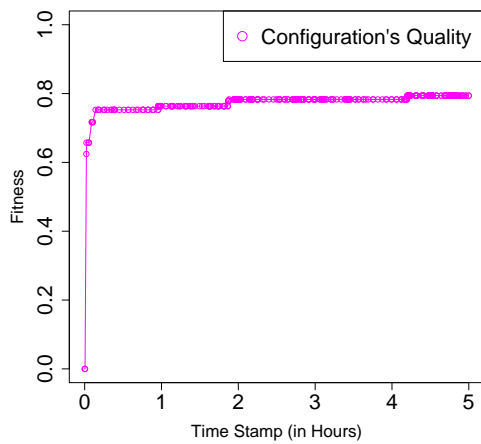
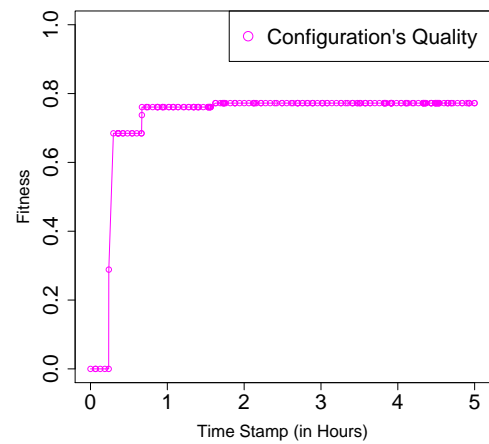
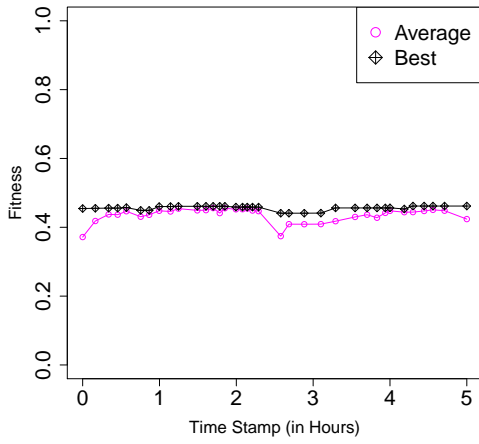
Figure 6.5: Convergence of fitness/quality values for the dataset *GPP*.(a) Auto-MEKA<sub>GPP</sub>: Small.(b) Auto-MEKA<sub>GPP</sub>: Large.(c) Auto-MEKA<sub>spGPP</sub>: Small.(d) Auto-MEKA<sub>spGPP</sub>: Large.(e) Auto-MEKA<sub>BO</sub>: Small.(f) Auto-MEKA<sub>BO</sub>: Large.

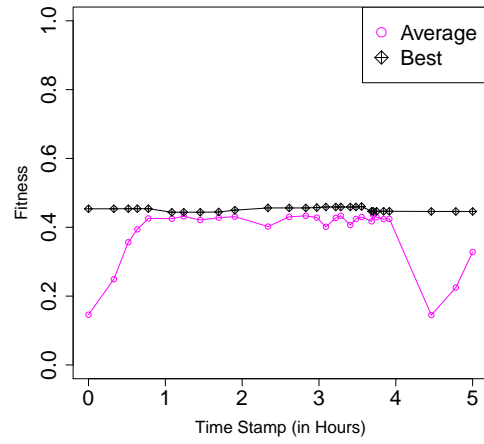


Figure 6.6: Convergence of fitness/quality values for the dataset *CAL*.

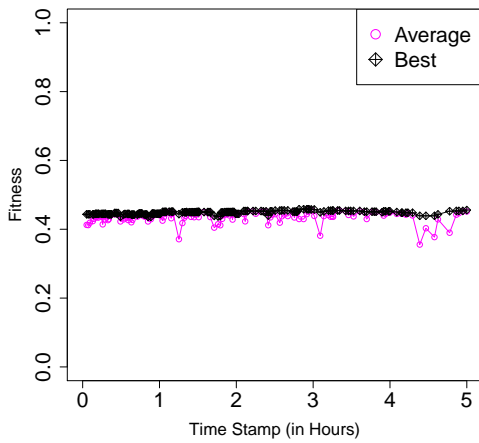
(a) Auto-MEKA<sub>GGP</sub>: Small.



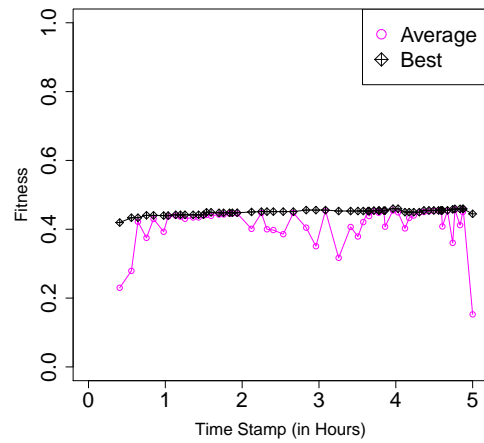
(b) Auto-MEKA<sub>GGP</sub>: Large.



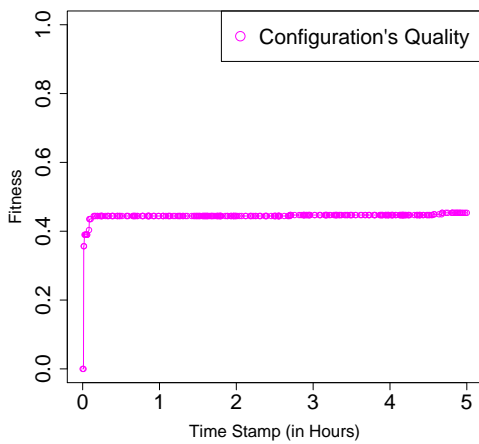
(c) Auto-MEKA<sub>spGGP</sub>: Small.



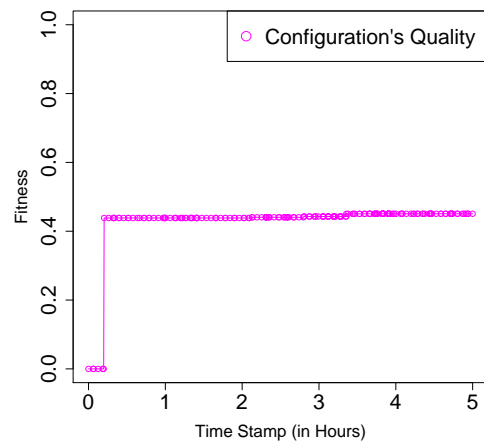
(d) Auto-MEKA<sub>spGGP</sub>: Large.



(e) Auto-MEKA<sub>BO</sub>: Small.



(f) Auto-MEKA<sub>BO</sub>: Large.



### 6.6.1 Final Remarks

As an overall conclusion, we understand that the sizes of the explored *search spaces* can indeed justify the premature convergence. When this size is enormous, the proposed AutoML methods could fall into local optima, possibly indicating that exploitation prevails over exploration. When this size is small, we can basically cover its limits quickly, and the convergence happens naturally. This subject is also related to RQ2. Complementing the results of Section 6.4, we show in this section that the methods are not actually balancing very well between exploration and exploitation. To fix this issue of their convergence behavior, we could try to change the way the methods work to provide a better trade-off between exploitation and exploration. For instance, learning more about the *search spaces* in execution time would give us a better chance to discover more successful MLC algorithms. In particular, we could have at the core of the evolutionary-based methods a model to help them to perform basic selection decisions. As the AutoML *search spaces* are usually enormous, this might help to find MLC algorithms with competitive predictive performance.

Furthermore, we varied time budgets applied to the AutoML methods. Distinct time budgets naturally influence how the methods search/optimize the algorithms. For specific *search spaces*, some of the proposed methods (especially Auto-MEKA<sub>BO</sub>) suffered from overfitting. Therefore, depending on the features of the *search method*, it is an alternative to stop (and restart) the search/optimization process after reaching convergence. This mechanism might help the methods to control their overfitting. However, in the same way, the time we give for the AutoML method to run may impact how much of the search space will be covered. Therefore, it might be good to restart the search process after convergence (as we are doing for the evolutionary methods now), but it might be interesting to use the information from other starts (e.g., we can encompass this information into a predictive model) and continue the search process without being completely blind from the beginning. This might be a challenging task for future work, i.e., to understand the *search spaces* in execution time and use it to improve the AutoML final predictive performance.

Finally, another alternative is to include other mechanisms to avoid overfitting, such as the one recommended in Mohr et al. [144], which is based on a two-phased model to control the error rates on the test set. These assessments and analyses are connected to the answers to Research Question 3 (RQ3) of Chapter 1 and, consequently and complementarily, address Issue 3.

# Chapter 7

## Conclusions and Future Work

This chapter draws the main conclusions about the results and analysis performed to tackle the open issues defined in Chapter 1 and our progress in the field. We organize our results and conclusions according to these issues.

### 7.1 Issue 1: Proposing AutoML Methods for the Multi-label Classification Context

In Chapter 5, we propose four *search methods* that tackle the AutoML task in the MLC context: (i) GA-Auto-MLC; (ii) Auto-MEKA<sub>GGP</sub>; (iii) Auto-MEKA<sub>spGGP</sub>; and (iv) Auto-MEKA<sub>BO</sub>. We show their flexibility by running them on top of three designed *search spaces* (Small, Medium, and Large). As a result, we show that these proposed AutoML methods can handle the complex *search spaces* of MLC algorithms and return configured MLC algorithms that present competitive predictive results – except for GA-Auto-MLC, which was not included in the main analysis of this thesis due to the complexity of the selected and configured MLC algorithms it produced (see Section 6.2 for more details about this topic).

Observing the results of Section 6.4, we conclude that the proposed AutoML methods mostly had similar multi-label classification performances across the 14 datasets. However, we noted some cases of statistical differences among the proposed methods themselves. For instance, Auto-MEKA<sub>GGP</sub> presented statistically better results than Auto-MEKA<sub>spGGP</sub> twice. When compared to Auto-MEKA<sub>BO</sub> in three scenarios (i.e., a combination of evaluation measure, time budget, and *search space*), Auto-MEKA<sub>GGP</sub> showed a statistically higher performance. Because of that, we found this method – Auto-MEKA<sub>GGP</sub> – the most prominent. Auto-MEKA<sub>GGP</sub> has also shown its competitiveness when it performed statistically better than Auto-MEKA<sub>RS</sub> in three scenarios.

In our perspective, by having equivalent predictive performances in most cases,

Auto-MEKA<sub>spGGP</sub> has also shown its competitiveness. As we explained in Chapter 6, this *search method* employs a multi-fidelity optimization approach during its search process. Therefore, we expected it not to reach the top performance in all cases. However, its equivalence to the other proposed *search method* (except for two cases) showed that Auto-MEKA<sub>spGGP</sub> is indeed a reliable AutoML *search method*. We intend in future work to run the other proposed methods with the specified multi-fidelity approaches as well (see Section 5.4 for more details) in order to give a better evaluation of these methods when compared to Auto-MEKA<sub>spGGP</sub>.

With respect to Auto-MEKA<sub>BO</sub>, given the predictive results obtained by the algorithms selected by this method, we believe it still needs to be enhanced with strategies to prevent overfitting before being applied to real-world MLC problems. We could incorporate into this method the same strategy used in Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>spGGP</sub>, i.e., resampling the learning and validation set every  $g$  generations, where  $g$  in our thesis took the value of five. Besides, this method needs a mechanism of convergence analysis during its execution time. By looking at Auto-MEKA<sub>BO</sub>'s convergence, we understand that this method is wasting time in its search as it could stop the optimization process and restart it in an unexplored part of the *search space*. Nevertheless, Auto-MEKA<sub>BO</sub> has achieved robustness in its results, evolving through time and presenting statistically better results than Auto-MEKA<sub>RS</sub> in three cases for the *search space* Large. This was not even achieved by Auto-MEKA<sub>GGP</sub>.

Overall, these results and conclusions indicate that AutoML can work for MLC problems as well as for SLC and regression problems, as pointed out in Issue 1 of Section 1.1, particularly for more complex *search spaces*. If we had to recommend the user one method to start AutoML for MLC problems, it would be Auto-MEKA<sub>GGP</sub> because of its inherent performance and consistency in recommending and configuring MLC algorithms.

As future work, we understand it is worth testing other ways of defining species for Auto-MEKA<sub>spGGP</sub>. Furthermore, adding a local operator to these global search methods could help improve exploitation.

## 7.2 Issue 2: Presence of an Exploration-Exploitation Trade-off in AutoML Methods

By looking at the results in Section 6.4, we observe that there is a high correlation between the size (and definition) of the search space and the effectiveness of AutoML methods to select and configure algorithms. With these results, we have a pure indication

that as the AutoML’s *search space* decreases, pure-exploration and/or pure-exploitation AutoML *search methods* tend to have similar results than robust AutoML methods (such as the ones proposed in this thesis).

However, we found that currently, the proposed *search methods* can not satisfactorily settle a trade-off between exploration and exploitation. As discussed earlier, we claim that this would only happen if the proposed *search methods* had beaten the baseline methods, which represent the two extremes: one favors exploration all the time while the second favors exploitation.

Although the three proposed *search methods* could improve when given more time to them (i.e., five hours) and beat random search (a.k.a., Auto-MEKA<sub>RS</sub>) in two measures (EM and fitness) in the *search space* Large, we believe this is still not enough. For future work, we expect to improve the proposed methods in order to reach a better balance between exploration and exploitation, consequently improving predictive results.

One way to do it is to focus on the coverage of the *search space*. As shown in Section 6.5, the proposed methods that performed better were those that selected simpler but effective MLC algorithms. For instance, Auto-MEKA<sub>RS</sub> and Auto-MEKA<sub>BS</sub> did not select meta algorithms at the MLC level so often. The opposite happened with Auto-MEKA<sub>BO</sub>, which is the method (among the proposed ones) with the highest percentage of selection of meta-algorithms. These results indicate that the combination of *search method* and *search space* is crucial to solving an AutoML problem, particularly in the MLC scenario. Therefore, it is not only novel *search methods* that matter in AutoML research, but also the *search space* that is explored (and exploited) by them. A fine-grained comprehension of the definition of the *search spaces* is a topic for future work in this thesis.

Recall that this further analysis is associated with the sufficiency and parsimony of the *search spaces* [11], as defined in Chapter 1. Whereas we want a powerful *search space* that has the most promising MLC algorithm configurations (*sufficiency*), we also want to keep only the necessary configurations not to enlarge the *search space* so much and made the search harder (*parsimony*). In Section 6.5, we discussed about this trade-off between sufficiency and parsimony. The idea is to test whether the SLC and MLC meta-algorithms are really necessary to the search.

This further study is also important to GA-Auto-MLC, which is a method that suffers from the complexity in the solution space. I.e., it tends to select very complex MLC algorithms (and meta-algorithms) for the MLC problems. In this regard, we would like to properly understand this issue on GA-Auto-MLC as well in future analysis.

Finally, we also want to analyze in depth the characteristics of the *search spaces* of AutoML problems, understanding the shape of their fitness landscape [164, 138].

## 7.3 Issue 3: The Impact of Constrained Time Budgets on the Performance of AutoML Methods

In the experiments reported in Chapter 6, the proposed AutoML methods were evaluated considering two contrasting time budgets for the whole search process (i.e., one and five hours).

We showed that when we increase the time budget from one hour to five hours, two behaviors appear in the *search methods*. First, for some *search spaces*, the proposed *search methods* improved their performance. A clear example of this behavior can be found in the execution of the methods in the *search space* Large when the measures EM, HL, and fitness were analyzed.

Second, we understand that giving the *search methods* more time to run increased their chances of overfitting. By analyzing the results of Section 6.6, we conclude that the chances of overfitting increase as the methods converge faster than expected. This is a serious issue for Auto-MEKA<sub>BO</sub>, which does not have specific mechanisms to avoid overfitting. Based on the predictive results, we identify that overfitting is likely to happen in smaller *search spaces*. For the proposed *search methods*, overfitting occurred only once for the *search space* Large. We believe this is strictly associated with the convergence of the methods, as convergence generally happened faster for smaller *search spaces*.

In future work, we aim to include specific mechanisms into Auto-MEKA<sub>BO</sub> to control convergence and overfitting. Besides, we would like to investigate novel ways to control overfitting, which can also be included in the search mechanisms of Auto-MEKA<sub>GGP</sub> and Auto-MEKA<sub>spGGP</sub>.

## 7.4 Final Remarks

This chapter discussed how the results presented in this thesis match the expected contributions defined in Chapter 1. Considering our formalization of the AutoML problem in Chapters 4 and 5, we claim we completed the first expected contribution of this thesis.

In addition, given the four proposed *search methods* and the three designed *search spaces*, we understand that the second and third expected contributions were achieved: the proposal of novel AutoML methods for the multi-label classification context and the mod-

eling of specific *search spaces* for this scenario. Furthermore, by showing the competitive predictive performances of the proposed AutoML methods in Chapter 6, we demonstrated that this thesis reached satisfactory results regarding the fourth contribution.

The fifth and sixth expected contributions are basically topics of Sections 6.4, 6.5 and 6.6 of Chapter 6, i.e., the analysis of the influence of the size of the *search spaces* and time budgets on the proposed *search methods*. As a consequence, these contributions were naturally covered.

It is important to emphasize that, as far as we know, no study in the literature tries to understand the predictive performance of AutoML methods in terms of the defined *search spaces*, selected algorithms, and achieved convergence. Therefore, the analysis made in Chapter 6 turns this thesis into an important contribution to the AutoML community. We agree that novel methods should be proposed, but we perceive a lack of studies trying to understand the whys in AutoML, i.e., to give an assessment of the search or optimization processes before going ahead into novel methods. We believe we made a first step towards this more in-depth understanding of what works or not and why in AutoML, particularly for multi-label classification problems.

## 7.5 Publications

Given our achieved results, this thesis has generated so far the following publications:

- **Alex G. C. de Sá**, Cristiano G. Pimenta, Alex A. Freitas, and Gisele L. Pappa. A robust experimental evaluation of automated multi-label classification methods. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 175–183, ACM, 2020.
- **Alex G. C. de Sá**, Alex A. Freitas, and Gisele L. Pappa. Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming. In Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN), 308-320, Springer, 2018.
- **Alex G. C. de Sá**, Alex A. Freitas, and Gisele L. Pappa. Multi-label classification search space in the MEKA software. arXiv preprint, arXiv:1811.11353v2, 2019.
- **Alex G. C. de Sá**, Gisele L. Pappa, and Alex A. Freitas. Towards a method for automatically selecting and configuring multi-label classification algorithms. In

Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO Companion), 1125-1132, ACM, 2017.

During the PhD, we also collaborated with other professors and students, generating the following publications that are related to the field of AutoML:

- Márcio P. Basgalupp, Rodrigo C. Barros, **Alex G. C. de Sá**, Gisele L. Pappa, Rafael G. Mantovani, André C. P. L. F. de Carvalho, Alex A. Freitas. An extensive experimental evaluation of automated machine learning methods for recommending classification algorithms. arXiv, 2020.
- Cristiano G. Pimenta, **Alex G. C. de Sá**, Gabriela Ochoa, and Gisele L. Pappa. Fitness landscape analysis of automated machine learning search spaces. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP), 114–130, Springer, 2019.
- **Alex G. C. de Sá**, Adriano César M. Pereira, and Gisele L. Pappa. A customized classification algorithm for credit card fraud detection. Engineering Applications of Artificial Intelligence, 72 C, 21-29, Elsevier, 2018.
- **Alex G. C. de Sá**, Walter José G. S. Pinto, Luiz Otávio V. B. Oliveira, and Gisele L. Pappa. RECIPE: a grammar-based framework for automatically evolving classification pipelines. In Proceedings of the European Conference on Genetic Programming (EuroGP), 246-261, Springer, 2017.
- **Alex G. C. de Sá**, Gisele L. Pappa, and Adriano César M. Pereira. Generating personalized algorithms to learn Bayesian network classifiers for fraud detection in Web transactions. In Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia), 179-186, ACM, 2014.
- **Alex G. C. de Sá**, and Gisele L. Pappa. A hyper-heuristic evolutionary algorithm for learning Bayesian network classifiers. In Proceedings of the Ibero-American Conference on Artificial Intelligence (IBERAMIA), Santiago, Chile, 2014.

We still intend to submit our enhancements on the proposed methods and their respective results and analyses to high-level journals directly related to the field of artificial intelligence, evolutionary computation, and machine learning.



# Bibliography

- [1] Michael Abramovici, Manuel Neubach, Madjid Fathi, and Alexander Holland. Competing fusion for Bayesian applications. In *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, IPMU'08, pages 378–385, Málaga, Spain, 2008. Department of Applied Mathematics, University of Málaga.
- [2] Charu Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer, New York, NY, USA, 2012.
- [3] David Aha, Dennis Kibler, and Marc Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
- [4] Reem Al-Otaibi, Peter Flach, and Meelis Kull. Multi-label classification: A comparative study on threshold selection methods. In *Proceedings of the International Workshop on Learning over Multiple Contexts*, LMCE/ECML-PKDD'14, 2014.
- [5] Ahmed Alaa and Mihaela van der Schaar. AutoPrognosis: Automated clinical prognostic modeling via Bayesian optimization with structured kernel learning. In *Proceedings of the International Conference on Machine Learning*, volume 80 of *ICML'18*, pages 139–148. Proceedings of Machine Learning Research (PMLR), 2018.
- [6] Ahmed Alaa and Mihaela van der Schaar. Prognostication and risk factors for cystic fibrosis via automated machine learning. *Scientific Reports*, 8(1):11242, 2018.
- [7] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. DENSER: Deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20(1):5–35, 2019.
- [8] Christopher Atkeson, Andrew Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1):11–73, 1997.
- [9] Thomas Back, David Fogel, and Zbigniew Michalewicz. *Evolutionary computation 2: Advanced algorithms and operators*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1999.
- [10] Thomas Back and Hans-Paul Schwefel. Evolutionary computation: An overview. In *Proceedings of International Conference on Evolutionary Computation*, CEC'96, pages 20–29, New York, NY, USA, 1996. IEEE.

- 
- [11] Wolfgang Banzhaf, Frank Francone, Robert Keller, and Peter Nordin. *Genetic programming: An introduction*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1998.
- [12] Rodrigo Barros, André de Carvalho, and Alex Freitas. *Automatic design of decision-tree induction algorithms*. Springer, AG, Switzerland, 2015.
- [13] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, Red Hook, NY, USA, 2011. Curran Associates, Inc.
- [14] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.
- [15] Concha Bielza and Pedro Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, 47(1):5, 2014.
- [16] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. In *Proceedings of the International Conference on Machine Learning*, ICML'98, pages 55–63, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers, Inc.
- [17] Jasmin Bogatinovski. Automating machine learning for structured output prediction. Technical report, Department of Knowledge Technologies, Jozef Stefan Institute, Ljubljana, Slovenia, 2018.
- [18] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: Probability and Statistics*, 9:323–375, 2005.
- [19] Remco Bouckaert. *Bayesian belief networks: From construction to inference*. PhD thesis, The University of Utrecht, Utrecht, Netherlands, 1995.
- [20] Remco Bouckaert. Bayesian network classifiers in WEKA. Technical report, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 2007.
- [21] Matthew Boutell, Jiebo Luo, Xipeng Shen, and Christopher Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.
- [22] Jeffrey Bradford, Clayton Kunz, Ron Kohavi, Clifford Brunk, and Carla Brodley. Pruning decision trees with misclassification costs. In *Proceedings of the European Conference on Machine Learning*, ECML'98, pages 131–136, Berlin/Heidelberg, Germany, 1998. Springer-Verlag.

- [23] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Meta-learning: Applications to data mining*. Springer-Verlag, Berlin/Heidelberg, Germany, 2008.
- [24] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [25] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [26] Klaus Brinker, Johannes Fürnkranz, and Eyke Hüllermeier. A unified model for multilabel classification and ranking. In *Proceedings of European Conference on Artificial Intelligence, ECAI’06*, pages 489–493, Amsterdam, The Netherlands, 2006. IOS Press.
- [27] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Proceedings of the International Conference on Algorithmic Learning Theory, ALT’09*, pages 23–37, Berlin/Heidelberg, Germany, 2009. Springer-Verlag.
- [28] Alberto Cano, Amelia Zafra, Eva Gibaja, and Sebastián Ventura. A grammar-guided genetic programming algorithm for multi-label classification. In *Proceedings of the European Conference on Genetic Programming, EuroGP’13*, pages 217–228, Berlin/Heidelberg, Germany, 2013. Springer-Verlag.
- [29] Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits. In *Proceedings of the International Conference on Machine Learning, ICML’15*. Proceedings of Machine Learning Research (PMLR), 2015.
- [30] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. Getting the most out of ensemble selection. In *Proceedings of the International Conference on Data Mining, ICDM’06*, pages 828–833, New York, NY, USA, 2006. IEEE.
- [31] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the International Conference on Machine Learning, ICML’04*, New York, NY, USA, 2004. ACM.
- [32] Saskia Cessie and Johannes van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.
- [33] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.
- [34] Francisco Charte, Antonio Rivera, Mará del Jesus, and Francisco Herrera. Addressing imbalance in multilabel classification: measures and random resampling algorithms. *Neurocomputing*, 2015.

- 
- [35] Lena Chekina, Lior Rokach, and Bracha Shapira. Meta-learning for selecting a multi-label classification algorithm. In *Proceedings of the International Conference on Data Mining Workshops*, ICDMW'11, pages 220–227, New York, NY, USA, 2011. IEEE.
- [36] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO'18, pages 402–409, New York, NY, USA, 2018. ACM.
- [37] Linlin Chen, Degang Chen, and Hui Wang. Alignment based kernel selection for multi-label learning. *Neural Processing Letters*, pages 1–21, 2018.
- [38] Weizhu Chen, Jun Yan, Benyu Zhang, Zheng Chen, and Qiang Yang. Document transformation for multi-label feature selection in text categorization. In *Proceedings of the International Conference on Data Mining*, ICDM'07, pages 451–456, New York, NY, USA, 2007. IEEE.
- [39] François Chollet. Keras: Deep learning library for Theano and Tensorflow. <https://keras.io>, 2016.
- [40] Amanda Clare and Ross King. Knowledge discovery in multi-label phenotype data. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD'01, pages 42–53, Berlin/Heidelberg, Germany, 2001. Springer-Verlag.
- [41] John Cleary and Leonard Trigg. K\*: An instance-based learner using an entropic distance measure. In *Proceedings of the International Conference on Machine Learning*, ICML'95, pages 108–114, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers, Inc.
- [42] William Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, ICML'95, pages 115–123, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers, Inc.
- [43] Gregory Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.
- [44] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [45] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 2006.

- [46] Joseph Cruz and David Wishart. Applications of machine learning in cancer prediction and prognosis. *Cancer Informatics*, 2:59–77, 2006.
- [47] Silvia das Dôres, Carlos Soares, and Duncan Ruiz. Bandit-based automated machine learning. In *Proceedings of the Brazilian Conference on Intelligent Systems*, BRACIS’18, pages 121–126, New York, NY, USA, 2018. IEEE.
- [48] Alex de Sá and Gisele Pappa. Towards a method for automatically evolving bayesian network classifiers. In *Proceedings of the Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO’13 Companion, page 1505–1512, New York, NY, USA, 2013. ACM.
- [49] Alex de Sá and Gisele Pappa. A hyper-heuristic evolutionary algorithm for learning Bayesian network classifiers. In *Proceedings of the Ibero-American Conference on Artificial Intelligence*, IBERAMIA’14, pages 430–442, Cham, Switzerland, 2014. Springer.
- [50] Alex de Sá, Walter José Pinto, Luiz Otávio Oliveira, and Gisele Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. In *Proceedings of the European Conference on Genetic Programming*, EuroGP’17, pages 246–261, Cham, Switzerland, 2017. Springer.
- [51] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [52] Krzysztof Dembczyński, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the International Conference on Machine Learning*, ICML’10, pages 279–287, Madison, WI, USA, 2010. Omnipress.
- [53] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.
- [54] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- [55] Thomas Desautels, Andreas Krause, and Joel Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 15(1):3873–3923, 2014.
- [56] Bernard Dickman and Michael Gilman. Monte Carlo optimization. *Journal of Optimization Theory and Applications*, 60(1):149–157, 1989.

- [57] Laura Dioşan, Alexandrina Rogozan, and Jean-Pierre Pecuchet. Improving classification performance of support vector machine by genetically optimising kernel shape and hyper-parameters. *Applied Intelligence*, 36(2):280–294, 2012.
- [58] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- [59] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.
- [60] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings 1995*, pages 194–202. Elsevier, Amsterdam, The Netherlands, 1995.
- [61] Agoston Eiben and Cornelis Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.
- [62] Agoston Eiben and James Smith. *Introduction to evolutionary computing*, volume 53. Springer-Verlag, Berlin/Heidelberg, Germany, 2003.
- [63] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Proceedings of the International Conference on Neural Information Processing Systems*, NIPS’01, pages 681–687, Cambridge, MA, USA, 2001. MIT Press.
- [64] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search. In Hutter et al. [109], pages 69–86. Available at <http://automl.org/book>.
- [65] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.
- [66] Kutluhan Erol, James Hendler, and Dana Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, AIPS’94, pages 249–254, Palo Alto, CA, USA, 1994. AAAI Press.
- [67] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7:1079–1105, 2006.
- [68] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 2*, NIPS’90, pages 524–532, Cambridge, MA, USA, 1990. MIT Press.

- [69] Rong-En Fan and Chih-Jen Lin. A study on threshold selection for multi-label classification. Technical report, Department of Computer Science, National Taiwan University, Taipei City, Taiwan, 2007.
- [70] Usama Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'93*, pages 1022–1027, San Francisco, CA, USA, 1993. Morgan Kaufmann publishers.
- [71] Alan Ferrenberg and Robert Swendsen. Optimized Monte Carlo data analysis. *Computers in Physics*, 3(5):101–104, 1989.
- [72] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Practical automated machine learning for the AutoML challenge 2018. In *Proceedings of the International Workshop on Automatic Machine Learning, AutoML'18*, 2018.
- [73] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Hutter et al. [109], pages 3–38. Available at <http://automl.org/book>.
- [74] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Manuel Blum, and Frank Hutter. Methods for improving Bayesian optimization for AutoML. In *Proceedings of the International Workshop on Automatic Machine Learning, AutoML'15*, 2015.
- [75] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 2, NIPS'15*, pages 2755–2763, Red Hook, NY, USA, 2015. Curran Associates, Inc.
- [76] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI'15*, pages 1128–1135, Palo Alto, CA, USA, 2015. AAAI Press.
- [77] Ronald Fisher. *Statistical methods and scientific inference*. Hafner Publishing Co. Limited, UK, 2nd edition, 1956.
- [78] Eibe Frank and Remco Bouckaert. Naïve Bayes for text classification with unbalanced classes. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML-PKDD'06*, pages 503–510, Berlin/Heidelberg, Germany, 2006. Springer-Verlag.

- [79] Eibe Frank, Mark Hall, and Bernhard Pfahringer. Locally weighted naïve Bayes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 249–256, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers, Inc.
- [80] Eibe Frank and Ian Witten. Generating accurate rule sets without global optimization. In *Proceedings of the International Conference on Machine Learning*, ICML'98, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers, Inc.
- [81] Alex Freitas. A review of evolutionary algorithms for data mining. In *Data Mining and Knowledge Discovery Handbook*, pages 371–400. Springer, Boston, MA, USA, 2009.
- [82] Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning*, ICML'96, pages 148–156, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers, Inc.
- [83] Yoav Freund and Robert Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.
- [84] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.
- [85] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
- [86] Nicolo Fusi, Rishit Sheth, and Huseyn Melih Elibol. Probabilistic matrix factorization for automated machine learning. In *Proceedings of the International Conference on Neural Information Processing Systems*, NIPS'18, pages 3348–3357, Red Hook, NY, USA, 2018. Curran Associates, Inc.
- [87] Eva Gibaja and Sebastián Ventura. A tutorial on multilabel learning. *ACM Computing Surveys*, 47(3):52:1–52:38, 2015.
- [88] Pieter Gijsbers, Joaquin Vanschoren, and Randal Olson. Layered TPOT: Speeding up tree-based pipeline optimization. In *Proceedings of the International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms*, AutoML'17, pages 49–68. CEUR-WS.org, 2017.
- [89] Walter Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC Press, Boca Raton, FL, USA, 1995.
- [90] Christophe Giraud-Carrier and Foster Provost. Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper? In *Proceedings of the ICML Workshop on Meta-Learning*, pages 9–16, 2005.



- [91] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD'04*, pages 22–30, Berlin/Heidelberg, Germany, 2004. Springer-Verlag.
- [92] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google Vizier: A service for black-box optimization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'17*, pages 1487–1495, New York, NY, USA, 2017. ACM.
- [93] Taciana Gomes, Ricardo Prudêncio, Carlos Soares, André Rossi, and André de Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.
- [94] Joshua Griffin and Tamara Kolda. Nonlinearly constrained optimization using heuristic penalty methods and asynchronous parallel generating set search. *Applied Mathematics Research eXpress*, 2010(1):36–62, 2010.
- [95] Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI'11*, pages 1300–1305, Palo Alto, CA, USA, 2011. AAAI Press.
- [96] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. Analysis of the AutoML challenge series 2015-2018. In Hutter et al. [109], pages 191–236. Available at <http://automl.org/book>.
- [97] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11:10–18, 2009.
- [98] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
- [99] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In *Proceedings of the Conference on Advances in Neural Information Processing Systems, NIPS'97*, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.
- [100] David Heckerman, Dan Geiger, and David Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 85–96, Palo Alto, CA, USA, 1994. AAAI Press.

- [101] Francisco Herrera, Francisco Charte, Antonio Rivera, and Mara del Jesus. *Multilabel classification: Problem analysis, metrics and techniques*. Springer, AG, Switzerland, 1st edition, 2016.
- [102] Alireza Hesar, Hamid Tabatabaee, and Mehrdad Jalali. Structure learning of Bayesian networks using heuristic methods. In *Proceedings of International Conference on Information and Knowledge Management, ICIKM’12*, pages 246–250, Singapore, Singapore, 2012. IACSIT Press.
- [103] Geoffrey Hinton and Russ Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [104] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
- [105] John Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. The University of Michigan, Ann Arbor, MI, USA, 1975.
- [106] Robert Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.
- [107] Holger Hoos and Frank Hutter. Programming by optimization: A practical paradigm for computer-aided algorithm design, 2017. Tutorial at the International Joint Conference on Artificial Intelligence.
- [108] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization, LION’11*, pages 507–523, Berlin/Heidelberg, Germany, 2011. Springer-Verlag.
- [109] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated machine learning: Methods, systems, challenges*. Springer, New York, NY, USA, 2019. Available at <http://automl.org/book>.
- [110] Frank Hutter and Steve Ramage. *Manual for SMAC version v2.10.03-master*. Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, 2015.
- [111] Ronald Iman and James Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics*, 9(6):571–595, 1980.
- [112] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the International Conference on*

- Artificial Intelligence and Statistics*, AISTATS'16, pages 240–248. Proceedings of Machine Learning Research (PMLR), 2016.
- [113] Rodolphe Jenatton, Cedric Archambeau, Javier González, and Matthias Seeger. Bayesian optimization with tree-structured dependencies. In *Proceedings of the International Conference on Machine Learning*, ICML'17, pages 1655–1664. Proceedings of Machine Learning Research (PMLR), 2017.
- [114] Thorsten Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic publishers, Hingham, MA, USA, 2002.
- [115] George John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers, Inc.
- [116] Donald Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.
- [117] Donald Jones, Cary Perttunen, and Bruce Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.
- [118] Donald Jones, Matthias Schonlau, and William Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.
- [119] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel text classification for automated tag suggestion. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases: Discovery Challenge*, ECML-PKDD Discovery Challenge 2008, pages 75–83, 2008.
- [120] Sathiya Keerthi, Shirish Shevade, Chiranjib Bhattacharyya, and Krishna Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computing*, 13(3):637–649, 2001.
- [121] Scott Kirkpatrick, Daniel Gelatt, and Mario Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [122] Dragi Kocev. *Ensembles for predicting structured outputs*. PhD thesis, Department of Knowledge Technologies, Jozef Stefan Institute, Ljubljana, Slovenia, 2011.

- [123] Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. Ensembles of multi-objective decision trees. In *Proceedings of the European Conference on Machine Learning, ECML'07*, pages 624–631, Berlin/Heidelberg, Germany, 2007. Springer-Verlag.
- [124] Patrick Koch, Oleg Golovidov, Steven Gardner, Brett Wujek, Joshua Griffin, and Yan Xu. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'18*, pages 443–452, New York, NY, USA, 2018. ACM.
- [125] Ron Kohavi. The power of decision tables. In *Proceedings of the European Conference on Machine Learning, ECML'95*, pages 174–189, Berlin/Heidelberg, Germany, 1995. Springer-Verlag.
- [126] Sotiris Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real World AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, 2007. IOS Press.
- [127] Lars Kotthoff, Chris Thornton, Holger Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(1):826–830, 2017.
- [128] Lars Kotthoff, Chris Thornton, and Frank Hutter. User guide for Auto-WEKA (version 2.6). Technical report, Computer Science Department at University British Columbia, Vancouver, BC, Canada, 2017.
- [129] Konstantina Kourou, Themis Exarchos, Konstantinos Exarchos, Michalis Karamouzis, and Dimitrios Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8–17, 2015.
- [130] John Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.
- [131] Tomáš Křen, Martin Pilát, and Roman Neruda. Automatic creation of machine learning workflows with strongly typed genetic programming. *International Journal on Artificial Intelligence Tools*, 26(05):1760020:1–1760020:24, 2017.
- [132] Tomáš Křen, Martin Pilát, and Roman Neruda. Multi-objective evolution of machine learning workflows. In *Proceedings of the IEEE Symposium Series on Computational Intelligence, SSCI'17*, pages 1–8, New York, NY, USA, 2017. IEEE.

- 
- [133] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1):161–205, 2005.
- [134] David Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the European Conference on Machine Learning, ECML’1998*, pages 4–15, Berlin/Heidelberg, German, 1998. Springer-Verlag.
- [135] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [136] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104, 2012.
- [137] Samir Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, The University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1995.
- [138] Katherine Malan and Andries Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013.
- [139] Andrew McCallum and Kamal Nigam. A comparison of event models for naïve Bayes text classification. In *Proceedings of the Workshop on Learning for Text Categorization*, pages 41–48, Palo Alto, CA, USA, 1998. AAAI Press.
- [140] Michael McKay. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceedings of the Conference on Winter Simulation, WSC’92*, pages 557–564, New York, NY, USA, 1992. ACM.
- [141] Robert McKay, Nguyen Hoai, Peter Whigham, Yin Shan, and Michael O’Neill. Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11(3):365–396, 2010.
- [142] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Proceedings of the International Workshop on Automatic Machine Learning, AutoML’16*, pages 58–65, 2016.
- [143] Patricia Miquilini, Rafael Rossi, Marcos Quiles, Vinícius de Melo, and Márcio Basgalupp. Automatically design distance functions for graph-based semi-supervised learning. In *Proceedings of the IEEE International Conference on Big Data Science and Engineering, BigDataSE’17*, pages 933–940, New York, NY, USA, 2017. IEEE.

- [144] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107:1495–1515, 2018.
- [145] David Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
- [146] Jose Moyano, Eva Gibaja, Krzysztof Cios, and Sebastián Ventura. Review of ensembles of multi-label classifiers: Models, experimental study and prospects. *Information Fusion*, 44:33–45, 2018.
- [147] Jose Moyano, Eva Gibaja, Krzysztof Cios, and Sebastián Ventura. An evolutionary approach to build ensembles of multi-label classifiers. *Information Fusion*, 50:168–180, 2019.
- [148] Andreas Müller and Sarah Guido. *Introduction to machine learning with Python: A guide for data scientists*. O’Reilly Media, Inc., Sebastopol, CA, USA, 1st edition, 2017.
- [149] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.
- [150] John Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.
- [151] Vlad Niculae. polylearn: A library for factorization machines and polynomial networks for classification and regression in Python. <https://github.com/scikit-learn-contrib/polylearn>, 2016.
- [152] Peter Norvig. *Paradigms of artificial intelligence programming: Case studies in common LISP*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1992.
- [153] Randal Olson, Nathan Bartley, Ryan Urbanowicz, and Jason Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO’16*, pages 485–492, New York, NY, USA, 2016. ACM.
- [154] Randal Olson and Jason Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the International Workshop on Automatic Machine Learning, AutoML’16*, pages 66–74, 2016.
- [155] Randal Olson, Ryan Urbanowicz, Peter Andrews, Nicole Lavender, La Creis Kidd, and Jason Moore. Automating biomedical data science through tree-based pipeline optimization. In *Proceedings of the European Conference on the Applications of*

- Evolutionary Computation*, EvoApplications'16, pages 123–137, AG, Switzerland, 2016. Springer.
- [156] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.
- [157] Fernando Otero, Tom Castle, and Colin Johnson. EpochX: genetic programming in Java with statistics and event monitoring. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO Companion'12, pages 93–100, New York, NY, USA, 2012. ACM.
- [158] Arjun Pakrashi and Brian Mac Namee. CascadeML: An automatic neural network architecture evolution and training algorithm for multi-label classification. *arXiv*, arXiv:1904.10551:1–16, 2019.
- [159] Gisele Pappa and Alex Freitas. *Automating the design of data mining algorithms: An evolutionary computation approach*. Springer-Verlag, Berlin/Heidelberg, Germany, 2009.
- [160] Gisele Pappa, Gabriela Ochoa, Matthew Hyde, Alex Freitas, John Woodward, and Jerry Swan. Contrasting meta-learning and hyper-heuristic research: The role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35, 2014.
- [161] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [162] Rafael Pereira, Alexandre Plastino, Bianca Zadrozny, and Luiz Merschmann. Correlation analysis of performance measures for multi-label classification. *Information Processing and Management*, 54(3):359–369, 2018.
- [163] John Pestian, Chris Brew, Pawel Matykiewicz, Dj Hovermale, Neil Johnson, Kevin Cohen, and Wlodzislaw Duch. A shared task involving multi-label classification of clinical free text. In *Proceedings of the Workshop on BioNLP: Biological, Translational, and Clinical Language Processing*, BioNLP'07, pages 97–104, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
- [164] Erik Pitzer and Michael Affenzeller. A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, pages 161–191. Springer-Verlag, Berlin/Heidelberg, Germany, 2012.

- [165] John Platt. Advances in kernel methods. In Bernhard Schölkopf, Christopher Burges, and Alexander Smola, editors, *Advances in Kernel Methods*, chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.
- [166] John Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1993.
- [167] Carl Edward Rasmussen and Christopher Williams. *Gaussian processes for machine learning*. MIT Press, Cambridge, MA, USA, 2006.
- [168] Jesse Read. A pruned problem transformation method for multi-label classification. In *Proceedings of the New Zealand Computer Science Research Student Conference, NZCSRS'08*, pages 143–150, Christchurch, Canterbury, New Zealand, 2008. The University of Canterbury.
- [169] Jesse Read. *Scalable multi-label classification*. PhD thesis, The University of Waikato, Hamilton, New Zealand, 2010.
- [170] Jesse Read, Luca Martino, and David Luengo. Efficient Monte Carlo optimization for multi-label classifier chains. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP'13*, pages 3457–3461, New York, NY, USA, 2013. IEEE.
- [171] Jesse Read, Luca Martino, and David Luengo. Efficient Monte Carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3):1535–1546, 2014.
- [172] Jesse Read, Luca Martino, Pablo Olmos, and David Luengo. Scalable multi-output label prediction: From classifier chains to classifier trellises. *Pattern Recognition*, 48(6):2096–2109, 2015.
- [173] Jesse Read and Fernando Perez-Cruz. Deep learning for multi-label classification. *ArXiv*, arXiv:1502.05988:1–8, 2014.
- [174] Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Multi-label classification using ensembles of pruned sets. In *Proceedings of the International Conference on Data Mining*, pages 995–1000, New York, NY, USA, 2008. IEEE.
- [175] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.
- [176] Jesse Read, Peter Reutemann, Bernhard Pfahringer, and Geoff Holmes. MEKA: A multi-label/multi-target extension to WEKA. *Journal of Machine Learning Research*, 17(21):1–5, 2016.



- [177] Elaine Rich. *Artificial intelligence*. McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1983.
- [178] Elaine Rich and Kevin Knight. *Artificial intelligence*. McGraw-Hill, Inc., New York, NY, USA, 2nd edition, 1990.
- [179] Adriano Rivolli and André de Carvalho. O uso seletivo de classificadores binários na solução de problemas multirrótulos. In *Proceedings of the National Meeting on Artificial and Computational Intelligence*, ENIAC'15, pages 1–9, Porto Alegre, Brazil, 2015. Brazilian Computer Society.
- [180] Adriano Rivolli and Andre de Carvalho. The utiml package: Multi-label classification in R. *The R Journal*, 10(2):24–37, 2018.
- [181] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [182] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the International Conference on Machine Learning*, ICML'08, pages 880–887, New York, NY, USA, 2008. ACM.
- [183] Steven Salzberg. Book Review: C4.5: Programs for machine learning by J. Ross Quinlan, Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, 16(3):235–240, 1994.
- [184] Aécio Santos, Sonia Castelo, Cristian Felix, Jorge Piazzentin Ono, Bowen Yu, Sungsoo Ray Hong, Cláudio Silva, Enrico Bertini, and Juliana Freire. Visus: An interactive system for automatic machine learning model building and curation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA'19, pages 6:1–6:7, New York, NY, USA, 2019. ACM.
- [185] Cullen Schaffer. A conservation law for generalization performance. In *Proceedings of the International Conference on International Conference on Machine Learning*, ICML'94, pages 259–265, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers, Inc.
- [186] Robert Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
- [187] Matthias Schonlau, William Welch, and Donald Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series, New Developments and Applications in Experimental Design*, 34:11–25, 1998.

- [188] Eric Scott and Kenneth De Jong. Evaluation-time bias in quasi-generational and steady-state asynchronous evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO'16, pages 845–852, New York, NY, USA, 2016. ACM.
- [189] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery*, ECML-PKDD'11, pages 145–158, Berlin/Heidelberg, Germany, 2011. Springer-Verlag.
- [190] Huan Shao, GuoZheng Li, GuoPing Liu, and YiQin Wang. Symptom selection for multi-label data of inquiry diagnosis in traditional Chinese medicine. *Science China Information Sciences*, 56(5):1–13, 2013.
- [191] Eric Siegel. *Predictive analytics: The power to predict who will click, buy, lie, or die*. John Wiley & Sons, Inc., Hoboken, New Jersey, USA, 2nd edition, 2016.
- [192] Michael Sipser. *Introduction to the theory of computation*. Cengage Learning, Inc., Boston, MA, USA, 3rd edition, 2012.
- [193] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, Red Hook, NY, USA, 2012. Curran Associates, Inc.
- [194] Andrew Sohn, Randal Olson, and Jason Moore. Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO'17, pages 489–496, New York, NY, USA, 2017. ACM.
- [195] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the International Conference on Machine Learning*, ICML'10, pages 1015–1022, Madison, WI, USA, 2010. Omnipress.
- [196] Marc Sumner, Eibe Frank, and Mark Hall. Speeding up logistic model tree induction. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECML-PKDD'05, pages 675–683, Berlin/Heidelberg, Germany, 2005. Springer-Verlag.
- [197] Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. ATM: A distributed, collaborative, scalable system for automated machine learning. In *Proceedings of the International Conference on Big Data*, Big Data'17, pages 151–162, New York, NY, USA, 2017. IEEE.

- [198] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael Osborne. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. *arXiv*, arXiv:1409.4011:1–6, 2014.
- [199] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers, Inc.
- [200] Chris Thornton, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. AutoWEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD’13, pages 847–855, New York, NY, USA, 2013. ACM.
- [201] Konstantinos Trohidis, Grigorios Tsoumakas, George Kalliris, and Ioannis Vlahavas. Multi-label classification of music into emotions. In *International Conference on Music Information Retrieval*, volume 8 of *ISMIR’08*, pages 325–330, 2008.
- [202] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal on Data Warehousing and Mining*, 3(3):1–13, 2007.
- [203] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of the ECML-PKDD Workshop on Mining Multidimensional Data*, MMD’08, pages 30–44, 2008.
- [204] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. *Mining multi-label data*, pages 667–685. Springer, Boston, MA, USA, 2010.
- [205] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, 2011.
- [206] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of the European Conference on Machine Learning*, ECML’07, pages 406–417, Berlin/Heidelberg, Germany, 2007. Springer-Verlag.
- [207] Bülent Üstün, Willem Melssen, and Lutgarde Buydens. Facilitating the application of support vector regression by using a universal Pearson VII function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1):29–40, 2006.
- [208] Joaquin Vanschoren. Meta-learning. In Hutter et al. [109], pages 39–68. Available at <http://automl.org/book>.

- [209] Hua Wang, Heng Huang, and Chris Ding. Function-function correlated multi-label protein function prediction over interaction networks. In *Proceedings of the Annual International Conference on Research in Computational Molecular Biology*, RECOMB'12, pages 302–313, Berlin/Heidelberg, Germany, 2012. Springer-Verlag.
- [210] Tinghua Wang, Dongyan Zhao, and Shengfeng Tian. An overview of kernel alignment and its applications. *Artificial Intelligence Review*, 43(2):179–192, 2015.
- [211] Marcel Wever, Felix Mohr, Alexander Hetzer, and Eyke Hüllermeier. Automating multi-label classification extending ML-Plan. In *Proceedings of the International Workshop on Automatic Machine Learning*, AutoML'19, 2019.
- [212] Marcel Wever, Felix Mohr, and Eyke Hüllermeier. ML-Plan for unlimited-length machine learning pipelines. In *Proceedings of the International Workshop on Automatic Machine Learning*, AutoML'18, 2018.
- [213] Jonathan Wexler, Susan Haller, and Radhikha Myneni. An overview of SAS<sup>®</sup> visual data mining and machine learning on SAS<sup>®</sup> Viya. In *SAS Global Forum Conference*, Cary, NC, USA, 2017. SAS Institute, Inc.
- [214] Peter Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Rochester, NY, USA, 1995. The University of Rochester.
- [215] Peter Whittle. Multi-armed bandits and the Gittins index. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):143–149, 1980.
- [216] Ian Witten, Eibe Frank, Mark Hall, and Christopher Pal. *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 4th edition, 2016.
- [217] David Wolpert and William Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [218] David Wolpert and William Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, 2005.
- [219] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [220] Mohammed Zaki and Wagner Meira Jr. *Data mining and analysis: Fundamental concepts and algorithms*. Cambridge University Press, Cambridge, UK, 2nd edition, 2020.

- 
- [221] Julio Zaragoza, Luis Sucar, Eduardo Morales, Concha Bielza, and Pedro Larrañaga. Bayesian chain classifiers for multidimensional classification. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'11*, pages 2192–2197, Palo Alto, CA, USA, 2011. AAAI Press.
- [222] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv*, arXiv:1807.06906:1–11, 2018.
- [223] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: An overview. *Frontiers of Computer Science: Selected Publications from Chinese Universities*, 12(2):191–202, 2018.
- [224] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Proceedings of the IEEE International Conference on Granular Computing*, pages 718–721, New York, NY, USA, 2005. IEEE.
- [225] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.
- [226] Min-Ling Zhang and Zhi-Hua Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.
- [227] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014.

# Appendix A

## Multi-Label Classification Search Space in the MEKA Software

This appendix describes the Multi-Label Classification (MLC) search space in the MEKA software, including the traditional and meta MLC algorithms and the traditional, meta, and preprocessing Single-Label Classification (SLC) algorithms. The SLC search space is also studied because it is part of the MLC search space, as several methods use problem transformation methods to create a solution (i.e., a classifier) for an MLC problem. This was done to understand better the MLC algorithms.

### A.1 Search Space – Algorithms from WEKA

In this section, we study **22** traditional (single label) classification algorithms from the WEKA software [97]. This is done in order to understand the whole search space of multi-label methods. All parameters in this section were set in accordance with the search space definition from Auto-WEKA [127, 128, 200]. The methods and their respective (hyper-)parameters were defined after studying the code, logs, and configuration files of Auto-WEKA, which is considered a stable and robust approach for automatically selecting and configuring machine learning algorithms.

#### A.1.1 C4.5

The method for generating a C4.5 decision tree [166]. This algorithm can decide whether it will use the default C4.5's error-based pruning method [22, 183, 216] or not. If the algorithm decides to use pruning, the C4.5's pruning method is applied to the tree,

and an estimation of the error rate of every subtree is done. After that, the pruning method will replace the subtree with a leaf node if the estimated error of the leaf is lower than a threshold [183]. **Parameters:**

- Confidence factor (*cf*)[-C]: It is used for C4.5's error-based pruning method (smaller values incur more pruning) and is defined by the interval:  
 $\{cf \in \mathbb{R} \mid 0.0 \leq cf \leq 1.0\}$ .  
**Default value:** 0.25.
- Minimum number of objects (*mno*)[-M]: The minimum number of instances per leaf. It can take values in the interval:  
 $\{mno \in \mathbb{Z} \mid 1 \leq mno \leq 64\}$ .  
**Default value:** 2.
- Collapse tree (*ct*)[-O]: It is used to decide if internal nodes will be collapsed to avoid overfitting. This parameter is used with C4.5's error-based pruning method to enhance the final decision tree. It collapses a subtree to a node only if the training error of the subtree does not increase when compared to the entire tree. It is applied to every subtree in the tree, where subtrees are collapsed (pruned) if pruning does not increase its classification error. For example, if there is a subtree with two leaf nodes having the same classification on the training data, this subtree will be replaced by a single leaf. It can take Boolean values (true or false).  
**Default value:** true.
- Unpruned (*u*)[-U]: It decides whether pruning is performed or not. It can take Boolean values (true or false).  
**Default value:** false.
- Binary splits (*bs*)[-B]: It decides whether C4.5 will use binary splits on nominal attributes when building the trees. It can take Boolean values (true or false).  
**Default value:** false.
- Use MDL correction (*umc*)[-J]: It decides whether the MDL correction is used when finding splits on numeric attributes. It can take Boolean values (true or false).  
**Default value:** true.
- Use Laplace (*ul*)[-A]: It decides if the counts of instances at leaves are smoothed based on the Laplace correction. It can take Boolean values (true or false).  
**Default value:** false.
- Subtree raising (*sr*)[-S]: It is used for C4.5's error-based pruning and decides whether the algorithm will consider the subtree raising operation when pruning. It can take

Boolean values (true or false).

**Default value:** true.

**Dependencies/Constraints:**

1. If the parameter `unpruned` is set to “true”, the parameters “confidence factor”, “collapse tree” and “subset raising” are not used (omitted).

## A.1.2 Logistic Model Trees

The method for building Logistic Model Trees (LMT) [133, 196], which are classification trees with logistic regression functions at the leaves. This is done by using the LogitBoost algorithm. In this case, boosting is used (aiming) to build very effective decision trees. The idea of LMT is to use LogitBoost to induce trees with linear-logistic regression models at the leaves. LogitBoost performs additive logistic regression. Thus, at each iteration of the boosting algorithm, it creates a simple regression model by going through all the attributes, finding the simple regression function with the smallest error, and adding it into the additive model [216]. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes, and missing values.

**Parameters:**

- Minimum number of objects (`mno`)[-M]: The minimum number of instances per leaf. It can take values in the interval:  
 $\{mno \in \mathbb{Z} \mid 1 \leq mno \leq 64\}$ .

**Default value:** 15.

- Convert Nominal (`cn`)[-B]: It decides if the method will convert all nominal attributes to binary ones before building the tree. This means that all splits in the final tree will be binary. It can take Boolean values (true or false).

**Default value:** false.

- Split on residuals (`sor`)[-R]: It decides whether the method will set the splitting criterion based on the residuals of LogitBoost. There are two possible splitting criteria for LMT: the default is to use the C4.5 splitting criterion that uses information gain on the class variable. The other splitting criterion tries to improve the purity in the residuals produced when fitting the logistic regression functions. It can take Boolean values (true or false).

**Default value:** false.



- Fast Regression (*fr*)[-C]: It decides whether the method will use a heuristic that avoids the use of cross-validation to optimize the number of Logit-Boost iterations at every node. In the case of using this heuristic, LMT will fit the logistic regression functions at a leaf node using the LogitBoost algorithm, applying a 5-fold cross-validation procedure to determine how many iterations to run just once. Then, it employs the same number of iterations throughout the tree, instead of cross-validating at every node. This heuristic reduces the running time considerably, with little effect on accuracy [216]. It can take Boolean values (true or false).

**Default value:** true.

- Error on probabilities (*eop*)[-P]: It decides if the method will minimize the error on classification probabilities instead of the misclassification error when cross-validating the number of LogitBoost iterations. When this parameter is set to ‘true’, the number of LogitBoost iterations that minimizes the error on classification probabilities instead of the misclassification error is chosen. It can take Boolean values (true or false).

**Default value:** false.

- Weight trim beta(*wtb*)[-W]: It sets the beta value used for weight trimming in LogitBoost. Only instances carrying  $(1 - \text{beta})\%$  of the weight from the previous iteration are used in the next iteration. The value zero (0) means no weight trimming, which is the default value. The values are restricted to the interval :  $\{wtb \in \mathbb{R} \mid 0.0 \leq wtb \leq 1.0\}$ .

**Default value:** 0.0.

- Use AIC (*uaic*)[-A]: It decides if the method will use the AIC (Akaike’s Information Criterion) measure to determine when to stop LogitBoost’s iterative process. More precisely, if *uaic* takes the value ‘true’, the best number of iterations will be defined by an information criterion measure (currently, AIC). If false, the stopping criterion will be determined by the best number of iterations in a 5-fold cross-validation procedure. It can take Boolean values (true or false).

**Default value:** false.

There are no dependencies/constraints between the parameters of LMT.

### A.1.3 Decision Stump

The method for building and applying a Decision Stump (DS) model [216], which is considered a weak learner. Because of that, it is usually used in conjunction with a boosting algorithm.

The DS's classification is based on the entropy measure, and a missing value is treated as a separate value. The DS algorithm constructs a simple decision tree that has only one level, i.e., a decision tree that has only one internal (root) node, which is directly linked to the leaves. It also creates an extra branch for missing values.

In the case of nominal attributes at the root node, there are two possibilities. The first possibility is to build a stump that contains a leaf for each possible feature value. The second possibility is to consider a stump with two leaves; one of them is mapped to some category, and the other to all other categories. The DS from WEKA employs the latter approach. This method has no explicit parameters.

### A.1.4 Random Forest

The Random Forest (RF) method for constructing a forest of random trees [25].

#### Parameters:

- Number of trees ( $nt$ )[-I]: The number of trees to be generated by the algorithm. It is an integer value bounded by the interval:  $\{nt \in \mathbb{Z} \mid 2 \leq nt \leq 256\}$ .

**Default value:** 100.

- Number of features ( $nf$ )[-K]: It sets the number of randomly sampled attributes used as candidate attributes at each tree node. It is an integer value bounded by the interval:  $\{nf \in \mathbb{Z} \mid 2 \leq nf \leq 32\}$ . However, it may also take the value zero (0), which means  $nf$  will be just used as a flag to indicate that the real value produced by the equation  $\log_2(\text{number\_of\_attributes} + 1)$  rounded to the nearest integer is automatically used for this parameter.

**Default value:** 0.

- Maximum depth ( $md$ )[-depth]: The maximum depth of the tree. It is bounded by the interval:  $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$ . However, it may also take the value zero (0) as a flag, and, in this case, the depth of the tree can be unlimited.

**Default value:** 0.

There are no dependencies/constraints between the parameters of RF.

### A.1.5 Random Tree

The method for constructing a tree that considers  $K$  randomly sampled attributes as candidate attributes at each node, i.e., a Random Tree (RTree) [216]. It is important to mention that this version of RT performs no pruning. **Parameters:**

- Minimum weight ( $mw$ )[-M]: The minimum total weight of the instances in a leaf. It is restricted by the interval:  $\{mw \in \mathbb{Z} \mid 1 \leq mw \leq 64\}$ .  
**Default value:** 1.
- Number of features ( $nf$ )[-K]: It sets the number of randomly sampled attributes used as candidate attributes at each tree node. It is an integer value bounded by the interval:  $\{nf \in \mathbb{Z} \mid 2 \leq nf \leq 32\}$ . However, it may also take the value zero (0), which means  $nf$  will be just used as a flag to indicate that the real value produced by the equation  $\log_2(\text{number\_of\_attributes} + 1)$  rounded to the nearest integer is automatically used for this parameter.  
**Default value:** 0.
- Maximum depth ( $md$ )[-depth]: The maximum depth of the tree. It is bounded by the interval:  $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$ . However, it may also take the value zero (0) as a flag, and, in this case, the depth of the tree can be unlimited.  
**Default value:** 0.
- Number of folds for back-fitting and for growing the tree ( $nfbgt$ )[-N]: It determines the amount of data used for back-fitting and for growing the tree. One fold is used for back-fitting, i.e., for making a preliminary estimation of class probabilities based on a hold-out set. The other ( $nf - 1$ ) folds are used for growing the tree. It is bounded by the interval:  $\{nfbf \in \mathbb{Z} \mid 2 \leq nfbf \leq 5\}$ . It can also use the value zero (0), which means no back-fitting will be performed in this case. It can not take the value one (1) because we would have zero folds for growing the tree. In the case of taking the value one, the algorithm returns an error and does not run. It is important to mention that Auto-WEKA allows this error, ignoring the RT algorithm with this configuration (when it occurs) and continuing the search from this point.  
**Default value:** 0.

There are no constraints/dependencies between the parameters of RT.

### A.1.6 REPTree

The method for the fast decision tree learner, which is well-known as REPTree [216]. It builds a decision tree using information gain and prunes it using reduced-error pruning (with back-fitting). It only sorts values for numeric attributes once at the start of the algorithm. Missing values are dealt with by splitting the corresponding instances into pieces (i.e., as in C4.5). **Parameters:**

- Minimum weight ( $mw$ )[-M]: The minimum total weight of the instances in a leaf. It is restricted by the interval:  
 $\{mw \in \mathbb{Z} \mid 1 \leq mw \leq 64\}$ .  
**Default value:** 2.
- Maximum depth ( $md$ )[-L]: The maximum tree depth. It can take integer values considering the interval:  $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$ . However, it may also take the value  $-1$  as a flag, and, in this case, the depth of the tree the depth will not be restricted.  
**Default value:** -1.
- Use pruning ( $up$ )[-P]: It decides whether REPTree will use reduced-error pruning or not. In the case of using this pruning method, a simple hold-out set ( $\frac{1}{3}$  of the training data) is used to estimate the error of a node, instead of using cross-validation. It can take Boolean values (true or false).  
**Default value:** false.

There are no dependencies/constraints between the parameters of REPTree.

### A.1.7 Decision Table

The method for building and using a simple Decision Table (DT) classifier [125].  
**Parameters:**

- Evaluation Measure ( $em$ )[-E]: The measure used to evaluate the performance of attribute combinations used in the decision table. It can take one of the four cat-

egorical values: 1. accuracy (*acc*); 2. root mean squared error (*rmse*) of the class probabilities; 3. mean absolute error (*mae*) of the class probabilities; 4. area under the ROC curve (*auc*). The two measures *rmse* and *mae* are adapted to be used in the classification context.

**Default value:** *acc*.

- Use IBk (*uibk*)[-I]: It sets whether a k-nearest neighbor (k=1) classifier should be used instead of the majority class in order to classify non-matching instances. It can take Boolean values (true or false).

**Default value:** false.

- Search method (*sm*)[-S]: It sets the search method that will be used to find good attribute combinations for the decision table. It can take the values Greedy Stepwise or Best First.

**Default value:** Best First.

- Cross-Validation (*crv*)[-X]: It sets the number of folds for the internal cross-validation procedure to evaluate the attribute sets. It may take the values one (1), two (2), three (3), or four (4). If the value one (1) is set for this hyper-parameter, a leave-one-out procedure is applied.

**Default value:** 1.

There are no dependencies/constraints between the parameters of DT.

### A.1.8 JRip

The method that implements a propositional rule learner algorithm, namely Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [42]. **Parameters:**

- Minimum total weight (*mtw*)[-N]: This parameter determines the minimum total weight of the instances in a rule. It can take values considering the interval:  $\{mtw \in \mathbb{R} \mid 1.0 \leq mtw \leq 5.0\}$ .

**Default value:** 2.0.

- Check error rate (*cer*)[-E]: It decides whether JRip will consider the “error rate greater or equal than 0.5” as a stopping criterion. It can take Boolean values (true or false).

**Default value:** true.

- Use pruning (*up*)[-P]: It decides whether JRip will use reduced error pruning or not. In the case of using this pruning method, a 3-fold cross-validation procedure is applied to prune the rules. Otherwise, no pruning method is used. It can take Boolean values (true or false).

**Default value:** false.

- Optimizations (*o*)[-O]: The number of optimization runs. It can take integer values considering the interval:  $\{o \in \mathbb{Z} \mid 1 \leq o \leq 5\}$ .

**Default value:** 2.

There are no dependencies/constraints between the parameters of JRip.

### A.1.9 One Rule

The method for building and using a One Rule (OneR) classifier [106]. In other words, it uses the minimum-error attribute for prediction, discretizing numeric attributes.

**Parameters:**

- Minimum bucket size (*mbs*)[-B]: It is used for discretizing numeric attributes. It is limited by the interval:  $\{mbs \in \mathbb{Z} \mid 1 \leq mbs \leq 32\}$ .

**Default value:** 6.

OneR has only one parameter, and consequently, it has no dependencies/constraints.

### A.1.10 PART

The method for generating a PART decision list [80]. PART uses the separate-and-conquer paradigm: It builds a partial C4.5 decision tree in each iteration and makes the “best” leaf into a rule. **Parameters:**

- Minimum number of objects (*mno*)[-M]: The minimum number of instances per leaf. It can take values in the interval:

$\{mno \in \mathbb{Z} \mid 1 \leq mno \leq 64\}$ .

**Default value:** 2.

- Binary splits (*bs*)[-B]: It decides whether C4.5 will use binary splits on nominal attributes when building the trees. It can take Boolean values (true or false).

**Default value:** false.

- Reduced-error pruning(*rep*)[-R]: It is used to decide whether reduced-error pruning is used instead of C4.5's default pruning (error-based pruning). If C4.5's error-based pruning is chosen, a (default) confidence factor of 0.25 is used to prune the tree. If not (i.e., the reduced-error pruning is chosen), the method will consider each node for pruning, and the removal of a subtree at a node is done if the resulting tree performs no worse than the original one on the validation set. The size of the validation set is determined by the next parameter (*nr*). It can take Boolean values (true or false).

**Default value:** true.

- Number of folds (*nr*)[-N]: It determines the amount of data used for reduced-error pruning. One fold is used for pruning, and the rest for growing the tree. It can take the values two (2), three (3), four (4) or five (5).

**Default value:** not used.

#### **Dependencies/Constraints:**

1. If the reduced-error pruning method is not set to “true”, the parameter “number of folds” is not used.

### **A.1.11 Zero Rule**

The method for building and using a Zero Rule (ZeroR) classifier [216]. The ZeroR classifier simply predicts the majority category (class), ignoring the predictor attributes. This method has no explicit parameters.

### **A.1.12 K-Nearest Neighbors**

The method for K-Nearest Neighbors (KNN) classifier [3]. KNN can select an appropriate value of K based on internal leave-one-out evaluation and can also compute distances based on instance weighting. **Parameters:**

- Number of neighbors ( $k$ )[-K]: The number of neighbors to use. The value of  $k$  is bounded by the interval:  $\{k \in \mathbb{Z} \mid 1 \leq k \leq 64\}$ .

**Default value:** 1.

- Leave-one-out ( $loo$ )[-X]: It decides whether leave-one-out evaluation on the training data will be used or not to select the best  $k$  value between 1 and the value specified as the KNN parameter. If set as false, the selected  $k$  value is used. It can take only Boolean values (true or false).

**Default value:** false.

- Distance weighting ( $dw$ ): It sets the used distance weighting method. It may take the unique following values:

- -I: Weight neighbors by the inverse of their distance.
- -F: Weight neighbors by one minus their distance.
- None: No distance weighting method is applied.

**Default value:** None.

There are no dependencies/constraints between the parameters of KNN.

### A.1.13 $K^*$

The method  $K^*$  is an instance-based classification algorithm [41]. Thus, in order to classify a test instance,  $K^*$  considers the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners by using an entropy-based distance function. **Parameters:**

- Global blending ( $gb$ )[-B]: The parameter is a percentage for global blending. This parameter controls the “sphere of influence” by specifying how many of the neighbors of the instance  $i$  should be considered important (although there is no hard cut-off at the edge of the sphere – it is more related to a gradual decreasing of importance). The values are restricted to the interval  $\{gb \in \mathbb{Z} \mid 1 \leq gb \leq 100\}$ . Thus, selecting zero (0) for this parameter gives the nearest neighbor algorithm (this is why AutoWEKA does not allow one to choose it), and choosing 100 gives equally weighted instances. Intermediate values are interpolated linearly.

**Default value:** 20.



- Entropic auto-blending (*ea*b)[-E]: It decides whether entropy-based blending will be used or not. It can take Boolean values (true or false).

**Default value:** false.

- Missing Mode (*mm*)[-M]: It determines how missing attribute values are treated. It can take one of the four categorical values: 1. average column entropy curves (a); 2. ignore the instances with missing values (d); 3. treat missing values as maximally different (m); 4. normalize over the attributes (n).

**Default value:** a.

There are no dependencies/constraints between the parameters of K\*.

### A.1.14 Voted Perceptron

The Voted Perceptron (VP) algorithm was created by Freund and Schapire [83]. It globally replaces all missing values with their default values. More precisely, VP replaces all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. Additionally, it transforms nominal attributes into binary ones. **Parameters:**

- Number of iterations (*i*)[-I]: The number of iterations to be performed by VP. This parameter varies in accordance to the interval:

$$\{i \in \mathbb{Z} \mid 1 \leq i \leq 10\}.$$

**Default value:** 1.

- Max K(*mk*)[-M]: The maximum number of alterations to the perceptron, i.e., the maximum number of perceptrons used in the iterative process. It can take values of the interval:

$$\{mk \in \mathbb{Z} \mid 5,000 \leq mk \leq 50,000\}$$

**Default value:** 1,000.

- Exponent (*e*)[-E]: The exponent for the polynomial kernel. It can take values of the interval:  $\{e \in \mathbb{R} \mid 0.2 \leq e \leq 5.0\}$

**Default value:** 1.0.

There are no dependencies/constraints between the parameters of VP.

### A.1.15 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) uses the traditional and well-known back-propagation algorithm [181] to create a neural model to classify the instances. MLP creates just one hidden layer (for now), and all its nodes use sigmoid activation functions (except for when the class is numeric, in which case the output nodes become unthresholded linear units). **Parameters:**

- Learning rate (*lr*)[-L]: The amount by which the weights are updated during training. It is restricted by the interval:

$$\{lr \in \mathbb{R} \mid 0.1 \leq lr \leq 1.0\}.$$

**Default value:** 0.3.

- Momentum (*m*)[-M]: It is applied to the weights during updating. It is restricted by the interval:  $\{m \in \mathbb{R} \mid 0.0 \leq m \leq 1.0\}$ .

**Default value:** 0.2.

- Number of hidden nodes (*nhn*)[-H]: It defines the number of hidden nodes in the hidden layer of the neural network. This parameter may take four predefined nominal values (a, i, o, and t), which represent the following integer values:

- a =  $\frac{(\text{number\_of\_attributes} + \text{number\_of\_classes})}{2}$ , always using the default floor function to convert it to an integer value.

- i = number of attributes.

- o = number of classes.

- t =  $(\text{number\_of\_attributes} + \text{number\_of\_classes})$ .

**Default value:** a.

- Nominal to binary filter (*n2b*)[-B]: It decides whether the algorithm will transform nominal attributes to binary ones or not. This could help improve performance if there are nominal attributes in the data. It can take Boolean values (true or false).

**Default value:** true.

- Reset (*r*)[-R]: It decides whether the algorithm will use the reset approach. In this case, the algorithm will allow the network to reset with a lower learning rate. If the network diverges from the answer, this will automatically reset the network with a lower learning rate and begin training again. It can take Boolean values (true or false).

**Default value:** true.

- Decay (*d*)[-D]: It decides whether the algorithm will cause the learning rate to decrease. This will divide the starting value of the learning rate by the sequential number of the current epoch in order to determine what the current learning rate should be. This may help to stop the network from diverging from the target output, as well as improving general performance. It can take Boolean values (true or false).  
**Default value:** false.

There are no dependencies/constraints between the parameters of MLP.

### A.1.16 Stochastic Gradient Descent

The method that implements the well-known Stochastic Gradient Descent (SGD) approach [216] for learning various linear models (binary class SVM, binary class logistic regression, squared loss, Huber loss, and epsilon-insensitive loss linear regression).

**Parameters:**

- Loss function (*lf*)[-F]: It sets the loss function to be minimized. It can take the following integer values associated with three approaches:
  - (0): hinge loss (SVM).
  - (1): log loss (logistic regression).
  - (2): squared loss (regression).**Default value:** 0.
- Learning rate (*lr*)[-L]: The learning rate. If normalization is turned off, then the default learning rate will need to be reduced. It is restricted by the interval:  $\{lr \in \mathbb{R} \mid 0.00001 \leq lr \leq 1.0\}$ .  
**Default value:** 0.01.
- Ridge (*r*)[-R]: It sets the Ridge value in the log-likelihood. This parameter can take any value of the given set:  
 $\{r \in \mathbb{R} \mid 10^{-12} \leq r \leq 10.0\}$   
**Default value:** 0.0001
- Do not normalize (*nn*)[-N]: It decides whether normalization will be turned off or not. It can take Boolean values (true or false).  
**Default value:** false.

- Do not replace missing values (*nrmv*)[-M]: It decides whether the global replacement of missing values will be turned off or not. In the case of being turned off, the missing values will be ignored. Otherwise, SGD will replace all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. It can take Boolean values (true or false).

**Default value:** false.

There are no dependencies/constraints between the parameters of SGD.

### A.1.17 Sequential Minimal Optimization

This method implements John Platt's Sequential Minimal Optimization (SMO) algorithm for training a support vector classifier (SVC) [99, 120, 165]. It globally replaces all missing values with their default values. More precisely, SMO (like VP) replaces all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. Additionally, it transforms nominal attributes into binary ones. **Parameters:**

- Cost (*c*)[-C]: It defines the complexity parameter, which is the penalty parameter of the error term and is defined by the interval:  $\{c \in \mathbb{R} \mid 0.5 \leq c \leq 1.5\}$ . This is a parameter that controls the trade-off between training error and model complexity. It is important to mention that a low value of  $c$  will increase the number of training errors, whereas a high value of  $c$  will lead to a behavior similar to that of a hard-margin SVM [114].

**Default value:** 1.0.

- Filter type (*ft*)[-N]: It determines how/if the data will be transformed. It may take the values zero (0, i.e., normalize the training data – it sets all the numeric attributes in the given dataset into the interval  $[0,1]$ ), one (1, i.e., standardize the training data – it standardizes all numeric attributes in the given dataset to have zero mean and unit variance) or two (2, i.e., no normalization/standardization is applied to the data).

**Default value:** 0.

- Build Calibration Models (*bcm*)[-M]: It decides whether the model will fit calibration models to SVM's outputs (for proper probability estimates). It can take Boolean values (true or false).

**Default value:** false.

- Kernel( $k$ )[-K]: The kernel to use. It can take one of the following possible kernels (and associated constrained parameters):
  - PolyKernel: The standard polynomial kernel. **Parameters:**
    1. Exponent ( $exp$ )[-E]: It determines the exponent value and is defined by the interval:  $\{exp \in \mathbb{R} \mid 0.2 \leq exp \leq 5.0\}$ .
    2. Use Lower-Order ( $ulo$ )[-L]: It decides whether the method will use lower-order terms or not. It can take Boolean values (true or false).
  - NormalizedPolyKernel: The normalized polynomial kernel. **Parameters:**
    1. Exponent ( $exp$ )[-E]: It determines the exponent value and is defined by the interval:  $\{exp \in \mathbb{R} \mid 0.2 \leq exp \leq 5.0\}$ .
    2. Use Lower-Order ( $ulo$ )[-L]: It decides whether the method will use lower-order terms or not. It can take Boolean values (true or false).
  - Puk: The Pearson VII function-based universal kernel [207]. **Parameters:**
    1. Omega ( $om$ )[-O]: The omega value. It is defined by the interval:  $\{om \in \mathbb{R} \mid 0.1 \leq om \leq 1.0\}$ .
    2. Sigma ( $sig$ )[-S]: The sigma value. It is defined by the interval:  $\{sig \in \mathbb{R} \mid 0.1 \leq sig \leq 10.0\}$ .
  - RBF: The RBF kernel. **Parameters:**
    1. Gamma ( $g$ )[-G]: The gamma value. It is defined by the interval:  $\{g \in \mathbb{R} \mid 0.0001 \leq g \leq 1.0\}$ .

**Default value:** PolyKernel with ‘Exponent’ equals to 1.0 and ‘Use Lower-Order’ equals to true.

The constraints/dependencies for SMO are only in the selection of the kernel and its respective parameters.

### A.1.18 Logistic Regression

Method for building and using a multinomial Logistic Regression (LogR) model with a ridge estimator [32]. **Parameters:**

- Ridge ( $r$ )[-R]: It sets the Ridge value in the log-likelihood. This parameter can take any value of the given set:  
 $\{r \in \mathbb{R} \mid 10^{-12} \leq r \leq 10.0\}$   
**Default value:** 0.00000001

LogR has one parameter, and consequently, it has no dependencies/constraints.

### A.1.19 Simple Logistic

The method for constructing Simple Logistic (SL) regression models [133, 196]. LogitBoost, with simple regression functions as base learners, is used for fitting the logistic models. **Parameters:**

- Weight trim beta (*wtb*)[-W]: It sets the beta value used for weight trimming in LogitBoost. Only instances carrying  $(1 - \text{beta})\%$  of the weight from the previous iteration are used in the next iteration. The value zero (0) means no weight trimming, which is the default value. The values are restricted to the interval :  $\{wtb \in \mathbb{R} \mid 0.0 \leq wtb \leq 1.0\}$ . It also can be omitted and take the default value of zero.

**Default value:** 0.0.

- Use Cross-Validation (*ucv*)[-S]: It decides if SL will try to find the best number of LogitBoost iterations using an internal 5-fold cross-validation procedure or simply using the number of iterations that minimizes error on the training set. Thus, if not set to 'true', the number of LogitBoost iterations which is used is the one that minimizes the error on the training set (misclassification error). It can take Boolean values (true or false).

**Default value:** true.

- Use AIC (*uaic*)[-A]: It decides if the method will use the AIC (Akaike's Information Criterion) measure to determine when to stop the LogitBoost iterative process. More precisely, if *uaic* takes the value 'true', the best number of iterations will be defined by an information criterion measure (currently, AIC). If false, the stopping criterion will be determined by the best number of iterations in an internal 5-fold cross-validation procedure or simply in accordance to the error on the training set, as explained in the previous item. It can take Boolean values (true or false).

**Default value:** false.

There are no dependencies/constraints between the parameters of SL.

### A.1.20 Naïve Bayes

The Naïve Bayes (NB) classifier using estimator classes [115]. This algorithm builds a fixed structure (model) given the attributes of the dataset. **Parameters:**

- Use kernel estimator (*uke*)[-K]: It decides whether NB will use a kernel estimator for numeric attributes rather than a (single) Gaussian distribution. In the case of using the kernel estimator, NB will apply one Gaussian kernel per observed data value (for more details, see Flexible Naïve Bayes' section in [115]). It can take Boolean values (true or false). It is important to mention that a discrete estimator is automatically used for nominal attributes, which is a simple discrete probability estimator based on nominal values' counts. This also means the Laplace correction is applied in order to perform the estimation.

**Default value:** false.

- Use supervised distribution (*usd*)[-D]: It decides whether NB will use supervised discretization to convert numeric attributes to nominal ones. Discretization is performed by [70]'s method. This method uses a criterion based on the Minimum Description Length (MDL) principle to define the number of intervals produced over the continuous space [60]. It can take Boolean values (true or false).

**Default value:** false.

#### Dependencies/Constraints:

1. If the parameter “use kernel estimator” is activated, the parameter “use supervised distribution” must not be activated, and vice-versa. This constraint must be enforced in the grammar.

### A.1.21 Bayesian Network Classifier

This method is used to learn a Bayesian Network Classifier (BNC) [20] based on various search algorithms and a local Bayesian scoring metric [43, 100]. **Parameters:**

- Search Method (*sm*)[-Q]: For BNC algorithms, the optimization occurs just on the method used for searching network structures. Thus, the search method can be one of the following: 1. Tree Augmented Naïve Bayes (TAN) [85]; 2. K2 [43]; 3. Hill Climbing (HC) [19, 102]; 4. Look Ahead in Good Directions Hill Climbing

(LAGDHC) [1]; 5. Simulated Annealing (SA) [19]; 6. Tabu Search (TS) [19]. All the search methods use the parameter “maximum number of parents” set to two (including the class node), except for TAN and SA, which do not use the “maximum number of parents” as a parameter. In addition, the methods use the (default) Bayesian scoring metric to search for appropriate Bayesian networks for data.

**Default value:** there is no default value for this hyper-parameter because all these search methods are important algorithms in the literature. Therefore, we use all search methods in our space.

BNC has one parameter, and consequently, it has no dependencies/constraints.

### A.1.22 Naïve Bayes Multinomial

The method for building and using Naïve Bayes Multinomial (NBM) [2, 78, 134, 139, 216]. This algorithm was particularly designed for text classification, and for this reason, it changes how the traditional Naïve Bayes calculates the probabilities. This is done to take into account the number of times a word appears in the document.

This method has no explicit parameters, and consequently, it has no dependencies/constraints.

## A.2 Search Space – Meta Classification Algorithms from WEKA

In this section, the search space of **5** traditional (single label) meta-classification algorithms from WEKA [97] is studied. This is also done in order to extend and improve the search space of multi-label methods. All parameters in this section were also set in accordance with the search space definition from Auto-WEKA [200, 127, 128]. The methods and their respective (hyper-)parameters were defined after studying the code, logs, and configuration files of Auto-WEKA, which is considered a stable and robust approach for automatically selecting and configuring machine learning algorithms.



### A.2.1 Locally Weighted Learning

The Locally Weighted Learning (LWL) method [8, 79]. It uses an instance-based algorithm to assign instance weights, which are then used by a specified Weighted Instances Handler. In other words, LWL assigns weights using an instance-based method, and after this step, another classification algorithm is used to build a classifier from the weighted instances. For example, it can do the classification by using a naïve Bayes classifier or a decision stump (default) from these weighted instances. **Parameters:**

- Classifier ( $c$ )[-W]: The classifier to be used. It can be one classification algorithm from Section A.1, except for the algorithms LMT, OneR, K\*, SGD, and VP, as the classifiers produced by these algorithms do not handle weighted instances.
- Number of neighbors ( $k$ )[-K]: It sets how many neighbors are used to determine the width of the weighting function. It may take the following values:  $\{-1, 10, 30, 60, 90, 120\}$ . A negative value means that all neighbors will be considered.

**Default value:** -1.

- Weighting kernel ( $wk$ )[-U]: It determines the weighting function and may take the five following integer values:
  - (0) Linear.
  - (1) Epnechnikov.
  - (2) Tricube.
  - (3) Inverse.
  - (4) Gaussian.

It can be omitted with 50% of probability, and then LWL will use the default value zero for this parameter, i.e., the linear function. The five (5) not omitted values jointly take the other 50% of probability, which represents at the end that the value 0 has 60% of probability to be chosen.

**Default value:** 0.

There are no dependencies/constraints between the parameters of LWL.

### A.2.2 Random Subspace

This method constructs an ensemble classifier that consists of multiple models systematically constructed by randomly selecting subsets of components of the feature vector, i.e., the classification models are constructed according to Random Subspaces (RSS) [104]. More precisely, for each classifier, a certain percentage of the number of attributes is randomly sampled and then used to build the classifier. **Parameters:**

- Classifier ( $c$ )[-W]: The classifier to be used. It can be any classification algorithm from Section A.1.
- Subspace size ( $sss$ )[-P]: It defines the size of each sub-space as a percentage of the number of attributes. It could take values in the range:  $\{sss \in \mathbb{R} \mid 0.1 \leq sss \leq 1.0\}$ .  
**Default value:** 0.5.
- Number of iterations ( $ni$ )[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range:  $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 64\}$ .  
**Default value:** 10.

There are no dependencies/constraints between the parameters of RSS.

### A.2.3 Bagging of Single-Label Classifiers

The method for bagging a classifier in order to reduce variance [24]. **Parameters:**

- Classifier ( $c$ )[-W]: The classifier to be used for each member of the ensemble. It can be any classification algorithm from Section A.1.
- Bag size percent ( $bsp$ )[-P]: It defines the size of each bag, as a percentage of the training set size. It may take values in the range:  $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$ . It makes sampling with replacement. Thus, even if the bag size percent is 100%, it will sample different sets with the same size of the training set.  
**Default value:** 100.
- Number of iterations ( $ni$ )[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range:  $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 128\}$ .  
**Default value:** 10.

- Calculate out-of-bag (*coob*)[-O]: It decides whether the out-of-bag error is calculated. It can take Boolean values (true or false).

**Default value:** false.

**Dependencies/Constraints:**

1. If the parameter “calculate out-of-bag” is activated (set to true), the parameter “bag size percent” must be equal to 100. This is a constraint of WEKA, and only an internal modification in the WEKA code could suppress it. This can happen with 50% of probability. I.e., in half of the cases, the parameter “bag size percent” is set to 100. In the other part of the cases, “bag size percent” may take values between 10 and 100, because the parameter “calculate out-of-bag” is not activated (set to false).

## A.2.4 Random Committee

The method for building an ensemble of randomizable base classifiers from the WEKA software [216], creating a Random Committee (RC) of classifiers. For this reason, the only classifiers (at the base level) that can be used in this meta-algorithm are Random Forest (RF), Random Tree (RT), REP Tree (REPTree), Stochastic Gradient Descent (SGD) and Multilayer Perceptron (MLP). The creation of a randomizable classifier is done by using an input (pseudo-random) seed. It is important to mention that the classifiers in the ensemble differ in terms of the structure of their models. For instance, a random seed can define how the random trees are constructed in RF, RT, and REPTree, how the linear models are defined in SGD, and how the network connection weights are first defined in MLP. Nevertheless, all classifiers are constructed using the same data, differently from Bagging and RSS. Thus, in the end, the final prediction is based on the average of the class probabilities generated by the base classifiers. **Parameters:**

- Classifier (*c*)[-W]: The classifier to be used for each member of the ensemble. It is restricted to one of the five (5) aforementioned algorithms, i.e., RF, RT, REPTree, SGD, and MLP.
- Number of iterations (*ni*)[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range:  $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 64\}$ .

**Default value:** 10.

There are no dependencies/constraints between the parameters of RC.

### A.2.5 Ada Boost M1

The method for boosting a nominal class classifier using the Adaboost M1 (AdaM1) approach [82]. **Parameters:**

- Classifier ( $c$ )[-W]: The classifier to be used. It can be one classification algorithm from Section A.1, except for the algorithms LMT, OneR,  $K^*$ , SGD, and VP, as the classifiers produced by these algorithms do not handle weighted instances. It is important to mention that Auto-WEKA allows any classifier at the base level of AdaM1, including those that can not handle weights in the instances. In this case, Auto-WEKA ignores that algorithm (with its configuration) and proceeds with the search.
- Weight threshold ( $wt$ )[-P]: It defines the weight threshold for weighted pruning, i.e., it only selects instances with weights that contribute to the specified quantile of the weight distribution. It may take values in the range:  
 $\{wt \in \mathbb{Z} \mid 50 \leq wt \leq 100\}$ .  
**Default value:** 100.
- Number of iterations ( $ni$ )[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take the values in the range:  
 $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 128\}$ .  
**Default value:** 10.
- Use resampling ( $ur$ )[-Q]: It decides whether AdaM1 will use resampling instead of reweighting. Thus, it is possible to generate an unweighted dataset from the weighted data by resampling. In this case, instances are chosen with probability proportional to their weight. As a result, instances with high weight are replicated frequently, and the ones with low weight may never be selected. Once the new dataset becomes as large as the original one, it is fed into the learning approach instead of the weighted data [216]. It can take Boolean values (true or false).  
**Default value:** false.

There are no dependencies/constraints between the parameters of AdaM1.

## A.3 Search Space – Preprocessing Algorithms from WEKA

In this section, the search space of (single label) preprocessing classification algorithms from WEKA [97] is studied. This is also done in order to extend and improve the search space of multi-label methods. Instead of using just a single-label classification (SLC) algorithm at the SLC base level, a wrapper containing preprocessing methods is first used and, just after that, SLC is performed.

### A.3.1 Attribute Selection Classifier

The method reduces the dimensionality of training and test data by performing attribute selection (using the training set only) before the data is set as input to a classifier [216], constructing an Attribute Selection Classifier (ASC). **Parameters:**

- Classifier (*c*)[-W]: The classifier to be used. It can be any classification algorithm from Section A.1.
- Search method (*sm*)[-S]: The search method for selecting the attribute subset to be used as input by the classifier. It may take two values:
  1. Best First: It searches the space of attribute subsets by greedy hill-climbing augmented with a backtracking facility.
  2. Greedy Stepwise: It performs a greedy forward search through the space of attribute subsets.

Both methods use the evaluator “CfsSubsetEval”, which evaluates the worth of a subset of attributes by considering the individual predictive ability of each attribute, along with the degree of redundancy between them. Hence, ASC is conceptually equivalent to using the CFS (Correlation-based Feature Selection) attribute selection method followed by the use of the chosen classifier with the attributes selected by CFS.

**Default value:** Best First.

There are no dependencies/constraints between the parameters of ASC.

## A.4 Studying the Search Space of Multi-Label Classification Algorithms

We studied **26** multi-label and meta multi-label classification algorithms from the MEKA software [176], which are described in the following two sections. It is important to mention that most algorithms in (this version of) MEKA could define a **threshold** to perform the classification using the model’s confidence outputs (typically, class probabilities). For the general multi-label context, it is in general better to optimize the threshold than simply using an arbitrary threshold of 0.5 [69, 175]. This parameter (*pred\_tshd*) [*-threshold*] could take the following values:

- Proportional Cut method by Instance (PCut1) [169]: It takes into account the label cardinality of the dataset, which is simply the average number of labels associated with each instance of this dataset. Thus, PCut1 automatically calibrates the prediction confidence threshold, by minimizing the difference between the label cardinality of the training set and the label cardinality obtained with a given set of predicted labels – where the latter set is determined by the threshold value. This does not require access to the true predictions in the test set.
- Proportional Cut method by Label (PCutL): It is used to calibrate the prediction confidence threshold the same way as PCut1, but for each label individually.
- The threshold could also take a unique real value between zero (0.0) and one (1.0) for all instances being classified. Formally, the threshold can also be defined by the following interval:  $\{threshold \in \mathbb{R} \mid 0.0 < threshold < 1.0\}$ .

**Default value:** PCut1.

## A.5 Search Space – Multi-Label Classification Algorithms

### A.5.1 Binary Relevance

The standard Binary Relevance (BR) method [204]. It creates a binary classification problem for each label and learns a model for each label individually. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in BR.

### A.5.2 The ‘Quick’ Version of Binary Relevance

The Quick BR (BRq) [175] is a version of BR that is able to downsample the number of training instances across the binary models. It is intended for use in an ensemble (but it also works in a standalone fashion).

**Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Down-sample ratio (*dsr*)[-P]: It is a ratio used to reduce the number of instances across the binary models. Low values mean more removals and high values mean fewer removals, as BRq uses the following formula  $(1 - dsr) * number\_of\_instances$  to calculate the number of instances to remove. This parameter is constrained by the interval:

$$\{dsr \in \mathbb{R} \mid 0.2 \leq dsr \leq 0.8\}.$$

There is no explanation about this parameter in the original paper and in other papers in the multi-label classification literature. The justification – for the used interval – is that we would like to have (at least) 20% of the instances from the original data to construct the model (otherwise, the method may not have sufficient instances to build the model). Additionally, we would like to have (at most) 80% of the instances from the original data in order to learn the classifier (otherwise, the method would be very similar to BR).

**Default value:** 0.75.

There are no dependencies/constraints between the parameters of BRq.

### A.5.3 Classifier Chain

The Classifier Chain (CC) method [175] is also similar to BR, but the label outputs predicted by a classifier become new inputs for the next classifiers in the chain. It uses a single random order of labels in the chain.

**Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in CC.

### A.5.4 The ‘Quick’ Version of Classifier Chains

The Quick CC (CCq) [175] is a version of CC that is able to down-sample the number of training instances across the binary models. It is also intended for use in an ensemble (but it also works in a standalone fashion). **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Down-sample ratio (*dsr*)[-P]: It is a ratio used to reduce the number of instances across the binary models. Low values mean more removals, and high values mean fewer removals, as CCq uses the following formula  $(1 - dsr) * number\_of\_instances$  to calculate the number of instances to remove. This parameter is constrained by the interval:

$$\{dsr \in \mathbb{R} \mid 0.2 \leq dsr \leq 0.8\}.$$

There is no explanation about this parameter in the original paper and in other papers in the multi-label classification literature. The justification – for the used interval – is that we would like to have (at least) 20% of the instances from the original data to construct the model (otherwise, the method may not have sufficient instances to build the model). Additionally, we would like to have (at most) 80% of the instances from the original data in order to learn the classifier (otherwise, the method would be very similar to CC).

**Default value:** 0.75.



There are no dependencies/constraints between the parameters of CCq.

### A.5.5 Bayesian Classifier Chain

The Bayesian classifier chain (BCC) method [221] creates a maximum spanning tree based on marginal dependencies, defines a Bayesian network from it, and then employs a classifier chain (CC) using the order of the labels found in the Bayesian network model. The original paper used Naïve Bayes as a base classifier, but other types of classifiers can be used. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Dependency type (*dp*)[-X]: The way to measure and find the dependencies. It may take ten categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf (Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. No label dependence; 10. L (The “LEAD” method for finding conditional dependence).

**Default value:** Ibf.

There are no dependencies/constraints between the parameters of BCC.

### A.5.6 (Bayes Optimal) Probabilistic Classifier Chain

The Probabilistic Classifier Chain (PCC) method [52] acts exactly like CC at training time, but explores all possible paths as inference at test time (hence, “Bayes optimal”). **Parameters:**

- Base classifier (*bc*)[-W]: It can be any classifier from WEKA.

**Dependencies/Constraints:**

1. PCC has poor scalability, i.e., it is very slow when the number of labels is greater than a certain threshold. In the PCC’s original paper [52], it is said that this threshold should be 15 labels. In the future, we might consider it to scale the

proposed solution to evolve a multi-label learning algorithm. For instance, we must impose a constraint in the grammar that specifies the use of PCC only if the number of labels is less than 15. We can also specify a time budget for all MLC algorithms, depending on the size of the dataset. This will consequently limit the effectiveness of the PCC algorithm in more complex types of data.

### A.5.7 Monte-Carlo Classifier Chains

The methods based on Monte-Carlo Classifier Chains (MCC and M2CC) [170, 171], apply classifier chains with Monte Carlo optimization, using a maximum number of inference and chain-order trials. MCC has a tractable label prediction scheme only at the test time (MCC), whereas M2CC performs an additional search for the optimal chain sequence at the training time. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Inference Iterations (*ii*)[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval:  $\{ii \in \mathbb{Z} \mid 1 < ii \leq 100\}$ .

**Default value:** 10.

- Chain Iterations (*chi*)[-Is]: The number of iterations to search the chain space at training time. This parameter is bounded by the values in the interval:  $\{chi \in \mathbb{Z} \mid 1 < chi \leq 1500\}$ . It can also take the value zero and the MCC algorithm is used instead of M2CC. This will happen with 50% of probability, i.e., MCC and M2CC have the same chances of being selected.

**Default value:** 0.

- Payoff function (*pof*)[-P]: It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss, limited by the number of labels; 13. Log loss limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18.  $F_1$  micro averaged; 19.  $F_1$  macro averaged by example; 20.  $F1$  macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23. Levenshtein distance.

**Default value:** Exact match.

There are no dependencies/constraints between the parameters in MCC and M2CC. Additionally, we studied the range of the parameters in the works of Read et al. [170, 171]. However, the authors did not employ proper parameter tuning at the single-label level nor at the multi-label level. In the work of Read et al. [171], a search is performed to find the proper number of chain iterations in accordance with the payoff function. We are using part of this study to define the range of the parameters.

### A.5.8 Population of Monte-Carlo Classifier Chains

The Population of Monte-Carlo Classifier Chains (PMCC) [170, 171] is a method that has similar properties when compared to MCC and M2CC. However, it is considered an extension of both methods. The difference is that PMCC creates a population of  $M$  chains at training time (from  $Is$  candidate chains, using Monte Carlo sampling) and uses all of them at test time. This is not a typical majority-vote ensemble method. The simulated annealing search [121] can also be applied to the chain structures (produced by MCC or M2CC) in order to find the best one.

#### Parameters:

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Inference Iterations ( $ii$ )[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval:  $\{ii \in \mathbb{Z} \mid 1 < ii \leq 100\}$ .

**Default value:** 10.

- Chain Iterations ( $chi$ )[-Is]: The number of iterations to search the chain space at training time. This parameter is bounded by the values in the interval:  $\{chi \in \mathbb{Z} \mid 50 < chi \leq 1500\}$ .

**Default value:** 50.

- Beta ( $\beta$ )[-B]: It sets the factor with which the temperature (and thus the acceptance probability of steps in the wrong direction in the search space) is decreased in each iteration of the simulated annealing search. This parameter is bounded by the interval:  $\{\beta \in \mathbb{Z} \mid 0.01 \leq \beta \leq 0.99\}$ .

**Default value:** 0.03.

- Temperature switch ( $ts$ )[-O]: It sets the use of simulated annealing search and, when it is activated, it cools the chain down over time (from the beginning of the chain). It may take the values zero (0) or one (1). The value zero (0) means that no

temperature is used, i.e., the parameter  $\beta$  is ignored internally by PMCC. If using  $ts = 1$ , this sets the use of the  $\beta$  constant.

**Default value:** 0.

- Population size ( $ps$ )[-M]: It sets the population size. It should be always smaller than the total number of chains evaluated ( $Is$ ). This parameter takes one of the values defined by the following interval:

$$\{ps \in \mathbb{Z} \mid 1 \leq ps \leq 50\}.$$

**Default value:** 10.

- Payoff function ( $pof$ )[-P]: It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss limited by the number of labels; 13. Log loss, limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18.  $F_1$  micro averaged; 19.  $F_1$  macro averaged by example; 20.  $F1$  macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23. Levenshtein distance.

**Default value:** Exact match.

**Dependencies/Constraints:**

1. The parameter “population size” must be smaller than the parameter “chain iterations”.

Again, we studied the range of the parameters in the works of Read et al. [171]. However, the authors did not employ proper parameter tuning at the single-label level nor at the multi-label level. During the work of Read et al. [171], a search is performed to find the proper number of chain iterations in accordance with the payoff function. We are using part of this study to define the range of the parameters. Nevertheless, parameters  $\beta$ , temperature, and population size are not properly studied for the multi-label scenario.

### A.5.9 Classifier Trellis

The Classifier Trellis (CT) method [172] builds classifier chains in a trellis structure (rather than a cascaded chain). It is possible to set the width and type/connectivity of the trellis and optionally to change the payoff function, which guides the placement of nodes (labels) within the trellis. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Width (*w*)[-H]: it determines the width of the trellis (0 for chain, i.e.,  $w = L$ ; -1 for a square trellis, i.e.,  $w = \sqrt{L}$ , always using the default floor function to convert it to an integer value). Thus, the trellis structure will always have  $w$  rows and  $L$  nodes, in total, connected using directed edges.  
**Default value:** -1.
- Dependency type (*dp*)[-X]: The way to measure and find the label dependencies. It may take nine categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf ( Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. No label dependence.  
**Default value:** Ibf.
- Inference Iterations (*ii*)[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval:  $\{ii \in \mathbb{Z} \mid 1 \leq ii \leq 100\}$ .  
**Default value:** 10.
- Chain Iterations (*chi*)[-Is]: The number of iterations to search the chain space at train time. This parameter is bounded by the values in the interval:  $\{chi \in \mathbb{Z} \mid 1 < chi \leq 1500\}$ .  
**Default value:** 0.
- Density (*d*)[-L]: It determines the neighborhood density (the number of neighbors for each node in the trellis). The default value for the density parameter is one (1), and zero (0) indicates a BR classifier. Thus, this parameter is not allowed to take the value zero, being restricted by the interval:  $\{d \in \mathbb{Z} \mid 1 \leq d \leq \sqrt{L} + 1\}$ , where  $L$  is the total number of labels.  
**Default value:** 1.
- Payoff function (*pof*)[-P]: It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss limited by the number of labels; 13. Log loss, limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18.  $F_1$  micro averaged; 19.  $F_1$  macro averaged by example; 20.  $F1$  macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23.

Levenshtein distance.

**Default value:** Exact match.

**Dependencies/Constraints:**

1. If the width  $w = L$  ( $w = 0$ ), the density  $d = 1$ . Otherwise, if  $w = \sqrt{L}$  ( $w = -1$ ), the density  $d$  should be  $\sqrt{L} + 1$ , at most, i.e.,  $d \leq \sqrt{L} + 1$ .

### A.5.10 Conditional Dependency Networks

The Conditional Dependency Networks (CDN) method [95] builds a fully connected undirected network, where each node (label) is connected to each other node (label). Each node is a binary classifier that predicts  $p(y_j|x, y_1, \dots, y_{j-1}, \dots, y_L)$ . Then, inference is done using the Gibbs Sampling method over  $I$  iterations. Additionally, the final  $I_c$  iterations are used to collect the marginal probabilities, which become the prediction ( $y[]$ ).

**Parameters:**

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Iterations ( $i$ )[-I]: The total number of iterations to perform in CDT. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 100 < i \leq 1000\}$ .

**Default value:** 1000.

- Collection iterations ( $ci$ )[-Ic] The number of collection iterations used to compute the output class probabilities in the Gibbs Sampling method. The parameter  $ci$  is restricted by the interval:  $\{ci \in \mathbb{Z} \mid 1 \leq ci \leq 100\}$ .

**Default value:** 100.

**Dependencies/Constraints:**

1. The collections will happen just after  $(i - ci)$  iterations. So,  $i$  should be substantially greater than  $ci$  in order to make the algorithm work properly.

### A.5.11 Conditional Dependency Trellis

The Conditional Dependency Trellis (CDT) method [95, 172] is similar to the CDN approach. However, it constructs a trellis structure (like CT) instead of a fully connected

network. **Parameters:**

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Width ( $w$ )[-H]: it determines the width of the trellis (0 for chain, i.e.,  $w = L$ ; -1 for a square trellis, i.e.,  $w = \sqrt{L}$ , always using the default floor function to convert it to an integer value). Thus, the trellis structure will always have  $w$  rows and  $L$  nodes, in total, connected using directed edges.

**Default value:** -1.

- Dependency type ( $dp$ )[-X]: The way to measure and find the label dependencies. It may take nine categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf ( Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. None (Using empty).

**Default value:** None.

- Density ( $d$ )[-L]: It determines the neighborhood density (the number of neighbors for each node in the trellis). The default value for the density parameter is one (1), and zero (0) indicates a BR classifier. Thus, this parameter is not allowed to take the value zero, being restricted by the interval:  $\{d \in \mathbb{Z} \mid 1 \leq d \leq \sqrt{L} + 1\}$ , where  $L$  is the total number of labels.

**Default value:** 1.

- Iterations ( $i$ )[-I]: The total number of iterations to perform in CDT. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 100 < i \leq 1000\}$ .

**Default value:** 1000.

- Collection iterations ( $ci$ )[-Ic] The number of collection iterations used to compute the output class probabilities in the Gibbs Sampling method. The parameter  $ci$  is restricted by the interval:  $\{ci \in \mathbb{Z} \mid 1 \leq ci \leq 100\}$ .

**Default value:** 100.

#### **Dependencies/Constraints:**

1. If the width  $w = L$  ( $w = 0$ ), the density  $d = 1$ . Otherwise, if  $w = \sqrt{L}$  ( $w = -1$ ), the density  $d$  should be  $\sqrt{L} + 1$ , at most, i.e.,  $d \leq \sqrt{L} + 1$ .
2. The collections will happen just after  $(i - ci)$  iterations. So,  $i$  should be substantially greater than  $ci$  in order to make the algorithm work properly.

### A.5.12 Four-Class Pairwise Classification

The Four-class PairWise Classification (FW) method [176] trains a multi-class base classifier for each pair of labels. Thus, the number of classifiers is  $\frac{L*(L-1)}{2}$  in total (where L is the number of labels), each one with four possible class values (00,01,10,11) representing the possible combinations of relevant (1)/irrelevant (0) values for each label in the label pair. It uses a voting and a threshold scheme at testing time where, e.g., 01 from pair  $jk$  gives one vote to label  $k$ , and any label with a number of votes above the threshold is considered relevant. It uses the same threshold specified in Section A.4 to define the relevance of a label. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in FW.

### A.5.13 Ranking and Threshold

The Ranking and Threshold (RT) method [169] duplicates each multi-labeled example and assigns one of the labels (only) to each copy. After that, it trains a regular multi-class base classifier. At test time, a threshold separates relevant from irrelevant labels using the posterior probability for each class value (i.e., label). It uses the same threshold specified in Section A.4 to define the relevance of a label. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in RT.

### A.5.14 Label Combination

The Label Combination (LC) method [204], also known as Label Powerset (LP), treats each label combination as a single class in a multi-class learning scheme. The set of possible values of each class is the powerset of the set of labels.

- Base classifier (bc)[-W] : It can be any classifier from WEKA.



There are no dependencies/constraints in LC.

### A.5.15 Pruned Sets

The Pruned Sets (PS) method [168, 169] was created to use the power of LC's labelset-based paradigm without the disadvantages of such method. In order to do this, this algorithm has two important steps: a pruning step and a label-set subsampling step. The pruning step removes infrequently occurring label sets from the training data. This removes unnecessary complexity from the LC-transformed data by reducing the number of labelsets. Nevertheless, PS does not simply discard the pruned examples. Instead of doing that, PS subsamples the labelsets of these examples for label subsets, which occur more frequently in the training data. It then attaches these label sets to the example, creating new examples and reintroducing them into the training. It subsamples these labelsets  $pv$  times to produce  $pv$  new examples, where  $pv$  is the pruning value (defined in the following items).

After these steps, it trains a standard LC classifier. The idea of the method is to reduce the number of unique class values that would otherwise need to be learned by LC. PS achieves its best performance when used in an Ensemble (e.g., EnsembleML).

#### Parameters:

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Pruning value ( $pv$ )[-P]: It defines an infrequent labelset as one which occurs less than  $p$  times in the data.  $p = 0$  would mean that the LC classifier is learned. Thus, this parameter is bounded by the following interval:  $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$ .

**Default value:** 0.

- Subsampling value ( $sv$ )[-N]: The label set of each pruned example (in accordance with the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The PS method subsamples the label sets of pruned examples to create examples that do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:  $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$ .

**Default value:** 0.

There are no dependencies/constraints between the parameters of PS. Additionally, there is a proper study in the work of Read [169] about the range of these two parameters.

### A.5.16 Pruned Sets with Threshold

The Pruned Sets method with a Threshold (PSt) [168, 169, 174], which is a modification of PS that can form new label sets at classification (i.e., test) time by using a threshold function. Given the posterior of the label classes (combinations) and the number of labels, it returns the distribution across labels. Using the threshold (defined in Section A.4) could make the method to predict labelsets not seen in the training set, differently from PS. **Parameters:**

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Pruning value ( $pv$ )[-P]: It defines an infrequent labelset as one which occurs less than  $p$  times in the data.  $p = 0$  would mean that the LC classifier is learned. Thus, this parameter is bounded by the following interval:  $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$ .

**Default value:** 0.

- Subsampling value ( $sv$ )[-N]: The label set of each pruned example (in accordance with the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The PSt method subsamples the label sets of pruned examples to create examples that do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:  $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$ .

**Default value:** 0.

There are no dependencies/constraints between the parameters of PSt. Additionally, there is a proper study in the work of Read [169] about the range of these two parameters ( $pv$  and  $sv$ ). The parameters are the same as PS. The main thing that is changed in PSt when compared to PS occurs at the test time.

### A.5.17 Random k-Label Pruned Sets

The RANdom k-labEL Pruned Sets (RAkEL) method [169, 206, 205] randomly draws  $M$  subsets of labels, each with  $k$  labels, from the set of labels, and trains PS upon each one. Finally, it combines label votes from the PS classifiers to get a label-vector prediction. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.
- Pruning value (*pv*)[-P]: It prunes an infrequent labelset when it occurs less than *pv* times in the data.  $pv = 0$  means that the LC classifier is learned. Thus, this value is not allowed for RAKEL, which makes this parameter bounded by the following interval:  
 $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$ .  
**Default value:** 0.
- Subsampling value (*sv*)[-N]: The label set of each pruned example (in accordance with the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. This version of RAKEL in MEKA subsamples the label sets of pruned examples to create examples which do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:  $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$ .  
**Default value:** 0.
- Number of labels for each subset (*les*)[-k]: It defines the number of labels in each label subset. This parameter should be bounded by the interval [136]:  
 $\{les \in \mathbb{Z} \mid 1 \leq les \leq \frac{L}{2}\}$ , where  $L$  is the number of labels.  
**Default value:** 3.
- Number of subsets to run in an ensemble (*sre*)[-M]: This parameter controls the number of models to build in an ensemble and take values in accordance to the following interval [136]:  
 $\{sre \in \mathbb{Z} \mid 2 \leq sre \leq \min(2 \cdot L, 100)\}$ , where  $L$  is the number of labels.  
**Default value:** 10.

There are no dependencies/constraints between the parameters of RAKEL. Additionally, we followed the work of Read [169] about the range of the subsampling and pruning values. The other two parameters (number of labels in each subset and number of models to build in an ensemble) were defined in accordance with the work of Madjarov et al. [136].

### A.5.18 Random k-Label Disjoint Pruned Sets

The RAndom k-labEL Disjoint Pruned Sets (RAkELd) method [169, 206, 205] takes a random partition of labels, but unlike RAKEL, the labelsets are disjoint/non-

overlapping subsets. **Parameters:**

- Base classifier ( $bc$ )[-W] : It can be any classifier from WEKA.
- Pruning value ( $pv$ )[-P]: It prunes an infrequent labelset when it occurs less than  $p$  times in the data.  $pv = 0$  means that the LC classifier is learned. Thus, this value is not allowed for RAKEL, which makes this parameter bounded by the following interval:  
 $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$ .

**Default value:** 0.

- Subsampling value ( $sv$ )[-N]: The label set of each pruned example (in accordance with the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The version of RAKEd in MEKA subsamples the label sets of pruned examples to create examples that do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:  $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$ .

**Default value:** 0.

- Number of subsets to run in an ensemble ( $sre$ )[-M]: This parameter controls the number of models to build in an ensemble and take values in accordance to the following interval [136]:  
 $\{sre \in \mathbb{Z} \mid 2 \leq sre \leq \min(2 \cdot L, 100)\}$ , where  $L$  is the number of labels.

**Default value:** 10.

There are no dependencies/constraints between the parameters of RAKELd. Additionally, we followed the work of Read [169] again to set the range of the subsampling and pruning values. The other two parameters (number of labels in each subset and number of models to build in an ensemble) were defined in accordance with the work of Madjarov et al. [136].

### A.5.19 Multi-Label Back-Propagation Neural Network

The Multi-Label Back-Propagation Neural Network (ML-BPNN) method [173, 225] is a standard Back-Propagation Neural Network [181] with multiple outputs that correspond to multiple labels. That is, each node in the output layer corresponds to a different class label. **Parameters:**

- Number of epochs ( $ne$ )[-E]: It is the number of iterations to train the neural network. It is restricted by the interval:  $\{ne \in \mathbb{Z} \mid 10 \leq ne \leq 1000\}$ .  
**Default value:** 100.
- Number of hidden units ( $nhu$ )[-H]: It defines the number of hidden units in the neural network. It is important to mention that the version of ML-BPNN in MEKA is limited to one hidden layer with  $nhu$  hidden units. This parameter takes values in proportion to the number of attributes (received as input). Thus, the number of hidden units of the network can vary from 20% to 100% of the number of attributes:  $\{nhu \in \mathbb{Z} \mid 0.2 \cdot \text{number\_of\_attributes} \leq nhu \leq \text{number\_of\_attributes}\}$ . The proportion will always be rounded to the nearest integer.  
**Default value:** 10.
- Learning rate ( $lr$ )[-r]: The amount by which the weights are updated during training. It is restricted by the interval:  
 $\{lr \in \mathbb{R} \mid 0.001 \leq lr \leq 0.1\}$ .  
**Default value:** 0.1.
- Momentum ( $m$ )[-m]: It is applied to the weights during updating. It is restricted by the interval:  $\{m \in \mathbb{R} \mid 0.1 \leq m \leq 0.8\}$ .  
**Default value:** 0.1.

There are no dependencies/constraints between the parameters of ML-BPNN. Additionally, the range of values for the parameters number of epochs, momentum, and learning rate were set following the work of Read and Perez-Cruz [173]. The only parameter that was defined based on a different work [225] was the number of hidden units,  $nhu$ .

## A.6 Search Space – Multi-Label Meta Classification Algorithms

In this section, we describe the search space of multi-label meta-algorithms in MEKA. It is important to say that some of the multi-label classifiers (presented in the last section) do not perform very well when used as the multi-label base classifier in a meta classifier. This is due to the poor scalability of such combination (meta multi-label and base multi-label). Examples of methods that would not scale up well are: MCC, PCC, PMCC, CDN, and CDT (these two methods involve Gibbs sampling, which may be

too expensive in an ensemble), RAKEL and RAKELd (these two methods are ensembles by themselves, and using an ensemble as base classifier would lead to a very slow ensemble of ensembles). This must be considered in the grammar or directly in the execution of the algorithm (i.e., setting a time budget for such algorithms when they are used at the multi-label base level).

### A.6.1 Subset Mapper

The Subset Mapper (SM) method [186] maps the output of a multi-label classifier to a known label combination using the Hamming distance, i.e., it checks what label combination (label subsets) from the training set has the closest distance to the predicted label combination on the test instance using the probability distribution of the label subset for this instance. In order to do that, SM transforms the probability distribution array of the label subset into a binary array. For each label subset in the training set (also represented by a binary array), it calculates the Hamming distance to the binary probability distribution array, outputting the closest label subset to the predicted distribution array. SM will map this label subset to this particular test instance. **Parameters:**

- Multi-label classifier(*mlc*)[-W]: The multi-label method that creates a model at the multi-label classification level.

#### **Dependencies/Constraints:**

1. The multi-label classification method can be any one described in Section A.5.

### A.6.2 Bagging of Multi-Label Classifiers

The Bagging of Multi-Label Classifiers (BaggingML) [169] is a method that combines several multi-label classifiers using Bootstrap AGGREGatING (Bagging) [24]. It randomly sets weights higher than zero to certain instances, on only those instances are chosen for the bag. The parameter “bag percent size” is then not used as the number of instances in the bag is just based on the weight values. Thus, the members of the ensemble could have 100% of the instances if all of them have a weight assigned. **Parameters:**

- Multi-label classifier (*mlc*)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Number of iterations (*i*)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ . The range of the number of iterations for this ensemble method was defined by Read’s thesis [169].

**Default value:** 10.

**Dependencies/Constraints:**

1. The multi-label classification method can be any one described in Section A.5, except for BCC, which is not suitable for this meta-learner.

### A.6.3 Bagging of Multi-Label Classifiers with Duplicates

BaggingML with Duplicates (BaggingMLDup) [169] is a method that also combines several multi-label classifiers using Bootstrap AGGREGatING. However, it uses the parameter “bag size percent” to define a specific number of instances for each member (classifier) of the ensemble. After that, it randomly samples instances, being able to sample the same instance (*duplicates*) for the bag. This method does not use any weight to select the instances for the members of the ensemble. **Parameters:**

- Multi-label classifier (*mlc*)[-W]: The multi-label method that creates a model at the multi-label classification level.
- Bag size percent (*bsp*)[-P]: The size of the bag in the percentage of the training set size (number of training instances) and it is defined by the interval:  $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$ .

**Default value:** 67.

- Number of iterations (*i*)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ .

**Default value:** 10.

**Dependencies/Constraints:**

1. The multi-label classification method can be any one described in Section A.5, except for BCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined by Read’s thesis [169]. The parameter “bag size percent” is defined in the MEKA documentation.

#### A.6.4 Ensemble of Multi-Label Classifiers

The Ensemble of Multi-Label Classifiers (EnsembleML) [169] is a method that combines several multi-label classifiers in a simple-subset ensemble. This method is very similar to BaggingMLDup. The only difference is that BaggingMLDup allows sampling with replacement for each model, whereas EnsembleML uses sampling without replacement.

##### Parameters:

- Multi-label classifier (*mlc*)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Bag size percent (*bsp*)[-P]: The size of the bag in the percentage of the training size (number of training instances) and it is defined by the interval:  $\{bsp \in \mathbb{Z} \mid 52 \leq bsp \leq 72\}$ .

**Default value:** 67.

- Number of iterations (*i*)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ .

**Default value:** 10.

##### Dependencies/Constraints:

1. The multi-label classification method can be any one described in Section A.5, except for BCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined by Read’s thesis [169]. Additionally, in his thesis, the author mentioned that they found that values around 62% are the best ones for the parameter “bag size percent” in an ensemble without replacement, which is the case. Thus, we are trying to set the range for this parameter introducing lower and upper bounds close to this value (10% smaller and 10% greater).



### A.6.5 Random Subspace Multi-Label

The Random Subspace Multi-Label (RSML) method [25] combines several multi-label classifiers in an ensemble where the attribute space and the instance space used for building each model are random subsets from the original space. In other words, RSML subsamples the attribute space and instance space randomly for each ensemble member. Basically, it is a generalized version of Random Forests. Additionally, it is computationally cheaper than EnsembleML for the same number of models in the ensemble and the same value of bag size percent.

**Parameters:**

- Multi-label classifier (*mlc*)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Bag size percent (*bsp*)[-P]: The size of the bag in the percentage of the training set size (number of training instances), and it is defined by the interval:  $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$ .

**Default value:** 67.

- Number of iterations (*i*)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ .

**Default value:** 10.

- Attribute percent (*ap*)[-A]: The size of the attribute space, as a percentage of total attribute space size (number of attributes). This parameter is bounded by the following interval:  $\{ap \in \mathbb{Z} \mid 10 \leq ap \leq 100\}$ .

**Default value:** 50.

**Dependencies/Constraints:**

1. The multi-label classification method can be any one described in Section A.5.

The range of the number of iterations for this ensemble method was defined based on Read's thesis [169]. The range of values for this parameter also considers scalability issues, as we need to run a multi-label algorithm many times in an ensemble. The parameter "bag size percent" is defined in the MEKA documentation. The attribute percentage was set in accordance with the single-label version for the same method. This was done because there is not any work that studies this algorithm for multi-label classification.

### A.6.6 Expectation Maximization

In the Expectation Maximization (EMax) method [53], a specified multi-label classifier is built on the training data. This model is then used to classify the training data. The confidence with which instances are classified is used to reweight them. This data is then used to retrain the classifier. This cycle continues ('EM'-style) for  $I$  iterations. The final model is used to classify the test data. Because of the weighting, it is advised to use a classifier that gives good confidence (probabilistic) outputs. **Parameters:**

- Multi-label classifier ( $m_l c$ )[-W]: The multi-label method that creates a model at the multi-label classification level.
- Number of iterations ( $i$ )[-I]: The number of iterations to perform. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ .

**Default value:** 10.

#### Dependencies/Constraints:

1. The classifier at the base multi-label classification level should be capable of producing probabilistic predictions. However, in our preliminary tests, most multi-label classification methods described in Section A.5 were suitable for this meta-learner, except for PMCC. Thus, we will use all the suitable methods in accordance with these experiments at the base multi-label classification level.

The range of the number of iterations for this ensemble method was defined based on Read's thesis [169]. The range of values for this parameter also considers scalability issues, as we need to run a multi-label algorithm many times in an ensemble. This was done because there is no appropriate work that studies this algorithm for multi-label classification.

### A.6.7 Classification Maximization

The Classification Maximization (CMax) method [53, 176] trains a classifier with labeled and unlabeled data (semi-supervised) learning using the Classification Expectation algorithm, which is a hard version of the EM algorithm, as it does not update the instance weights using (a product factor of) the probability distribution produced by the classifier. Instead, it sets to zero (0.0) or one (1.0) the weight of any instance in the dataset. Unlike

EM, it can use any classifier, not necessarily one that gives good probabilistic outputs.

**Parameters:**

- Multi-label classifier (*mlc*)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Number of iterations (*i*)[-I]: The number of iterations to perform. This parameter is restricted by the interval:  $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$ .

**Default value:** 10.

**Dependencies/Constraints:**

1. The multi-label classification method can be any one described in Section A.5, except for PMCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined based on Read's thesis [169]. The range of values for this parameter also considers scalability issues, as we need to run a multi-label algorithm many times in an ensemble. This was done because there is no appropriate work that studies this algorithm for multi-label classification.

# List of Abbreviations and Acronyms

**|L|** Number of Labels.

**3SR** 3sources\_reuters1000.

**AA** Algorithm Adaptation.

**AdaM1** Ada Boost M1.

**AI** Artificial Intelligence.

**ASC** Attribute Selection Classifier.

**ATM** Auto-Tuned Models.

**AutoML** Automated Machine Learning.

**Auto-sklearn** Automated Scikit-Learn.

**Auto-WEKA** Automated WEKA.

**Avg. Rank.** Average Ranking.

**Avg. Val.** Average Values.

**AvgPrec** Average Precision.

**BaggingML** Bagging of Multi-Label Classifiers.

**BaggingMLDup** Bagging of Multi-Label Classifiers with Duplicates.

**BCC** Bayesian Classifier Chain.

**BNC** Bayesian Network Classifier.

**BNF** Backus Naur Form.

**BO** Bayesian Optimization.

**BR** Binary Relevance.

**BRD** Birds.

**BRq** Quick Version of Binary Relevance.

- 
- BS** Beam Search.
- BTX** Bibtex.
- CAL** CAL500.
- Card.** Cardinality.
- CASH** Combined Algorithm Selection and Hyper-parameter.
- CC** Classifier Chain.
- CCq** Quick Version of Classifier Chain.
- CD** Critical Difference.
- CDN** Conditional Dependency Networks.
- CDT** Conditional Dependency Trellis.
- CFG** Context-Free Grammar.
- CG** Current Number of Generations.
- CHD** CHD<sub>49</sub>.
- CMax** Classification Maximization.
- Cov** Coverage.
- CT** Classifier Trellis.
- DBM** Stacked Boltzmann Machine.
- Dens.** Density.
- Div.** Diversity.
- DS** Decision Stump.
- DT** Decision Table.
- DTree** Decision Tree.
- EA** Evolutionary Algorithm.
- EAS-NIS** Exponential Attribute Selection, No Instance Selection.
- EAS-PIS** Exponential Attribute Selection, Polynomial Instance Selection.

- 
- EBR** Ensemble of Binary Relevance.
- ECC** Ensemble of Classifier Chains.
- EI** Expected Improvement.
- EKF** Expert Knowledge Filter.
- ELP** Ensemble of Label Powersets.
- EM** Exact Match.
- EMax** Expectation Maximization.
- EME** Evolutionary Multi-Label Ensemble.
- EMT** Emotions.
- ENR** Enron.
- Ens.** Ensemble.
- EnsembleML** Ensemble of Multi-Label Classifiers.
- EPS** Ensemble of Pruned Sets.
- F<sub>1</sub>** F<sub>1</sub>-measure.
- FLG** Flags.
- FM**  $F_1$  Macro Averaged by Label.
- FN** False Negatives.
- FP** False Positives.
- FW** Four-Class Pairwise Classification.
- G3P-ML** Grammar-Guided Genetic Programming Algorithm for Multi-Label Classification.
- GA** Genetic Algorithm.
- GA-Auto-MLC** Genetic Algorithm for Automated Multi-Label Classification.
- GBS** Genbase.
- GGP** Grammar-based Genetic Programming.
- GP** Genetic Programming.

**GPC** Number of Generations to Population's Convergence.

**GPG** GpositiveGO.

**GP-ML** Genetic Programming for Machine Learning.

**GPP** GpositivePseAAC.

**GSS** Generating Set Search.

**HL** Hamming Loss.

**HOMER** Hierarchy of Multi-Label Classifiers.

**HP** Hyper-Parameters.

**HPA** HumanPseAAC.

**HTN** Hierarchical Task Networks.

**JRip** Java Version for RIPPER.

**K** Number of Cross-validation folds.

**K\*** K Star.

**KNN** K-Nearest Neighbors.

**L** Labels.

**LF** Loss Function.

**LGL** Langlog.

**LHS** Latin Hypercube Sample.

**LMT** Logistic Model Trees.

**LogR** Logistic Regression.

**LP** Label Powerset.

**LR** Label Ranking.

**Lr** Learning Set.

**LPOT** Layered TPOT.

**LWL** Locally Weighted Learning.

- m** Number of Attributes/Features.
- M2CC** Monte-Carlo Classifier Chain (Second Version).
- MBR** Meta Binary Relevance.
- MCC** Monte-Carlo Classifier Chain (First Version).
- MDR** Multifactor Dimensionality Reduction.
- MED** Medical.
- Meta-MLC** Meta-Algorithms for Multi-Label Classification.
- Meta-SLC** Meta-Algorithms for Single-Label Classification.
- mGP-ML** Multi-objective Version of GP-ML.
- ML** Machine Learning.
- ML<sup>2</sup>-Plan** Multi-Label ML-Plan.
- ML-BPNN** Multi-Label Back Propagation Neural Network.
- MLC** Multi-Label Classification.
- ML-C4.5** Multi-Label Version of C4.5 Decision Tree Algorithm.
- ML-DBPNN** Multi-Label Deep Back Propagation Neural Network.
- ML-KNN** Multi-Label K-Nearest Neighbors.
- MLP** Multi-Layer Perceptron.
- MLR** Multi-Label Ranking.
- n** Number of Instances/Examples.
- NAS-NIS** No Attribute Selection, No Instance Selection.
- NAS-PIS** No Attribute Selection, Polynomial Instance Selection.
- NB** Naïve Bayes.
- NBM** Naïve Bayes Multinomial.
- NFL** No Free Lunch.
- NSGA-II** Non-dominated Sorting Genetic Algorithm II.
- OneR** One Rule.



**PAS-NIS** Polynomial Attribute Selection, No Instance Selection.

**PAS-PIIS** Polynomial Attribute Selection, Polynomial Instance Selection.

**PCC** Probabilistic Classifier Chains.

**PCT** Prediction Clustering Tree.

**PMCC** Population of MCC.

**PMF-AutoML** Probabilistic Matrix Factorization for AutoML.

**PPA** PlantPseAAC.

**PS** Pruned Sets.

**PSt** Pruned Set with Threshold.

**PT** Problem Transformation.

**q** Number of Labels.

**RAkEL** Random k-Label Pruned Sets.

**RAkELd** Random k-Label Disjoint Pruned Sets.

**Rank-SVM** Ranking-based Support Vector Machines.

**RBM** Restricted Boltzmann Machine.

**RC** Random Committee.

**RECIPE** Resilient Classification Pipeline Evolution.

**RF** Random Forest.

**RF-PCT** Random Forest of Predictive Clustering Trees.

**RIPPER** Repeated Incremental Pruning to Produce Error Reduction.

**RL** Ranking Loss.

**ROC** Receiver Operating Characteristic.

**RQ** Research Question.

**RS** Random Search.

**RSML** Random Subspace Multi-Label.

- 
- RSS** Random Subspace.
- RT** Ranking and Threshold.
- RTree** Random Tree.
- SC** Stopping Criteria.
- SCN** Scene.
- SGD** Stochastic Gradient Descent.
- sklearn** Scikit-Learn.
- SL** Simple Logistic.
- SLC** Single-Label Classification.
- SM** Subset Mapper.
- SMAC** Sequential Model-based Algorithm Configuration.
- SMBO** Sequential Model-Based Optimization.
- SMO** Sequential Minimal Optimization.
- Stat. Comp.** Statistical Comparison.
- STGP** Strongly Typed Genetic Programming.
- SVM** Support Vector Machines.
- TN** True Negatives.
- TP** True Positives.
- TPE** Tree-structured Parzen Estimator.
- TPOT** Tree-based Pipeline Optimization Tool.
- Ts** Test Set.
- Val** Validation Set.
- VP** Voted Perceptron.
- VPA** VirusPseAAC.
- WEKA** Waikato Environment for Knowledge Analysis.

**WQT** Water-quality.

**XML** Extensible Markup Language.

**YST** Yeast.

**ZeroR** Zero Rules.