UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

CYNTHIA MOREIRA MAIA

On Multi-Label Meta-Learning for Automated Pipeline Recommendation

Recife

2025

CYNTHIA MOREIRA MAIA

On Multi-Label Meta-Learning for Automated Pipeline Recommendation

Thesis presented to the Post-graduation Program in Computer Science - PPGCC of the Centro de Informática of the Universidade Federal de Pernambuco, as a partial requirement for obtaining the title of Doctor in Computer Science.

Concentration Area: Computational Intelligence

Supervisor: George Darmiton da Cunha Cavalcanti

Co-supervisor: Rafael Menelau Oliveira e Cruz

Recife

2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Maia, Cynthia Moreira.
   On Multi-Label Meta-Learning for automated pipeline recommendation / Cynthia Moreira Maia. - Recife, 2025.
   159f.: il.

   Tese (Doutorado)- Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2025.
   Orientação: George Darmiton da Cunha Cavalcanti.
   Coorientação: Rafael Menelau Oliveira e Cruz.

   1. Fluxos; 2. Meta-aprendizagem; 3. Multirrótulo. I. Cavalcanti, George Darmiton da Cunha. II. Cruz, Rafael Menelau Oliveira e. III. Título.

UFPE-Biblioteca Central

Cynthia Moreira Maia

“On Multi-Label Meta-Learning for Automated Pipeline Recommendation”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovada em: 15/10/2025.

_____

Orientador: Prof. Dr. George Darmiton da Cunha Cavalcanti

BANCA EXAMINADORA

_____

Prof. Dr. Tsang Ing Ren
Centro de Informática / UFPE

_____

Prof. Dr. Adiel Teixeira de Almeida Filho
Centro de Informática / UFPE

_____

Prof. Dr. Sylvio Barbon Junior
Departamento de Engenharia e Arquitetura / University of Trieste

_____

Prof. Dr. Rafael Gomes Mantovani
UTFPR / Campus Apucarana

_____

Prof. Dr. . Alceu de Souza Brito Junior
Programa de Pós Graduação em Informática Aplicada /PUC/PR

ACKNOWLEDGEMENTS

Primeiramente, agradeço a Deus por me permitir chegar até este momento. Agradeço também a mim mesma, pelo esforço contínuo e por nunca desistir, mesmo diante de tantas dificuldades.

Meus agradecimentos especiais ao meu esposo, **Julio Cartier**, por sempre me incentivar e compreender os diversos momentos em que não pude estar tão presente devido à dedicação à tese. À minha família, que foi meu alicerce: minha avó **Maria Moreira**, que ao longo deste processo recebeu o diagnóstico de Alzheimer, mas ainda pude mostrar a ela sua primeira neta doutora; minha mãe, **Gerucilania Moreira**; meu padrasto, **Vlademir Diógenes**; e minhas irmãs, **Navi Diógenes** e **Cibelle Diógenes**.

Agradeço também aos meus colegas, em especial **Lucas Amorim**, por toda sua disponibilidade em colaborar nas produções científicas. Aos meus orientadores, **George Darmiton da Cunha Cavalcanti e Rafael Menelau Oliveira e Cruz**, deixo minha profunda gratidão pelo apoio constante, pela dedicação e paciência, que foram fundamentais para a conclusão deste trabalho. Com eles, aprendi a me tornar uma pesquisadora melhor, aperfeiçoei meu inglês e desenvolvi minha escrita científica.

Por fim, agradeço à **CAPES** pelo apoio essencial para a realização e conclusão desta etapa.

RESUMO

O Aprendizado de Máquina Automatizado (Automated Machine Learning - AutoML) visa automatizar etapas do processo de aprendizado de máquina, como seleção de algoritmos, pré-processamento e ajuste de hiperparâmetros. Um de seus principais desafios é projetar um espaço de busca que atenda a diferentes problemas, garantindo a melhor relação entre desempenho e custo computacional. As abordagens tradicionais de AutoML exploram principalmente o espaço de busca em tempo de execução (online), aplicando estratégias de otimização como a Otimização Bayesiana para encontrar a melhor configuração dentro de um prazo determinado. Embora eficazes, tais estratégias frequentemente resultam em altos custos computacionais. Em contraste, nossa proposta busca evitar estratégias de busca online empregando meta-aprendizado para abordar tais desafios. Essa abordagem utiliza as meta-características dos problemas para recomendar soluções apropriadas à sua natureza, eliminando assim a necessidade de busca exaustiva em tempo de execução. Dessa forma, propomos o MetaML, primeiro estudo desta tese, uma abordagem de meta-aprendizado baseada em algoritmos multirrótulos para recomendação de pipelines em AutoML. Para tanto, apresentamos um projeto de espaço de busca com curadoria que reduz automaticamente o número de pipelines candidatos, com base em dados históricos de repositórios online, incluindo apenas os pipelines mais utilizados e com melhor desempenho em um número significativo de conjuntos de dados. Além disso, propomos recomendações encadeadas usando algoritmos multirrótulos que consideram as interdependências entre as etapas do pipeline. Experimentos em diferentes conjuntos de dados demonstram a eficácia da abordagem, com o MetaML alcançando resultados satisfatórios e, em alguns casos, resultados superiores a um custo computacional menor do que os métodos AutoML atuais. No entanto, os pipelines derivados dos experimentos do repositório online apresentaram pouca representatividade em relação ao uso de técnicas de pré-processamento. Como alternativa, propomos o meta-dataset PIPES, o segundo estudo da tese, que consiste em uma coleção de experimentos envolvendo múltiplos pipelines, projetados para representar todas as combinações selecionadas de técnicas incluindo diferentes blocos de pré-processamento e um bloco de classificação. Após a construção do PIPES, utilizamos este meta-dataset no terceiro estudo da tese, o MetaML 2.0, para verificar se é possível obter resultados ainda melhores com uma representatividade mais ampla dos pipelines. Os experimentos demonstraram que, de fato, a abordagem proporcionou desempenhos melhores em determinados conjuntos de dados.

Palavras-chave: Fluxos. Meta-Aprendizagem. Multirrótulo. Aprendizado de Máquina Automatizado.

ABSTRACT

Automated Machine Learning (AutoML) aims to automate stages of the machine learning process, such as algorithm selection, data preprocessing, and hyperparameter tuning. One of its main challenges is designing a search space that can handle different problems while ensuring the best trade-off between performance and computational cost. Traditional AutoML approaches primarily explore the search space online, utilizing optimization strategies such as Bayesian Optimization to identify the optimal configuration within a specified time budget. Although effective, such methods often result in high computational costs. In contrast, our proposal seeks to avoid online search strategies by employing meta-learning to address these challenges. This approach leverages the meta-features of problems to recommend solutions appropriate to their nature, thereby eliminating the need for exhaustive search at runtime. Accordingly, we propose MetaML, the first study of this thesis, a meta-learning approach based on multi-label algorithms for pipeline recommendation in AutoML. To this end, we present a curated search space design that automatically reduces the number of candidate pipelines, based on historical data from online repositories, including only the most frequently used pipelines with the best performance across a significant number of datasets. Additionally, we propose chained recommendations utilizing multi-label algorithms that take into account the interdependencies between pipeline stages. Experiments conducted on different datasets demonstrate the effectiveness of the approach, with MetaML achieving satisfactory results and, in some cases, superior outcomes at a lower computational cost compared to current AutoML methods. However, the pipelines derived from the repository experiments showed limited representativeness with respect to preprocessing techniques. As an alternative, we propose the PIPES meta-dataset, the second study of this thesis, which consists of a collection of experiments involving multiple pipelines, designed to represent all selected combinations of techniques, including different preprocessing blocks and a classification block. After constructing PIPES, we employed this meta-dataset in the third study of the thesis, MetaML 2.0, to investigate whether broader pipeline representativeness could yield even better results. The experiments demonstrated that this approach indeed achieved improved performance in specific datasets.

Keywords: Pipeline. Meta-Learning. Multi-Label. Automated Machine Learning.

LIST OF FIGURES

Figure 1 – Representation of the four components ASP (RICE, 1976), adapted from (SMITH-MILES, 2009). 26

Figure 2 – Representation of the base and meta levels of Metalearning, adapted from (BRAZDIL et al., 2008). 27

Figure 3 – Transformation of the multi-label problem into a single-label problem using the BR method. Adapted from (SOROWER, 2010). 31

Figure 4 – Transformation of the multi-label problem into a single-label problem using the CC method. Adapted from (MOYANO et al., 2019). 33

Figure 5 – Transformation of the multi-label problem into a single-label problem using the LP method. Adapted from (MOYANO et al., 2019). 34

Figure 6 – Comparison between traditional AutoML (a) and our proposed meta-learning based framework MetaML (b). 40

Figure 7 – Representation of the four components ASP, adapted from (RICE, 1976). . 42

Figure 8 – Recommendation with Metalearning, adapted from (BRAZDIL et al., 2008). . 43

Figure 9 – A representation of an example of pipeline as a tuple of blocks. Each block is a step in the ML pipeline. 45

Figure 10 – Representation typical AutoML components, adapted from (HUTTER; KOTTHOFF; VANSCHOREN, 2019). 46

Figure 11 – The MetaML framework process is divided into four main phases: (A) Search Space Definition, (B) Meta-Dataset Construction, (C) Training Phase, and (D) Recommendation Phase. The arrows indicate transitions between these stages. In phase (A), $\mathbb{G}$ represents the selection of the top n experiments for each dataset present in the collection $\mathbb{O}$, while $\mathbb{P}$ corresponds to the k most frequent pipelines among the experiments in $\mathbb{G}$. From this filtering step, the best-performing experiment is selected for each dataset, provided that its pipeline belongs to $\mathbb{P}$. This process defines the search space $\mathbb{S}$. 54

Figure 12 – Critical difference diagram of the average ranking of meta-model algorithms considering their base-level accuracy. 64

Figure 13 – Critical difference diagram of the average ranking of the base-level performance of the AutoML methods, including (a) MetaML zero-shot, (b) MetaML 3-shot. 68
Figure 14 – Performance distributions of the pipelines recommended by each AutoML method on the tested datasets. 69
Figure 15 – Comparison of AutoML methods using different recommendation time budgets. 70
Figure 16 – Recommendation frequency of each pipeline for MetaML zero-shot (a) and MetaML 3-shot (b). 72
Figure 17 – MetaML's base-level performance the full search space versus when using the curated search space under different values for the parameters k and n. 75
Figure 18 – Meta-model inference time of the MetaML when using the full search space versus the curated one. 76
Figure 19 – Representation of an example of a specific pipeline. 86
Figure 20 – Frequency of execution of each technique, per pipeline block, according to OpenML records. Most of the abbreviations for FP and scaling are given in Table 10. Additional abbreviations: Classifiers: LR - LogisticRegression, XGB - XGBoost, NuSVC - Nu-Support Vector Classification, LGBM - LightGBM, Bag - Bagging, Perc - Perceptron, SE - StackingEstimator, Voting - VotingClassifier. Scaling: Bin - Binarizer. 90
Figure 21 – Quantity of pipelines that employ each number of preprocessing blocks. Considering all OpenML classification pipelines that use Scikit-learn. 91
Figure 22 – The graphs show how frequently each technique appears in the best pipelines for the 192 datasets in common. Most abbreviations for FP and scaling are given in Table 10. Additional abbreviations: VT - VarianceThreshold, FA - FactorAnalysis, SFM - SelectFromModel. Scaling: MA — Max Absolute Scaler. 93
Figure 23 – PIPES' datasets organized according to a UMAP representation of their meta-features' space. 94
Figure 24 – Pipelines performance on 110 datasets, with and without preprocessing blocks. 101

Figure 25 – The graphs show the frequency distribution of each meta-target across 117 datasets of MetaML 2.0 (a) and 290 datasets of MetaML 1.0 (b). Additional abbreviations are provided below: SI - Simple Imputer, OHE - OneHotEncoder, OE - OrdinalEncoder, PF - PolynomialFeatures, ETP - ExtraTrees pre, KPCA - KernelPCA, RTE - RandomTreesEmbedding, PT - PowerTransforme, RS - RobustScaler, QT - QuantileTransformer, MM - MinMaxScaler, SS - StandardScaler, HGB - HistGradientBoosting, ET - ExtraTrees, RF - RandomForest, MLP - Multi-layer Perceptron, SVC - Support Vector Classification, VT - VarianceThreshold, DT - Decision Tree. 102
Figure 26 – Representation of the meta-features space with UMAP. 129
Figure 27 – Distribution of 12 complexity measures used in the construction of the meta-dataset. 130
Figure 28 – Accuracy comparison of AutoMLs considering specific dataset characteristics. 146
Figure 29 – Representation of the pipelines. 148
Figure 30 – Comparison of AutoML methods using different recommendation time budgets. 151

LIST OF TABLES

Table 1 – Types of Meta-features. 44

Table 2 – Comparison of the most popular open-source AutoML frameworks. FE - Feature Engineering, MS - Model Selection, HO - Hyperparameters Optimization, DP - Data Preprocessing. 48

Table 3 – Mathematical notation. 55

Table 4 – The five pipelines included in the MetaML's search space. 59

Table 5 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML zero-shot (a) and 3-shot (b) against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 5.56e\text{-}04$, and the total recommendation time taken for datasets for which all methods ran successfully. 65

Table 6 – All the six different pipelines recommended by the MetaML for the 290 test datasets. 72

Table 7 – The nineteen pipelines included in the MetaML's full search space. The binary digits indicate whether or not each technique is present in the pipeline. SP - SelectPercentile, SI - SimpleImputer, PCA -Principal component analysis, RSC - RobustScaler, PF - PolynomialFeatures, OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest, AdaBoost - Adaptive Boosting, LDA - LinearDiscriminantAnalysis, XGB - eXtreme Gradient Boosting, ExtraTrees, LR - LogisticRegression, LinearSVC - Linear Support Vector Classification, MLP - Multi-layer Perceptron, SGD - Stochastic Gradient Descent. 73

Table 8 – The eight pipelines search space curated (k=8). The binary digits indicate whether or not each technique is present in the pipeline. OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest, LinearSVC - Linear Support Vector Classification, MLP - Multi-layer Perceptron, SGD - Stochastic Gradient Descent. 74

Table 9 – A base-level analysis of the mean performance (accuracy), Win/tie/loss of the Curated search space (n=5 and k=5) against the others, mean ranking, p-value of the Wilcoxon signed rank test with α =8.33e-03. 75
Table 10 – Pipeline blocks and their possible techniques. 88
Table 11 – A analysis of the mean accuracy, Wins/ties/losses of the $\mathbf{M}_{\text{PIPES}}$ against $\mathbf{M}_{\text{OpenML}}$, mean ranking, p-value of the Wilcoxon signed rank test ($\alpha = 0.05$.) 93
Table 12 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML 2.0 zero-shot (a) and 3-shot (b) against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 4.55e\text{-}04$, and the total recommendation time taken for datasets for which all methods ran successfully. 103
Table 13 – Comparison pipelines of OpenML. RSC - RobustScaler, OHE - One-Hot Encoder, VT - Variance Threshold, SS - StandardScale, DT - Decision Tree. 117
Table 14 – Datasets used to construct the meta-dataset. DID - Data ID, TID - Task ID. 118
Table 15 – Datasets used to construct the meta-dataset. 131
Table 16 – Datasets that were used in the comparative analysis. DID - Data ID, TID - Task ID. 137
Table 17 – Wins, ties of losses of MetaML using three different multi-label approaches versus single label approaches. The p-values, resulting from a Wilcoxon signed rank test that are lower than $\alpha = 0.004$ are highlighted in bold. . . . 143
Table 18 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML 3-shot in real (a) and artificial (b) datasets against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 5.56e\text{-}04$. 144
Table 19 – Meta-features used to construct the meta-dataset MetaMl 2.0. 152
Table 20 – Meta-features excluded. 157

LIST OF ABBREVIATIONS AND ACRONYMS

AB	AdaBoost
AMOEA	Asynchronous Multi-Objective Evolutionary Algorithm
ASP	Algorithm Selection Problem
ASHA	Asynchronous Successive Halving Algorithm
AUTOML	Automated Machine Learning
Bag	Bagging
Bin	Binarizer
BNB	BernoulliNB
BR	Binary Relevance
CANE	Cumulative Average Normalized Error
CASH	Combined Algorithm Selection and Hyperparameter Optimization
CC	Classifier Chains
CF	Collaborative Filtering
CNN	Convolutional Neural Network
DP	Data Preprocessing
DT	Decision Tree
ET	ExtraTrees
ETP	ExtraTrees prep
ExtraTrees	Extremely Randomized Trees
FA	Factor Analysis
FAGG	Feature Agglomeration

FE	Feature Engineering
FICA	FastICA
GAMA	General Automated Machine learning Assistant
GLMs	Large Language Models
GN	GaussianNB
GP	Genetic Programming
GU	Generic Univariate Select
HGB	HistGradient Boosting
HPO	Hyperparameter Optimization
KNN	K-Nearest Neighbor
KPCA	KernelPCA
LDA	Linear Discriminant Analysis
LGBM	LightGBM
LLMs	Large Language Models
LODOCV	Leave-one-dataset-out cross-validation
LP	Label Powerset
LR	Logistic Regression
LSVC	LinearSVC
LSVCP	LinearSVC prep
MA	Max Absolute Scaler
ML	Machine Learning
MLP	Multilayer Perceptron
MM	MinMaxScaler

MNB	MultinomialNB
MS	Model Selection
MtL	Meta-Learning
NAS	Neural Architecture Search
NN-SMFO	Nearest Neighbor Sequential Model-Free Optimization
Nor	Normalizer
NuSVC	Nu-Support Vector Classification
NY	Nystroem
OE	Ordinal Encoder
OHE	One-Hot Encoding
PA	Passive Aggressive
PCA	Principal Component Analysis
PCC	Probabilistic Classifier Chains
Perc	Perceptron
PF	Polynomial Features
POSH	Portfolio Successive Halving
PT	PowerTransformer
QT	Quantile Transformer
QDA	Quadratic Discriminant Analysis
RBFS	Radial Basis Function Sampler
RF	Random Forest
RS	Robust Scaler
RTE	Random Trees Embedding

SE	Stacking Estimator
SGD	Stochastic Gradient Descent
SFM	Select From Model
SI	Simple Imputer
SMAC	Sequential Model-based Algorithm Configuration
SP	Select Percentile
SS	Standard Scaler
SVC	Support Vector Classifier
SVR	Support Vector Regressor
SVM	Support Vector Machine
TabPFN	Tabular Prior-Data Fitted Network
TPOT	Tree-based Pipeline Optimization Tool
TPE	Tree of Parzen Estimators
TSVD	Truncated SVD
UMAP	Uniform Manifold Approximation and Projection for Dimension Reduction
VT	Variance Threshold
XGB	XGBoost

LIST OF SYMBOLS

$\mathbf{D}$	Original dataset.
$\mathcal{D}$	Multi-label dataset.
$\mathbb{D}$	The set of datasets.
$\mathbf{D_e}$	Dataset used in experiment $\mathbf{e}$.
$\mathbf{e}$	An ML experiment.
$\mathbf{f}$	Meta-features vector.
$\mathbf{M}$	Meta-dataset.
$\mathbf{N}$	A new (query) dataset.
$\mathbb{O}$	Record of past ML experiments.
π	An ML pipeline.
$\hat{\pi}$	Recommended pipeline.
π_i	Pipeline representation for dataset i.
$\mathbb{P}$	Set of k most frequent pipelines.
$\mathbf{R_N}$	Ranking of promising pipelines for $\mathbf{N}$.
$\mathbb{S}$	Search space.
λ	Meta-classifier.
m	Number of meta-classes.
n	Number of experiments.
q	Number of pipelines.
x	Number of meta-features.
z	Number of datasets.
l	Number of labels in multi-label dataset.

$\mathbf{y}_i$	The vector of labels of the i-th instance.
$\mathbf{x}_i$	Feature vector of the i-th instance.
$\mathbb{T}$	The set of imputation techniques.
$\mathbb{E}$	The set of encoding techniques.
$\mathbb{P}$	The set of scaling techniques.
$\mathbb{A}$	The set of feature preprocessing, transformation and feature selection.
$\mathbb{C}$	The set of classifiers.

SUMMARY

1 INTRODUCTION 22
1.1 OBJECTIVE 24
1.2 MAIN CONTRIBUTIONS AND THESIS STRUCTURE 24
2 BASIC CONCEPTS 26
2.1 META-LEARNING 26
2.1.1 Meta-Features 28
2.1.2 Meta-Target 28
2.2 MULTI-LABEL ALGORITHMS 29
2.2.1 Binary Relevance (BR) 30
2.2.2 Classifier Chains (CC) and Probabilistic Classifier Chains (PCC) . . 31
2.2.3 Label Powerset (LP) 33
2.3 AUTOMATED MACHINE LEARNING 34
2.4 FINAL CONSIDERATIONS 36
3 METAML: A MULTI-LABEL META-LEARNING APPROACH FOR PIPELINE RECOMMENDATION 37
3.1 INTRODUCTION 37
3.2 BACKGROUND 41
3.2.1 Meta-Learning 42
3.2.2 Multi-Label Algorithms 44
3.2.3 Automated Machine Learning 45
3.3 RELATED WORK 47
3.3.1 AutoML using Bayesian Optimization 48
3.3.2 AutoML using Random Search 50
3.3.3 AutoML with Genetic Programming 51
3.3.4 AutoML other optimization types 51
3.4 METAML 53
3.4.1 Search space definition 53
3.4.2 Meta-dataset construction 56
3.4.3 Training 57
3.4.4 Recommendation 57

3.5 EXPERIMENTAL SETUP . . . 58
3.5.1 Datasets and Search Space . . . 58
3.5.2 Meta-dataset construction . . . 60
3.5.3 Meta-Model . . . 60
3.5.4 Data Preprocessing . . . 61
3.5.5 Evaluation procedure . . . 61
3.5.6 Software and Hardware . . . 63
3.6 RESULTS AND DISCUSSION . . . 63
3.6.1 Comparing meta-model algorithms . . . 63
3.6.2 Comparing the performances of recommended pipelines . . . 64
3.6.3 Comparing recommendation times . . . 70
3.6.4 Pipeline Recommendation Analysis . . . 71
3.6.5 The effect of search space curation . . . 73
3.7 THREATS TO VALIDITY . . . 77
3.8 CONCLUSION . . . 78
4 PIPES: A META-DATASET OF MACHINE LEARNING PIPELINES 80
4.1 INTRODUCTION . . . 80
4.2 BACKGROUND AND RELATED WORK . . . 83
4.3 PROPOSED META-DATASET . . . 85
4.4 META-DATASET CONSTRUCTION . . . 87
4.4.1 Datasets . . . 87
4.4.2 Pipeline blocks . . . 87
4.4.3 Meta-Features . . . 88
4.4.4 Hardware and Software . . . 89
4.5 ANALYSIS . . . 89
4.5.1 Exploratory analysis of the pipelines from OpenML . . . 89
4.5.2 Comparing PIPES and OpenML in a meta-learning task . . . 91
4.5.3 PIPES' datasets diversity . . . 94
4.6 LIMITATIONS . . . 94
4.7 CONCLUSION . . . 95
5 METAML 2.0 . . . 96
5.1 INTRODUCTION . . . 96
5.2 EXPERIMENTAL SETUP . . . 97

5.2.1 Datasets and Search Space . . . 97
5.2.2 Meta-dataset construction and Meta-Model . . . 98
5.2.3 Data Preprocessing . . . 98
5.2.4 Software and Hardware . . . 99
5.2.5 Evaluation procedure . . . 99
5.3 RESULTS AND DISCUSSION . . . 100
5.3.1 Meta-level analysis . . . 100
5.3.2 The impact of using preprocessing blocks . . . 100
5.3.3 Comparing Pipeline Diversity: PIPES vs. OpenML . . . 101
5.3.4 Comparing the performances of recommended pipelines . . . 102
5.4 CONCLUSION . . . 105
6 CONCLUSION AND FUTURE WORK . . . 106
REFERENCES . . . 108
APPENDIX A – PIPELINE REDUNDANCY ANALYSIS . . . 116
APPENDIX B – DATASETS USED IN META-DATASET CONSTRUCTION . . . 118
APPENDIX C – METAML DATASETS DIVERSITY . . . 129
APPENDIX D – META-FEATURES . . . 131
APPENDIX E – DATASETS USED IN COMPARATIVE EXPERIMENTS . . . 137
APPENDIX F – COMPARATIVE PERFORMANCE OF META-MODELS: MULTI-LABEL VS. SINGLE-LABEL APPROACHES 143
APPENDIX G – PERFORMANCE ANALYSIS: REAL VS. ARTIFICIAL DATASETS . . . 144
APPENDIX H – COMPARISON OF AUTOMLS CONSIDERING DATASETS CLASSES AND INSTANCES . . . 145
APPENDIX I – EXAMPLES OF PIPELINES RECOMMENDED BY METAML AND NAIVE AUTOML . . . 148
APPENDIX J – COMPARISON OF AUTOML METHODS USING DIFFERENT TIME BUDGETS . . . 150
APPENDIX K – META-FEATURES . . . 152
APPENDIX L – LIST OF EXCLUDED META-FEATURES . . . 157

1 INTRODUCTION

A goal of Automated Machine Learning (AutoML) is to streamline the machine learning pipeline by automating tasks such as data preprocessing, model selection, and hyperparameter optimization, thereby reducing dependency on manual expertise while producing robust models (ELSHAWI; MAHER; SAKR, 2019; BAHRI et al., 2022) (DÔRES; SOARES; RUIZ, 2018). However, its use faces challenges (BAHRI et al., 2022; ELSHAWI; MAHER; SAKR, 2019): (1) time budget allocation, larger budgets increase the system's chances of finding the optimal configuration (ELSHAWI; MAHER; SAKR, 2019), but they also lead to longer user waits and higher computational costs, including memory and CPU usage, which may incur monetary expenses. Conversely, smaller budgets reduce wait times but decrease the chances of finding an optimal configuration. (2) search space design: the set of all possible configurations that the system can explore defines the search space. Its design is challenging because a small search space may fail to include configurations that achieve good performance for certain types of problems. In contrast, a large search space increases computational cost due to the greater number of possibilities that must be evaluated (BRAZDIL et al., 2022b). Studies show that smaller, but carefully selected, search spaces (with high-performance classifiers) can achieve comparable results to a space that encompasses a broad search space (ELDEEB et al., 2022).

The performance of AutoML systems varies, as no single method performs best across all tasks: it depends on the dataset (GIJSBERS et al., 2019). These systems can be improved using Meta-Learning (MtL), which leverages dataset characteristics (meta-features) to recommend promising configurations for a given task (HUTTER; KOTTHOFF; VANSCHOREN, 2019). Meta-Learning (MtL) enables learning from metadata generated by previous machine learning experiments. To achieve this, a meta-dataset is constructed, comprising meta-examples. Each meta-example represents a dataset through its characteristics (meta-features) and is associated with the performance of configurations on that dataset (meta-targets) (BRAZDIL et al., 2008). A challenge in using MtL lies in the construction of these metadata, as their generation entails a high computational cost due to the need to evaluate the performance of multiple configurations across diverse datasets (KHAN et al., 2020). A promising solution to alleviate these costs involves utilizing available machine learning experiments from public repositories, which contain hundreds of experiments, thereby enabling reuse and accelerating research progress in the field (BRAZDIL et al., 2022a).

In this context, we propose MetaML, a meta-learning approach for pipeline recommendation. The primary objective of this work is to address the challenge of selecting the most suitable machine learning pipeline for a given dataset based on its meta-features. The proposal brings two main contributions: (1) reducing the search space through the automatic curation of the most promising pipelines based on historical data from online repositories; and (2) using multi-label classification as a metamodel to explore label interdependencies and recommend a set of candidate pipelines, modeling the relationship between blocks composed of classification algorithms and data preprocessing algorithms. In this way, for example, decision tree-based algorithms can be recommended without the need for a data scaling block, since one is not necessary. This approach differs from studies (WANG et al., 2014; ZHANG; SONG, 2015; DANTAS; POZO, 2018; KHAN et al., 2020; ZHU et al., 2021; GOLSHANRAD et al., 2021; WEGMETH; BEEL, 2022) in that they do not rely on pipeline recommendations that take into account the interdependence of the steps.

However, the approach of using results from previous machine learning experiments from online repositories presented a limitation. While online repositories provide comprehensive experimental data, they exhibit inconsistent quality and limited representativeness of the recorded pipelines. A particular concern is that preprocessing techniques are rarely employed, and even when they are, just a few are explored.

We carried out an analysis of 22,298 machine learning pipelines in OpenML, of these pipelines and found that only 47.09% included at least one preprocessing block; 23.20% of the pipelines included a function transformer block, 7.70% included the scaling block; 6.84% included feature preprocessing; 3.93% applied missing value imputation methods; 3.64% employed encoding for categorical variables; only 0.16% included data resampling techniques. Additionally, 1.60% registered the use of preprocessing techniques but did not specify which methods were applied.

Research has demonstrated their significant impact on model performance (AMORIM; CAVALCANTI; CRUZ, 2023a) (RAJU et al., 2020) (OBAID; DHEYAB; SABRY, 2019). To address these challenges, our second study presents PIPES, a meta-dataset of machine learning experiments that strives for completeness and diversity of pipelines, including several preprocessing blocks and a classification block.

Building on PIPES and aiming to overcome the limitations observed in the original MetaML approach, we propose MetaML 2.0. This new version is built upon the PIPES meta-dataset and aims to identify promising pipelines and recommend rankings of pipelines most suitable for

specific datasets. We seek to compare the results of MetaML 2.0 with the original MetaML to evaluate the impact of this enrichment. MetaML 2.0 introduces greater diversity of pipelines by incorporating a broader range of preprocessing techniques into the search space. As a result, we observed improvements in some of the evaluated datasets compared to the original MetaML.

1.1 OBJECTIVE

The main objective of this thesis is to propose an AutoML method based on meta-learning that automates pipeline recommendation by learning interdependencies between pipeline blocks from past experience.

1.2 MAIN CONTRIBUTIONS AND THESIS STRUCTURE

The main contributions of this thesis are:

- **Algorithm design a curated search space**: a novel way to heuristically reduce the search space by automatically curating the most relevant candidate pipelines based on historical data.
- **Chained recommendations**: a method of composing pipelines that is based on the chained recommendations of their steps in such a way that the interdependence of the steps is taken into account.
- **A meta-dataset of machine learning experiments**: collection of experiments involving multiple pipelines, a total of 9,408 combinations, i.e., pipelines, (2 imputation techniques $\times$ 3 categorical encoders $\times$ 7 scalers $\times$ 14 feature preprocessing techniques $\times$ 16 classification algorithms). The meta-dataset includes full details of the experiments.
- **Comparative analysis**: demonstration of the performance benefits of the proposed method through comparison with state-of-the-art methods.

This thesis is structured as follows:

- **Chapter 2** presents the basic concepts of this thesis.

- **Chapter 3** introduces our first contribution through the paper: "MetaML: A Multi-Label Meta-Learning Approach for Pipeline Recommendation". This chapter has been submitted to a Machine Learning journal.
- **Chapter 4** details our second contribution with paper: "PIPES: A Meta-dataset of Machine Learning Pipelines". This chapter has been published as a paper in the International Joint Conference on Neural Networks (IJCNN) 2025.
- **Chapter 5** presents MetaML 2.0, which integrates contributions from Chapters 3 and 4.
- **Chapter 6** concludes the thesis with final considerations, limitations, and future works.

2 BASIC CONCEPTS

This thesis encompasses three main research areas: meta-learning, multi-label algorithms, and automated machine learning. Thus, Section 2.1 addresses meta-learning. Section 2.2 presents multi-label algorithms, and Section 2.3 briefly discusses the concepts of automated machine learning.

2.1 META-LEARNING

Rice (1976) formulated the Algorithm Selection Problem (ASP), defined as the problem of mapping algorithm performance to dataset characteristics. This formulation, illustrated in Figure 1, addresses the issue of selecting the most appropriate algorithm for a given task by leveraging the dataset properties (RICE, 1976). ASP is one of the research focuses in Meta-Learning, where the goal is to automate the selection of algorithms for a given dataset based on its meta-features (BRAZDIL et al., 2022a).

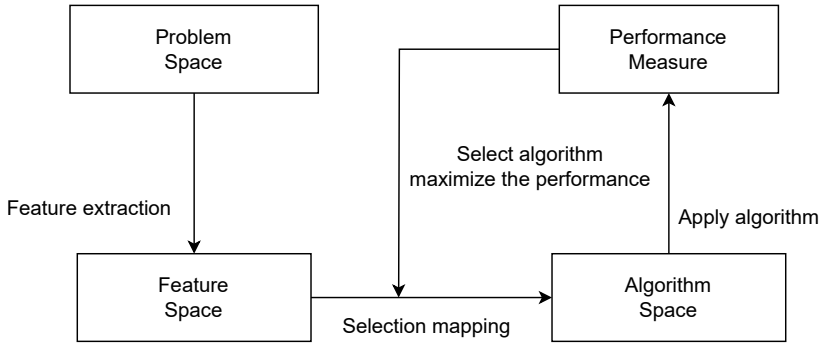


Figure 1 – Representation of the four components ASP (RICE, 1976), adapted from (SMITH-MILES, 2009).

In addition to ASP, another key focus of MtL is Hyperparameter optimization (HPO) (FEURER; HUTTER, 2019), Combined algorithm selection and hyperparameter optimization (CASH) (KOTTHOFF et al., 2019), and Workflow (pipeline) (BRAZDIL et al., 2008). In HPO, given a dataset, HPO aims to recommend the best hyperparameter settings for a specific algorithm. CASH extends HPO by simultaneously recommending the best algorithm and its optimal hyperparameters for a given dataset. Workflow aims to recommend complete machine learning pipelines, which may include multiple algorithms from the different blocks that make up the pipeline. One area that has been benefiting from meta-learning is Automated Machine Learning (AutoML), which refers to the automation of the machine learning pipeline.

In AutoML systems, meta-learning has been used to initialize searches, complementing recommendation approaches (BAHRI et al., 2022). For example, in AutoSklearn, meta-learning is employed as an initial step to identify suitable instantiations in new datasets, which can then guide searches using the Bayesian optimization technique for optimal recommendations (FEURER et al., 2020).

Figure 2 presents a typical architecture in meta-learning that contains a base level and a meta-level. The base level defines the set of candidate algorithms that the system can recommend and the evaluation metric used to evaluate the algorithm's performance. These definitions guide the construction of the meta-dataset $\mathbf{M} = \{(\mathbf{f}_1, \pi_1), (\mathbf{f}_2, \pi_2), \ldots, (\mathbf{f}_z, \pi_z)\}$, which consists of meta-examples, each representing a given dataset $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_z\}$, through pairs of (1) $\mathbf{f}_i = \{f_i^1, f_i^2, \ldots, f_i^x\}$ with x meta-features, allow the characterization of data sets by providing information, such as the number of classes, attributes; and (2) performance data of candidate algorithms on this dataset, $\pi_i = \{\pi_i^1, \pi_i^2, \ldots, \pi_i^m\}$,. At the meta-level, the metamodel λ is trained on this meta-dataset $\mathbf{M}$ to learn the relationship between dataset features and algorithm performance, enabling automated algorithm recommendation for new datasets $\mathbf{N}$, given $\mathbf{N} \notin \mathbb{D}$.

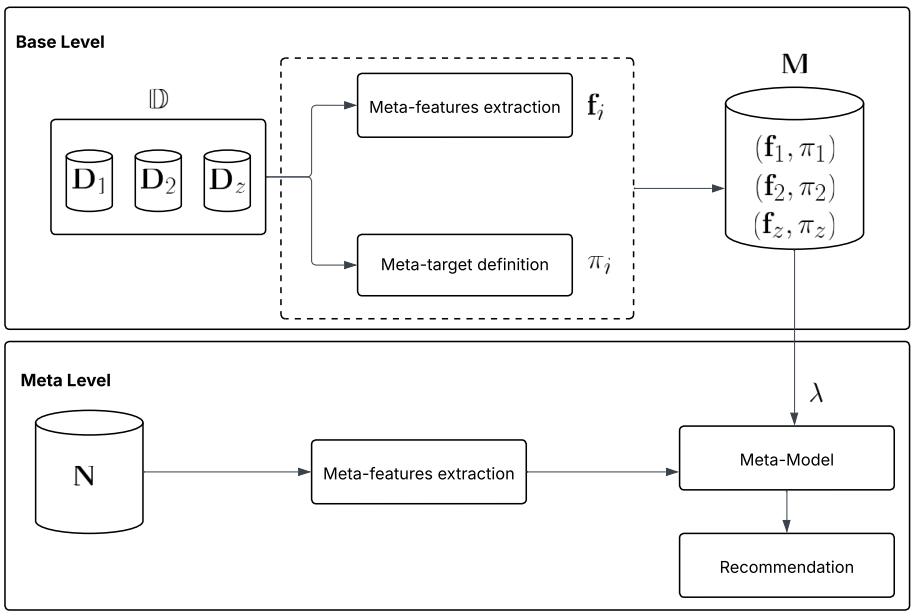


Figure 2 – Representation of the base and meta levels of Metalearning, adapted from (BRAZDIL et al., 2008).

Some important decisions when using meta-learning are: what types of meta-features will be used to represent the datasets, and what type of meta-target will be used for recommendation. The following subsection describes some of the types of meta-features and meta-targets.

2.1.1 Meta-Features

Meta-features allow us to characterize datasets and can be organized into different groups. The main groups of meta-features are described below (BRAZDIL et al., 2022a) (CASTIELLO; CASTELLANO; FANELLI, 2005):

- **Simple, Statistical, Information theory**: The Simple are meta-features that contain more general information about the datasets, for example: number of instances, proportion of discrete attributes, and number of classes. Statistical measures take into account the statistical properties of the datasets, for example: skewness, kurtosis, class probabilities, and correlations between features. Information theory, originating in information theory, encompasses meta-features such as class and attribute entropy, as well as mutual information (BRAZDIL; GAMA; HENERY, 1994).
- **Model-based**: these are meta-features derived from a model's properties. For instance, in the case of a decision tree model, meta-features could include the number of nodes per feature and the number of leaves per class (PENG et al., 2002).
- **Landmarkers**: are measures of the predictive performance of models on datasets. For example, the performance of the linear discriminant algorithm, characterizing linear separability (BENSUSAN; GIRAUD-CARRIER, 2000).
- **Complexity-based**: enable the analysis of data set complexity, including non-linearity of a linear classifier, the average number of features per dimension, and the volume of the overlapping region (HO; BASU, 2002).

2.1.2 Meta-Target

Meta-targets define the type of recommendation in a meta-learning problem (BRAZDIL et al., 2008). The main types include:

- **(i) Best algorithm**, which returns the single algorithm expected to perform best on the dataset. For example, given a search space $\mathcal{A} = \{a_1, \ldots, a_4\}$, the meta-model may recommend a_4 if it is expected to yield the highest performance, rather than the others that make up the space;

- **(ii) Ranking**, which provides an ordered list of algorithms. A complete linear ranking, often used in meta-learning systems, implies that all algorithms are ordered with distinct ranks, e.g., $1^{\circ} \rightarrow a_2$, $2^{\circ} \rightarrow a_3$, $3^{\circ} \rightarrow a_1$, and $4^{\circ} \rightarrow a_4$;
- **(iii) Algorithm subset selection**, more than one algorithm is indicated, for example: $\{a_1, a_3, a_4\}$. In the event of a tie with other best algorithms, the first one is selected in the order in which they appear.

The choice of the meta-target also affects the choice of the meta-model. Thus, to recommend multiple algorithms simultaneously, some studies use multi-label classification algorithms as meta-models (ZHU et al., 2021) (KHAN et al., 2020).

2.2 MULTI-LABEL ALGORITHMS

Single-label classification associates each instance with a unique label, whereas multi-label classification allows an instance to be associated with multiple labels simultaneously (TSOUMAKAS; KATAKIS; VLAHAVAS, 2009). An example would be the semantic annotation of an image, which may have multiple related labels, such as trees, people, animals, grass, and clouds. Another example is the categorization of a news story. The story may be associated with several topics, for example, engineering, statistics, and biology. The multi-label problem is present in various segments, including text classification (BAKER; KORHONEN, 2017), image classification (DIMITROVSKI et al., 2011), musical data (TROHIDIS et al., 2008), and bioinformatics (TANG et al., 2019), among others.

Multi-label classification can be grouped into two approaches: problem transformation and algorithm adaptation (HERRERA et al., 2016). Algorithm adaptation is algorithm-dependent; machine learning algorithm methods that address single-label problems are extended to address multi-label classification. Examples of algorithm adaptation include (CLARE; KING, 2001), which modified the C4.5 decision tree algorithm to address multi-label classification by modifying the entropy measure. In (ZHANG; ZHOU, 2007), the k-nearest neighbors (kNN) algorithm is extended, called ML-kNN. In ML-kNN, the labels of a test instance are determined via the maximum a posteriori, which is based on the a priori and a posteriori probabilities for each label of the k nearest neighbors.

In problem transformation, the multi-label classification problem is transformed into a single-label problem, independent of the (TSOUMAKAS; KATAKIS; VLAHAVAS, 2009) algorithm.

Some problem transformation algorithms are described below, as this approach is used in this thesis. To illustrate the transformations in different algorithms, we will consider the following notation.

Let

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$$

be a multi-label dataset, where:

$$\mathbf{x}_i \in \mathbb{R}^d$$

is the feature vector of the i-th instance, and

$$\mathbf{y}_i \in \{0, 1\}^l$$

is the vector of labels of the i-th instance, with l possible labels.

For the case with $l = 3$ labels, we denote by $\mathbf{Y}_j$ the column vector corresponding to the label j, for $j \in \{1, 2, 3\}$.

2.2.1 Binary Relevance (BR)

Binary Relevance (BR) consists of transforming the multi-label problem into binary single-label problems. Suppose there is an original dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n},$$

that contains three labels $j \in \{1, 2, 3\}$. Then, three binary datasets are created, each corresponding to a label j, denoted by

$$\mathcal{D}_j = \{(x_i, y_{i,j})\}_{i=1}^{n},$$

where $y_{i,j}$ is the value of label j for instance i. Each dataset $\mathcal{D}_j$ is treated independently.

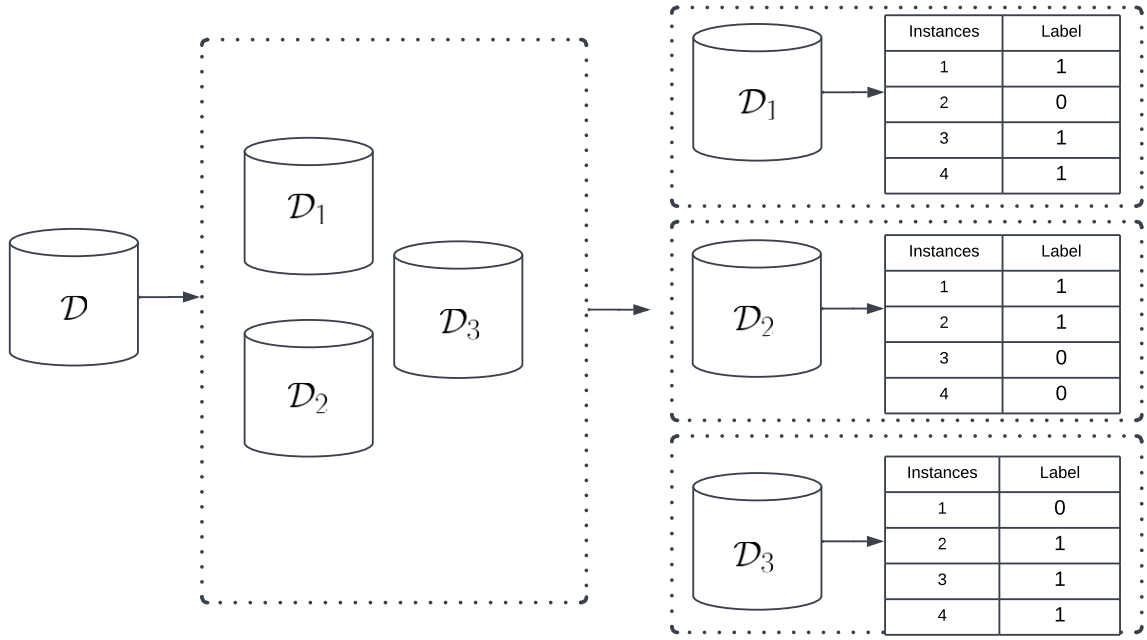


Figure 3 – Transformation of the multi-label problem into a single-label problem using the BR method. Adapted from (SOROWER, 2010).

These datasets contain the same instances as the original dataset, but each dataset will be labeled according to its corresponding label. Thus, if an instance belongs to the label, it will be labeled 1, meaning relevant; if not, it will be labeled 0, meaning irrelevant. After being labeled, a binary classifier is trained with the dataset. Finally, for a new instance, the method generates as output the union of the labels generated by the classifiers. This method checks the relevant labels in each binary classifier and then performs the combination ((TSOUMAKAS; KATAKIS, 2007) (SANTOS, 2012) (SOROWER, 2010). It is considered a simple and quite popular strategy, but one disadvantage is that it does not consider the correlation between the labels ((LUACES et al., 2012)).

2.2.2 Classifier Chains (CC) and Probabilistic Classifier Chains (PCC)

To capture dependencies between labels in multi-label classification, one effective approach is the Classifier Chains (CC) method. Instead of predicting each label independently, CC links a sequence of binary classifiers so that the output of one becomes part of the input for the next. For example, given an original multi-label dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad \text{where } y_i = (y_{i,1}, y_{i,2}, y_{i,3}) \in \{0,1\}^3$$

three binary models are generated, one for each label. However, the predictions of a previous model are used for new predictions by a subsequent model, where each model (except the first) incorporates predictions from preceding steps as additional features.

Classifier for Y_1: Trained on the dataset

$$\mathcal{D}_1 = \{(x_i, y_{i,1})\}_{i=1}^{n}$$

$$f_1 : \mathbb{R}^d \to \{0, 1\}.$$

Classifier for Y_2: Uses the original features x_i and the true label $y_{i,1}$ as an additional feature. Trained on the dataset

$$\mathcal{D}_2 = \{(x_i, y_{i,1}), y_{i,2}\}_{i=1}^{n}$$

$$f_2 : \mathbb{R}^{d+1} \to \{0, 1\}.$$

Classifier for Y_3: Uses x_i and the true labels $y_{i,1}$ and $y_{i,2}$ as additional features. Trained on the dataset

$$\mathcal{D}_3 = \{(x_i, y_{i,1}, y_{i,2}), y_{i,3}\}_{i=1}^{n}$$

$$f_3 : \mathbb{R}^{d+2} \to \{0, 1\}.$$

Prediction Phase: For a new instance x, predictions are made sequentially:

$$\hat{y}_1 = f_1(x), \quad \hat{y}_2 = f_2(x, \hat{y}_1), \quad \hat{y}_3 = f_3(x, \hat{y}_1, \hat{y}_2).$$

The final prediction is the vector:

$$(\hat{y}_1, \hat{y}_2, \hat{y}_3).$$

For new instances, the three label predictions are considered (MOYANO et al., 2019) (TIDAKE; SANE, 2018). This process is illustrated in Figure 4.

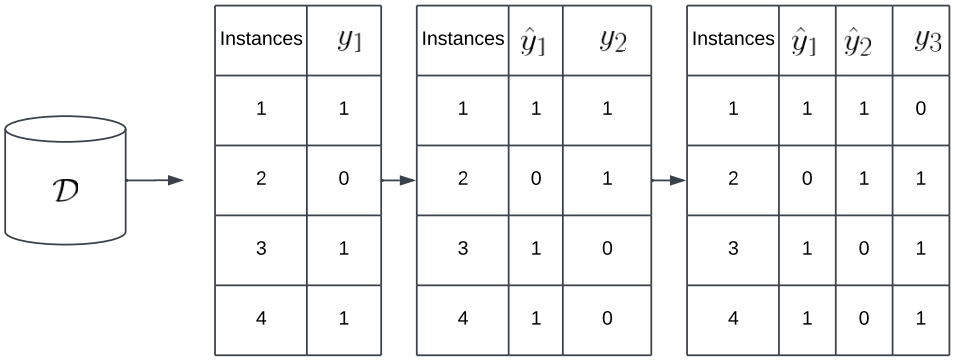


Figure 4 – Transformation of the multi-label problem into a single-label problem using the CC method. Adapted from (MOYANO et al., 2019).

PCC is a probabilistic extension of the multi-label CC algorithm, in which the conditional probability of each label combination is computed. As an extension of CC, PCC models label dependencies, where each classifier predicts a label using the original features along with the probabilistic predictions of the previous labels as additional inputs (CHEKINA et al., 2013).

For example, given original multi-label dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad \text{where } y_i = (y_{i,1}, y_{i,2}, y_{i,3}) \in \{0,1\}^3$$

For a new instance x, the prediction of the label combination (y_1, y_2, y_3) is done via:

$$P(y_1, y_2, y_3 \mid x) = P(y_1 \mid x) \cdot P(y_2 \mid x, y_1) \cdot P(y_3 \mid x, y_1, y_2).$$

The final prediction can be obtained by the label combination that maximizes the joint probability:

$$\hat{y} = \arg \max_{(y_1,y_2,y_3)\in\{0,1\}^3} P(y_1, y_2, y_3 \mid x).$$

This approach explicitly calculates the joint probability using the probabilistic chain rule, capturing the dependencies between labels. One disadvantage is computational complexity, and a larger number of labels will result in higher computational costs (HERRERA et al., 2016).

2.2.3 Label Powerset (LP)

The Label Powerset (LP) method formulates multi-label classification as a multi-class problem, where each unique label combination constitutes a class, as illustrated in Figure 5.

For example, given an original multi-label dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad \text{where } y_i = (y_{i,1}, y_{i,2}, y_{i,3}) \in \{0,1\}^3$$

each unique combination of labels is considered a class:

$$\mathcal{C} = \{y_i \mid i = 1, \ldots, n\} \subseteq \{0,1\}^l$$

For a new instance, the classifier predicts a class that corresponds to a set of labels (SOROWER, 2010).

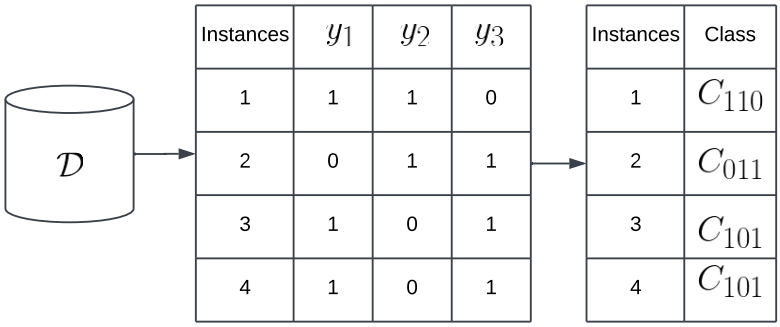


Figure 5 – Transformation of the multi-label problem into a single-label problem using the LP method. Adapted from (MOYANO et al., 2019).

One advantage is that it considers the correlation between labels. Still, a disadvantage is that if the problem has a large number of label sets, many classes may be associated with a small number of examples (GANDA; BUCH, 2018).

2.3 AUTOMATED MACHINE LEARNING

In (ZÖLLER; HUBER, 2021), AutoML is defined as a way to favor the use of machine learning by domain experts, without them having extensive knowledge of the area itself. Domain experts are individuals with extensive knowledge in the domain where machine learning will be applied (KARMAKER et al., 2021).

In the AutoML approach, when considering the task of selecting the best pipeline for a given dataset, the process typically contains the following components: a predefined computational budget (which limits the computational cost of the search) and an evaluation metric, which allows for comparing different pipeline configurations during the search for the best solution. AutoML explores a search space composed of all possible combinations. To navigate this space,

optimization strategies such as Grid Search, Random Search, and Bayesian optimization are applied, aiming to find the best configuration within the established constraints (NGUYEN et al., 2022).

There are different optimization techniques for finding optimal pipelines. These strategies include: Grid and Random Search, Bayesian Optimization, and Genetic Programming. Each of these strategies is briefly described below.

- Grid and Random Search: Grid search is a foundational technique for exploring a predefined configuration space through an exhaustive search (ZÖLLER; HUBER, 2021). It works by evaluating all possible combinations, checking each one until the best-performing configuration is identified. In random search, combinations are not analyzed in the order in which they were predefined; that is, they are analyzed randomly (ALSHAREF et al., 2022).

- Bayesian optimization: This is one of the most popular strategies in AutoML systems (CELIK; VANSCHOREN, 2021), which is based on the Bayes theorem. It involves exploring the search configuration space by building a model that takes previous evaluations into account. The model then makes decisions about where to explore, based on previous evaluations, evaluating new regions and promising configurations. There are two important decisions when using Bayesian optimization: the first involves selecting a prior function that makes assumptions about the function being optimized. The second involves choosing an acquisition function that will allow deciding which configuration to evaluate next, based on the posterior model. (SNOEK; LAROCHELLE; ADAMS, 2012).

- Genetic programming: based on Charles Darwin's theory of evolution. (ALSHAREF et al., 2022). Initially, a population of random pipelines, also called a population of individuals, is created. This population of *pipelines* is evaluated at each generation, and in each generation, crossovers and mutations are selected and performed to achieve the best solution at the end (OLSON et al., 2016) (GIJSBERS, 2022).

AutoML systems are designed with robust approaches to find the best solutions to problems. Despite efforts to build efficient AutoML systems, these systems face limitations that hinder their use. A comparative study evaluated the performance of various AutoML tools under different time limits, revealing that memory and time issues are among the main causes of failure (GIJSBERS et al., 2022). It was observed that the more time available for the search, the greater the tendency for failures to occur. This is partly because AutoML systems often

store a large number of candidate models during the search process, which can exceed the available memory constraints. Additionally, the size of the dataset represents another challenge. Large datasets require more computational resources, potentially leading to interruptions in the optimization process, as they tend to demand more computational resources, such as memory (ERICKSON et al., 2020).

2.4 FINAL CONSIDERATIONS

The objective of this chapter is to provide a basis for the main concepts of this work. The concepts of Meta-Learning, Multi-Label Algorithms, and Automated Machine Learning (AutoML) were discussed.

Meta-learning was introduced as an approach that enables learning from past machine learning experiences through metadata, supporting tasks such as algorithm selection and hyperparameter optimization. Multi-label algorithms were presented as strategies for dealing with problems in which each instance can be associated with multiple labels, illustrating transformation and adaptation methods such as Binary Relevance (BR), Classifier Chains (CC), and Label Powerset (LP). Finally, AutoML was described as a method to automate the entire machine learning pipeline, aiming to make model development more accessible and efficient.

The integration of these three areas provides the foundation for this work, enabling the development of an AutoML method based on meta-learning that automates pipeline recommendation through the use of multi-label algorithms to suggest a set of candidate pipelines.

3 METAML: A MULTI-LABEL META-LEARNING APPROACH FOR PIPELINE RECOMMENDATION

Cynthia Moreira Maia, Lucas B. V. de Amorim, George D. C. Cavalcanti, Rafael M. O. Cruz

This chapter has been submitted Machine Learning journal.

Abstract

In the machine learning (ML) literature, AutoML refers to the automated definition of a sequence of necessary steps to achieve an ML task, such as classification. Each of these steps of the ML pipeline, involving data preprocessing and algorithm selection, normally allows extensive variation, leading to large search spaces which makes it hard to find optimal pipelines for a certain problem. Most of the approaches so far presented to carry out AutoML rely on Bayesian optimization methods that have shown to be successful, albeit at high computational costs. Therefore, we propose a method that employs meta-learning (MtL) for recommending pipelines, taking into account the interdependence of its steps. MtL allows us to shift the computational complexity to an offline training phase. At the same time, we approach the search space complexity problem by designing an algorithm that carefully curates the pipeline candidates based on past ML experiments, optimizing the training and effective performance of the pipeline recommendation model. An analysis using 152 datasets shows that MetaML achieves final classification performance equivalent or superior to state-of-the-art methods but incurs much lower computational times. The source code for the experiments is available at the project's repository[1].

Keywords: Meta-Learning, Pipeline, Multi-Label, AutoML.

3.1 INTRODUCTION

When machine learning (ML) practitioners are faced with a new task, they must develop a pipeline for dealing with all the steps necessary to extract useful insights from the data. This includes data preprocessing procedures (ALASADI; BHAYA, 2017), such as data balancing (AVELINO; CAVALCANTI; CRUZ, 2024), scaling (AMORIM; CAVALCANTI; CRUZ, 2023b), feature

[1] https://github.com/cynthiamaia/MetaML

selection (ZEBARI et al., 2020) and the choice and configuration of the ML algorithm to use (KOTSIANTIS; ZAHARAKIS; PINTELAS, 2006). This can be a time-consuming task that frequently demands expert knowledge. Consequently, researchers have invested in ways to automatize this task, giving rise to the area of automated machine learning (AutoML), which aims at generating effective machine learning pipelines with little to no user interaction and without requiring ample domain knowledge (ELSHAWI; MAHER; SAKR, 2019; BAHRI et al., 2022).

Traditionally, AutoML has been approached via optimization techniques, including genetic programming (OLSON; MOORE, 2016), Bayesian optimization (THORNTON et al., 2013; KOTTHOFF et al., 2017), and random search (LEDELL; POIRIER, 2020). For every new dataset, these techniques iteratively search for good pipelines within a search space defined by all possible configurations of pipelines to be searched. While they have been successful in finding useful, if not optimal, pipelines, these techniques often take a blind or naive approach, using a broad set of configurations without prior study to identify the most effective ones, which leads to a costly optimization procedure with redundant or irrelevant configurations. Although researchers have also employed collaborative filtering to avoid the cost of more robust and extensive optimization techniques (YANG et al., 2019), it has the limitation of still relying on past results of some of the pipeline candidates on the query dataset, which is a computationally demanding task to overcome during the recommendation phase.

Defining a wide and yet viable search space and the time budget for an AutoML process is still a challenge (BAHRI et al., 2022; ELSHAWI; MAHER; SAKR, 2019). While a more comprehensive search space increases the chances of finding an optimal solution, it demands a greater time budget and more computational resources such as memory, CPU and GPU (ELSHAWI; MAHER; SAKR, 2019; BRAZDIL et al., 2022b). Nonetheless, this trade-off can be circumvented, as a small search space, with carefully curated configurations, can lead to superior performance than a wider search space under the same time constraint (ELDEEB et al., 2022).

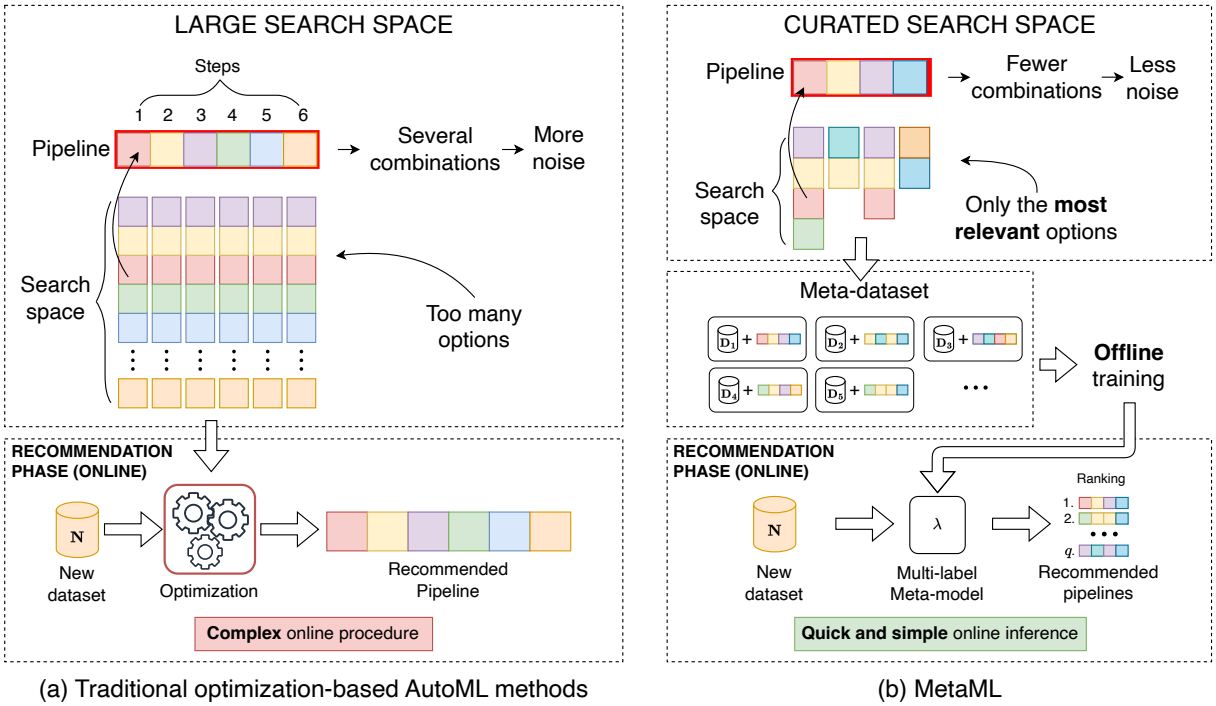
Therefore, the design of an AutoML system that is effective in finding good pipelines and still computationally efficient is a challenge. A desired solution to the problem is to find an algorithm that, given a query dataset, can instantly recommend one or more suitable pipelines after efficiently probing its characteristics. For this purpose, meta-learning is a reasonable solution (BRAZDIL et al., 2008). It learns predictive models that can quickly recommend pipelines based on dataset characteristics (a.k.a. meta-features) (HUTTER; KOTTHOFF; VANSCHOREN, 2019). To do this, it models the relationship between past datasets' meta-features and a meta-target, e.g., the performances achieved by all candidate pipelines on these datasets.

Meta-learning leverages knowledge from previously observed meta-examples stored in a meta-dataset to provide informed initial solutions, reducing the number of evaluations needed during the search. This allows optimization algorithms to explore the space more efficiently, accelerating convergence to suitable settings. Some studies have proposed the use of meta-learning to guide the optimization process by recommending an initial population of promising configurations. For example, Gomes et al. (2012) and Miranda et al. (2014) combined meta-learning with optimization algorithms to recommend parameters for predictor algorithms. Other studies use meta-learning to predict whether or not a hyperparameter optimization approach is likely to improve a classifier (MANTOVANI et al., 2019). Probst, Boulesteix e Bischl (2019) and Rijn e Hutter (2018) investigate the importance of hyperparameter tuning through meta-learning. The studies highlighted above use metamodels based on classical machine learning algorithms (such as k-Nearest Neighbors or Random Forest) to recommend specific hyperparameters or decide whether tuning is necessary.

On the other hand, works such as (WANG et al., 2014; ZHANG; SONG, 2015; DANTAS; POZO, 2018; KHAN et al., 2020; ZHU et al., 2021; GOLSHANRAD et al., 2021; WEGMETH; BEEL, 2022) expand this scope by adopting multi-label models, where each label encodes the option of each hyperparameter and the algorithm. These applications have performed well both in terms of recommendation quality and computational cost. However, the literature has not shown a significant trend in utilizing meta-learning as the primary process in an AutoML system that recommends an entire ML pipeline. In particular, when it comes to learning the relationship between multiple steps in a pipeline, such as data preprocessing, feature selection, and the predictive algorithm.

Considering the described context, we hypothesize that by (i) designing a carefully curated, reduced search space and (ii) using meta-learning with multi-label meta-models that take into account the interdependence of pipeline steps, it is possible to recommend pipelines that are equivalent or superior to the ones generated by state-of-the-art AutoML systems but incurring much smaller computational cost. Using these two main directives, we therefore propose the MetaML framework, a meta-learning approach based on probabilistic multi-label classification algorithms for recommending rankings of full machine-learning pipelines. Multi-label classifiers are ideal for this application because they can learn the relationships between the steps of a pipeline, e.g., a pipeline that uses random forest as its classifier will not need data scaling. By learning these relationships, MetaML is able to perform well by focusing only on the relevant configurations.

Figure 6 contrasts traditional optimization-based AutoML methods (a) with our proposal (b). While traditional methods rely on wide and generalist search spaces, our method curates the search space based on previously executed ML experiments, taking only the most relevant pipeline configurations.



(a) Traditional optimization-based AutoML methods (b) MetaML

Figure 6 – Comparison between traditional AutoML (a) and our proposed meta-learning based framework MetaML (b).

As search spaces grow, as in Figure 6, the likelihood of generating redundant pipelines can increase (FEURER et al., 2020). This is analogous to hyperparameter optimization, where small variations in certain hyperparameters often do not significantly affect performance on certain datasets, which can lead to redundant configurations (BERGSTRA; BENGIO, 2012). The idea behind search space curation is focusing on less, but statistically more relevant pipelines that are expected to be complimentary. This also enables a clearer, less noisy meta-dataset to train the meta-model. For a new dataset, traditional methods perform a search for the optimal pipeline within their wide search space. This is done online during the recommendation phase, which delays the system's response. On the other hand, our method learns offline the relationships between dataset features and their optimal pipelines. For a new dataset, our method extracts dataset features and performs a quick meta-model inference to recommend a ranking of n pipelines based on class probabilities obtained from the multi-label meta-model, where the n is user-defined and can be set to just one pipeline or multiple pipelines to be evaluated through a cross-validation procedure to select the best one. This difference in

approaches directly impacts the performance that can be achieved with limited time budgets. When comparing the performance achieved by the MetaML versus the performance of the state-of-the-art AutoML methods with various time budgets, our analysis shows that MetaML achieves high performance in a short time, while, on average, the other methods require more time to find good pipelines.

The main contributions of this paper are:

- A new approach to AutoML, using meta-learning with multi-label algorithms recommending a set of n candidate pipelines.
- A novel way to heuristically reduce the search space by automatically curating the most relevant candidate pipelines based on historical data. This allows for more efficient meta-model training and more effective pipeline recommendations.
- A method of composing pipelines that is based on the chained recommendations of their steps in such a way that the interdependence of the steps is taken into account.
- A comparative performance analysis, using 152 datasets, in terms of accuracy and computational time of the proposed framework against several state-of-the-art AutoML methods.
- We show that MetaML recommends pipelines with performance comparable to or better than state-of-the-art methods while requiring significantly less computational time.

This paper is organized as follows: Section 3.2 presents an overview of the main concepts of automated machine learning (AutoML), meta-learning, and multi-label classification algorithms; Section 3.3 exhibits a review of the related works. Section 3.4 presents the proposed framework. The experimental methodology is detailed in Section 5.2. Sections 5.3, 3.7 and 3.8 discuss the results, threats to validity and conclude this research, respectively.

3.2 BACKGROUND

This work involves three main research areas: meta-learning, multi-label algorithms, and automated machine learning. Therefore, this section briefly discusses these main areas of research.

3.2.1 Meta-Learning

The Algorithm Selection Problem (ASP) was formulated by Rice in 1976 and refers to the challenge of choosing the most appropriate algorithm to solve a specific problem, considering that there are several algorithms available. Rice's model for representing the ASP is composed of four basic components (KHAN et al., 2020)(RICE, 1976), as illustrated in Figure 7.

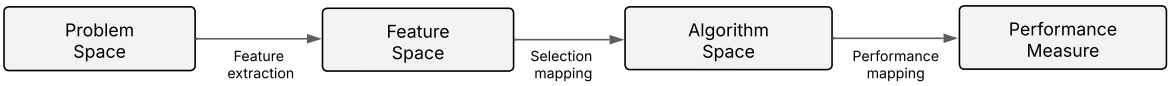


Figure 7 – Representation of the four components ASP, adapted from (RICE, 1976).

Figure 7 provides a representation of the relationships between the components of the ASP. The four components are described below:

- **Problem space** refers to the domain of the problem that needs to be solved, e.g., classification task.
- **Feature space** corresponds to the characteristics that describe the problem, such as the number of instances.
- **Algorithm space** involves the set of algorithms that can be applied to solve the problem.
- **Performance measure** represents the metric used to evaluate the performance of each algorithm in the context of the specific problem, e.g., accuracy.

ASP is one of the focuses of research in the area of Meta-Learning (MtL) (SMITH-MILES, 2009) (SONG; WANG; WANG, 2012) (SOUTO et al., 2008) (PIMENTEL; CARVALHO, 2019) (FERRARI; CASTRO, 2015). Another key focus of MtL is CASH (Combined Algorithm Selection and Hyperparameter Optimization) (BRAZDIL et al., 2022a), which integrates the algorithm selection problem with hyperparameter optimization.

MtL makes it possible to learn from previous machine-learning experiences (HUTTER; KOTTHOFF; VANSCHOREN, 2019). For example, considering the algorithm recommendation problem, the approach contains a base-level and a meta-level task. The representation of algorithm recommendation is illustrated in Figure 8. At the base level (A) of Figure 8, it is decided which algorithms the system can recommend and the performance metric that will be used to evaluate the performance of the algorithms on the datasets. Subsequently, the meta-dataset, which is composed of meta-examples, is built. Meta-examples are datasets,

represented by their characteristics (meta-features), associated with the performances of the algorithms on those datasets (meta-targets). At the meta-level (B) of Figure 8, a predictive algorithm is employed to build a meta-model, which recommends the base-level algorithm. The type of meta-model algorithm depends on the type of meta-target that will be used for recommendation (BRAZDIL et al., 2008), but classification and regression algorithms are among the most commonly used. Thus, given a new dataset, its meta-features are extracted and the meta-model is used to recommend the base-level algorithms that are more likely to perform well on this dataset.

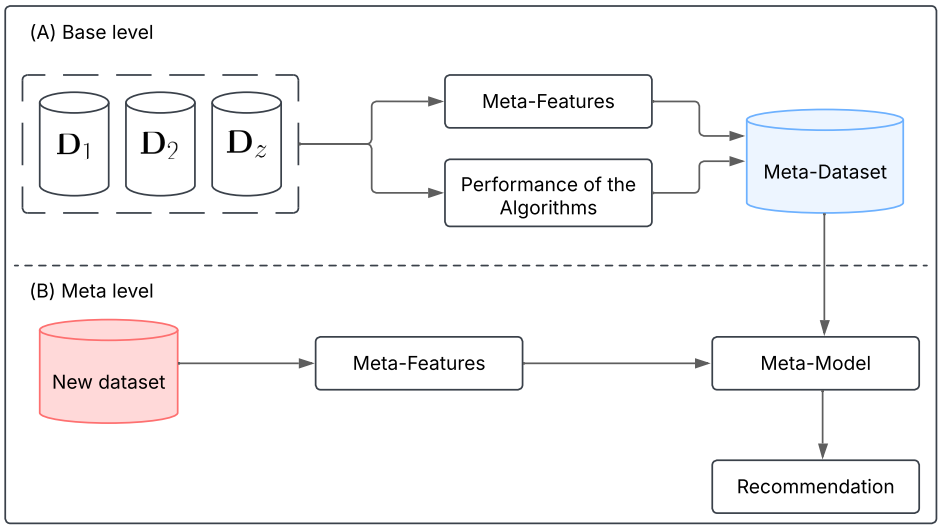


Figure 8 – Recommendation with Metalearning, adapted from (BRAZDIL et al., 2008).

Meta-features are a collection of measures designed to characterize datasets. They can be organized into different groups (BRAZDIL et al., 2022a): simple and statistical, information theory, model-based, complexity-based and landmarkers. Table 1 summarizes these groups, providing a brief description and examples of each. Simple meta-features provide general information about datasets, such as the number of classes, attributes, and instances. Statistical meta-features collect statistical measures from datasets, such as standard deviation, mean, and coefficient of variation. The information theory group includes measures such as entropy of the class and attributes (CASTIELLO; CASTELLANO; FANELLI, 2005). Model-based are measurements that reflect the properties of a specific model; for example, the number of leaf nodes of a simple decision tree trained on the dataset (BRAZDIL et al., 2008). Performance-based meta-features, also known as landmarking, are predictive performance measures of specific models on the dataset; for example, the performance of the Naive Bayes (BRAZDIL et al., 2022a) algorithm. Finally, complexity meta-features allow analyzing the complexity of datasets in different

aspects, such as class separability and attribute overlap (RIVOLLI et al., 2022).

Table 1 – Types of Meta-features.

Group	Description	Examples
Simple	General information about the datasets.	Number of classes, attributes.
Statistical	Statistical measures of the together of datasets.	Skewness, kurtosis.
Info-theory	Measures based on information theory.	Class entropy and mutual information.
Model-based	Measures based on the properties of a model.	Number of leaf nodes (decision tree).
Landmarking	Measures predictive performance of the algorithm on the datasets.	Performance of the naive bayes algorithm.
Complexity	Allow analyzing the complexity of sets of data in different aspects.	The separability of classes.

A meta-target is the predicted variable in the meta-level problem. It can be formed by different approaches: best algorithm, ranking, and a subset of algorithms (BRAZDIL et al., 2008). If the best algorithm approach is considered, a single algorithm will be returned as the meta-model prediction, the one that is expected to perform better on the dataset than the others in the search space. When using the ranking approach, a ranking of the best-performing algorithms w.r.t their probabilities is returned, whereas when you want to return a subset of algorithms, more than one algorithm with equivalent performances is indicated. The choice of meta-target also affects the choice of the meta-model, thus, to recommend several algorithms simultaneously, some studies make use of multi-label classification algorithms such as meta-models (ZHU et al., 2021) (KHAN et al., 2020).

3.2.2 Multi-Label Algorithms

Multi-label classification allows an instance to be associated with more than one label simultaneously (TSOUMAKAS; KATAKIS; VLAHAVAS, 2009). It can be grouped into two approaches: problem transformation and algorithm adaptation. In algorithm adaptation, which is algorithm-dependent, the methods of machine learning algorithms that deal with single-label problems are extended so that they address the multi-label classification problem. In problem transformation, the multi-label classification problem is transformed into one or more single-label problems, being independent of the algorithm (TSOUMAKAS; KATAKIS; VLAHAVAS, 2009). Some problem transformation algorithms are Label Powerset (LP) (TSOUMAKAS; KATAKIS; VLAHAVAS, 2010), Classifier Chains (CC) (READ et al., 2009), and Probabilistic Classifier Chains (PCC) (CHENG; HÜLLERMEIER; DEMBCZYNSKI, 2010).

In the Label Powerset approach, the multi-label problem is transformed into a multi-class problem where the target can be any unique combination of the original labels, and then, for a new instance, the classifier predicts a set of labels (SOROWER, 2010). In the Classifier Chains

(CC) method, a chain of classifiers is formed in a way that each classifier predicts one of the multiple class labels, such that previous label predictions are considered to predict subsequent labels in the chain. First, a classifier is trained using only the original meta-features. Then, the first output label is added as a new input feature, forming a new input space composed of the original meta-features and the previous label prediction. As an extension of the classifier chain approach, Probabilistic Classifier Chains (PCC) considers the correlation between labels and employs Bayesian methods to optimize the chaining order of the classifiers (HERRERA et al., 2016).

3.2.3 Automated Machine Learning

AutoML can be described as the automatic definition of machine learning pipelines, avoiding manual efforts for pipeline configuration by machine learning experts as well as non-experts (BAHRI et al., 2022). A pipeline is illustrated in Figure 9. It consists of a tuple of blocks, where each block belongs to a specific category, such as Data Preprocessing (DP), Feature Engineering (FE), and Model Selection (MS). For example, a pipeline may include a Simple Imputer (SI) belonging to the set of imputation techniques, One-Hot Encoding (OHE) as part of the categorical encoding techniques, and a Standard Scaler (SS) included in the set of scaling techniques. All these techniques fall under the Data Preprocessing (DP) category. In the Feature Engineering (FE) category, methods such as Principal Component Analysis (PCA) belong to the set of Feature Transformation techniques. Meanwhile, the Model Selection (MS) category consists of a set of classifiers, which may include algorithms such as the Support Vector Classifier (SVC).

Figure 9 – A representation of an example of pipeline as a tuple of blocks. Each block is a step in the ML pipeline.

Some of the objectives of AutoML are: reducing human effort for applying machine learning, improving model performance, and reproducibility of scientific studies (HUTTER; KOTTHOFF; VANSCHOREN, 2019). Furthermore, high computational efficiency is sought to ensure favorable performance within a limited computational budget (YAO et al., 2018).

The typical components of an AutoML system are search space, metrics, computational budget, and optimization technique. These components are illustrated in Figure 10. Given a

dataset, the AutoML system performs a search process to find the best pipeline. All possible pipelines that the AutoML system can recommend are contained in its search space. The computational budget is the time and computational power allocated to the AutoML system so that it searches for the optimal pipeline within that budget (ELSHAWI; MAHER; SAKR, 2019). Optimization techniques allow finding the best pipeline in the search space according to the defined evaluation metric (NGUYEN et al., 2022), that is, the metric allows evaluating the pipelines found so that it can recommend the best one in the end. There are different optimization techniques in the search for optimal pipelines. These strategies include grid and random searches (ZÖLLER; HUBER, 2021), bayesian optimization (CELIK; VANSCHOREN, 2021), and genetic programming (ALSHAREF et al., 2022) (GIJSBERS, 2022) .

Grid search performs an exhaustive search in a predefined space. The combinations are analyzed in the order in which they were predefined. All these combinations are checked individually until the best one is returned. In random search, the combinations are not analyzed in the order in which they were predefined; that is, they are analyzed randomly (ZÖLLER; HUBER, 2021). Bayesian optimization (CELIK; VANSCHOREN, 2021) is one of the most popular strategies in AutoML systems, which is based on Bayes' theorem. It involves exploring the search space using probabilistic models that consider previous evaluations. Then, the model decides where to explore, evaluating new regions and promising configurations, focusing on the most likely to produce good results to determine the optimal pipeline configuration. Genetic Programming (GIJSBERS, 2022), is a kind of evolutionary algorithm, which is based on Charles Darwin's theory of evolution (ALSHAREF et al., 2022). Initially, a population of random pipelines is created, also called a population of individuals. This population of pipelines is evaluated at each generation, and in each generation, crossovers and mutations are carried out on the individuals, which are then selected to achieve the best solution in the end.

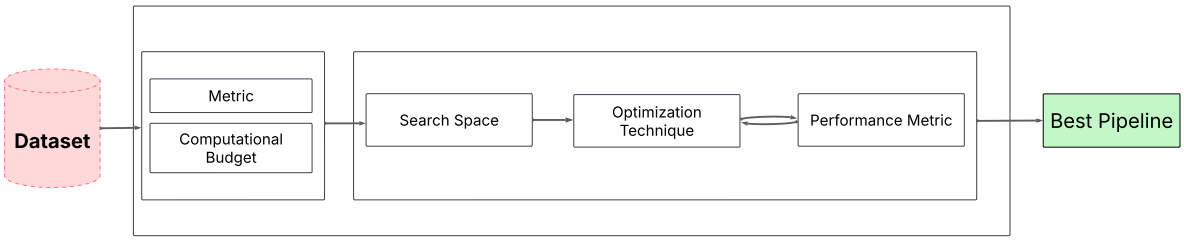


Figure 10 – Representation typical AutoML components, adapted from (HUTTER; KOTTHOFF; VANSCHOREN, 2019).

In addition to the typical components listed above, some AutoML systems complement

their approaches by using meta-learning to "warm-start" the search process, thus reducing overall complexity. That is, instead of starting the optimization by randomly selecting configurations, they start the search process using past performances of configurations on similar datasets to define an initial configuration to be further optimized (KARMAKER et al., 2021).

3.3 RELATED WORK

This section contrasts the state-of-the-art AutoML frameworks with the proposed MetaML. There are several studies in the literature that have explored meta-learning to automate specific steps of the machine-learning pipeline, focusing primarily on isolated components rather than end-to-end pipeline recommendations.

For example, in Gomes et al. (GOMES et al., 2012), meta-learning was combined with two search algorithms, Particle Swarm Optimization and Tabu Search to recommend two parameters for Support Vector Regressor (SVR): the RBF kernel parameter and the regularization constant C. In Miranda et al. (MIRANDA et al., 2014), a hybrid architecture that combines meta-learning with multi-objective optimization (MOO) techniques is used to select the γ and C parameters, the success rate and number of support vectors (which indicate complexity), of an Support Vector Machine (SVM) classifier. In Mantovani et al. (MANTOVANI et al., 2019), a meta-learning-based recommendation system is proposed to predict whether tuning will bring significant performance gains for SVM, in which it avoids unnecessary optimizations without compromising predictive performance.

In Zhang and Song (ZHANG; SONG, 2015), a method was proposed for recommending the Kernel hyperparameter of the SVM algorithm, which used meta-learning. In the work of Zhu et al. (ZHU et al., 2021), the use of meta-learning was proposed to recommend classification algorithms. While these works focus on deciding whether hyperparameter tuning is necessary, or on recommending algorithms, they do not seek the recommendation of various pipeline steps, which is the main focus of this work.

Thus, we now discuss AutoML frameworks that address multiple pipeline steps, such as FE - Feature Engineering, MS - Model Selection, HO - Hyperparameters Optimization, and DP - Data Preprocessing, which is similar to our goal of providing recommendations at different steps of the ML pipeline.

Data Preprocessing (DP) refers to the transformations applied to raw data to prepare it for modeling, such as imputing missing values, scaling, and encoding categorical. Feature

Engineering (FE), on the other hand, involves transformations aimed at creating new features or selecting the most informative features.

Table 2 lists the most popular open-source AutoML frameworks and their characteristics and includes MetaML for comparison.

The optimization methods employed by the various AutoML frameworks exhibit a diverse range of strategies. Bayesian optimization (FEURER et al., 2015; THORNTON et al., 2013; KOTTHOFF et al., 2017), random search (MOHR; WEVER, 2023; GIJSBERS; VANSCHOREN, 2020; LEDELL; POIRIER, 2020), genetic programming (OLSON; MOORE, 2016), and collaborative filtering (YANG et al., 2019) are among the techniques employed to navigate extensive search spaces and identify optimal machine learning pipelines. We organized the section according to the optimization types used in AutoML frameworks.

Table 2 – Comparison of the most popular open-source AutoML frameworks. FE - Feature Engineering, MS - Model Selection, HO - Hyperparameters Optimization, DP - Data Preprocessing.

AutoML Framework	Optimization	Search Space (SS)				SS Curation	Meta-Learning	Online Search
		FE	MS	HO	DP			
Auto-WEKA(THORNTON et al., 2013; KOTTHOFF et al., 2017)	SMAC	√	√	√	×	×	×	√
AutoSklearn 1.0(FEURER et al., 2015)	SMAC	√	√	√	√	×	√	√
AutoSklearn 2.0(FEURER et al., 2022)	SMAC	√	√	√	√	√	√	√
GAMA(GIJSBERS; VANSCHOREN, 2020)	RS;ASHA;AMOEA	√	√	√	√	×	×	√
Hyperopt-Sklearn(KOMER; BERGSTRA; ELIASMITH, 2019)	RS;TPE;Annealing	√	√	√	√	×	×	√
ML-Plan(MOHR; WEVER; HÜLLERMEIER, 2018)	PHRT	√	√	√	√	×	×	√
Naive AutoML(MOHR; WEVER, 2023)	RS	√	√	√	√	×	×	√
TPOT(OLSON; MOORE, 2016)	GP	√	√	√	√	×	×	√
TABPFN(HOLLMANN et al., 2025)	Transformer-based	√	√	√	√	×	√	×
AutoGluon(ERICKSON et al., 2020)	Multi-Layer Stack	×	√	√	×	×	×	×
FLAML-Zero(KAYALI; WANG, 2022)	Meta-learning only	×	√	√	×	√	√	×
H2O AutoML(LEDELL; POIRIER, 2020)	RS	×	√	√	×	×	×	√
OBOE(YANG et al., 2019)	CF	×	√	√	×	×	√	×
MetaML	Meta-learning only	√	√	×	√	√	√	×

3.3.1 AutoML using Bayesian Optimization

The frameworks Auto-WEKA (THORNTON et al., 2013; KOTTHOFF et al., 2017), AutoSklearn 1.0 (FEURER et al., 2015) and AutoSklearn 2.0 (FEURER et al., 2022) are examples that uses Bayesian Optimization with Sequential Model-based Algorithm Configuration (SMAC). Auto-Weka pioneered the use of Bayesian optimization in the search for the optimal pipeline, for which it uses the SMAC. AutoSklearn 1.0 uses it only to warm start a SMAC optimization procedure, which constitutes the bulk of its recommendation strategy. It uses meta-learning to initialize the bayesian optimizer SMAC, in which the meta-learner selects the k configurations to be used by optimization. Thus, when dealing with a new dataset, its meta-features are evaluated, and the instances are selected considering the k closest datasets for evaluation.

Autosklearn 2.0 is an extension of Autosklearn 1.0, with improvements focused on reducing the search space and improving the efficiency of the results. The main changes include: a) the use of iterative learning algorithms, limiting preprocessing techniques (feature preprocessing); b) an improved meta-learning strategy, in which a fixed portfolio of pipelines is formed. Instead of using the Auto-sklearn 1.0 approach, where the hot search is initialized based on the meta-features of a new dataset (evaluating the k closest instances to choose the best configuration), Auto-sklearn 2.0 uses POSH (Portfolio Successive Halving). This strategy uses a static portfolio of pipelines generated offline. This portfolio is calculated from multiple datasets to ensure the initial set covers many potential problems. The SH strategy is used to efficiently allocate the budget between different configurations, progressively adjusting the resource allocation in order to focus more on promising pipelines while discarding the less promising ones. The idea of using a portfolio is in line with the proposal by Wistuba et al. (WISTUBA; SCHILLING; SCHMIDT-THIEME, 2015), an approach for hyperparameter tuning based on a portfolio of hyperparameter configurations. The idea is to reduce the computational cost of exploring the hyperparameter space on new problems. To achieve this, the NN-SMFO (Nearest Neighbor Sequential Model-Free Optimization) strategy is used to optimize the hyperparameter search considering the similarity between datasets. Instead of evaluating all hyperparameter configurations, it possible focuses on those that are most similar to the data already evaluated, using the KTRC distance function to measure this similarity. In addition, the CANE (Cumulative Average Normalized Error) metric measures how efficiently a strategy converges quickly to the best hyperparameter configuration. This combination of techniques makes the search for hyperparameter configurations more efficient.

Autosklearn 2.0 has similar goals to MetaML in curating search spaces that address a wide range of problems. Autosklearn 2.0 focuses on building a portfolio of pipelines based on evaluations on different datasets. In contrast, our goal is not to build a portfolio through pipeline evaluations. Instead, we leverage historical data from repositories that provide results from machine learning experiments. This allows us to identify more representative pipelines that are capable of covering a wide of problems without the need for extensive evaluations. In this way, we aim to reduce the computational overhead associated with evaluating different pipelines, which we highlight as a unique contribution of our work.

These methods perform an online search. The pipeline steps considered within the search space by the AutoML frameworks vary in their inclusiveness. The frameworks Auto-WEKA (THORNTON et al., 2013; KOTTHOFF et al., 2017), AutoSklearn 1.0 (FEURER et al., 2015) and AutoSklearn

2.0 (FEURER et al., 2022) address the full machine learning pipeline, encompassing DP[2], FE, MS, and HO.

3.3.2 AutoML using Random Search

Frameworks like General Automated Machine learning Assistant (GAMA) (GIJSBERS; VANSCHOREN, 2020), Hyperopt-Sklearn (KOMER; BERGSTRA; ELIASMITH, 2019), Naive AutoML (MOHR; WEVER, 2023), and H2O AutoML (LEDELL; POIRIER, 2020) use the Random Search (RS) technique.

In the GAMA three optimization techniques are available, which are: RS, Asynchronous Successive Halving Algorithm (ASHA) and Asynchronous Multi-Objective Evolutionary Algorithm (AMOEA) (GIJSBERS; VANSCHOREN, 2020). Finally, a post-processing technique can be used to create a committee of the evaluated pipelines. In Hyperopt-Sklearn, there are different search strategies, such as: RS, TPE (Tree of Parzen Estimators) and Annealing.

The idea behind Naive AutoML is to propose a simple baseline method to which the more complex ones can be compared. While most methods consider the steps of a pipeline to be interdependent, Naive AutoML proposes a "naive" approach where each step is independently optimized via RS, and the final pipeline is built by simply using the best algorithm for each step. The greatest advantage of this approach is that it reduces the search space considerably, making it more efficient in terms of computational cost. Surprisingly, this simpler approach has been shown to achieve comparable and sometimes better results than more complex methods (MOHR; WEVER, 2023). Instead of assuming inter-independence within the steps to reduce the search space, MetaML focuses on a strategy to automatically curate the instances in the search space, leaving only the most relevant pipelines based on historical data. This optimizes MetaML's training and effective performance, allowing quick recommendations of full ML pipelines that take into account the relationship between the steps of the pipeline.

Instead of assuming inter-independence within the steps to reduce the search space, MetaML focuses on a strategy to automatically curate the instances in the search space, leaving only the most relevant pipelines based on historical data. This optimizes MetaML's training and effective performance, allowing quick recommendations of full ML pipelines that take into account the relationship between the steps of the pipeline.

H2O AutoML makes use of RS technique and its framework encompasses the use of

[2] Except Auto-WEKA.

stacked for post-processing, i.e., Stacked involves training a high-level model to combine the predictions of lower-level models to obtain a better final prediction (TING; WITTEN, 1999).

The frameworks GAMA (GIJSBERS; VANSCHOREN, 2020), Hyperopt-Sklearn (KOMER; BERGSTRA; ELIASMITH, 2019) and Naive AutoML (MOHR; WEVER, 2023) cover the entire machine learning pipeline, including DP, FE, MS, and HO. In contrast, H2O AutoML (LEDELL; POIRIER, 2020) focuses specifically on MS and HO. All these methods perform an online search.

3.3.3 AutoML with Genetic Programming

Tree-based Pipeline Optimization Tool (TPOT) (OLSON; MOORE, 2016) uses Genetic Programming (GP) as the method for optimizing tree-based pipeline operators. There are three operators: the supervised classification algorithm, the attribute preprocessing algorithm, and the attribute selection algorithm. It addresses the entire machine learning pipeline and performs an online search.

3.3.4 AutoML other optimization types

In ML-Plan (MOHR; WEVER; HÜLLERMEIER, 2018), the strategy is through Hierarchical Task Network Planning, in which it explores a graph, through a *best-first* algorithm to identify good *pipelines*. The Tabular Prior-Data Fitted Network (TabPFN) (HOLLMANN et al., 2025) uses a transformer-based generative architecture. Initially, synthetic tabular datasets are generated, with different relationships between features and targets (meta-learning). Then, a transformer model is trained to predict the targets from these synthetic datasets. The transformer training is performed offline; it is done only once on these synthetic data. After this training, the model can make predictions for new datasets without additional training. However, it has limitations that correspond to the size of small to medium-sized datasets with limits of up to 10,000 samples and 500 features. Another limitation corresponds to the fact that although TabPFN is fast for training, it may not be ideal for real-time inference.

AutoGluon (ERICKSON et al., 2020) uses a multi-layer stack strategy, i.e., the first layer consists of several trained base models, then the predictions of the base models are concatenated and serve as input to the next layer, which is composed of *stackers* models. The predictions of the second layer are fed to the last layer, where they are aggregated into an *ensemble* to produce the final output in a *weighted* form.

The framework FLAML-Zero (KAYALI; WANG, 2022) uses meta-learning as its main recommendation strategy. Similarly to MetaML, FLAML-Zero is a zero-shot approach, hence it does not need to perform an online search. During its offline stage, it builds a set of candidate pipeline configurations, which are evaluated for performance on various tasks. After this phase, the search space is reduced to form a smaller portfolio of configurations that meet a predefined performance criteria on all of its training tasks. During the online stage, given a new test task, a decision function is used to select which portfolio configuration to apply. For this selection, a kNN meta-classifier is used. The main drawback of this approach is that it is evaluated with a search space that only includes variations of decision tree-based classifiers and their hyperparameters, neglecting the importance of preprocessing techniques. Additionally, its meta-model relies on only four simple meta-features: the number of instances, the number of features, the number of classes, and the percentage of numeric features. These simple meta-features are unlikely to provide enough predictive power if a wider search space focusing on the full pipeline was to be considered. In contrast to the FLAML approach, which relies on evaluating candidate configurations to find effective solutions, MetaML adopts a search space curation strategy based on historical data. Instead of trying out new configurations, it uses the knowledge gained from previous experiments to focus on options that have already demonstrated promising results.

AutoGluon (ERICKSON et al., 2020) and FLAML-Zero (KAYALI; WANG, 2022), focus specifically on MS and HO. OBOE (YANG et al., 2019), which is also concerned only with MS and HO, employs Collaborative Filtering (CF) as its main search strategy. However, OBOE still requires executing part of the pipelines in its search space for every new query dataset to predict the remaining pipelines' performance using CF and recommend the optimal one. This impacts its responsiveness during the recommendation phase. On the other hand, MetaML's use of meta-learning ensures that, given a query dataset, a ranking of q complete pipelines can be recommended in a fraction of the recommendation times required by these other methods. In summary, we highlight some key points from the discussion:

- Notice that other frameworks also use meta-learning to some extent: AutoSklearn 1.0 (FEURER et al., 2015), AutoSklearn 2.0 (FEURER et al., 2022) and OBOE (YANG et al., 2019).
- Notably, most frameworks adopt online search strategies, conducting computationally intensive searches during the recommendation phase. In contrast, MetaML uses meta-learning as its main and only recommendation strategy, delegating much of the compu-

tational complexity to an offline meta-model training phase, allowing for an efficient and responsive recommendation phase.

- MetaML focuses on the entire pipeline and could include hyperparameter optimization (HO). However, HO is not considered in its current version, evaluated in this study, as we prioritized the broader stages of the pipeline due to the added complexities HO introduces.
- It is important to note that none of the discussed approaches can learn a search space from historical data, as highlighted in Table 2. While FLAML-Zero Shot builds a curated search space, it initially evaluates many candidate configurations to form its final portfolio. In contrast, our approach reduces the computational cost of evaluating numerous configurations by utilizing results from previous OpenML experiments to build a more efficient curated search space. Furthermore, this approach can help to avoid biases in the meta-learning system by ensuring that the meta-level selection of pipelines is based on a diverse set of experiments.

3.4 METAML

The proposed framework, MetaML (Figure 11), comprises four phases that are detailed in the following subsections: search space definition, meta-dataset construction, training, and recommendation. The mathematical notation used in this section is listed in Table 3.

3.4.1 Search space definition

For the search space definition, we propose an algorithm that considers a historical record, with results of previous ML experiments on different datasets, to automatically design a curated search space with reduced complexity. The goal is that the search space is composed only of frequently used and high-performing pipelines in past experiments to form a more effective search space.

The rationale behind the algorithm is that in the space of all possible pipelines, there is (i) redundancy, i.e., different pipelines that achieve similar results, (ii) lots of naive and unnecessary pipelines, such as pipelines composed of combinations of steps that do not make sense (e.g., algorithms based on decision trees preceded of a data scaling step) and even (iii)

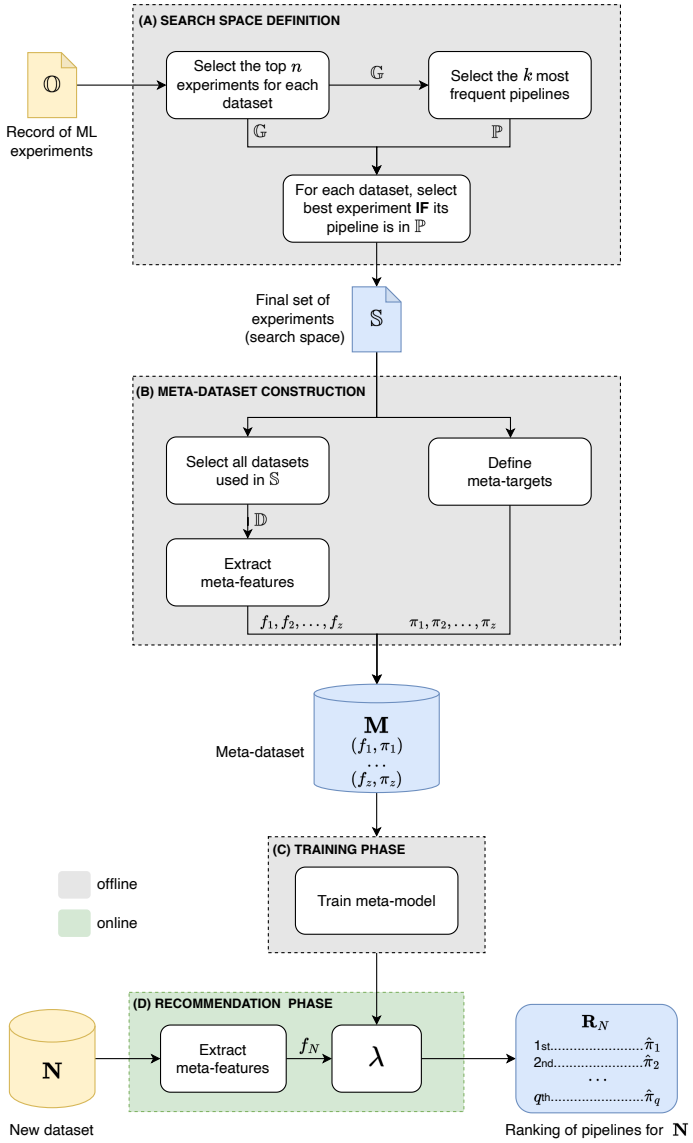


Figure 11 – The MetaML framework process is divided into four main phases: (A) Search Space Definition, (B) Meta-Dataset Construction, (C) Training Phase, and (D) Recommendation Phase. The arrows indicate transitions between these stages. In phase (A), $\mathbb{G}$ represents the selection of the top n experiments for each dataset present in the collection $\mathbb{O}$, while $\mathbb{P}$ corresponds to the k most frequent pipelines among the experiments in $\mathbb{G}$. From this filtering step, the best-performing experiment is selected for each dataset, provided that its pipeline belongs to $\mathbb{P}$. This process defines the search space $\mathbb{S}$.

pipelines that make sense but are almost never used in practice because they are only applicable to niche cases. Therefore, by selecting only the most frequent and high-performing pipelines from historical data, we avoid adding unnecessary noise to the search space and, consequently, to the training of the recommendation model. An analysis was performed on OpenML tasks to support these assumptions, and the results are presented in Appendix A.

The steps performed within the algorithm are described in the component A of Figure 11. We start with a collection, denoted as $\mathbb{O}$, which contains records of past machine learning

Table 3 – Mathematical notation.

Notation	Description
$\mathbf{D}$	A matrix representing one dataset.
$\mathbb{D}$	The set of the datasets used in $\mathbb{S}$.
$\mathbf{D_e}$	The dataset used in the experiment $\mathbf{e}$.
$\mathbf{e}_i = (\mathbf{D}, \pi, p)$	A tuple that represents an ML experiment.
$\mathbf{f}_i = \{f_i^1, f_i^2, \ldots, f_i^x\}$	A meta-feature vector with x meta-features.
$\mathbb{G}$	The set of the top n experiments (w.r.t. performance) for every unique dataset in $\mathbb{O}$.
k	Number of most frequent pipelines.
$\mathbf{M} = \{(\mathbf{f}_1, \pi_1), (\mathbf{f}_2, \pi_2), \ldots, (\mathbf{f}_z, \pi_z)\}$	The meta-dataset. Each pair $(\mathbf{f}_i, \pi_i)$ represents the meta-features and the meta-classes of the dataset $\mathbf{D}_i$.
m	Number of meta-classes.
$\mathbf{N}$	A new (query) dataset.
n	Number of experiments.
$\mathbb{O}$	The record of previously executed ML experiments.
$\mathbb{P}$	The set of the k most frequent pipelines in $\mathbb{G}$.
p	A scalar representing the value of a performance metric.
q	Number of pipelines.
$\mathbf{R_N}$	A ranking of the q most promising pipelines for $\mathbf{N}$.
$\mathbb{S}$	The search space.
x	Number of meta-features.
z	Number of datasets.
π	An ML pipeline.
$\hat{\pi}$	Recommended pipeline.
$\pi_i = \{\pi_i^1, \pi_i^2, \ldots, \pi_i^y\}$	The best pipeline for dataset $\mathbf{D}_i$, where y is the number of meta-classes, and each component π^j can be either 0 or 1, representing whether or not a certain preprocessing step is included.
λ	The meta-classifier.

experiments. Each experiment, represented as $\mathbf{e}_i$, comprises three components: a dataset $\mathbf{D}$, a pipeline π, and its corresponding performance measure p. In other words, $\mathbf{e}_i = (\mathbf{D}, \pi, p)$. From this collection, we construct another set, $\mathbb{G}$, which contains the top n performing experiments for each unique dataset present in $\mathbb{O}$.

Formally, we can define $\mathbb{G}$ as in Eq. 3.1:

$$\mathbb{G} = \{e \mid e \in \textit{top}(n, \mathbb{O})\} \tag{3.1}$$

where $\textit{top}(n, \mathbb{O})$ is a function that returns the set of the top n experiments, in terms of performance, for every unique dataset in $\mathbb{O}$. Therefore, if there are d unique datasets in $\mathbb{O}$, then $|\mathbb{G}| = n \times d$.

Then, to form the search space $\mathbb{S}$, for each unique dataset in the tuples of $\mathbb{G}$, the experi-

ment with the highest performance p in $\mathbb{G}$ is selected only if its pipeline is in a set $\mathbb{P}$ of the k most frequent pipelines in $\mathbb{G}$, else, the experiment is discarded. Consequently, $\mathbb{S}$ contains only one experiment record for each dataset.

Then, $\mathbb{P}$ can be defined by Eq. 3.2:

$$\mathbb{P} = \{\pi \mid \pi \in \textit{most\_freq}(k, \mathbb{G})\} \tag{3.2}$$

where *most_freq*$(k, \mathbb{G})$ is a function that returns the k most frequent pipelines in $\mathbb{G}$. Finally, the search space can be obtained from Eq. 3.3:

$$\mathbb{S} = \{e \mid \pi_e \in \mathbb{P} \wedge e \in \textit{best\_exps}(\mathbb{G})\} \tag{3.3}$$

where *best_exps*$(\mathbb{G})$ is a function that returns a set with the best experiment for every unique dataset in $\mathbb{G}$. Note that, the traditional concept of search space, as a set of pipelines, can be obtained by taking every unique pipeline from $\mathbb{S}$, that is $\{\pi_e | e \in \mathbb{S}\}$, where π_e is the pipeline used in the experiment e. However, we consider $\mathbb{S}$ our search space because, as a precursor for the construction of the meta-dataset, it contains not only a set of pipelines but the association of each dataset with its best pipeline. This information is necessary for the training of the proposed meta-model.

3.4.2 Meta-dataset construction

This phase, represented in component B of Figure 11, takes as input the set of experiments, $\mathbb{S}$, defined in the previous phase (Search space definition), from which it obtains the datasets and the associated best pipeline. All the datasets used in the experiments in $\mathbb{S}$ are selected, i.e., $\mathbb{D} = \{\mathbf{D}_e | e \in \mathbb{S}\} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_z\}$. The Meta-features extraction module represents each dataset $\mathbf{D}_i$ using a meta-feature vector $\mathbf{f}_i = \{f_i^1, f_i^2, \ldots, f_i^x\}$ with x meta-features. In parallel, the Meta-target definition module associates each dataset $\mathbf{D}_i$ with a target vector, a pipeline, $\pi_i = \{\pi_i^1, \pi_i^2, \ldots, \pi_i^m\}$, where m is the number of meta-classes, and π^j can be either 0 or 1, representing whether or not a certain preprocessing step is included. π_i is the best pipeline for dataset $\mathbf{D}_i$, according to the associations in $\mathbb{S}$. As a result, this phase outputs the meta-dataset $\mathbf{M} = \{(\mathbf{f}_1, \pi_1), (\mathbf{f}_2, \pi_2), \ldots, (\mathbf{f}_z, \pi_z)\}$, where each pair $(\mathbf{f}_i, \pi_i)$ represents the meta-features and the meta-classes of the dataset $\mathbf{D}_i$.

3.4.3 Training

In this phase, represented by component C of Figure 11, the meta-classifier λ is trained using the meta-dataset $\mathbf{M}$. λ is a multi-label classifier capable of modeling the relationship between the meta-features and each step of the pipeline while also considering the correlations between the steps by chaining the individual predictions. By learning from the described curated meta-dataset, which contains only high-performing and historically relevant pipelines, we hypothesize that it is possible to predict good pipelines for any new dataset. It represents a robust alternative to optimization processes in terms of computational cost and performance for recommending pipelines.

3.4.4 Recommendation

The recommendation phase, depicted in component D of Figure 11, is the only online step of our approach. It is when the trained meta-classifier (λ) recommends a set of promising pipelines for a new dataset $\mathbf{N}$. Given a new dataset $\mathbf{N} \notin \mathbb{D}$, the first step is to extract its meta-features composing the vector $\mathbf{f_N}$. The vector $\mathbf{f_N}$ is provided as input to the multi-label meta-classifier λ. This way, λ generates an ordered list (ranking) $\mathbf{R_N}$ of the q most promising pipelines for $\mathbf{N}$, where, $\mathbf{R_N} = (\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_q)$. The pipelines in $\mathbf{R_N}$ are ranked in the descending order of their class probabilities calculated by the trained meta-model. The rationale is to provide a few options that the user can choose from instead of predicting a single pipeline that may not be the best performing or, even if it is, it may not be the best option for the user due to other requirements such as computational complexity or model interpretability.

As a final step, the user can choose to use MetaML's recommendation as a zero-shot approach, where no further pipeline evaluation is necessary, or a few-shot approach, where the few recommended pipelines are compared prior to selection. The zero-shot approach can be achieved either by setting $q = 1$ or by selecting the 1st pipeline in the ranking when $q > 1$. In the few-shot (q-shot in this case) approach, the recommended pipelines in the ranking are tested on the query dataset using a cross-validation procedure in order to elect the best one.

3.5 EXPERIMENTAL SETUP

In this section, we present the experimental setup in detail. First, we describe how the search space design and the datasets used to construct the meta-dataset (Section 3.5.1). Then, the entire meta-dataset creation process is described (Section 3.5.2), followed by the process employed to select the meta-model algorithm to be used by the MetaML (Section 3.5.3). Furthermore, the data preprocessing step is detailed (Section 3.5.4). Finally, in Section 3.5.5, the research questions are presented and the evaluation procedure designed to answer them is explained.

3.5.1 Datasets and Search Space

The search space is based on historical record, consisting only of frequently used and high-performing pipelines from past experiments. In this first version, we use OpenML as our main source of historical data. It is important to note that our approach is not limited to OpenML and can be applied to any repository of historical ML experiments. The datasets and the record of previously executed experiments $\mathbb{O}$ used in the experiments were collected from the OpenML open-source platform[3] in April 2024. This record served as input to the procedure described in Section 3.4.1, using parameters $n = 5$ and $k = 5$ (the choice of these parameters was empirically defined, as is explained in Section 3.6.5); however, we applied a few filtering criteria as follows. We selected only experiments using datasets with a number of instances in the interval [100, 8000000], number of classes in [2, 100], number of attributes in [4, 100], and no missing values. From these experiments, only the ones that performed a classification task, reported performance in accuracy, applied 10-fold cross-validation, and used the Scikit-learn library (PEDREGOSA et al., 2011a)[4] were included.

Initially, the procedure resulted in a total of 11,435 experiments in $\mathbb{G}$ using 329 unique datasets and a variety of classification algorithms used in the pipelines, as *K-Nearest Neighbor* (KNN), *Decision Tree* (DT), *Support Vector Classifier* (SVC), *Random Forest* (RF), *Multilayer Perceptron* (MLP) and *Extremely Randomized Trees* (ExtraTrees), and an equally diverse list of preprocessing techniques, e.g., *Principal Components Analysis (PCA)*, *OneHotEnconder*, *Variance Threshold*, *Standard Scaler* and *Polynomial Features*.

[3] https://www.openml.org/
[4] For ease of implementation.

However, after the last filtering step of the search space definition, to select only the most frequent pipelines, the final number of experiments in $\mathbb{S}$ is 296, and the number of unique datasets was consequently the same since we selected only one (the best) experiment for each dataset. Some complexity meta-features could not be calculated for six of these datasets, which were then removed. Therefore, our final number of instances in the meta-dataset was 290. Appendix B presents the final list of datasets used in the meta-dataset. To ensure the diversity of this selection, we performed an analysis presented in Appendix C. For example, for the complexity meta-features, which describe the difficulty of the problems, the analysis reveals a heterogeneous distribution covering a wide range of scenarios. Thus, there are datasets of different complexity.

The pipelines included in the search space are detailed in Table 4. The binary digits indicate whether or not each technique is present in the pipeline. OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest. From the 290 datasets selected, 222 are binary, and 68 are multi-class problems. Pipelines π^3, π^4, and π^5 were the top-performing solutions in 117, 89, and 84 experiments, respectively, and the preprocessing techniques were only employed in 26 experiments. No pipeline employing RF applied preprocessing techniques. Note that these numbers indicate an imbalanced, multi-label meta-dataset. The resulting search space consists of three classification algorithms (DT, SVC, and RF) and three preprocessing techniques (One-HotEncoder, Variance Threshold, and Standard Scaler). All algorithms were used with their default hyperparameter configurations. We prioritized on the broader stages of the pipeline, due to the complexities introduced by HO.

Table 4 – The five pipelines included in the MetaML's search space.

Pipeline	Preprocessing			Classifier		
	OHE	VT	SS	SVC	RF	DT
π^1	1	1	1	1	0	0
π^2	1	1	1	0	0	1
π^3	0	0	0	1	0	0
π^4	0	0	0	0	1	0
π^5	0	0	0	0	0	1

3.5.2 Meta-dataset construction

This step consists of extracting the dataset's meta-features and associating them with a meta-target. In this case, the meta-target is the best-performing pipeline for each dataset, which is obtained from the set of experiments in the search space $\mathbb{S}$. A summary of the 86 meta-features used to represent each dataset is presented in Appendix D. They belong to five groups: simple, info-theory, statistical, model-based, and complexity. Simple, info-theory, and statistical measures are commonly used (RIVOLLI et al., 2022), requiring low computational effort to calculate them (KHAN et al., 2020). Model-based measures are a robust way of characterizing the datasets with information from predictive models (RIVOLLI et al., 2022), but they yield significantly higher computational costs than the previous measures (KOTLAR et al., 2021). Complexity measures aim to describe a meta-learning task in terms of how intricate they are (LORENA et al., 2019; PARMEZAN; LEE; WU, 2017). They allow analyzing the complexity of datasets in different aspects, such as feature-based complexity measures, which evaluate how informative attributes are to separate classes, and linearity measures, which evaluate how linearly classes can be separated, which can contribute to improving the recommendation. The simple, info-theory, and statistical meta-features were fetched from OpenML (VANSCHOREN et al., 2014) records, and the PyMFE (ALCOBAÇA et al., 2020) package was used to extract the model-based and complexity ones. Overall, the set of meta-features employed in MetaML is diverse, which is desirable since it allows for capturing different and relevant information about the tasks.

3.5.3 Meta-Model

To select an algorithm to use as MetaML's meta-model, three candidate multi-label classifiers were considered: Label Powerset (LP), Classifier Chains (CC), and Probabilistic Classifier Chains (PCC). The advantage of these three algorithms is that they take the relationship between labels into account (HERRERA et al., 2016). This property allows MetaML to predict pipelines taking into account the relationships between its steps. We compared these three candidates, as seen in Section 3.6.1, and PCC presents the best base-level performance and, therefore, is the one we selected and employed in all remaining analyses. PCC is an extension of the Classifier Chains algorithm and uses Bayesian optimization to find the best chaining order of the classifiers. It also enables predictions of a ranking of q pipelines for a query dataset. This

is done by ranking the probabilities computed for the classifier chain. As the base estimator for the PCC, we used a Decision Tree classifier, with entropy as its splitting criterion and a minimum of 10 samples in a leaf node.

3.5.4 Data Preprocessing

The values of the meta-features used in this study vary in different intervals and may present outliers, therefore the Quantile Transformer scaling algorithm was employed because of its robustness to outliers (PEDREGOSA et al., 2011a) (AMORIM; CAVALCANTI; CRUZ, 2023b).

Of the 120 meta-features initially considered, we excluded those with more than 10% of missing values, resulting in 86 meta-features. Then, it was observed among these 86 meta-features, 24 meta-features related to the number of numerical attributes still presented missing values on 7.5% of the datasets, those without numerical attributes. Thus, we decided to impute their value as zero for these cases. Other meta-features with fewer missing values were fixed by a KNN imputation (ZHANG, 2012) ($k = 5$), namely, f2.mean (5.52%), f1.sd (3.79%), f1.mean (3.79%), nodes_repeated.sd (3.45%), nodes_per_level.sd (3.10%), and tree_imbalance.sd (3.10%).

3.5.5 Evaluation procedure

The experiments described in this section are designed to answer the following research questions:i

RQ1 Can a meta-learning based AutoML system, that takes into account the interdependence of pipeline steps, achieve pipeline recommendation performances equivalent to the state of the art but incurring much less computational time?

RQ2 Is it possible to obtain better pipeline recommendations using an algorithmically curated search space instead of a larger superset of such space?

For all the experimental analyses in this paper, the MetaML is trained and tested using a leave-one-dataset-out cross-validation procedure (LODOCV), where from the 290 datasets, 289 are used for training the meta-model, and one is left for testing. This procedure is repeated 290 times so that each dataset is the test instance exactly once. From this, the meta-model performance is calculated by taking the mean of the meta-classifier performances (in terms of

the F1, Accuracy, and Precision metrics) over the 290 test instances. The base-level performances are calculated by taking the pipeline recommended for each test dataset and evaluating its accuracy on such dataset using a 10-fold cross-validation procedure. Therefore, a final set of 290 base-level performances is obtained, one for each dataset.

However, for the analysis required to answer RQ1, only a subset of 152 of the MetaML's base-level performances is considered to compare it against the state of the art. This subset is presented in Appendix E. It was necessary to use this subset to maintain fairness in the comparison and dataset diversity. From the original 290 datasets, 85 were used in the AutoSklearn 1.0 meta-learning training procedure and were therefore removed. Other 47 datasets were excluded to reduce the number of datasets belonging to the same family, hence increasing the diversity within the test bed. Another 6 datasets were excluded because some methods failed to interpret them due to formatting issues, leading to 152 datasets. Some of the methods also failed to execute for less than 11.68% of the datasets due to memory errors or due to exceeding the computational time limit. TABPFN presented limitations when dealing with datasets with a high number of classes and instances, exceeding its maximum supported capacity. It was only capable of dealing with problems that had up to 10,000 training examples, 500 variables, and 10 classes. Thus, it presented errors in about 20% of the datasets. Hence, we imputed the missing results for this method using a Constant Predictor. The baseline, Constant Predictor, predicts the empirical class probabilities from the training data.

Using this subset of 152 datasets, the proposed MetaML is compared, at the base level, with the following frameworks: Auto-WEKA, AutoSklearn 1.0, AutoSklearn 2.0, AutoGluon, FLAML-Zero, H2O AutoML, Naive AutoML, TabPFN and TPOT. Since Auto-sklearn 2.0 does not support the accuracy metric, we configured it to use balanced accuracy. However, to enable a fair performance comparison across the 152 datasets with other AutoML methods, we computed the standard accuracy metric based on the predictions obtained from all cross-validation folds. FLAML-Zero was included because it is a method that can operate in zero-shot mode, akin to MetaML. Naive AutoML was included because it was proposed as a baseline method for AutoML studies and because, unlike MetaML, it does not consider the interdependence of the pipeline steps, which, therefore, helps us validate our hypothesis on the relationship between pipeline steps. The remaining frameworks were included because of their popularity, as shown by their large number of citations. To facilitate the comparison of the existing frameworks, we used the AutoML benchmark tool (GIJSBERS et al., 2019) (GIJSBERS et al., 2024) setting a runtime limit (i.e., time budget) of 3 hours per dataset for each compared

method. The software implementation details of the MetaML and the hardware used in the experiments are described in Section 3.5.6, and the source code for the experiments is available at the project's repository[5].

3.5.6 Software and Hardware

The MetaML was implemented using the Python language (version 3.9.1) and the following packages: Scikit-Learn (PEDREGOSA et al., 2011a) (version 1.2.2), Pandas (MCKINNEY, 2010) (version 1.5.3), Scikit-Multilearn (SZYMAŃSKI; KAJDANOWICZ, 2017) (version 0.2.0), OpenML (FEURER et al., 2020) (version 0.10.2), and PyMFE (ALCOBAÇA et al., 2020) (version 0.4.2). The experiments for the comparative analysis were performed using the AutoML Benchmark on the Linux operating system, using an eight-core, 3.7GHz AMD Ryzen 7 2700X processor with 16 GB of RAM for the lager datasets, and a six-core, 3.4GHz AMD Ryzen 5 processor with 16 GB of RAM for the smaller datasets.

3.6 RESULTS AND DISCUSSION

In this section we detail the experiments designed to evaluate MetaML and answer the research questions. We present a comparative analysis of the base-level performance of MetaML and the state of the art in Section 3.6.2. Section 3.6.3 compares their recommendation times. An analysis of the pipelines recommended by MetaML is presented in Section 3.6.4. Details on the effects of our search space curation method are presented in Section 3.6.5. Finally, in Section 3.6.1, we present the experiment where we compared three multi-label classification algorithms before adopting PCC as MetaML's meta-model.

3.6.1 Comparing meta-model algorithms

To decide on the choice of meta-classifier algorithm to be employed in the MetaML, we considered three multi-label algorithms as described in Section 3.5.3: LP, CC, and PCC. As previously discussed, PCC is the only one that gives us the possibility of recommending a ranking of pipelines (sorted according to their probabilities) and, because of this, MetaML can be used as a few-shot approach when using PCC. Therefore, we compared the average rankings

[5] https://github.com/cynthiamaia/MetaML

on base-level accuracy of these five options: MetaML-LP (zero-shot only), MetaML-CC (zero-shot only), MetaML-PCC zero-shot, MetaML-PCC two-shot and MetaML-PCC three-shot. Figure 12 illustrates these rankings in a CD diagram.

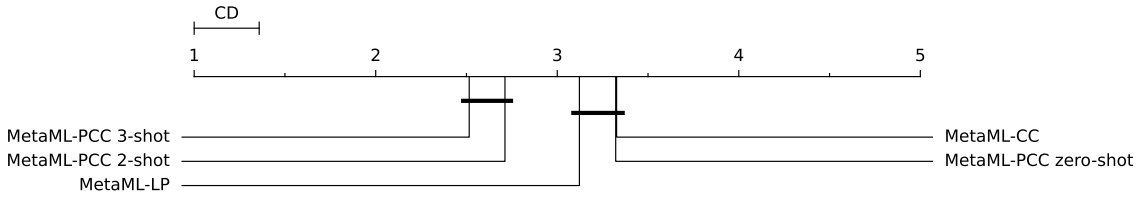


Figure 12 – Critical difference diagram of the average ranking of meta-model algorithms considering their base-level accuracy.

Note that the PCC with a 3-shot approach is the best choice of meta-model strategy for the MetaML, presenting the lowest (best) ranking and being significantly different from the 3 other approaches. This suggests that including more options in the recommendation allows for greater robustness, as it enables greater exploration of the set of candidate pipelines and increases the flexibility, the power of generalization and thus the chances of recommending the best pipeline for a given query dataset. Furthermore, recommending more than one pipeline allows the users to choose based on other criteria, such as interpretability or computational cost, according to their particular needs.

In addition to the three multi-label algorithms, we also studied the use of single-label meta-models through the decomposition of the multi-label problem in several, individual, single-label problems. The results of this analysis can be seen in Appendix F. While some of the single-label approaches were able to achieve slightly better performance than the chosen PCC approach, we emphasize that the motivation behind employing this multi-label algorithm is due to its ability to exploit label inter-dependencies. This capability is particularly valuable in pipeline recommendation tasks, where preprocessing and classification steps may have meaningful inter-dependencies. As the meta-dataset is enriched with a broader spectrum of pipelines, it is expected that this characteristic of the PCC algorithm will demonstrate its value and its advantages will become more evident.

3.6.2 Comparing the performances of recommended pipelines

Table 5 presents the results of the state-of-the-art methods and the MetaML regarding the performance (accuracy) of the recommended pipelines. Here we evaluate MetaML both as zero-shot, in Table 5(a), and as a 3-shot method, in Table 5(b). Recall that, in the zero-

shot approach, the first pipeline in the recommended ranking is selected, while in the 3-shot approach, the 3 pipelines in the ranking are tested on the dataset, using 10-fold cross-validation, and the one with maximum accuracy is selected. A comparative analysis was also performed considering the division between artificial and real datasets. This analysis is illustrated in Appendix G.

Table 5 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML zero-shot (a) and 3-shot (b) against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 5.56\text{e-}04$, and the total recommendation time taken for datasets for which all methods ran successfully.

	Mean acc.	Win/tie/loss	Mean rank	p-value	Time (hours)
Auto-WEKA	0.743	76/9/67	6.447	0.594	356
AutoSklearn 1.0	0.774	48/15/89	5.098	0.020	371
AutoSklearn 2.0	0.792	47/13/92	**4.171**	4.13e-06	361
AutoGluon	**0.830**	50/8/94	5.457	2.60e-08	13
H2O AutoML	0.789	72/3/77	6.388	0.211	366
Naive AutoML	0.811	50/16/86	4.763	4.13e-05	59
TPOT	0.746	61/13/78	5.447	0.275	357
FLAML-Zero	0.759	91/25/36	7.013	1.48e-08	0.001
TabPFN	0.767	57/13/82	4.223	0.060	23
MetaML Zero-shot	0.798	-	5.990	-	0.056

(a)

	Mean acc.	Win/tie/loss	Mean rank	p-value	Time (hours)
Auto-WEKA	0.743	100/11/41	6.611	1.39e-05	356
AutoSklearn 1.0	0.774	70/15/67	5.243	0.303	371
AutoSklearn 2.0	0.792	64/14/74	**4.286**	0.073	361
AutoGluon	**0.830**	74/9/69	5.618	0.176	13
H2O AutoML	0.789	92/2/58	6.516	0.017	366
Naive AutoML	0.811	66/22/64	4.888	0.646	59
TPOT	0.746	76/15/61	5.552	0.047	357
FLAML-Zero	0.759	107/23/22	7.111	6.19e-17	0.001
TabPFN	0.767	74/19/59	4.355	0.458	23
MetaML 3-shot	0.821	-	4.815	-	0.108

(b)

For this analysis, a time budget of 3 hours per dataset was imposed. In its last column, Table 5 also compares the total recommendation time elapsed, which for MetaML includes the time for feature extraction, meta-model inference, and, in the case of MetaML 3-shot, the selection of one of the 3 recommended pipelines, by evaluating them with 10-fold CV on

the query datasets. Most methods failed to execute for a few datasets. For fairness, the time reported here considers the recommendation time taken only for the 124 datasets for which all methods ran successfully. The p-values, from a Wilcoxon signed-rank test, that are under the $\alpha = 5.56e\text{-}04$ significance level, indicate cases where there is a significant difference between the compared method and MetaML's performance. This value for α was obtained by applying a Bonferroni correction to the typical value of 0.05 to control for family-wise Type I error (BENAVOLI; CORANI; MANGILI, 2016).

In Table 5 although AutoSklearn 2.0 had a slightly lower average rank (4.286) than MetaML 3-shot (4.815), however, this difference was not statistically significant (p-value $=$ 0.073), while MetaML achieved a higher average accuracy (0.821 vs. 0.792). In terms of win/tie/loss, Auto-sklearn 2.0 achieved more wins, but the difference does not indicate a dominant advantage.

Another relevant point is time, while Auto-sklearn 2.0 required 361 hours, MetaML achieved competitive results in a few minutes. This contrast can be attributed to each approach's different strategies in the online phase. Auto-sklearn 2.0 sequentially evaluates all pipelines in its fixed portfolio of pipelines (built offline) using the Successive Halving technique, which progressively allocates time to the most promising pipelines. If time is still available, an additional Bayesian Optimization step is triggered to refine the solution further. In contrast, MetaML uses a curated search space composed exclusively of the most frequent and best-performing pipelines extracted from publicly available experiments (such as OpenML). Auto-sklearn 2.0 uses approximately 100 pipelines, while MetaML includes only 5 candidate pipelines. This avoids redundant execution of configurations in the online phase. Thus, even though Auto-sklearn 2.0 shows an advantage in terms of average ranking, the results indicate that MetaML is capable of achieving competitive performance with reduced computational cost.

TabPFN, which ranks second in terms of mean ranking, exhibits lower mean accuracy and slower prediction speed compared to MetaML zero-shot and 3-shot. Additionally, with MetaML 3-shot, we achieved a higher number of wins. Furthermore, we note that TabPFN has some limitations. Its training time is low, but the prediction time is significantly higher; for example, for a dataset with 9285 instance and 4 features, TabPFN took 1.02 seconds to predict a single sample. This means the total time to predict all samples would be approximately 2.6 hours, as emphasized in the paper, TabPFN's inference speed is slower (HOLLMANN et al., 2025).

Note that MetaML 3-shot presented the second-best mean accuracy and third-best rank-

ing, consistently achieved higher number of wins (exception of Auto-sklearn 2.0), and the second-lowest recommendation time of all methods. MetaML zero-shot achieved numbers that, while inferior to the 3-shot variant, are still competitive for the amount of computational time it requires, since it presented the third-best mean accuracy and a mean ranking that is statistically equivalent to the other zero-shot technique, FLAML-Zero, and even to the technique with the best mean accuracy, AutoGluon. That said, in terms of base-level performance, this indicates that MetaML 3-shot has a better performance/cost balance than MetaML zero-shot, as its time penalty of just 1.51 seconds per dataset when compared to zero-shot method can be justified by its superior performance.

AutoGluon was the only method that presented a better mean accuracy than the MetaML 3-shot. However, the p-value indicates that their difference is not statistically significant. AutoGluon employs a neural network to stack and ensemble a few predefined models, including some high-performing classifiers, with a pre-defined sequence of pre-processing steps. Unlike the other methods, it does not recommend a pipeline for a query dataset, but instead uses the time budget to train and apply as many as possible of its base models on the specific query dataset, combining their outputs. This explains its shorter recommendation time as it does not need to search for an optimal pipeline. The downside is that its fixed predefined models may not specialize well for some problems (hence presenting only the 7th best mean ranking) which can lead to low performances for some datasets, even for bigger time budgets. In other words, even though AutoGluon numerically achieves higher mean performance, MetaML 3-shot still wins for 74 (48.7%) of the datasets against 69 (45.4%) wins for the AutoGluon. MetaML 3-shot also reaches a better mean ranking than AutoGluon (4.815 against 5.618).

When compared to Naive AutoML, MetaML 3-shot presents a lower (better) mean ranking, and the p-value indicates that this method's results are equivalent to MetaML's. MetaML 3-shot also presents higher mean accuracy at a much lower computational time of 0.108h against 59h for Naive AutoML.

FLAML-zero presents the lowest computational time, as expected due to its zero-shot meta-learning approach, which uses just a few simple meta-features. However, it presents a lower mean accuracy, losing for 91 (67.9%) and winning for only 36 (26.9%) datasets against the MetaML zero-shot. When compared to MetaML 3-shot it loses for 107 (70.4%) and wins for only 22 (14.5%). It also presents the worst mean ranking of all methods. Its simple pipelines, focusing only on the selection of DT-based models and the tuning of their hyperparameters, coupled with its choice of search space reduction strategy and the few and simple meta-features

it relies on, may be the culprits for its lower performances.

When compared to the remaining methods, which employ the traditional optimization technique, MetaML 3-shot presents even larger advantages in terms of either the win/loss ratio or the mean accuracy and rankings. This reiterates our assumption that using meta-learning and our heuristically curated search space it is possible to reduce costs and increase performance. The better win/loss ratio, mean accuracy, and ranking objectively show that MetaML 3-shot is the best option for most datasets, especially when considering its balance between performance and computational time.

Figure 13 presents a critical difference diagram comparing the rankings of the AutoML methods reported in Table 5, the post-hoc Nemenyi test was applied. In this type of diagram, the horizontal lines connect statistically equivalent methods. Note that MetaML 3-shot performs equivalently to robust AutoML methods such as AutoSklearn 2.0, AutoSklearn 1.0, TabPFN, Naive AutoML, AutoGluon while being significantly superior to FLAML-Zero, Auto-WEKA, and H2O AutoML. MetaML zero-shot presents an inferior result, but is still significantly better than FLAML-Zero.

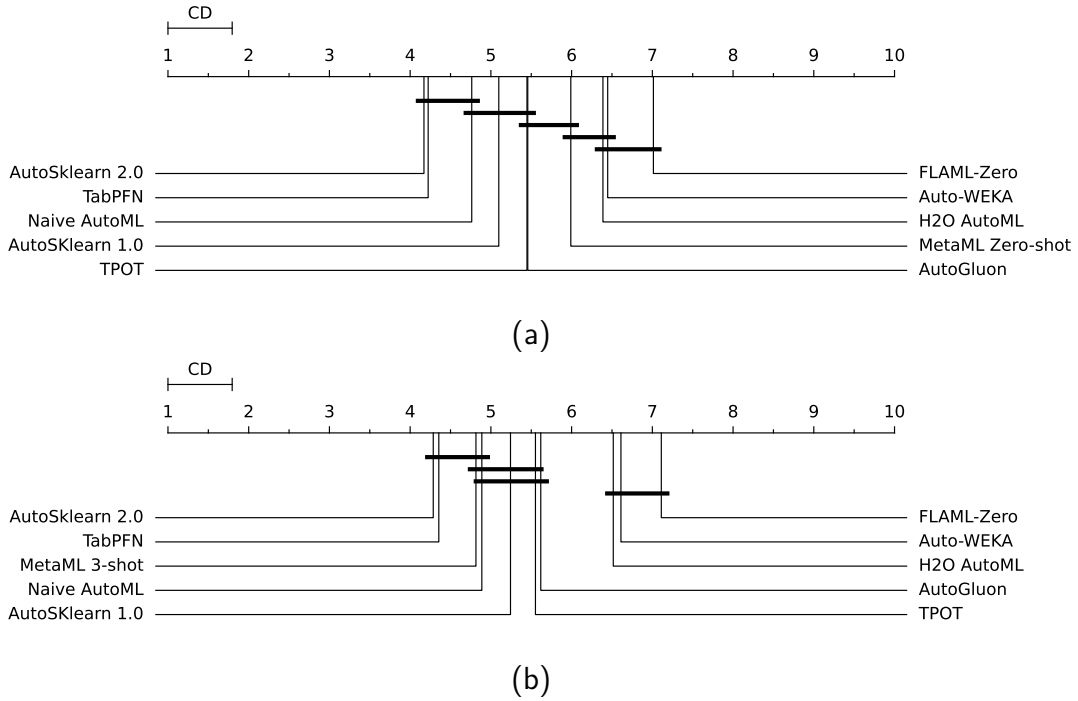


Figure 13 – Critical difference diagram of the average ranking of the base-level performance of the AutoML methods, including (a) MetaML zero-shot, (b) MetaML 3-shot.

Additionally, the violin plot in Figure 14 show that the distribution of the base-level accuracy achieved with MetaML 3-shot's recommended pipelines is equivalent, and sometimes superior to that of the state-of-the-art methods, even though it used a much smaller recommendation time than almost all the methods. Note that MetaML zero-shot is visibly superior

to FLAML-Zero, albeit with lower marks than MetaML 3-shot. We highlight that MetaML 3-shot presents a concentration in higher accuracy values and smaller variability than the other methods, as shown by the interquartile distances. This lower variability indicates a better generalization power that, in practice, translates to superior results in unseen datasets.

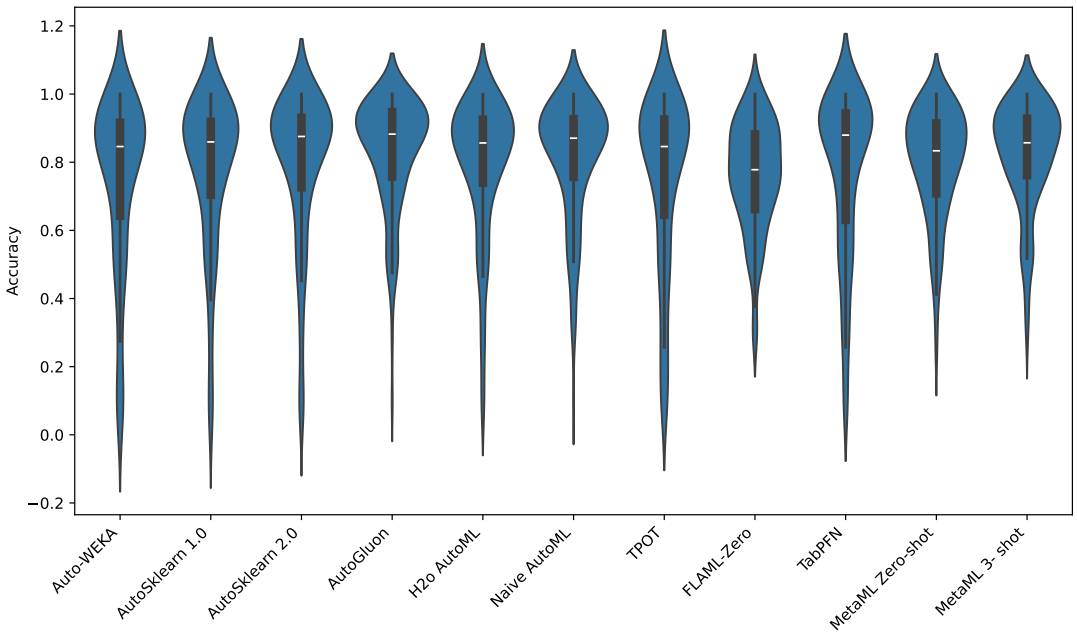


Figure 14 – Performance distributions of the pipelines recommended by each AutoML method on the tested datasets.

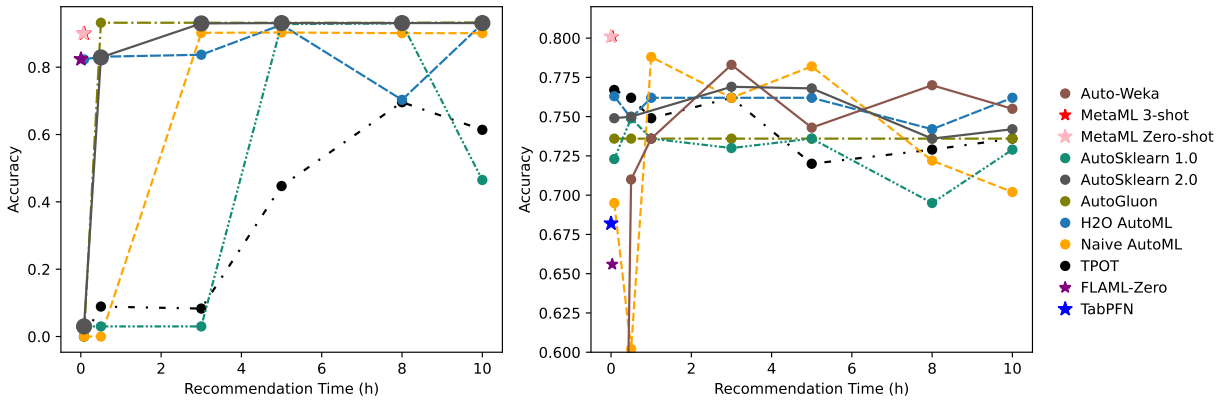
To further investigate the scenarios where MetaML outperforms or underperforms other AutoML methods, a comparison was conducted and is illustrated through heatmaps (Appendix H). This analysis considers specific dataset characteristics, such as the number of classes and instances, and provides an overview of how MetaML compares to other AutoML methods across various scenarios.

It is important to emphasize that, because of its curation technique, MetaML's search space is much smaller than the ones used by the other methods. The fact that its performance is on par with the competition demonstrates the benefit of employing our curation technique to reduce the search space. Note that MetaML was able to perform well even without tackling hyper-parameter optimization or using post-processing techniques such as ensembling (used by AutoSklearn 1.0 and H2O-AutoML). This shows that metaML is promising and can get even better as more meta-data becomes available. A more extensive meta-data collection procedure, such as using results from other libraries such as Weka and R, may make this possible. Figure 6 in Appendix I illustrates the representation of the pipelines. It displays the recommendations for four datasets, highlighting how the blocks significantly differ in quantity and, mainly, in

the adjusting of hyperparameters.

3.6.3 Comparing recommendation times

Figure 15 presents the mean accuracy of the recommended pipelines (base-level performance) of the state-of-the-art methods versus the MetaML, when applied to two example datasets under different time budgets, graphs from other datasets are included in Appendix J. Here, the star symbol represents the few-shot methods (MetaML Zero-shot, MetaML 3-shot, TabPFN and FLAML-zero).[6]



(a) BNG_lymph dataset with 1,000,000 instances.

(b) Tae dataset with 151 instances.

Figure 15 – Comparison of AutoML methods using different recommendation time budgets.

MetaML is positioned as a viable and advantageous method, achieving top performances in the shortest time budget on both zero-shot and three-shot variants. In Figure 15(a), for which a large dataset was used, we can see how most methods only improve their accuracy when using longer time budgets. Two of the methods, AutoSklearn 1.0 and H2O AutoML, sometimes even decrease their performance when given more time. AutoGluon, on the other hand, performed well for this dataset, with a high, constant accuracy, with a 30-minute time budget. FLAML-Zero has also achieved a high performance under low computational time for this larger dataset. However, when we look at Figure 15(b), using a small dataset, AutoGluon's performance is among the lowest, even using longer time budgets, and FLAML-Zero presents the lowest performance, while MetaML achieved the best performance in less than 3 seconds on both datasets. It is possible that FLAML-Zero's approach to search space complexity reduction, that keeps just a few portfolio pipelines based on their previously sampled performances, is not generalizing well, hence the contrasting results it obtained in our two example datasets. In the

[6] Figure 15(a) does not include Auto-WEKA and TABPFN because it failed for this dataset.

Tae dataset (151 instances), TabPFN performed competitively compared to other methods, highlighting its efficiency in smaller datasets. However, it showed limitations in dealing with problems with more than 10,000 instances, failing on the BNG_lymph dataset (1,000,000 instances), for instance.

This analysis reinforces our hypothesis that a focus on heuristically curating the search space and employing a meta-learning method that is aware of the interdependence of pipeline steps is a good direction for a holistic AutoML solution. Finally, this discussion enables us to answer RQ1 as follows:

RQ1 *Can a meta-learning based AutoML system, that takes into account the interdependence of pipeline steps, achieve pipeline recommendation performances equivalent to the state of the art but incurring much less computational time?*

By employing meta-learning, based on a multi-label classification algorithm, MetaML provides pipeline recommendations that consider the interdependencies between pipeline steps, achieving performances comparable to the state of the art while incurring a lower computational cost. MetaML consistently achieves high performance on datasets of various domains and sizes. In short, MetaML balances performance and computational cost across different domains, offering an adaptable approach to a wide range of problems that achieves state-of-the-art performances, as demonstrated in the analyses throughout Sections 3.6.2 and 3.6.3.

3.6.4 Pipeline Recommendation Analysis

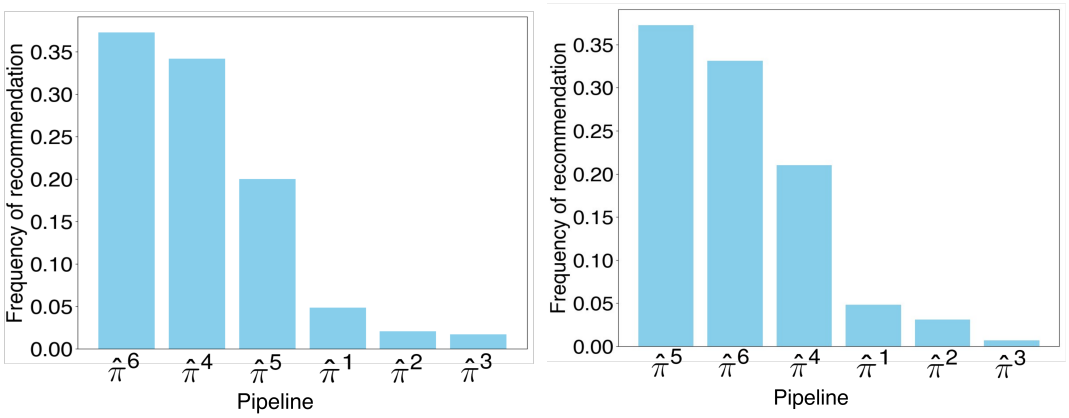
The resulting six pipelines, $\hat{\pi}^1$ to $\hat{\pi}^6$, recommended by the MetaML during its evaluation on the 290 datasets are listed in the Table 6. The binary digits indicate whether or not each technique is present in the pipeline. OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest. Notice how this set of pipelines differs from that of the search space (Table 4), as the meta-model was able to learn a pipeline $\hat{\pi}^2$ that was not explicitly in the search space. This is because MetaML uses Probabilistic Classifier Chains (PCC), which capture the correlation between steps. This allows the model to explore interdependencies that are not explicit in the original search space.

In Figure 16, we analyze how frequently each of these pipelines is recommended by the

Table 6 – All the six different pipelines recommended by the MetaML for the 290 test datasets.

Pipeline	Preprocessing			Models		
	OHE	VT	SS	SVC	RF	DT
$\hat{\pi}^1$	1	1	1	1	0	0
$\hat{\pi}^2$	1	1	1	0	1	0
$\hat{\pi}^3$	1	1	1	0	0	1
$\hat{\pi}^4$	0	0	0	1	0	0
$\hat{\pi}^5$	0	0	0	0	1	0
$\hat{\pi}^6$	0	0	0	0	0	1

MetaML under two different approaches: zero-shot and 3-shot.



(a) (b)

Figure 16 – Recommendation frequency of each pipeline for MetaML zero-shot (a) and MetaML 3-shot (b).

Note that, overall, MetaML's recommendations are diverse, as for example, pipelines $\hat{\pi}^4$, $\hat{\pi}^5$ and $\hat{\pi}^6$ are well represented, each being recommended for a significant portion of the datasets. Instead of learning a default pipeline to recommend to most datasets, MetaML takes into account relevant meta-features in its decision process and seeks to recommend the most suitable pipeline for each dataset. The recommendation of less frequent pipelines ($\hat{\pi}^1$, $\hat{\pi}^2$ and $\hat{\pi}^3$) that include preprocessing techniques is a testament of the MetaML's flexibility to specialize when necessary.

Additionally, when we compare Figures 16(a) and 16(b), we observe that, the three most frequent pipelines switch places. While for the zero-shot approach, the pipeline with the DT model was the most recommended, for the 3-shot approach, it was the one with the RF model. This suggests that the two approaches can really yield different recommendations and that relying only on a zero-shot method can lead to missing the opportunity for better recommendations that occur when we consider the 3-shot approach.

3.6.5 The effect of search space curation

To verify the contribution of the search space curation procedure described in Section 3.4.1, we conducted an analysis of the base-level performance by comparing the accuracies of the pipelines recommended by the MetaML when using a larger, unfiltered search space that we call "full search space", versus our proposed algorithmically curated search space. For the full search space, we started with 11,435 experiments in $\mathbb{G}$, covering 329 unique datasets and various classification algorithms. Due to errors in meta-feature extraction, six datasets were excluded, resulting in a final meta-dataset of 323 instances. This dataset includes 19 distinct pipelines, listed in Table 7.

Table 7 – The nineteen pipelines included in the MetaML's full search space. The binary digits indicate whether or not each technique is present in the pipeline. SP - SelectPercentile, SI - SimpleImputer, PCA -Principal component analysis, RSC - RobustScaler, PF - PolynomialFeatures, OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest, AdaBoost - Adaptive Boosting, LDA - LinearDiscriminantAnalysis, XGB - eXtreme Gradient Boosting, ExtraTrees, LR - LogisticRegression, LinearSVC - Linear Support Vector Classification, MLP - Multi-layer Perceptron, SGD - Stochastic Gradient Descent.

Pipeline	Preprocessing								Classifier
	SP	SI	PCA	RSC	PF	OHE	VT	SS	
π^1	0	0	0	0	0	1	1	1	SVC
π^2	0	0	0	0	0	1	1	1	DT
π^3	0	0	0	0	0	0	0	0	SVC
π^4	0	0	0	0	0	0	0	0	RF
π^5	0	0	0	0	0	0	0	0	DT
π^6	1	1	0	0	0	0	0	0	RF
π^7	0	1	0	0	0	1	1	1	ExtraTrees
π^8	0	1	0	0	0	0	0	0	XGB
π^9	0	0	1	0	0	0	0	0	DT
π^{10}	0	0	0	1	0	1	0	0	AdaBoost
π^{11}	0	0	0	0	1	0	0	0	AdaBoost
π^{12}	0	0	0	0	1	0	0	0	LDA
π^{13}	0	0	0	0	0	1	0	0	AdaBoost
π^{14}	0	0	0	0	0	1	1	0	RF
π^{15}	0	0	0	0	0	1	1	1	ExtraTrees
π^{16}	0	0	0	0	0	1	1	1	LinearSVC
π^{17}	0	0	0	0	0	1	1	1	MLP
π^{18}	0	0	0	0	0	1	1	1	SGD
π^{19}	0	0	0	0	0	0	0	0	LR

In the search space definition phase, the values of k and n[7], were empirically adjusted by evaluating these two different settings: (i) $n = 3$ or $n = 5$ and $k = 8$, resulting in 309 instances ($n = 3$ and $n = 5$ yields the same performance when k is fixed at 8); (ii) $n = 5$ and $k = 5$, resulting in a curated meta-dataset of 290 instances. The pipelines resulting from the choice of $k = 5$ in the curated search space were previously listed in Table 4, while the pipelines resulting from $k = 8$ are detailed in Table 8.

Table 8 – The eight pipelines search space curated (k=8). The binary digits indicate whether or not each technique is present in the pipeline. OHE - One-Hot Encoder, VT - Variance Threshold, SS - Standard Scaler, DT - Decision Tree, SVC - Support Vector Classifier, RF - Random Forest, LinearSVC - Linear Support Vector Classification, MLP - Multi-layer Perceptron, SGD - Stochastic Gradient Descent.

Pipeline	Preprocessing			Classifier
	OHE	VT	SS	
π^1	1	1	1	SVC
π^2	1	1	1	DT
π^3	0	0	0	SVC
π^4	0	0	0	RF
π^5	0	0	0	DT
π^6	1	1	1	LinearSVC
π^7	1	1	1	MLP
π^8	1	1	1	SGD

We compared these two versions of the curated search space with each other and with the full search space. For fairness in comparison, only the 290 datasets that are present in all search spaces were taken into account for measuring the mean base-level accuracy. The box plots in Figure 17 show the distribution of these base-level accuracies. In the first curated search space ($n = 3 \mid n = 5$ and $k = 8$) and in the full search space, represented by the box plots on the left and on the right, respectively, the values highlighted in red indicate invalid pipelines, e.g., pipelines including only preprocessing techniques without an associated classification model or two classification models in the same pipeline. Although they present a similar median, it is clear that the curated search space ($n = 5$ and $k = 5$) presents a greater concentration of values with higher accuracy, indicating superior performance compared to the others.

[7] Recall that k controls how many of the most frequent pipelines are selected, and n, the number of top experiments for each dataset. In other words, these two parameters control the degree of curation of the search space (see Section 3.4)

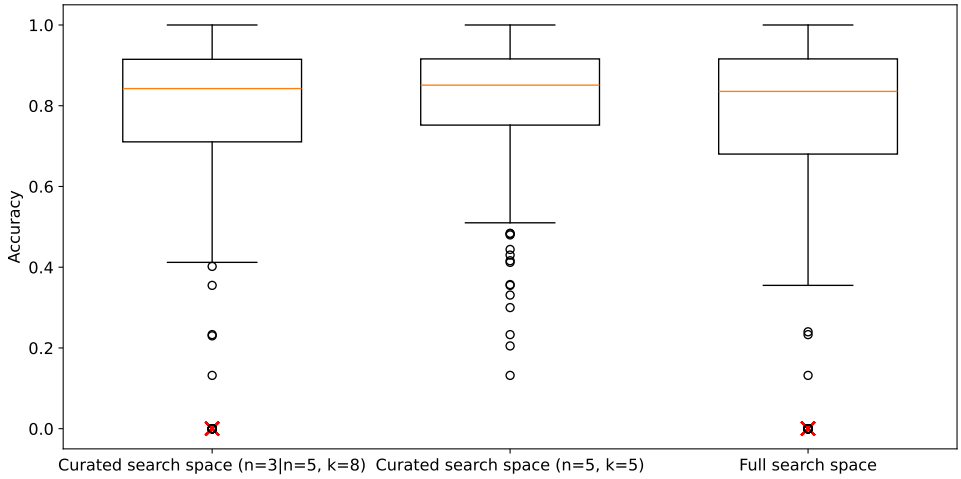


Figure 17 – MetaML's base-level performance the full search space versus when using the curated search space under different values for the parameters k and n.

Table 9 presents the average accuracy, ranking, and p-values of a Wilcoxon signed-rank test, again using the corrected significance level of α =8.33e-03. It can be seen that the curated search space (with n=5 and k=5) obtained the best average accuracy and ranking, presenting a higher number of wins compared to the others. The low performance observed in the search space with k=8 or the full search space can be attributed to the increased number of less common pipelines, which leads to a higher degree of class imbalance, hindering performance.

Table 9 – A base-level analysis of the mean performance (accuracy), Win/tie/loss of the Curated search space (n=5 and k=5) against the others, mean ranking, p-value of the Wilcoxon signed rank test with α =8.33e-03.

	Mean acc.	Win/tie/loss	Mean rank	p-value
Full search space	0.755	67/170/53	2.036	0.008
Curated search space (n=3|n=5 and k=8)	0.770	60/178/52	2.001	0.077
Curated search space (n=5 and k=5)	**0.814**	-	**1.962**	-

The box plot in Figure 18 compares the curated search space (n=5 and k=5) with the full search space with regards to meta-model inference time. The curated search space not only achieves better performance, as already highlighted in Figure 17, but also significantly reduces computational cost.

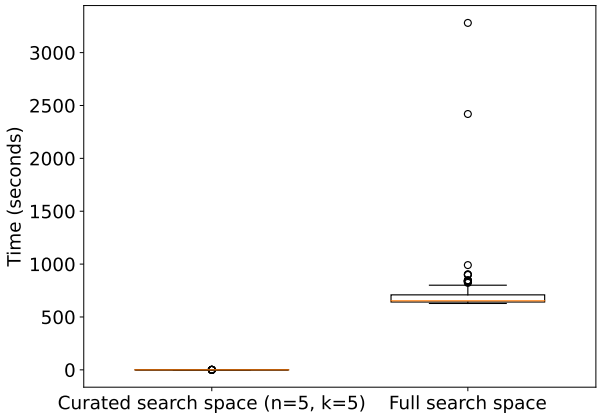


Figure 18 – Meta-model inference time of the MetaML when using the full search space versus the curated one.

For the full search space, the median of the meta-model inference time taken per dataset is 653.8 seconds, with outliers beyond 2000 seconds, while for the curated search space that time is drastically reduced to 0.085 seconds and is virtually uniform for all datasets. The larger number of labels in the full search space (18 pipelines) explains the extra complexity, as MetaML models the dependencies between the labels. This reinforces that a curated search space allows for a better representation of the space of possible pipelines enabling greater performance at lower computational complexity.

The clear boost in performance achieved by using the proposed curated search space when compared to the full search space (seen in Table 9 and Figures 17 and 18) reiterate the importance of such procedure in AutoML systems. This component of our proposal has a clear contribution in enabling its performance to be equivalent or superior to state-of-the-art AutoML approaches. The traditional approaches typically explore a wide range of possibilities, which makes the process challenging due to the vast number of options in the search for the best configuration. This not only increases computational demands but also risks diluting efforts in less promising configurations. Furthermore, expansive and complex search spaces can significantly increase the likelihood of overfitting, impairing the generalization ability (ELDEEB et al., 2022). With this in mind, we can answer RQ2:

RQ2 Is it possible to obtain better pipeline recommendations using an algorithmically curated search space instead of a larger superset of such space?

Yes. The results show that our proposed curated search space led to higher base-level performances and lower computational costs. Therefore, the algorithm is able to learn from historical data to reduce search space complexity while keeping the most relevant pipelines, which ultimately leads to the recommendation of better pipelines within shorter

times compared to the full search space. Note that the multi-label approach allowed MetaML to learn a new pipeline $\hat{\pi}^2$ that was not present in the search space. This tells us that, even when using a reduced search space, the method has the potential to adapt and recombine steps when necessary.

3.7 THREATS TO VALIDITY

As with any meta-learning system, no matter how solid the approach is, MetaML's output quality strongly depends on the quality of the meta-data provided. In this study, MetaML was evaluated by using historical data from machine learning experiments available from OpenML, one of the largest public machine learning repositories. Although this repository offers a wide range of datasets, the quality and representativeness of the experiments logged there may vary. For instance, we have noticed a strong imbalance in the choice of classification models. Another problem is that preprocessing techniques are rarely employed, and even when they are, just a few are explored. It is possible that users perform custom preprocessing and model choosing in their pipelines but do not log that information on OpenML. This may have affected the diversity and comprehensiveness of pipeline configurations available for further curation by MetaML's search space curation component, ultimately degrading the meta-model performance. Even so, MetaML achieves performances close to or even superior to current AutoML systems, with a lower computational cost.

We highlight that this is a data-related issue that can be fixed by future investments in building better meta-datasets covering a wider range of preprocessing techniques and machine learning models. Additionally, it is important to note that only scikit-learn datasets were used, and experiments performed with Weka or R, for example, were not considered and may be included in future analyses.

Another issue is that meta-learning approaches always have a costly offline phase. However, note that, unlike optimization procedures, this high cost only has to be paid once. For every new recommendation, only the online (fast) phase of the meta-learning process has to be performed. In practice, MetaML can be (costly) built once from a collection of previously run meta-learning experiments and then it is ready to be employed as many times as needed in a fast and efficient way. Additionally, to reduce the cost of the offline phase, we use previously performed machine learning experiments stored in OpenML, which include hundreds of experiments ready for reuse. This significantly reduces the computational cost associated with

evaluating algorithm performance on different datasets, a common limitation in the field, as highlighted in the paper by Khan et al., (KHAN et al., 2020).

Furthermore, another limitation is that the datasets used do not present missing values, which was a methodological choice we adopted during the process.

3.8 CONCLUSION

We proposed MetaML, a multi-label meta-learning method for recommending machine-learning pipelines. Using information from previous machine-learning experiments, a technique was devised to reduce search space complexity by selecting only the most relevant pipelines. For this particular instantiation of the MetaML method, we opted to use OpenML; however, MetaML is designed to learn from historical data across any repository, not limited to OpenML. MetaML can recommend a ranking of pipelines, which the user can then assess. Experiments performed on 152 datasets of different sizes and domains demonstrated the effectiveness of the proposed method. We evaluated MetaML both as a zero-shot and as a 3-shot approach. In terms of base-level performance MetaML 3-shot has a better performance/cost balance than MetaML zero-shot. Overall, MetaML 3-shot achieved classification performance better than or equivalent to the state of the art, but requiring much less computational time.

MetaML's multi-label approach, using the PCC meta-model, which considers the dependency between the steps of the pipeline, allowed recommendations of rankings of pipelines based on their probabilities. This expands the available options, which can be an important criterion for users who want to select, for example, a pipeline based on the interpretability of the model. Additionally, when this is used in tandem with a few-shot approach, it enables efficient selection of the pipeline candidates.

The proposed search space curation technique proved to be an important component of the MetaML, providing a boost in performance while at the same time reducing its computational complexity. By learning from historical data, the technique was able to select the most relevant pipelines reducing the search space drastically, without leading to the loss of generalization power by the meta-model.

Future work can focus on improving the set of meta-features, on the diversity of historical records of ML experiments, identifying/generating unrepresented areas of the search space to build a stronger meta-dataset, on exploring the interpretability of the meta-model, and on using stacking, ensembling, and other approaches to combine the outputs of the top q pipelines to

produce the final prediction. We can also explore integrating Large Language Models (LLMs) with MetaML to enhance pipeline recommendations optimization and interpretability. Moreover, we plan to explore efficient ways of integrating hyperparameter optimization into future versions of the method. Furthermore, incorporating datasets with varying percentages of missing values will be adopted for further analyses.

4 PIPES: A META-DATASET OF MACHINE LEARNING PIPELINES

Cynthia Moreira Maia, Lucas B. V. de Amorim, George D. C. Cavalcanti, Rafael M. O. Cruz

This chapter has been published as a paper in International Joint Conference on Neural Networks (IJCNN) 2025.

Abstract

Solutions to the Algorithm Selection Problem (ASP) in machine learning face the challenge of high computational costs associated with evaluating various algorithms' performances on a given dataset. To mitigate this cost, the meta-learning field can leverage previously executed experiments shared in online repositories such as OpenML. OpenML provides an extensive collection of machine learning experiments. However, an analysis of OpenML's records reveals limitations. It lacks diversity in pipelines, specifically when exploring data preprocessing steps/blocks, such as scaling or imputation, resulting in limited representation. Its experiments are often focused on a few popular techniques within each pipeline block, leading to an imbalanced sample. To overcome the observed limitations of OpenML, we propose PIPES, a collection of experiments involving multiple pipelines designed to represent all combinations of the selected sets of techniques, aiming at diversity and completeness. PIPES stores the results of experiments performed applying 9,408 pipelines to 300 datasets. It includes detailed information on the pipeline blocks, training and testing times, predictions, performances, and the eventual error messages. This comprehensive collection of results allows researchers to perform analyses across diverse and representative pipelines and datasets. PIPES also offers potential for expansion, as additional data and experiments can be incorporated to support the meta-learning community further. The data, code, supplementary material, and all experiments can be found at https://github.com/cynthiamaia/PIPES.git.

Keywords: Meta-Learning, Pipelines, Meta-Dataset.

4.1 INTRODUCTION

The Algorithm Selection Problem (ASP) was formulated by Rice in 1976 and refers to the challenge of choosing the most appropriate algorithm to solve a specific problem, considering

several algorithms available (RICE, 1976). ASP is one of the research focuses in the area of Meta-Learning (MtL) (SMITH-MILES, 2009; SONG; WANG; WANG, 2012; SOUTO et al., 2008; PIMENTEL; CARVALHO, 2019; FERRARI; CASTRO, 2015; KHAN et al., 2020), a field that allows to learn from previous machine learning experiences and transfer the acquired knowledge to new (HUTTER; KOTTHOFF; VANSCHOREN, 2019) tasks. MtL achieves this by mapping the characteristics (meta-features) of datasets with information describing the performance of algorithms. However, a limitation in this field is the high computational cost associated with evaluating algorithms' performances on datasets and extracting meta-features (KHAN et al., 2020). A promising approach to alleviate these costs is to use machine learning experiments available in public online repositories, where hundreds of experiments are released to the community, with a large set of machine learning techniques, which can be used to facilitate reuse and provide faster advancement in the area (BRAZDIL et al., 2022a). The effectiveness of these repositories depends heavily on community contributions, mainly through complete and efficient experiment registries. These shared registries help reduce the computational burden associated with individual experimentation.

One of the largest public repositories that enable sharing of experiments in machine learning is OpenML. It offers a wide range of datasets, experiments, and results. It makes the datasets available and provides detailed information about the experiments performed on this data, including descriptions of the pipeline blocks — such as scaling, encoding, feature preprocessing, imputation, class balancing, classification, clustering and regression algorithms, along with their corresponding hyperparameters and the evaluation metrics employed. This makes OpenML a rich metadata source, allowing researchers to explore, reproduce, and analyze results (BISCHL et al., 2017). OpenML offers a web API that makes it easy to submit new results by integrating it into popular machine learning tools (RIJN et al., 2013).

Although this repository offers a wide range of datasets, the quality and representativeness of recorded experiments may vary. One problem observed is the low usage of preprocessing blocks in pipelines. Even when these blocks are applied, limited exploration of their possible techniques is observed. This happens possibly because users can perform custom preprocessing and model selection in their pipelines without logging this information to OpenML. The experiments are typically not executed on OpenML's servers and may instead be run locally (BRAZDIL et al., 2022a), which hinders the creation of complete experiment registries. OpenML currently presents data about 22,298 machine learning pipelines. We carried out an analysis of these pipelines and found that only 47.09% included at least one preprocessing block; 23.20%

of the pipelines included function transformer block, 7.70% included the scaling block; 6.84% included feature preprocessing; 3.93% applied missing value imputation methods; 3.64% employed encoding for categorical variables; only 0.16% included data resampling techniques. Additionally, 1.60% registered the use of preprocessing techniques but did not specify which methods were applied. This indicates that most of the records in OpenML are focused on the classifier, neglecting the preprocessing steps of the pipeline (KÜHN et al., 2018), (PERRONE et al., 2018). This imbalance can induce bias in meta-learning systems that rely on OpenML for its meta-data.

Previous studies highlight the importance of preprocessing techniques (OBAID; DHEYAB; SABRY, 2019), (RAJU et al., 2020). A recent study investigated the impact of scaling techniques on the performance of classification algorithms, for example, (AMORIM; CAVALCANTI; CRUZ, 2023a). Its results show the importance of this preprocessing step and how it can significantly influence model performance. While the use of preprocessing techniques is beneficial, it is essential to have a diverse dataset that covers a wide range of scenarios for preprocessing algorithms. However, building such a comprehensive and diverse collection is challenging (PIO et al., 2024).

In this context, the main goal of this paper is to present PIPES, a meta-dataset for meta-learning that we are making available to the community. PIPES aims to support meta-learning by providing researchers with a comprehensive and representative collection of results covering different pipeline blocks (classifiers and data preprocessing) evaluated on multiple datasets. To this end, several pipelines are evaluated on datasets of various sizes. By incorporating multiple preprocessing blocks, each including many possible techniques in a representative way, PIPES aims to overcome the limitations observed in OpenML and provide a more complete collection of pipelines along with their results when applied to diverse datasets, facilitating machine learning and meta-learning research.

This comprehensive evaluation results in a set of records that can be used to advance meta-learning research, better understand the outcomes of different combinations of machine learning models and preprocessing techniques on different datasets and allow for important insights. The main contributions of this paper are:

- We present PIPES, a meta-dataset of machine learning experiments seeking completeness and diversity of the pipelines, including several preprocessing blocks and a classification block. The meta-dataset includes full details of the experiment setups and outcomes to

ensure easy replication by researchers aiming to advance research in meta-learning.

- We provide an API through which the user can fetch the meta-data and also expand PIPES. Easing interaction with the repository[1].
- We exemplify a use of PIPES in meta-learning research for pipeline recommendation.
- We analyze and compare the representativeness and completeness of pipelines from PIPES with those obtained from OpenML, one of the largest public repositories.

4.2 BACKGROUND AND RELATED WORK

Meta-Learning enables learning from prior machine learning experiences to tackle tasks like algorithm recommendation (HUTTER; KOTTHOFF; VANSCHOREN, 2019). This process involves two levels: base level and meta level. At the base level, algorithms and evaluation metrics are defined to evaluate their performance on datasets. At the meta level, a meta-model is trained using meta-datasets that comprise meta-examples — datasets represented by meta-features (characteristics of the dataset) and meta-targets (e.g., algorithm performances on the dataset). This trained meta-model can be used to recommend the most suitable base-level algorithm for new datasets. The type of meta-model depends on the meta-target (BRAZDIL et al., 2008).

Meta-features, which describe datasets, fall into categories like simple, statistical, information theory, model-based, complexity-based, and performance-based (landmarking) (BRAZDIL et al., 2022a). Simple meta-features capture general dataset properties (e.g., number of classes), while statistical ones focus on measures like mean and standard deviation. Information theory meta-features assess attributes like entropy (CASTIELLO; CASTELLANO; FANELLI, 2005), and model-based ones reflect properties of trained models (e.g., decision tree leaf count) (BRAZDIL et al., 2008). Landmarking meta-features evaluate the performance of simple, fast-to-train algorithms (such as Naive Bayes) on a dataset, providing an indication of the dataset's properties through the relative success of these algorithms (BRAZDIL et al., 2022a). Complexity meta-features analyze aspects like class separability and attribute overlap (RIVOLLI et al., 2022). Together, these meta-features enable effective algorithm recommendations for new datasets.

In meta-learning, most of the studies have focused on recommending predictive models, such as classifiers and regressors (ZHU et al., 2018; MISIR; SEBAG, 2017; PIMENTEL; CARVALHO,

[1] <https://github.com/cynthiamaia/PIPES.git>

2019; WANG et al., 2014; DANTAS; POZO, 2018). On the other hand, only a few studies have focused on recommending preprocessing algorithms (PIO et al., 2024; KHAN et al., 2023; AVELINO; CAVALCANTI; CRUZ, 2024; AMORIM; CAVALCANTI; CRUZ, 2024), and recommending hyperparameters for classifiers (ZHANG; SONG, 2015). Studies agree that one of the limitations of the meta-learning area is the computational cost associated with the meta-base construction phase. This cost arises because algorithms need to be executed on multiple datasets, making the process expensive (PIO et al., 2024), (KHAN et al., 2020). However, collaborative structures that allow sharing of experimental results can help reduce computational costs. Investigating how these tools are used in practice can significantly contribute to advances in the area. This type of analysis is crucial to understanding how the results obtained can effectively promote reproducibility and drive advances in meta-learning.

Vanschoren et al. (VANSCHOREN et al., 2012) present a structure aimed at facilitating the sharing of experiments in machine learning. Experiments were carried out on 84 datasets, evaluating 54 Weka algorithms. Fifty-six meta-features were calculated for each dataset, although the study does not explicitly state the types of meta-features considered. Furthermore, the experiments are not accessible, despite the link being provided in the study, they do not allow access to the structure. Nonetheless, this study was the precursor to the design of OpenML (VANSCHOREN et al., 2014). OpenML is an open-source platform that allows the sharing of datasets and experiment meta-data. It is integrated into popular platforms such as Weka, R, MOA, and Scikit-learn. It offers a website with access to 5,866 datasets and 22,298 machine learning pipelines, and also provides access to some kinds of meta-features, such as simple measures, statistics, information theory, and landmarking.

Kaggle is also an open-source platform that strives for reproducibility, but focuses on machine learning competitions and challenges (BOJER; MELDGAARD, 2021). Unlike OpenML, which offers a robust system for recording metadata in a standardized way, Kaggle doesn't provide standardization in describing and sharing experiments, limiting its application to structured analysis and reproducibility.

NAS-Bench-101 (YING et al., 2019) is an architecture meta-dataset designed for Neural Architecture Search (NAS). This dataset includes a comprehensive record of training and evaluation results for a wide range of Convolutional Neural Network (CNN) algorithms. The publicly available meta-dataset covers data, search space, and training code, aiming to promote reproducibility. However, unlike OpenML, which supports various machine learning tasks and pipelines, NAS-Bench-101 is focused on CNN architectures.

While OpenML is a comprehensive repository that supports a variety of machine learning tasks, its results have limitations. One is a lack of representativeness, especially when using pipeline blocks encompassing data preprocessing techniques. Many users perform experiments locally and may not log each step of the pipeline, compromising the integrity of the model performances provided and ultimately hindering progress in the meta-learning field. Less than half (47.09%) of the pipelines currently available in OpenML employ at least one preprocessing block. Among these, 23.20% uses function transformers, 7.70% uses scaling techniques, 6.84% uses feature preprocessing, 3.93% applies missing value imputation and 3.64% uses encoding strategies for categorical variables. The statistics highlight the need for greater exploration of data preprocessing techniques in experiments since many are little applied (or logged) regardless of their known importance to classification performance.

PIPES aims to provide large-scale record of machine learning experiments covering a diverse range of pipelines and datasets. It is therefore expected that it enables advances in meta-learning research, especially with regards to the effects of previously underrepresented pipeline blocks and techniques.

4.3 PROPOSED META-DATASET

A meta-dataset is an indispensable input in a meta-learning process. In the context of the algorithm selection problem, for example, a meta-learning process consists of recommending the most suitable algorithm for a specific task based on metadata from previous experiments in machine learning. In this sense, this study proposes a meta-dataset that enables the sharing of well-structured experiments, taking into account the interactions among the several pipeline blocks and on different datasets. Therefore, the proposed PIPES meta-dataset consists of a comprehensive collection of pipeline details associated with their outcomes for different datasets.

The goal is to provide the machine learning research community with a valuable meta-dataset, which can be used to train models in a holistic way, to recommend complete pipelines (i.e., including the predictive model and the data preprocessing blocks). Since the models will be trained on a large meta-dataset that is built with a focus on completeness, it is expected that they will present a high generalization power. This way, PIPES can help the community identify the most promising pipelines to solve specific problems based on frequency of use and performance in previous experiments. This knowledge can help in designing more effective and

targeted search spaces.

Formally, the construction of the PIPES meta-dataset is defined as follows. Consider the set of datasets used, $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_z\}$. For each dataset $\mathbf{D}_i \in \mathbb{D}$, all the different pipelines are applied, with each pipeline consisting of a chain of blocks (steps) that each have several possible techniques. Given the following set definitions:

$\mathbb{T} = \{t_1, t_2, \ldots, t_k\}$: The set of imputation techniques.

$\mathbb{E} = \{e_1, e_2, \ldots, e_l\}$: The set of encoding techniques.

$\mathbb{P} = \{p_1, p_2, \ldots, p_m\}$: The set of scaling techniques.

$\mathbb{A} = \{a_1, a_2, \ldots, a_n\}$: The set of feature preprocessing, transformation and feature selection.

$\mathbb{C} = \{c_1, c_2, \ldots, c_p\}$: The set of classifiers.

A pipeline is a tuple containing five elements (blocks). The first block is populated by an imputation technique $t \in \mathbb{T}$, the second block corresponds to a categorical encoding technique $e \in \mathbb{E}$, then comes the pipeline block responsible for data scaling with a technique $p \in \mathbb{P}$, followed by a feature preprocessing techinique $a \in \mathbb{A}$, and, finally a classification algorithm $c \in \mathbb{C}$. In Fig. 19, we illustrate an example of a specific pipeline.

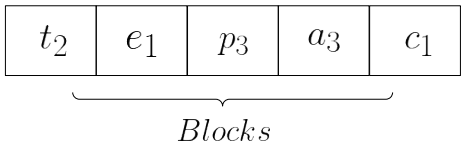


Figure 19 – Representation of an example of a specific pipeline.

Thus, for each dataset $\mathbf{D}_i \in \mathbb{D}$, the set of possible pipelines $\mathbb{S}$ is given by the combination of all elements of the sets $\mathbb{T}, \mathbb{E}, \mathbb{P}, \mathbb{A}$ and $\mathbb{C}$, that is:

$$\mathbb{S} = \mathbb{T} \times \mathbb{E} \times \mathbb{P} \times \mathbb{A} \times \mathbb{C} \tag{4.1}$$

Finally, considering that each dataset $\mathbf{D}_i$ is represented by a vector $\mathbf{f}_i = \{f_i^{(1)}, f_i^{(2)}, \ldots, f_i^{(x)}\}$, with x meta-features that characterize its properties, the meta-dataset $\mathbb{S}^*$ is generated, bringing together all pipelines and the results of each dataset in each fold, covering all possibilities, not limited to just the best performance. Thus, we have the following formulation:

$$\mathbb{S}^* = \left\{ (\mathbf{f}_i, S_j, H, \tau_{\text{train}}, \tau_{\text{test}}, \epsilon) \mid \begin{array}{r} \forall i \in \{1, 2, 3, \ldots, z\}, \\ \forall S_j \in \mathbb{S}, \\ H, \tau_{\text{train}}, \tau_{\text{test}} \in \mathbb{R} \end{array} \right\} \tag{4.2}$$

Where $\mathbb{R}$ is the set of real numbers. Therefore, the set $\mathbb{S}^*$ contains a detailed collection of experiments using different pipelines, including dataset characteristics $\mathbf{f}_i$, the achieved performance H, the training and testing times τ_{train} and τ_{test} and a text field with information about errors (ϵ) occurred during processing, if any. These errors may refer, for example, to situations where a dataset contains missing values and, in certain pipelines, the imputation technique is absent.

4.4 META-DATASET CONSTRUCTION

In this section, we detail the procedure performed to build the proposed meta-dataset according to the formal description in Section 4.3.

4.4.1 Datasets

The datasets were collected from the open-source OpenML platform in September 2024. We selected only datasets with the number of instances in the range [100, 100000], the number of classes in [2, 100], and the number of attributes in [5, 100]. Initially, the procedure resulted in a total of 1,248 datasets. From these datasets, 867 were excluded to reduce the number of datasets belonging to the same family and to remove repeated datasets the had been logged with different names, resulting in 381 distinct datasets. From the 381 datasets selected, 273 are binary-class problems, and 108 are multi-class problems. The final list of datasets used is presented in the supplementary material.

4.4.2 Pipeline blocks

Given the wide variety of different classification algorithms and preprocessing methods, we limited the selection of the techniques to populate the pipeline blocks based on the methods used in Auto-Sklearn (FEURER et al., 2015), which uses the Scikit-learn package (PEDREGOSA et al., 2011b). The pipelines' blocks and their possible techniques are presented in Table 10. Note that each preprocessing block includes an option of not being executed (None), while the classification block is the only one that is mandatory.

This results in a total of 9,408 combinations, i.e., pipelines, (2 imputation techniques $\times$ 3 categorical encoders $\times$ 7 scalers $\times$ 14 feature preprocessing techniques $\times$ 16 classification

Table 10 – Pipeline blocks and their possible techniques.

Imputation ($\mathbb{T}$)	Encoding ($\mathbb{E}$)
SimpleImputer(SI), None.	OrdinalEncoder (OE), OneHotEncoder (OHE), None.

Scaling ($\mathbb{P}$)
MinMaxScaler (MM), StandardScaler (SS), PowerTransformer (PT), QuantileTransformer (QT), RobustScaler (RS), Normalizer (Nor), None.

Feature Preprocessing ($\mathbb{A}$)
ExtraTrees prep (ETP), FastICA (FICA), Nystroem (NY), FeatureAgglomeration (FAGG), GenericUnivariateSelect (GU), LinearSVC prep (LSVCP), Principal component analysis (PCA), KernelPCA (KPCA), Radial Basis Function Sampler (RBFS), PolynomialFeatures (PF),RandomTreesEmbedding (RTE), SelectPercentile (SP), TruncatedSVD (TSVD), None.

Classification ($\mathbb{C}$)
AdaBoost (AB), BernoulliNB (BNB), DecisionTree (DT), ExtraTrees (ET), GaussianNB (GNB), HistGradientBoosting (HGB), K-Nearest Neighbors (KNN), LinearDiscriminantAnalysis (LDA), LinearSVC (LSVC), Multi-layer Perceptron (MLP), MultinomialNB (MNB), QuadraticDiscriminantAnalysis (QDA), PassiveAggressive (PA), Support Vector Classification (SVC), Stochastic Gradient Descent (SGD), RandomForest (RF).

algorithms). We consider the hyperparameters in their default values for this first analysis; however, we highlight the potential to extend our meta-dataset in future work to also address hyperparameter variations. Apart from these techniques, Auto-sklearn also incorporates a class balancing method that is embedded within AutoML system and not as easy to replicate. Therefore, we focus on techniques and algorithms that are easily accessible and configurable in Scikit-learn.

Following the definition in Eq. 4.1, all pipeline possibilities are explored. Some of these pipelines may be invalid or lead to errors for some datasets. Nonetheless, all of them were documented and included in the meta-dataset. The meta-dataset, therefore, covers all results, not just the best one for each dataset, allowing analyses of the performance of all pipelines across all datasets.

4.4.3 Meta-Features

The extracted meta-features belong to six groups: simple (12 features), information theory (13 features), statistics (48 features), model-based (24 features), landmarking (14 features), and complexity (34 features), with a total of 145 meta-features, which are detailed in the supplementary material.

4.4.4 Hardware and Software

We employed three computing clusters and ran all the pipelines on 300 datasets through several parallelized jobs, each job running a fraction of the pipelines on a given dataset. After the executions were completed, the results were combined into a structured table containing the various details necessary for composing the meta-dataset. We used the Python language (version 3.9.1) and the following packages: Scikit-Learn (version 1.2.2) (PEDREGOSA et al., 2011b), (version 1.5.3), OpenML (version 0.15.0) (FEURER et al., 2021), and PyMFE (version 0.4.2) (ALCOBAÇA et al., 2020) .

4.5 ANALYSIS

This section details the experiment performed and seeks to answer the following research questions:

RQ1 Does PIPES overcome OpenML's limitations regarding the biases and data imbalance of pipelines and contribute to the advancement of meta-learning?

RQ2 Are the datasets selected in PIPES diverse?

To answer RQ1, we perform two analyses that we present in subsections 4.5.1 and 4.5.2. First, we perform an exploratory analysis of the pipelines available in OpenML (subsection 4.5.1). The idea is to study how frequently each technique is used within each pipeline block. Then, in subsection 4.5.2, we present an example of using PIPES for a meta-learning task to recommend the optimal techniques for Feature Preprocessing and Scaling, given a fixed classifier, the SVC. These two particular blocks and the classifier were chosen because of their high frequency in OpenML data, based on the outcomes of the exploratory analysis in subsection 4.5.1. We compare the classification performance achieved using the optimal pipeline from OpenML data versus PIPES data. Finally, in subsection 4.5.3 we answer RQ2 by studying the diversity of the PIPE's datasets according to their meta-feature representations.

4.5.1 Exploratory analysis of the pipelines from OpenML

In this first analysis, we explore the pipelines recorded in OpenML w.r.t. the frequencies of each block's techniques. In Fig. 20, the bar plots show these frequencies for all five pipeline

blocks covered in our study. These graphs consider 6,729,117 classification experiments using Scikit-learn present in OpenML records.

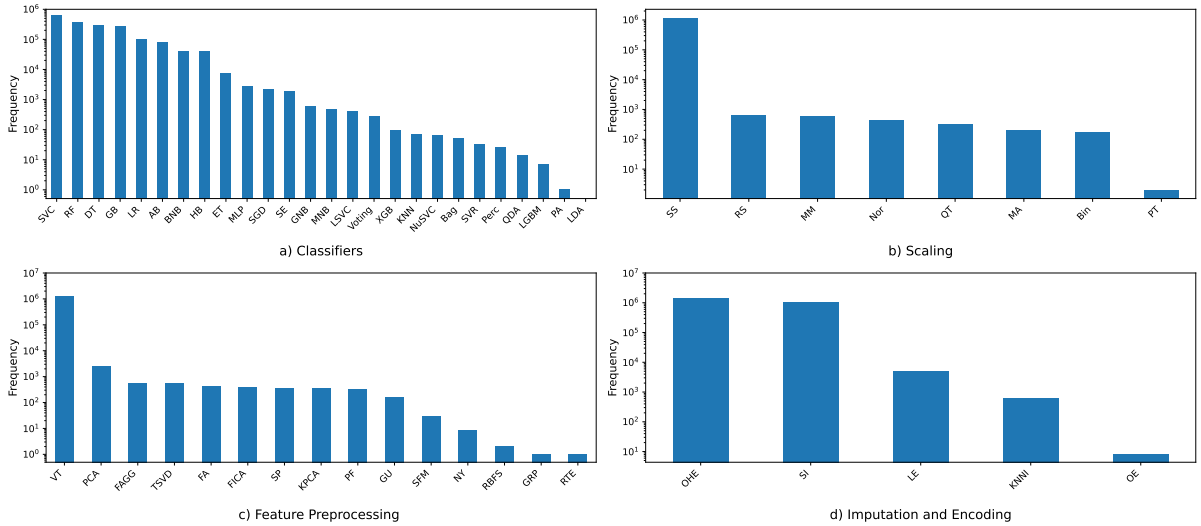


Figure 20 – Frequency of execution of each technique, per pipeline block, according to OpenML records. Most of the abbreviations for FP and scaling are given in Table 10. Additional abbreviations: Classifiers: LR - LogisticRegression, XGB - XGBoost, NuSVC - Nu-Support Vector Classification, LGBM - LightGBM, Bag - Bagging, Perc - Perceptron, SE - StackingEstimator, Voting - VotingClassifier. Scaling: Bin - Binarizer.

In Fig. 20(a), we observe that a large number of classifiers have been logged to OpenML, but their frequencies show that while some models have been used more than 100,000 times (SVC, RF, DT, GB), the majority of the models appear for less than 1,000 times, with some extreme cases presenting less than 100 examples. There is clearly a preference for SVM and decision-tree-based algorithms. The same kind of imbalance can be observed for the remaining blocks. We highlight the scaling block, where StandardScaler is employed approximately 1 million times, while the remaining techniques appear less than 1,000 times, with PowerTransformer being used less than 10 times. Moreover, upon further inspection of the data, we note that 87% of the examples using MinMaxScaler use only six datasets, 83% of RobustScaler's appearances occur using only seven datasets, and 100% of PowerTransformer's examples use just one single dataset. This concentration means that, although there are many experiments available in OpenML's records, there is a limitation in the diversity of contexts in which these techniques are applied. This imbalance in the representativeness of machine learning pipeline blocks can induce bias and negatively affect the quality of meta-learning models that rely on this repository for meta-data.

Another important aspect is the lack of diversity in the combinations of preprocessing techniques in the pipelines. Fig. 21 shows that most of the classification pipelines recorded on OpenML use only one or no preprocessing block at all. As the number of preprocessing

blocks used increases, the number of pipelines decreases. Only 2.74% of the pipelines employ all four pipeline blocks. This leads to bias in the way the blocks are combined. For example, VarianceThreshold, which is the most frequent Feature Preprocessing technique is, in most cases, associated with the StandardScaler scaling technique, while FeatureAgglomeration is only used with StandardScaler in two examples, and KernelPCA, Nystroem, FastICA and TruncatedSVD are never combined with scaling techniques. These observations reflect a limited exploration of different combinations of techniques within pipelines and the use of preprocessing techniques in OpenML records.

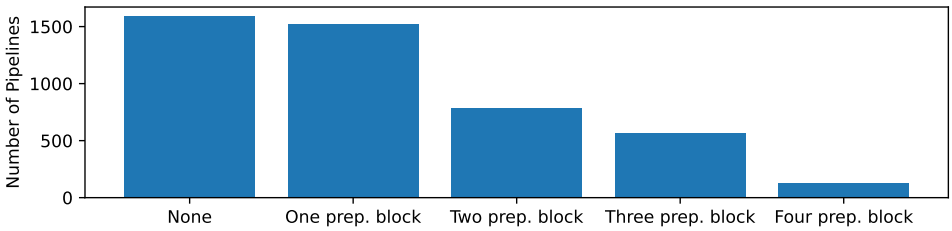


Figure 21 – Quantity of pipelines that employ each number of preprocessing blocks. Considering all OpenML classification pipelines that use Scikit-learn.

Therefore, the main limitations observed in OpenML records include a large concentration of examples using a few of the techniques within each block, the lack of exploration of the possible combinations of pipeline blocks, and a large portion of the examples using just a few datasets. PIPES addresses these limitations by exploring pipeline blocks in depth, with greater representation and diversity in use. In our proposed PIPES meta-dataset, every possible pipeline is applied to every dataset. All the techniques within each pipeline block are equally explored, providing a balanced meta-dataset, which allows for an unbiased training of recommendation meta-models.

4.5.2 Comparing PIPES and OpenML in a meta-learning task

In this subsection, we present an example comparing the use of PIPES and OpenML in a meta-learning task, contributing to answering RQ1. The meta-learning task in this example is to automatically recommend the optimal feature preprocessing and scaling techniques for the SVC classifier, given a particular dataset represented by its meta-features vector. In this example, two meta-datasets are built: one based on PIPES' results and the other using OpenML results. Subsequently, these meta-data sets are compared to assess their representativeness. The goal is to examine the registered pipelines and their representativeness in using feature preprocessing and scaling blocks. These two preprocessing blocks and the SVC classifier were

selected based on the analysis outcomes in subsection 4.5.1, choosing the most commonly used blocks in OpenML to retain a substantial amount of learning data, maintaining a fair comparison. We also restrict the datasets to only the 192 that appear in both PIPES and OpenML.

The OpenML meta-dataset, $\mathbf{M}_{\text{OpenML}}$, used for this analysis is composed as follows. First, $\mathbb{D}_{\text{selected}}$ contains all 192 datasets in common with PIPES. After, given the set $\mathbb{F}$ of pipelines that use SVC as the classifier, defined as $\mathbb{F} = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, we take only the pipelines that are the best for each selected dataset to form $\mathbb{F}_{\text{best}}$, defined in Eq. 4.3, which constitutes our meta-target.

$$\mathbb{F}_{\text{best}} = \{\sigma_{\text{best}}(\mathbf{D}_i) | \forall \mathbf{D}_i \in \mathbb{D}_{\text{selected}}\} \tag{4.3}$$

Then, the meta-features vectors $\mathbf{f}_i$ representing each dataset $\mathbf{D}_i$ are merged with the meta-targets. The resulting meta-dataset $\mathbf{M}_{\text{OpenML}}$ is composed of the tuples $(\mathbf{f}_i, \sigma_{\text{best}}(\mathbf{D}_i))$, allowing the recommendation of pipeline based on the datasets' meta-features, as shown in Eq. 4.4.

$$\mathbf{M}_{\text{OpenML}} = \{(\mathbf{f}_1, \sigma_{\text{best}}(\mathbf{D}_1)), (\mathbf{f}_2, \sigma_{\text{best}}(\mathbf{D}_2)), \ldots, (\mathbf{f}_z, \sigma_{\text{best}}(\mathbf{D}_z))\} \tag{4.4}$$

We define $\mathbf{M}_{\text{PIPES}}$ following the same procedure described for OpenML but using PIPES' best pipelines for each dataset.

$$\mathbf{M}_{\text{PIPES}} = \{(\mathbf{f}_1, \pi_{\text{best}}(\mathbf{D}_1)), (\mathbf{f}_2, \pi_{\text{best}}(\mathbf{D}_2)), \ldots, (\mathbf{f}_z, \pi_{\text{best}}(\mathbf{D}_z))\} \tag{4.5}$$

Where each pair $(\mathbf{f}_i, \pi_{\text{best}}(\mathbf{D}_i)$ represents the meta-features of the dataset $\mathbf{D}_i$ and the pipeline $\pi_{\mathbf{D}_i}$ which provided the best performance for dataset $\mathbf{D}_i$.

In Fig. 22, the frequency with which each FP and Scaling technique appear in the pipelines from the meta-datasets is presented. Fig. 22a and Fig. 22c show the results for $\mathbf{M}_{\text{OpenML}}$ and Fig. 22d and Fig. 22c for $\mathbf{M}_{\text{PIPES}}$. Notice how the best technique indicated using OpenML data is much less varied and distributed than indicated by PIPES results. The careful analysis of the results from meta-dataset $\mathbf{M}_{\text{PIPES}}$ can also reinforce the use of certain techniques over others according to the selected algorithm, which can help create curated search spaces for optimization processes, reducing computational cost and increasing performance for recommending pipelines.

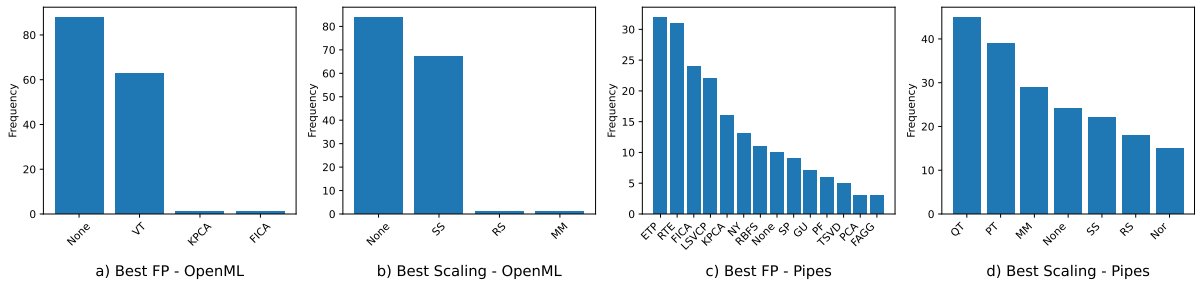


Figure 22 – The graphs show how frequently each technique appears in the best pipelines for the 192 datasets in common. Most abbreviations for FP and scaling are given in Table 10. Additional abbreviations: VT - VarianceThreshold, FA - FactorAnalysis, SFM - SelectFromModel. Scaling: MA — Max Absolute Scaler.

One possible explanation for the pipeline imbalance obtained from OpenML is incomplete pipeline reporting. Many users run experiments locally, and many may not report each step of the pipeline, harming the reliability of the records.

Considering the accuracy obtained by each pipeline on the 192 datasets using 5-fold cross-validation, we compare the two meta-datasets in Table 11, where we also show the number of wins, ties, and losses of PIPES versus OpenML, the mean ranking, and the p-value from a Wilcoxon signed-rank test comparing these rankings. Notice how the PIPES pipelines obtain a higher accuracy of 0.84 versus 0.73 from the OpenML pipelines. Additionally, PIPES only lost 3 times against OpenML, winning in 93% of the datasets. Consequently, the mean ranking of PIPES is 1.04, against 1.96 for OpenML (lower is better). The p-value obtained refutes the hypothesis that $\mathbf{M}_{\text{PIPES}}$ and $\mathbf{M}_{\text{OpenML}}$ have similar average rankings.

Table 11 – A analysis of the mean accuracy, Wins/ties/losses of the $\mathbf{M}_{\text{PIPES}}$ against $\mathbf{M}_{\text{OpenML}}$, mean ranking, p-value of the Wilcoxon signed rank test ($\alpha = 0.05$.)

	$\mathbf{M}_{\text{PIPES}}$	$\mathbf{M}_{\text{OpenML}}$
Mean acc.	**0.840**	0.735
Win/tie/loss	180/9/3	
Mean rank	**1.039**	1.960
p-value	1.021e-31	

Now, answering **RQ1: Does PIPES overcome OpenML's limitations regarding the biases and data imbalance of pipelines and contribute to the advancement of meta-learning?** Yes, PIPES' strategy of completeness-oriented meta-dataset construction, which gives each technique the same opportunity in the records, allows for less biased and more balanced meta-model learning that can potentially lead to new insights, contributing to the advancement of meta-learning.

4.5.3 PIPES' datasets diversity

To answer **RQ2: Are the datasets selected in PIPES diverse?**, in Fig. 23, the 280 datasets from PIPES are represented in terms of all their meta-features after a transformation to a bi-dimensional space using Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP)(MCINNES; HEALY; MELVILLE, 2018), configured with $k = 3$ neighbors. Due to computational failures during the execution process, only 280 datasets could be successfully processed and included in the analysis. We observe that the datasets tend to cover the extent of the space rather than appearing in a cluster, which indicates diversity. A meta-dataset composed of diverse pipelines contributes to the meta-learning area, as it allows the exploration of different combinations of pipeline blocks, taking into account the specific characteristics of each dataset, representing different possible real-life scenarios.

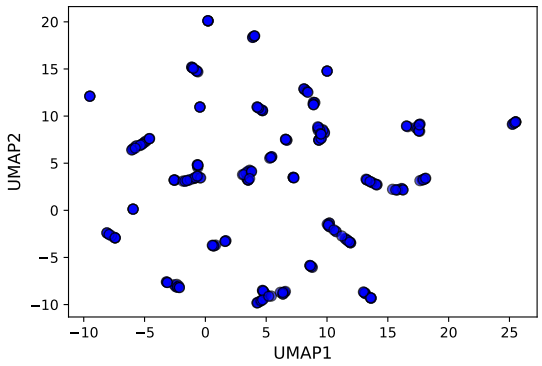


Figure 23 – PIPES' datasets organized according to a UMAP representation of their meta-features' space.

4.6 LIMITATIONS

PIPES offers a broad meta-dataset but is not without limitations. The main drawbacks are: (i) it does not take hyperparameters into account, (ii) it misses some other common pipeline blocks, such as data balancing. Additionally, its current architecture is not yet completely finalized, and improvements are needed, mainly in integration, to facilitate user visualization and metadata access.

4.7 CONCLUSION

We proposed PIPES, a meta-dataset for meta-learning. PIPES is designed for the machine learning community to enable meta-learning experimentation on a rich collection of experiments that does not neglect important pipeline blocks, particularly preprocessing, such as scaling, feature preprocessing, imputation, and encoding. Given the difficulty of obtaining collections of representative experiments from OpenML, one of the most significant learning and machine repositories, the idea is that PIPES can contribute to the reproducibility of experiments on a representative meta-dataset covering a wide range of techniques.

PIPES can be used in the meta-learning area for tasks such as recommending the most suitable pipeline for specific datasets, analyzing the impact of different block combinations across various scenarios, and identifying promising pipelines to refine search spaces in recommendation systems. Its main advantages include broad coverage of problem domains, representative pipeline blocks (covering classifiers and preprocessing techniques), and detailed training/testing time data for cost-benefit analysis. In addition, it provides all predictions, allowing researchers to apply various evaluation metrics. For future work, we intend to optimize the algorithms' hyperparameters, complete the execution of the datasets, add more data and pipeline blocks, and also create an accessible tool to facilitate users' insertion and retrieval of data.

5 METAML 2.0

Abstract

The original MetaML was developed using historical machine learning experiments available on OpenML; however, OpenML exhibited limited diversity in pipeline use, with low records of the use of preprocessing techniques. To overcome this limitation, we propose MetaML 2.0, which improves the original version by utilizing experiments available in PIPES to identify promising pipelines, covering a broader diversity of preprocessing blocks, and providing recommendations for datasets with specific characteristics. The meta-dataset comprises 117 datasets, and a comparative analysis was conducted on 78 of these datasets, demonstrating the effectiveness of the proposed method. This work aims to address the main research questions related to the performance of MetaML 2.0 following the inclusion of more diverse preprocessing blocks of PIPES.

Keywords: Meta-Learning, Multi-Label, Pipeline, Preprocessing.

5.1 INTRODUCTION

The idea of MetaML in Chapter 3 was constructed using historical data from machine learning experiments available from OpenML, one of the largest public machine learning repositories. One of the problems is that preprocessing techniques are rarely employed, and even when they are, just a few are explored. It is possible that users perform custom preprocessing and model choosing in their pipelines, but do not log that information on OpenML. This may have affected the diversity and comprehensiveness of pipeline configurations available for further curation by MetaML's search space curation component, ultimately degrading the meta-model performance. Given the difficulty of obtaining collections of representative experiments from OpenML, one of the most significant learning and machine repositories, we constructed PIPES, a meta-dataset of machine learning experiments seeking completeness and diversity of the pipelines, including several preprocessing blocks and a classification block. Building on PIPES and aiming to overcome the limitations observed in MetaML, we improve upon the version presented in Chapter 3 of this thesis, with Metaml 2.0. MetaML 2.0's primary objective is to improve the quality of metadata provided in the original version by expanding the set of available preprocessing techniques and machine learning algorithms. The Search

space definition, Meta-dataset construction, Training and Recommendation phases follow the same approach described in Chapter 3, Section 3.4, maintaining the methodological consistency presented previously. The core algorithmic Search space principle remains, designed to learn from historical data; the modification is that the new version expands this step by using a different metadata repository. The specific details of this extension are discussed in the following subsections. This new version is built upon the PIPES meta-dataset and aims to identify promising pipelines and recommend rankings of pipelines most suitable for specific

This study is organized as follows: Section 5.2 presents the experimental setup; Section 5.3 discusses the results and addresses the research questions; and Section 5.4 concludes the study.

5.2 EXPERIMENTAL SETUP

5.2.1 Datasets and Search Space

The datasets used in the experiments were collected from PIPES, as described in Chapter 5. Initially, 320 datasets were retrieved from the repository; however, 27 of them failed during execution, leaving 293 datasets for analysis. These datasets were used as input for the procedure described in Section 3.4.1, with the algorithm parameters set to $n = 800$ and $k = 500$. During the search space definition phase, we empirically tested two parameter configurations:(i) $n = 5$, $k = 5$ and (ii) $n = 800$, $k = 500$, to evaluate the effect on search space curation.

When analyzing the configuration with (i) $n = 5$ and $k = 5$, we observed that the final number of datasets available for analysis was limited to only 16 datasets. This small sample restricts the construction of the meta-dataset. To address this limitation and increase diversity, we decided to adjust configurations with larger values of $n = 800$ and $k = 500$. This decision was also motivated by the presence of many tied results in the experiments, where different pipeline combinations yield identical performance, making broader sampling.

However, after the last filtering step of the search space definition, which selects only the most frequent pipelines, the final number of experiments in S is 146, and consequently, the number of unique datasets is the same, as we selected only one (the best) experiment for each dataset. Some complexity meta-features could not be calculated for twenty-nine of these datasets, which were then removed. Therefore, our final number of instances in the meta-dataset was 117. The resulting search space consists of five classification algorithms

(*Multilayer Perceptron*, *Extremely Randomized Trees*, *Histogram-Based Gradient Boosting*, *Support Vector Classifier*, and *Random Forest*) and twelve preprocessing techniques (*Simple Imputer*, *One-Hot Encoder*, *Ordinal Encoder*, *Extra Trees Preprocessor*, *Polynomial Features*, *Kernel Principal Component Analysis*, *Random Trees Embedding*, *Power Transformer*, *Robust Scaler*, *Quantile Transformer*, *Min-Max Scaler*, and *Standard Scaler*).

5.2.2 Meta-dataset construction and Meta-Model

This step extracts meta-features for each dataset and maps them to the best-performing pipeline in the search space $\mathbb{S}$. The meta-features are grouped into five categories: simple, information-theoretic, statistical, model-based, and complexity, and are summarized in the Appendix K. To select an algorithm to use as MetaML's 2.0 meta-model, Probabilistic Classifier Chains (PCC) were considered. As the base estimator for the PCC, we used a Decision Tree classifier, with entropy as its splitting criterion and a minimum of 10 samples in a leaf node.

5.2.3 Data Preprocessing

The values of the meta-features used in this study vary across different intervals and may present outliers. Therefore, the Quantile Transformer scaling algorithm was employed due to its robustness to outliers (PEDREGOSA et al., 2011a) (AMORIM; CAVALCANTI; CRUZ, 2023b).

From the 130 meta-features initially considered, we excluded those with more than 10% of missing values, resulting in 81 meta-features. The list of the 49 excluded features can be found in Appendix L. Other meta-features with fewer missing values were imputed using KNN (ZHANG, 2012) ($k = 5$), namely, f1v.mean (9.40cov.sd (7.69%), cov.mean (7.69%), gravity (7.69%), iq_range.mean (7.69%), l1.mean (7.69%), l2.mean (7.69%), l3.mean (7.69%), eigenvalues.sd (7.69%), n3.mean (7.69%), n4.mean (7.69%), n4.sd (7.69%), iq_range.sd (7.69%), t2 (7.69%), mad.mean (7.69%), t4 (7.69%), mad.sd (7.69%), n3.sd (7.69%), nodes_repeated.sd (5.98%), nodes_per_level.sd (5.13%), tree_imbalance.sd (5.13%), cat_to_num (3.42%), t_mean.sd (0.85%), t_mean.mean (0.85%).

5.2.4 Software and Hardware

The MetaML was implemented using the Python language (version 3.9.1) and the following packages: Scikit-Learn (PEDREGOSA et al., 2011a) (version 1.2.2), Pandas (MCKINNEY, 2010) (version 1.5.3), Scikit-Multilearn (SZYMAŃSKI; KAJDANOWICZ, 2017) (version 0.2.0), and PyMFE (ALCOBAÇA et al., 2020) (version 0.4.2). The experiments for the comparative analysis were performed using the AutoML Benchmark on the Linux operating system, utilizing an eight-core, 3.7 GHz AMD Ryzen 7 2700X processor with 16 GB of RAM for the larger datasets, and a six-core, 3.4 GHz AMD Ryzen 5 processor with 16 GB of RAM for the smaller datasets.

5.2.5 Evaluation procedure

The final comparative analysis used 78 test datasets to evaluate MetaML 2.0 against the following frameworks: Auto-WEKA, Auto-Sklearn 1.0, Auto-Sklearn 2.0, AutoGluon, FLAML-Zero, H2O AutoML, Naive AutoML, TabPFN, and TPOT, using the accuracy metric. Initially, a subset of 152 test datasets was considered (as discussed in the Evaluation Procedure 3.5.5 Section 3.5.5, Chapter 3); however, some datasets were removed because they either appeared in or belonged to the same family as those used to construct the MetaML 2.0 meta-dataset.

The experiments described in this section are designed to answer the following research questions:

RQ1 How does including preprocessing blocks (e.g., scaling, feature selection) impact classification algorithm performance compared to classifier-only pipelines across different datasets in PIPES?

RQ2 Does PIPES significantly improve the diversity of pipeline configurations in MetaML 2.0 compared to OpenML-derived meta-datasets (MetaML), particularly regarding preprocessing techniques?

RQ3 Does incorporating MetaML 2.0 preprocessing blocks allow for recommending pipelines that achieve better predictive performance with lower computational cost compared to state-of-the-art methods?

The MetaML 2.0 is trained and tested using a leave-one-dataset-out cross-validation procedure (LODOCV) (see Chapter 3, Section 3.5.5).

5.3 RESULTS AND DISCUSSION

5.3.1 Meta-level analysis

The meta-model performance is calculated by taking the mean of the meta-model performances (in terms of the F1 and Precision metrics) over the 117 test instances. The meta-model achieved an F1-score of 0.224 and a Precision of 0.255.

5.3.2 The impact of using preprocessing blocks

The formation of the meta-targets in the meta-dataset, which corresponds to the results from the performance of the pipelines, demonstrates the impact of preprocessing blocks. We observed that of the 117 datasets in the meta-base, only seven datasets (6%) had the best pipeline, which consisted solely of the classifier without any preprocessing steps. This indicates a low number of cases where the classifier achieves better performance without the combination of preprocessing blocks. To investigate whether there is a significant difference in using preprocessing blocks versus using a classifier alone, we applied the paired Wilcoxon signed-rank test to determine whether the observed performance difference is statistically significant.

To focus the analysis on the impact of preprocessing blocks, we excluded these seven datasets from this analysis. For the remaining 110 datasets (where preprocessing was a component of the best pipeline), we compared the performance of the best pipeline (with preprocessing) with the performance of the classifier pipeline used alone (without preprocessing) on the same dataset. We present the results in a Figure 24. This approach allows us to analyze the effect of preprocessing on performance.

We define the two approaches compared as follows: Pipeline with preprocessing: the best-performing configuration for a dataset, which includes at least one preprocessing block. Classifier-only pipeline: performance of the classifier used alone, without any preprocessing blocks. In the Figure 24, pipelines that include preprocessing blocks show a higher concentration of higher accuracy values. The Wilcoxon test revealed statistically significant differences between pipelines with and without preprocessing (p-value $=$ 2.657e-18). Pipelines with preprocessing blocks showed a higher average accuracy of 0.879, compared to 0.686 for pipelines without preprocessing.

These results allow us to answer the research question: **[RQ1]** How does including pre-

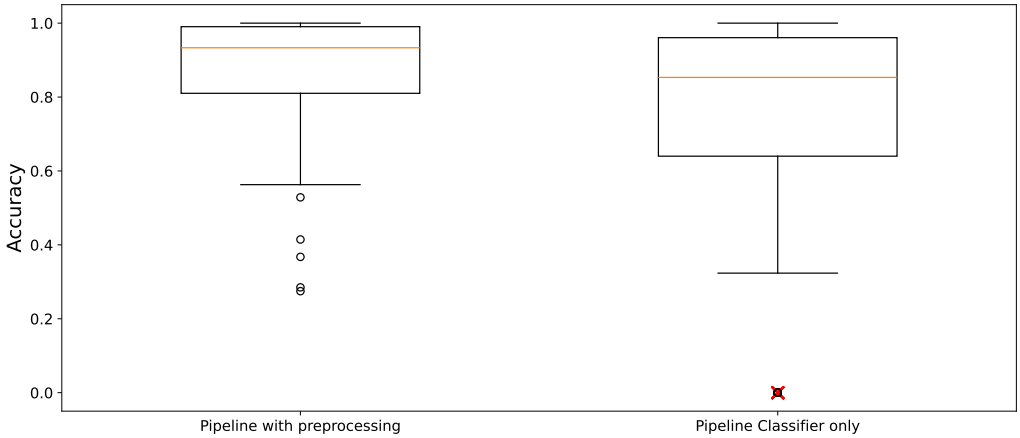


Figure 24 – Pipelines performance on 110 datasets, with and without preprocessing blocks.

processing blocks (e.g., scaling, feature selection) impact classification algorithm performance compared to classifier-only pipelines across different datasets in PIPES?

The inclusion of preprocessing blocks impacts significant classifier performance, resulting in higher accuracy compared to pipelines that use only the classifier. In some cases, their absence, such as imputation and encoding blocks, renders performance unfeasible for pipelines that use only the classifier, as indicated by the boxplot in Figure 24, marked by the red dots in pipelines that use only the classifier.

5.3.3 Comparing Pipeline Diversity: PIPES vs. OpenML

To answer the research question **[RQ2]** Does PIPES significantly improve the diversity of pipeline configurations in MetaML 2.0 compared to OpenML-derived meta-datasets (MetaML), particularly regarding preprocessing techniques? We illustrate in Figure 25 the frequency of meta-targets in MetaML 1.0, built with OpenML, and in MetaML 2.0 with PIPES, to represent their diversity.

We observed greater diversity in the preprocessing blocks of MetaML 2.0 (a) compared to MetaML 1.0 (b). For example, in the Feature Preprocessing block, MetaML 2.0 uses different techniques, while in MetaML 1.0, VarianceThreshold predominated. We also identified differences in the distribution of Scaling methods and Classifiers. In MetaML 1.0, the most frequent classifiers were tree-based (Decision Tree and Random Forest), followed by SVC. In MetaML 2.0, we observed the use of tree-based classifiers (HistGradientBoosting, ExtraTrees, and Random Forest), followed by MLP and SVC. The distribution among classifiers was more diverse, indicating a more exploration of the pipelines. Therefore, PIPES significantly improved

the diversity of pipeline configurations in MetaML, particularly in the preprocessing blocks.

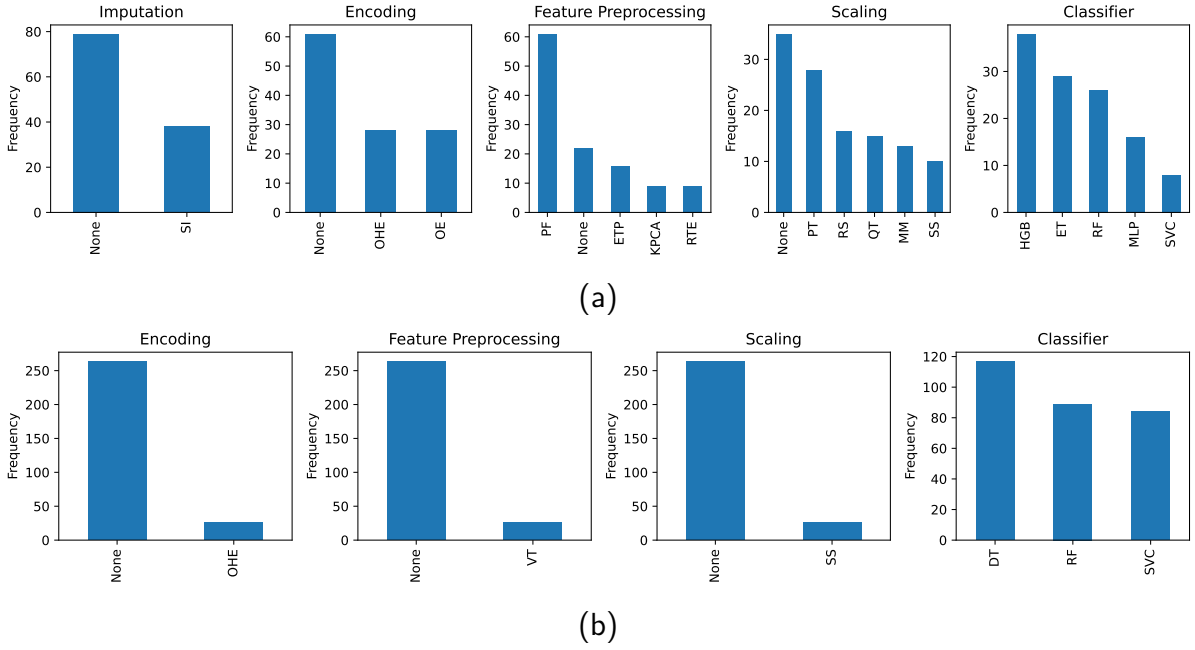


Figure 25 – The graphs show the frequency distribution of each meta-target across 117 datasets of MetaML 2.0 (a) and 290 datasets of MetaML 1.0 (b). Additional abbreviations are provided below: SI - Simple Imputer, OHE - OneHotEncoder, OE - OrdinalEncoder, PF - PolynomialFeatures, ETP - ExtraTrees pre, KPCA - KernelPCA, RTE - RandomTreesEmbedding, PT - PowerTransforme, RS - RobustScaler, QT - QuantileTransformer, MM - MinMaxScaler, SS - StandardScaler, HGB - HistGradientBoosting, ET - ExtraTrees, RF - RandomForest, MLP - Multi-layer Perceptron, SVC - Support Vector Classification, VT - VarianceThreshold, DT - Decision Tree.

5.3.4 Comparing the performances of recommended pipelines

To address item **[RQ3]** Does incorporating MetaML 2.0 preprocessing blocks allow for the recommendation of pipelines that achieve better predictive performance with lower computational cost compared to state-of-the-art methods? Table 12 compares the accuracy of the pipelines recommended by MetaML and the state-of-the-art methods. In Table 12(a), we evaluate MetaML 2.0 as a zero-shot method, and in Table 12(b), we evaluate it as a 3-shot method, using 10-fold cross-validation. In the zero-shot approach, the first recommended pipeline in the classification is selected. In contrast, the 3-shot approach tests all three pipelines on the dataset using 10-fold cross-validation and selects the one with the highest accuracy. A time budget of 3 hours per dataset was considered.

When comparing MetaML 2.0 in zero-shot mode (a) with MetaML 2.0 in 3-shot mode (b), a consistent performance improvement is observed with the use of 3-shot. This difference occurs because, in zero-shot mode, the metamodel sometimes generates structurally invalid pipelines, for example, by simultaneously recommending two preprocessing techniques. In these

Table 12 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML 2.0 zero-shot (a) and 3-shot (b) against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 4.55\text{e-}04$, and the total recommendation time taken for datasets for which all methods ran successfully.

	Mean acc.	Win/tie/loss	Mean rank	p-value	Time (hours)
Auto-WEKA	0.657	41/4/33	7.141	0.437	138
AutoSklearn 1.0	0.717	30/5/43	5.762	0.269	154
AutoSklearn 2.0	0.738	27/7/44	**4.576**	0.024	152
AutoGluon	**0.817**	21/4/53	5.237	0.000	4
H2O AutoML	0.744	40/4/34	7.006	0.865	152
Naive AutoML	0.785	32/2/44	4.762	0.080	24
TPOT	0.656	36/3/39	6.743	0.985	151
FLAML-Zero	0.744	41/4/33	6.512	0.420	**0.001**
TabPFN	0.674	35/4/39	5.846	0.907	18
MetaML Zero-shot	0.786	42/2/34	6.083	0.499	0.013
MetaML 2.0 Zero-shot	0.684	-	6.326	-	1.133

(a)

	Mean acc.	Win/tie/loss	Mean rank	p-value	Time (hours)
Auto-WEKA	0.657	53/6/19	7.512	5.47e-06	138
AutoSklearn 1.0	0.717	43/9/26	6.064	0.003	154
AutoSklearn 2.0	0.738	42/9/27	4.858	0.134	152
AutoGluon	0.817	34/12/32	5.602	0.541	4
H2O AutoML	0.744	53/5/20	7.256	2.302e-05	152
Naive AutoML	0.785	43/5/30	5.038	0.028	24
TPOT	0.656	50/6/22	7.044	6.603e-05	151
FLAML-Zero	0.744	55/5/18	6.955	1.286e-07	**0.001**
TabPFN	0.674	49/7/22	6.224	0.000	18
MetaML 3-shot	0.804	44/8/26	4.878	0.009	0.035
MetaML 2.0 3-shot	**0.825**	-	**4.564**	-	1.222

(b)

situations, a default performance value of zero was assigned, which impacted the results of MetaML zero-shot.

In MetaML 2.0 3-shot, the availability of additional options allowed for each block of the pipeline, resulting in only one valid strategy being recommended, which in turn led to executable pipelines. Furthermore, MetaML 2.0 3-shot explored a larger set of options compared to zero-shot.

Another relevant factor to consider is the increased complexity of the problem: the increase in the number of labels increases the difficulty of the classifier chain, with greater complexity.

Strategies can be used to improve such metamodel results, such as testing different label orders and experimenting with new base classifiers, like Random Forest.

When comparing the results of the original MetaML with MetaML 2.0 in the 3-shot configuration, we observed improvements on specific datasets. For example, on the Poker Hand dataset (data ID: 1567), the original MetaML presented the best pipeline using the Decision Tree classifier, achieving an average accuracy of 0.644. In contrast, in MetaML 2.0, the best pipeline was the combination of ExtraTrees + MLP preprocessing, which achieved an average accuracy of 0.977. Another example is the House 8L dataset (data ID: 843), where the best pipeline recommended by the original MetaML was again a Decision Tree, with a score of 0.843. In contrast, MetaML 2.0 proposed a pipeline consisting of PolynomialFeatures and HistGradientBoostingClassifier preprocessing, producing a better result of 0.893. These cases illustrate the improvement in specific datasets.

Regarding runtime, a significant increase was observed: while the original MetaML required 0.035 hours, MetaML 2.0 required approximately 1.133 hours. This increase is directly associated with the increased complexity of the problem, as the number of labels and the depth of the classifier chain increase, resulting in increased time for metamodel inference.

The results show that incorporating preprocessing blocks into MetaML 2.0 enabled it to recommend pipelines with improved predictive performance, with the MetaML 3-shot version emerging as the best performer.

In a direct comparison with AutoSklearn 2.0 in Table 12(b), MetaML 2.0 3-shot demonstrated superiority with a higher average precision (0.825 vs. 0.738) and a better average rank (lower average rank, 4.564 vs. 4.858), indicating better overall performance across all datasets. In terms of Win/Tie/Loss, MetaML 2.0 won on more datasets (42) than it lost (27) to AutoSklearn 2.0.

Regarding AutoGluon, MetaML 2.0 also performed better, with a higher average accuracy (0.825 vs. 0.817) and a lower average rank (4.564 vs. 5.602). Furthermore, it had a significantly lower computational cost, with an average time of 1.222 hours, while AutoGluon took 4 hours to complete the tasks.

It is worth noting that the zero-shot version of MetaML underperformed the other methods, which reinforces the effectiveness of the 3-shot approach in achieving superior results.

5.4 CONCLUSION

We propose MetaML 2.0 to overcome the limitations observed in MetaML 1.0. This new version is built upon the PIPES meta-dataset and is designed to identify promising pipelines and recommend rankings of the most suitable pipelines for datasets. Experiments demonstrated the effectiveness of the proposed method, and MetaML was evaluated in both zero-shot and 3-shot settings. In terms of base-level performance, the 3-shot approach achieves a better balance between performance and computational cost compared to the zero-shot approach.

Future work can focus on further experiments with adjustments to the parameters n and k to generate new meta-datasets and evaluate their performance. Additionally, incorporating hyperparameter optimization into future versions of the method could further enhance the predictive power and efficiency of MetaML 2.0.

6 CONCLUSION AND FUTURE WORK

This thesis aimed to present a meta-learning approach for pipeline recommendation, capable of handling various types of problems with lower computational cost compared to current AutoML methods. The three studies presented in this thesis, MetaML, PIPES, and MetaML 2.0, address a distinct layer of this objective, contributing complementary advances that, when combined, establish a comprehensive methodological framework. The first study designed a curated search space based on historical data from the online repository (OpenML), aiming to identify high-performance pipelines across a comprehensive set of datasets, ensuring diversity and representativeness. Based on the analysis of the investigated pipelines, it was found that a curated search space allows performance equivalent to that of the state-of-the-art, with significantly reduced computational time. A limitation observed in this study was the difficulty in building a balanced meta-dataset regarding the use of preprocessing techniques, due to the low representation of these techniques in the experiments made available by OpenML. To overcome this limitation, the second study proposed the construction of the PIPES meta-dataset, a collection of experiments involving multiple pipelines, with different combinations of preprocessing techniques and classification blocks. The goal was to create a more representative meta-dataset that could contribute to the diversity of experiments. Based on the PIPES meta-dataset, we present the third study, which uses PIPES to form a more comprehensive search space, resulting in MetaML 2.0. The experiments demonstrated the effectiveness of the proposed method, with greater diversity in the use of preprocessing techniques, enabling the recommendation of more comprehensive pipelines and with lower computational costs than current AutoML methods. Thus, this thesis demonstrates that a well-designed search space with high-performance pipelines on different datasets allows for performance equivalent to the state-of-the-art, while significantly reducing computational cost, thereby demonstrating the efficiency of the approach. Future directions include improving the metamodel's performance by introducing new metafeatures or reordering labels in the chain, testing new ranking algorithms, such as XGBoost Rank, and exploring new strategies for hyperparameter recommendation to evaluate their impact on performance. Future work also aims to investigate the importance and influence of the meta-features used in the metamodel's performance further. A promising possibility is to retrain the meta-classifier, considering only the most relevant meta-features, to evaluate the impact of this selection on the metamodel's recommendations. It is also intended

to assess in greater depth the cases where the recommendation was not the best, seeking to understand the reasons for this.

REFERENCES

ALASADI, S. A.; BHAYA, W. S. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, v. 12, n. 16, p. 4102–4107, 2017.

ALCOBAÇA, E.; SIQUEIRA, F.; RIVOLLI, A.; GARCIA, L. P.; OLIVA, J. T.; CARVALHO, A. C. D. Mfe: Towards reproducible meta-feature extraction. *The Journal of Machine Learning Research*, JMLRORG, v. 21, n. 1, p. 4503–4507, 2020.

ALSHAREF, A.; AGGARWAL, K.; KUMAR, M.; MISHRA, A. et al. Review of ml and automl solutions to forecast time-series data. *Archives of Computational Methods in Engineering*, Springer, p. 1–15, 2022.

AMORIM, L. B. de; CAVALCANTI, G. D.; CRUZ, R. M. The choice of scaling technique matters for classification performance. *Applied Soft Computing*, Elsevier, v. 133, p. 109924, 2023.

AMORIM, L. B. de; CAVALCANTI, G. D.; CRUZ, R. M. The choice of scaling technique matters for classification performance. *Applied Soft Computing*, Elsevier, v. 133, p. 109924, 2023.

AMORIM, L. B. de; CAVALCANTI, G. D.; CRUZ, R. M. Meta-scaler: A meta-learning framework for the selection of scaling techniques. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, v. 36, n. 3, p. 4805–4819, 2024.

AVELINO, J. G.; CAVALCANTI, G. D.; CRUZ, R. M. Resampling strategies for imbalanced regression: a survey and empirical analysis. *Artificial Intelligence Review*, Springer, v. 57, n. 4, p. 82, 2024.

BAHRI, M.; SALUTARI, F.; PUTINA, A.; SOZIO, M. Automl: state of the art with a focus on anomaly detection, challenges, and research directions. *International Journal of Data Science and Analytics*, Springer, p. 1–14, 2022.

BAKER, S.; KORHONEN, A. Initializing neural networks for hierarchical multi-label text classification. In: *BioNLP 2017*. [S.l.: s.n.], 2017. p. 307–315.

BENAVOLI, A.; CORANI, G.; MANGILI, F. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, JMLR. org, v. 17, n. 1, p. 152–161, 2016.

BENSUSAN, H.; GIRAUD-CARRIER, C. Discovering task neighbourhoods through landmark learning performances. In: SPRINGER. *European Conference on Principles of Data Mining and Knowledge Discovery*. [S.l.], 2000. p. 325–330.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *The journal of machine learning research*, JMLR. org, v. 13, n. 1, p. 281–305, 2012.

BISCHL, B.; CASALICCHIO, G.; FEURER, M.; GIJSBERS, P.; HUTTER, F.; LANG, M.; MANTOVANI, R. G.; RIJN, J. N. van; VANSCHOREN, J. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017.

BOJER, C. S.; MELDGAARD, J. P. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, Elsevier, v. 37, n. 2, p. 587–603, 2021.

BRAZDIL, P.; CARRIER, C. G.; SOARES, C.; VILALTA, R. *Metalearning: Applications to data mining*. [S.l.]: Springer Science & Business Media, 2008.

BRAZDIL, P.; GAMA, J.; HENERY, B. Characterizing the applicability of classification algorithms using meta-level learning. In: SPRINGER. *European conference on machine learning*. [S.l.], 1994. p. 83–102.

BRAZDIL, P.; RIJN, J. N. van; SOARES, C.; VANSCHOREN, J. *Metalearning: Applications to Automated Machine Learning and Data Mining*. [S.l.]: Springer Nature, 2022.

BRAZDIL, P.; RIJN, J. N. van; SOARES, C.; VANSCHOREN, J. Setting up configuration spaces and experiments. In: *Metalearning: Applications to Automated Machine Learning and Data Mining*. Cham: Springer Nature, 2022. p. 143–168.

CASTIELLO, C.; CASTELLANO, G.; FANELLI, A. M. Meta-data: Characterization of input features for meta-learning. In: SPRINGER. *International Conference on Modeling Decisions for Artificial Intelligence*. [S.l.], 2005. p. 457–468.

CELIK, B.; VANSCHOREN, J. Adaptation strategies for automated machine learning on evolving data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 43, n. 9, p. 3067–3078, 2021.

CHEKINA, L.; GUTFREUND, D.; KONTOROVICH, A.; ROKACH, L.; SHAPIRA, B. Exploiting label dependencies for improved sample complexity. *Machine learning*, Springer, v. 91, n. 1, p. 1–42, 2013.

CHENG, W.; HÜLLERMEIER, E.; DEMBCZYNSKI, K. J. Bayes optimal multilabel classification via probabilistic classifier chains. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 279–286.

CLARE, A.; KING, R. D. Knowledge discovery in multi-label phenotype data. In: SPRINGER. *European conference on principles of data mining and knowledge discovery*. [S.l.], 2001. p. 42–53.

DANTAS, A. L.; POZO, A. T. R. Selecting algorithms for the quadratic assignment problem with a multi-label meta-learning approach. In: IEEE. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2018. p. 175–180.

DIMITROVSKI, I.; KOCEV, D.; LOSKOVSKA, S.; DŽEROSKI, S. Hierarchical annotation of medical images. *Pattern Recognition*, Elsevier, v. 44, n. 10-11, p. 2436–2449, 2011.

DÔRES, S. C. N. das; SOARES, C.; RUIZ, D. Bandit-based automated machine learning. In: IEEE. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2018. p. 121–126.

ELDEEB, H.; MAHER, M.; MATSUK, O.; ALDALLAL, A.; ELSHAWI, R.; SAK, S. Automlbench: A comprehensive experimental evaluation of automated machine learning frameworks. *arXiv preprint arXiv:2204.08358*, 2022.

ELSHAWI, R.; MAHER, M.; SAKR, S. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

ERICKSON, N.; MUELLER, J.; SHIRKOV, A.; ZHANG, H.; LARROY, P.; LI, M.; SMOLA, A. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

FERRARI, D. G.; CASTRO, L. N. D. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Information Sciences*, Elsevier, v. 301, p. 181–194, 2015.

FEURER, M.; EGGENSPERGER, K.; FALKNER, S.; LINDAUER, M.; HUTTER, F. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074*, 2020.

FEURER, M.; EGGENSPERGER, K.; FALKNER, S.; LINDAUER, M.; HUTTER, F. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, v. 23, n. 261, p. 1–61, 2022.

FEURER, M.; HUTTER, F. Hyperparameter optimization. In: *Automated machine learning: Methods, systems, challenges*. [S.l.]: Springer International Publishing Cham, 2019. p. 3–33.

FEURER, M.; KLEIN, A.; EGGENSPERGER, K.; SPRINGENBERG, J.; BLUM, M.; HUTTER, F. Efficient and robust automated machine learning. *Advances in neural information processing systems*, v. 28, 2015.

FEURER, M.; RIJN, J. N. V.; KADRA, A.; GIJSBERS, P.; MALLIK, N.; RAVI, S.; MÜLLER, A.; VANSCHOREN, J.; HUTTER, F. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, v. 22, n. 100, p. 1–5, 2021.

FEURER, M.; RIJN, J. N. van; KADRA, A.; GIJSBERS, P.; MALLIK, N.; RAVI, S.; MUELLER, A.; VANSCHOREN, J.; HUTTER, F. Openml-python: an extensible python api for openml. *arXiv*, v. 1911.02490, 2020. Disponível em: <https://arxiv.org/pdf/1911.02490.pdf>.

GANDA, D.; BUCH, R. A survey on multi label classification. *Recent Trends in Programming Languages*, v. 5, n. 1, p. 19–23, 2018.

GIJSBERS, P. Systems for automl research. 2022.

GIJSBERS, P.; BUENO, M. L.; COORS, S.; LEDELL, E.; POIRIER, S.; THOMAS, J.; BISCHL, B.; VANSCHOREN, J. Amlb: an automl benchmark. *arXiv preprint arXiv:2207.12560*, 2022.

GIJSBERS, P.; BUENO, M. L.; COORS, S.; LEDELL, E.; POIRIER, S.; THOMAS, J.; BISCHL, B.; VANSCHOREN, J. Amlb: an automl benchmark. *Journal of Machine Learning Research*, v. 25, n. 101, p. 1–65, 2024.

GIJSBERS, P.; LEDELL, E.; THOMAS, J.; POIRIER, S.; BISCHL, B.; VANSCHOREN, J. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

GIJSBERS, P.; VANSCHOREN, J. Gama: A general automated machine learning assistant. In: SPRINGER. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. [S.l.], 2020. p. 560–564.

GOLSHANRAD, P.; RAHMANI, H.; KARIMIAN, B.; KARIMKHANI, F.; WEISS, G. Mega: Predicting the best classifier combination using meta-learning and a genetic algorithm. *Intelligent Data Analysis*, IOS Press, v. 25, n. 6, p. 1547–1563, 2021.

GOMES, T. A.; PRUDÊNCIO, R. B.; SOARES, C.; ROSSI, A. L.; CARVALHO, A. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, Elsevier, v. 75, n. 1, p. 3–13, 2012.

HERRERA, F.; CHARTE, F.; RIVERA, A. J.; JESUS, M. J. del. *Multilabel classification: problem analysis, metrics and techniques*. Cham: Springer International Publishing, 2016.

HO, T. K.; BASU, M. Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 24, n. 3, p. 289–300, 2002.

HOLLMANN, N.; MÜLLER, S.; PURUCKER, L.; KRISHNAKUMAR, A.; KÖRFER, M.; HOO, S. B.; SCHIRRMEISTER, R. T.; HUTTER, F. Accurate predictions on small data with a tabular foundation model. *Nature*, Nature Publishing Group UK London, v. 637, n. 8045, p. 319–326, 2025.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. *Automated machine learning: methods, systems, challenges*. [S.l.]: Springer Nature, 2019.

KARMAKER, S. K.; HASSAN, M. M.; SMITH, M. J.; XU, L.; ZHAI, C.; VEERAMACHANENI, K. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, ACM New York, NY, v. 54, n. 8, p. 1–36, 2021.

KAYALI, M.; WANG, C. Mining robust default configurations for resource-constrained automl. *arXiv preprint arXiv:2202.09927*, 2022.

KHAN, I.; ZHANG, X.; AYYASAMY, R. K.; ALI, R. Autofe-sel: A meta-learning based methodology for recommending feature subset selection algorithms. 2023.

KHAN, I.; ZHANG, X.; REHMAN, M.; ALI, R. A literature survey and empirical study of meta-learning for classifier selection. *IEEE Access*, IEEE, v. 8, p. 10262–10281, 2020.

KOMER, B.; BERGSTRA, J.; ELIASMITH, C. Hyperopt-sklearn. In: *Automated Machine Learning*. [S.l.]: Springer, Cham, 2019. p. 97–111.

KOTLAR, M.; PUNT, M.; RADIVOJEVIĆ, Z.; CVETANOVIĆ, M.; MILUTINOVIĆ, V. Novel meta-features for automated machine learning model selection in anomaly detection. *IEEE Access*, IEEE, v. 9, p. 89675–89687, 2021.

KOTSIANTIS, S. B.; ZAHARAKIS, I. D.; PINTELAS, P. E. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, Springer, v. 26, p. 159–190, 2006.

KOTTHOFF, L.; THORNTON, C.; HOOS, H. H.; HUTTER, F.; LEYTON-BROWN, K. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, v. 18, n. 25, p. 1–5, 2017. Disponível em: <http://jmlr.org/papers/v18/16-261.html>.

KOTTHOFF, L.; THORNTON, C.; HOOS, H. H.; HUTTER, F.; LEYTON-BROWN, K. Auto-weka: Automatic model selection and hyperparameter optimization in weka. In: *Automated machine learning*. [S.l.]: Springer, Cham, 2019. p. 81–95.

KÜHN, D.; PROBST, P.; THOMAS, J.; BISCHL, B. Automatic exploration of machine learning experiments on openml. *arXiv preprint arXiv:1806.10961*, 2018.

LEDELL, E.; POIRIER, S. H2o automl: Scalable automatic machine learning. In: *Proceedings of the AutoML Workshop at ICML*. [S.l.: s.n.], 2020. v. 2020.

LORENA, A. C.; GARCIA, L. P.; LEHMANN, J.; SOUTO, M. C.; HO, T. K. How complex is your classification problem? a survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 5, p. 1–34, 2019.

LUACES, O.; DÍEZ, J.; BARRANQUERO, J.; COZ, J. J. del; BAHAMONDE, A. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, Springer, v. 1, n. 4, p. 303–313, 2012.

MANTOVANI, R. G.; ROSSI, A. L.; ALCOBAÇA, E.; VANSCHOREN, J.; CARVALHO, A. C. de. A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves svm classifiers. *Information Sciences*, Elsevier, v. 501, p. 193–221, 2019.

MCINNES, L.; HEALY, J.; MELVILLE, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). *Proceedings of the 9th Python in Science Conference*. [S.l.: s.n.], 2010. p. 56 – 61.

MIRANDA, P. B.; PRUDÊNCIO, R. B.; CARVALHO, A. P. D.; SOARES, C. A hybrid meta-learning architecture for multi-objective optimization of svm parameters. *Neurocomputing*, Elsevier, v. 143, p. 27–43, 2014.

MISIR, M.; SEBAG, M. Alors: An algorithm recommender system. *Artificial Intelligence*, Elsevier, v. 244, p. 291–314, 2017.

MOHR, F.; WEVER, M. Naive automated machine learning. *Machine Learning*, Springer, v. 112, n. 4, p. 1131–1170, 2023.

MOHR, F.; WEVER, M.; HÜLLERMEIER, E. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, Springer, v. 107, n. 8, p. 1495–1515, 2018.

MOYANO, J. M.; GIBAJA, E. L.; CIOS, K. J.; VENTURA, S. An evolutionary approach to build ensembles of multi-label classifiers. *Information Fusion*, Elsevier, v. 50, p. 168–180, 2019.

NGUYEN, D. A.; KONONOVA, A. V.; MENZEL, S.; SENDHOFF, B.; BÄCK, T. An efficient contesting procedure for automl optimization. *IEEE Access*, IEEE, v. 10, p. 75754–75771, 2022.

OBAID, H. S.; DHEYAB, S. A.; SABRY, S. S. The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning. In: IEEE. *2019 9th annual information technology, electromechanical engineering and microelectronics conference (iemecon)*. [S.l.], 2019. p. 279–283.

OLSON, R. S.; BARTLEY, N.; URBANOWICZ, R. J.; MOORE, J. H. Evaluation of a tree-based pipeline optimization tool for automating data science. In: *Proceedings of the genetic and evolutionary computation conference 2016*. [S.l.: s.n.], 2016. p. 485–492.

OLSON, R. S.; MOORE, J. H. Tpot: A tree-based pipeline optimization tool for automating machine learning. In: PMLR. *Workshop on automatic machine learning*. [S.l.], 2016. p. 66–74.

PARMEZAN, A. R. S.; LEE, H. D.; WU, F. C. Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework. *Expert Systems with Applications*, Elsevier, v. 75, p. 1–24, 2017.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, JMLR. org, v. 12, p. 2825–2830, 2011.

PENG, Y.; FLACH, P. A.; SOARES, C.; BRAZDIL, P. Improved dataset characterisation for meta-learning. In: SPRINGER. *International conference on discovery science*. [S.l.], 2002. p. 141–152.

PERRONE, V.; JENATTON, R.; SEEGER, M. W.; ARCHAMBEAU, C. Scalable hyperparameter transfer learning. *Advances in neural information processing systems*, v. 31, 2018.

PIMENTEL, B. A.; CARVALHO, A. C. D. A new data characterization for selecting clustering algorithms using meta-learning. *Information Sciences*, Elsevier, v. 477, p. 203–219, 2019.

PIO, P. B.; RIVOLLI, A.; CARVALHO, A. C. d.; GARCIA, L. P. A review on preprocessing algorithm selection with meta-learning. *Knowledge and Information Systems*, Springer, v. 66, n. 1, p. 1–28, 2024.

PROBST, P.; BOULESTEIX, A.-L.; BISCHL, B. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, v. 20, n. 53, p. 1–32, 2019.

RAJU, V. G.; LAKSHMI, K. P.; JAIN, V. M.; KALIDINDI, A.; PADMA, V. Study the influence of normalization/transformation process on the accuracy of supervised classification. In: IEEE. *2020 third international conference on smart systems and inventive technology (icssit)*. [S.l.], 2020. p. 729–735.

READ, J.; PFAHRINGER, B.; HOLMES, G.; FRANK, E. Classifier chains for multi-label classification. In: SPRINGER. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part II 20*. [S.l.], 2009. p. 254–269.

RICE, J. R. The algorithm selection problem. In: . [S.l.]: Elsevier, 1976. v. 15, p. 65–118.

RIJN, J. N. V.; BISCHL, B.; TORGO, L.; GAO, B.; UMAASHANKAR, V.; FISCHER, S.; WINTER, P.; WISWEDEL, B.; BERTHOLD, M. R.; VANSCHOREN, J. Openml: A collaborative science platform. In: SPRINGER. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*. [S.l.], 2013. p. 645–649.

RIJN, J. N. V.; HUTTER, F. Hyperparameter importance across datasets. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. [S.l.: s.n.], 2018. p. 2367–2376.

RIVOLLI, A.; GARCIA, L. P.; SOARES, C.; VANSCHOREN, J.; CARVALHO, A. C. de. Meta-features for meta-learning. *Knowledge-Based Systems*, Elsevier, v. 240, p. 108101, 2022.

SANTOS, A. d. M. Investigando a combinação de técnicas de aprendizado semissupervisionado e classificação hierárquica multirrótulo. Universidade Federal do Rio Grande do Norte, 2012.

SMITH-MILES, K. A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 41, n. 1, p. 1–25, 2009.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, v. 25, 2012.

SONG, Q.; WANG, G.; WANG, C. Automatic recommendation of classification algorithms based on data set characteristics. *Pattern recognition*, Elsevier, v. 45, n. 7, p. 2672–2689, 2012.

SOROWER, M. S. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, v. 18, n. 1, p. 25, 2010.

SOUTO, M. C. D.; PRUDENCIO, R. B.; SOARES, R. G.; ARAUJO, D. S. D.; COSTA, I. G.; LUDERMIR, T. B.; SCHLIEP, A. Ranking and selecting clustering algorithms using a meta-learning approach. In: IEEE. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. [S.l.], 2008. p. 3729–3735.

SZYMAŃSKI, P.; KAJDANOWICZ, T. A scikit-based python environment for performing multi-label classification. *arXiv preprint arXiv:1702.01460*, 2017.

TANG, H.; WANG, Y.; TANG, S.; CHU, D.; LI, C. A randomized clustering forest approach for efficient prediction of protein functions. *IEEE Access*, IEEE, v. 7, p. 12360–12372, 2019.

THORNTON, C.; HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2013. p. 847–855.

TIDAKE, V. S.; SANE, S. S. Multi-label classification: a survey. *International Journal of Engineering and Technology*, v. 7, n. 4.19, p. 1045–1054, 2018.

TING, K. M.; WITTEN, I. H. Issues in stacked generalization. *Journal of artificial intelligence research*, v. 10, p. 271–289, 1999.

TROHIDIS, K.; TSOUMAKAS, G.; KALLIRIS, G.; VLAHAVAS, I. P. et al. Multi-label classification of music into emotions. In: *ISMIR*. [S.l.: s.n.], 2008. v. 8, p. 325–330.

TSOUMAKAS, G.; KATAKIS, I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, IGI Global, v. 3, n. 3, p. 1–13, 2007.

TSOUMAKAS, G.; KATAKIS, I.; VLAHAVAS, I. Mining multi-label data. *Data mining and knowledge discovery handbook*, Springer, p. 667–685, 2009.

TSOUMAKAS, G.; KATAKIS, I.; VLAHAVAS, I. Mining multi-label data. In: _____. *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2010. p. 667–685. Disponível em: <https://doi.org/10.1007/978-0-387-09823-4_34>.

VANSCHOREN, J.; BLOCKEEL, H.; PFAHRINGER, B.; HOLMES, G. Experiment databases: A new way to share, organize and learn from experiments. *Machine Learning*, Springer, v. 87, n. 2, p. 127–158, 2012.

VANSCHOREN, J.; RIJN, J. N. V.; BISCHL, B.; TORGO, L. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, ACM New York, NY, USA, v. 15, n. 2, p. 49–60, 2014.

WANG, G.; SONG, Q.; ZHANG, X.; ZHANG, K. A generic multilabel learning-based classification algorithm recommendation method. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM New York, NY, USA, v. 9, n. 1, p. 1–30, 2014.

WEGMETH, L.; BEEL, J. Camels: Cooperative meta-learning service for recommender systems. 2022.

WISTUBA, M.; SCHILLING, N.; SCHMIDT-THIEME, L. Sequential model-free hyperparameter tuning. In: IEEE. *2015 IEEE international conference on data mining*. [S.l.], 2015. p. 1033–1038.

YANG, C.; AKIMOTO, Y.; KIM, D. W.; UDELL, M. Oboe: Collaborative filtering for automl model selection. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. [S.l.: s.n.], 2019. p. 1173–1183.

YAO, Q.; WANG, M.; CHEN, Y.; DAI, W.; LI, Y.-F.; TU, W.-W.; YANG, Q.; YU, Y. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

YING, C.; KLEIN, A.; CHRISTIANSEN, E.; REAL, E.; MURPHY, K.; HUTTER, F. Nas-bench-101: Towards reproducible neural architecture search. In: PMLR. *International conference on machine learning*. [S.l.], 2019. p. 7105–7114.

ZEBARI, R.; ABDULAZEEZ, A.; ZEEBAREE, D.; ZEBARI, D.; SAEED, J. A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *Journal of Applied Science and Technology Trends*, v. 1, n. 1, p. 56–70, 2020.

ZHANG, M.-L.; ZHOU, Z.-H. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, Elsevier, v. 40, n. 7, p. 2038–2048, 2007.

ZHANG, S. Nearest neighbor selection for iteratively knn imputation. *Journal of Systems and Software*, Elsevier, v. 85, n. 11, p. 2541–2552, 2012.

ZHANG, X.; SONG, Q. A multi-label learning based kernel automatic recommendation method for support vector machine. *PloS one*, Public Library of Science San Francisco, CA USA, v. 10, n. 4, p. e0120455, 2015.

ZHU, X.; YANG, X.; YING, C.; WANG, G. A new classification algorithm recommendation method based on link prediction. *Knowledge-Based Systems*, Elsevier, v. 159, p. 171–185, 2018.

ZHU, X.; YING, C.; WANG, J.; LI, J.; LAI, X.; WANG, G. Ensemble of ml-knn for classification algorithm recommendation. *Knowledge-Based Systems*, Elsevier, v. 221, p. 106933, 2021.

ZÖLLER, M.-A.; HUBER, M. F. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, v. 70, p. 409–472, 2021.

APPENDIX A – PIPELINE REDUNDANCY ANALYSIS

To illustrate the redundancy of certain pipelines, we selected some OpenML flows, as presented in Table 13. The comparison was performed on 323 datasets, to verify whether the pipelines are really redundant in a large proportion of datasets. The metric used for the comparison was the accuracy. We observed that, for Decision Trees and algorithms based on decision trees, such as Random Forest, and AdaBoost, the employment of scaling techniques does not significantly impact performance. This highlights the redundancy of certain pipelines, since these transformations are unnecessary for these models.

For example, when analyzing the execution frequency of the OHE-VT-DT pipeline, we found 206 executions, while the OHE-VT-SS-DT pipeline was executed 595 times. Since SS does not influence the performance of Decision Trees, this difference highlights the redundancy in using pipelines that include this unnecessary transformation. Furthermore, our analysis suggests that eliminating redundant pipelines could significantly reduce the search space. This result supports the existence of (i) redundancy, where different pipelines lead to similar results, and (ii) naïve or unnecessary pipelines, as specific preprocessing steps (such as scaling for tree-based algorithms) do not contribute to performance improvement. Finally, we also identified (iii) rare and niche-specific pipelines. One example is a pipeline that first applies data imputation to handle missing values, then removes outliers using Isolation Forest, and finally classifies using Quadratic Discriminant Analysis (QDA), Flow ID 9678. When analyzed in OpenML, this pipeline was executed only 10 times for a single dataset (Autos, Data ID 9), indicating that it is rarely used in practice.

Table 13 – Comparison pipelines of OpenML. RSC - RobustScaler, OHE - One-Hot Encoder, VT - Variance Threshold, SS - StandardScale, DT - Decision Tree.

RandomForest (RF)			
Flow ID	**Pipeline**	**Mean acc**	**Mean rank**
4830	RF	0.839	1.517
17801	SS-RF	0.839	1.482
Decision Tree (DT)			
Flow ID	**Pipeline**	**Mean acc**	**Mean rank**
6946	OHE-VT-DT	0.801	1.473
7725	OHE-VT-SS-DT	0.800	1.526
Adaboost			
Flow ID	**Pipeline**	**Mean acc**	**Mean rank**
18144	OHE-RSC-Adaboost	0.796	1.504
18146	OHE-Adaboost	0.796	1.495

APPENDIX B – DATASETS USED IN META-DATASET CONSTRUCTION

Tables 14 shows the 290 datasets used to construct the meta-dataset.

Table 14 – Datasets used to construct the meta-dataset. DID - Data ID, TID - Task ID.

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
965	3828	zoo	2	17	101	Real
181	2073	yeast	10	9	1484	Real
753	3619	wisconsin	2	33	194	Real
187	2382	wine	3	14	178	Real
847	3712	wind	2	15	6574	Real
1511	9988	wholesale-customers	2	9	440	Real
979	3842	waveform-5000	2	41	5000	Artificial
1526	9942	wall-robot-navigation	4	5	5456	Real
1509	9945	walking-activity	22	5	149332	Real
1016	3879	vowel	2	14	990	Real
1546	9919	volcanoes-e5	5	4	1112	Artificial
1545	9918	volcanoes-e4	5	4	1252	Artificial
1544	9917	volcanoes-e3	5	4	1277	Artificial
1543	9916	volcanoes-e2	5	4	1080	Artificial
1542	9915	volcanoes-e1	5	4	1183	Artificial
1541	9923	volcanoes-d4	5	4	8654	Artificial
1540	9922	volcanoes-d3	5	4	9285	Artificial
1539	9921	volcanoes-d2	5	4	9172	Artificial
1538	9920	volcanoes-d1	5	4	8753	Artificial
1537	9933	volcanoes-c1	5	4	28626	Artificial
1536	9932	volcanoes-b6	5	4	10130	Artificial
1535	9931	volcanoes-b5	5	4	9989	Artificial
1534	9930	volcanoes-b4	5	4	10190	Artificial

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
1533	9929	volcanoes-b3	5	4	10386	Artificial
1532	9928	volcanoes-b2	5	4	10668	Artificial
1531	9927	volcanoes-b1	5	4	10176	Artificial
1530	9926	volcanoes-a4	5	4	1515	Artificial
1529	9925	volcanoes-a3	5	4	1521	Artificial
1528	9924	volcanoes-a2	5	4	1623	Artificial
1527	10103	volcanoes-a1	5	4	3252	Artificial
923	3786	visualizing_soil	2	5	8641	Real
736	3602	visualizing_environmental	2	4	111	Real
719	3585	veteran	2	8	137	Real
994	3857	vehicle	2	19	846	Real
1508	9944	user-knowledge	5	6	403	Real
1507	9943	twonorm	2	21	7400	Artificial
788	3653	triazines	2	61	186	Real
885	3748	transplant	2	4	131	Real
1506	9969	thoracic-surgery	2	17	470	Real
1115	3949	teachingAssistant	3	7	151	Real
955	3818	tae	2	6	151	Real
1004	3867	synthetic_control	2	61	600	Artificial
770	3636	strikes	2	7	625	Real
841	3706	stock	2	10	950	Real
1503	9966	spoken-arabic-digit	10	15	263256	Real
953	3816	splice	2	61	3190	Real
737	3603	space_ga	2	7	3107	Artificial
902	3765	sleuth_case2002	2	7	147	Real
1502	9965	skin-segmentation	2	4	245057	Real
826	3691	sensory	2	12	576	Real

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
958	3821	segment	2	20	2310	Real
1498	9961	sa-heart	2	10	462	Real
1520	9938	robot-failures-lp5	5	91	164	Real
717	3583	rmftsa_ladata	2	11	508	Artificial
1496	9959	ringnorm	2	21	7400	Artificial
816	3681	puma8NH	2	9	8192	Artificial
752	3618	puma32H	2	33	8192	Artificial
996	3859	prnn_fglass	2	10	214	Artificial
722	3588	pol	2	49	15000	Artificial
155	223	pokerhand	10	11	829201	Artificial
1567	9890	poker-hand	10	11	1025009	Real
354	3506	poker	2	11	1025010	Real
750	3616	pm10	2	8	500	Real
1490	9953	planning-relax	2	13	182	Real
1019	3882	pendigits	2	17	10992	Real
1069	3919	pc2	2	37	5589	Real
1167	4001	pc1_req	2	9	320	Real
1488	9951	parkinsons	2	23	195	Real
1021	3884	page-blocks	2	11	5473	Real
980	3843	optdigits	2	65	5620	Real
959	3822	nursery	2	9	12960	Real
886	3749	no2	2	8	500	Real
784	3649	newton_hema	2	4	140	Real
881	3745	mv	2	11	40768	Artificial
880	3744	mu284	2	11	284	Artificial
164	2373	molecular-biology_promoters	2	58	106	Real
995	3858	mfeat-zernike	2	48	2000	Real

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
962	3825	mfeat-morphological	2	7	2000	Real
1020	3883	mfeat-karhunen	2	65	2000	Real
971	3834	mfeat-fourier	2	77	2000	Real
1056	3907	mc1	2	39	9466	Real
733	3599	machine_cpu	2	7	209	Real
10	10	lymph	4	19	148	Real
941	3804	lowbwt	2	10	189	Real
977	3840	letter	2	17	20000	Real
1483	9974	ldpa	11	8	164860	Real
184	2076	kropt	18	7	28056	Real
1481	9972	kr-vs-k	18	7	28056	Real
807	3672	kin8nm	2	9	8192	Artificial
1045	3898	kc1-top5	2	95	145	Real
1066	3916	kc1-binary	2	95	145	Real
1048	3901	jEdit_4.2_4.3	2	9	369	Real
1073	3921	jEdit_4.0_4.2	2	9	274	Real
969	3832	iris	2	5	150	Real
823	3688	houses	2	9	20640	Real
843	3708	house_8L	2	9	22784	Real
821	3686	house_16H	2	17	22784	Real
1513	9948	heart-switzerland	5	13	123	Real
53	52	heart-statlog	2	14	270	Real
1565	9894	heart-h	5	14	294	Real
1026	3888	grub-damage	2	9	155	Real
1005	3868	glass	2	10	214	Real
714	3580	fruitfly	2	5	125	Real
901	3764	fried	2	11	40768	Artificial

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
805	3670	fri_c4_500_50	2	51	500	Artificial
838	3703	fri_c4_500_25	2	26	500	Artificial
855	3720	fri_c4_500_10	2	11	500	Artificial
918	3781	fri_c4_250_50	2	51	250	Artificial
933	3796	fri_c4_250_25	2	26	250	Artificial
863	3727	fri_c4_250_10	2	11	250	Artificial
932	3795	fri_c4_100_50	2	51	100	Artificial
868	3732	fri_c4_100_25	2	26	100	Artificial
878	3742	fri_c4_100_10	2	11	100	Artificial
797	3662	fri_c4_1000_50	2	51	1000	Artificial
723	3589	fri_c4_1000_25	2	26	1000	Artificial
751	3617	fri_c4_1000_10	2	11	1000	Artificial
937	3800	fri_c3_500_50	2	51	500	Artificial
749	3615	fri_c3_500_5	2	6	500	Artificial
896	3759	fri_c3_500_25	2	26	500	Artificial
936	3799	fri_c3_500_10	2	11	500	Artificial
873	3737	fri_c3_250_50	2	51	250	Artificial
744	3610	fri_c3_250_5	2	6	250	Artificial
832	3697	fri_c3_250_25	2	26	250	Artificial
793	3658	fri_c3_250_10	2	11	250	Artificial
716	3582	fri_c3_100_50	2	51	100	Artificial
916	3779	fri_c3_100_5	2	6	100	Artificial
768	3634	fri_c3_100_25	2	26	100	Artificial
783	3648	fri_c3_100_10	2	11	100	Artificial
806	3671	fri_c3_1000_50	2	51	1000	Artificial
813	3678	fri_c3_1000_5	2	6	1000	Artificial
715	3581	fri_c3_1000_25	2	26	1000	Artificial

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
740	3606	fri_c3_1000_10	2	11	1000	Artificial
920	3783	fri_c2_500_50	2	51	500	Artificial
792	3657	fri_c2_500_5	2	6	500	Artificial
879	3743	fri_c2_500_25	2	26	500	Artificial
869	3733	fri_c2_500_10	2	11	500	Artificial
877	3741	fri_c2_250_50	2	51	250	Artificial
911	3774	fri_c2_250_5	2	6	250	Artificial
794	3659	fri_c2_250_25	2	26	250	Artificial
830	3695	fri_c2_250_10	2	11	250	Artificial
922	3785	fri_c2_100_50	2	51	100	Artificial
726	3592	fri_c2_100_5	2	6	100	Artificial
775	3641	fri_c2_100_25	2	26	100	Artificial
762	3628	fri_c2_100_10	2	11	100	Artificial
866	3730	fri_c2_1000_50	2	51	1000	Artificial
912	3775	fri_c2_1000_5	2	6	1000	Artificial
903	3766	fri_c2_1000_25	2	26	1000	Artificial
913	3776	fri_c2_1000_10	2	11	1000	Artificial
766	3632	fri_c1_500_50	2	51	500	Artificial
870	3734	fri_c1_500_5	2	6	500	Artificial
779	3645	fri_c1_500_25	2	26	500	Artificial
824	3689	fri_c1_500_10	2	11	500	Artificial
769	3635	fri_c1_250_50	2	51	250	Artificial
730	3596	fri_c1_250_5	2	6	250	Artificial
746	3612	fri_c1_250_25	2	26	250	Artificial
935	3798	fri_c1_250_10	2	11	250	Artificial
876	3740	fri_c1_100_50	2	51	100	Artificial
829	3694	fri_c1_100_5	2	6	100	Artificial

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
812	3677	fri_c1_100_25	2	26	100	Artificial
789	3654	fri_c1_100_10	2	11	100	Artificial
837	3702	fri_c1_1000_50	2	51	1000	Artificial
743	3609	fri_c1_1000_5	2	6	1000	Artificial
917	3780	fri_c1_1000_25	2	26	1000	Artificial
910	3773	fri_c1_1000_10	2	11	1000	Artificial
888	3751	fri_c0_500_50	2	51	500	Artificial
884	3747	fri_c0_500_5	2	6	500	Artificial
926	3789	fri_c0_500_25	2	26	500	Artificial
943	3806	fri_c0_500_10	2	11	500	Artificial
732	3598	fri_c0_250_50	2	51	250	Artificial
776	3642	fri_c0_250_5	2	6	250	Artificial
773	3639	fri_c0_250_25	2	26	250	Artificial
763	3629	fri_c0_250_10	2	11	250	Artificial
850	3715	fri_c0_100_50	2	51	100	Artificial
754	3620	fri_c0_100_5	2	6	100	Artificial
889	3752	fri_c0_100_25	2	26	100	Artificial
808	3673	fri_c0_100_10	2	11	100	Artificial
904	3767	fri_c0_1000_50	2	51	1000	Artificial
799	3664	fri_c0_1000_5	2	6	1000	Artificial
849	3714	fri_c0_1000_25	2	26	1000	Artificial
845	3710	fri_c0_1000_10	2	11	1000	Artificial
1012	3875	flags	2	29	194	Real
1473	9984	fertility	2	10	100	Real
1044	3897	eye_movements	3	28	10936	Real
1011	3874	ecoli	2	8	336	Real
931	3794	disclosure_z	2	4	662	Artificial

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
795	3660	disclosure_x_tampered	2	4	662	Artificial
827	3692	disclosure_x_noise	2	4	662	Artificial
774	3640	disclosure_x_bias	2	4	662	Artificial
818	3683	diggle_table_a2	2	9	310	Artificial
819	3684	delta_elevators	2	7	9517	Artificial
803	3668	delta_ailerons	2	6	7129	Artificial
1075	3923	datatrieve	2	9	130	Real
735	3601	cpu_small	2	13	8192	Real
761	3627	cpu_act	2	22	8192	Real
796	3661	cpu	2	8	209	Real
150	218	covertype	7	55	581012	Real
987	3850	collins	2	23	500	Real
351	3505	codrna	2	9	488565	Real
983	3846	cmc	2	10	1473	Real
890	3753	cloud	2	8	108	Real
900	3763	chscase_census6	2	7	400	Artificial
906	3769	chscase_census5	2	8	400	Artificial
907	3770	chscase_census4	2	8	400	Artificial
908	3771	chscase_census3	2	8	400	Artificial
909	3772	chscase_census2	2	8	400	Artificial
1560	9898	cardiotocography	3	36	2126	Real
991	3854	car	2	7	1728	Real
23499	52945	breast-cancer-dropped	2	10	277	Real
825	3690	boston_corrected	2	21	506	Real
853	3718	boston	2	14	506	Real
778	3644	bodyfat	2	15	252	Real
1463	10094	blogger	2	6	100	Real

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
725	3591	bank8FM	2	9	8192	Real
833	3698	bank32nh	2	33	8192	Real
997	3860	balance-scale	2	5	625	Real
1121	3955	badges2	2	11	294	Artificial
463	3554	backache	2	32	180	Real
745	3611	auto_price	2	16	159	Real
1553	9905	autoUniv-au7-700	3	13	700	Artificial
1554	9903	autoUniv-au7-500	5	13	500	Artificial
1552	9906	autoUniv-au7-1100	5	13	1100	Artificial
1549	9904	autoUniv-au6-750	8	41	750	Artificial
1551	9907	autoUniv-au6-400	8	41	400	Artificial
1555	9902	autoUniv-au6-1000	8	41	1000	Artificial
1547	9909	autoUniv-au1-1000	2	21	1000	Artificial
756	3622	autoPrice	2	16	159	Real
951	3814	arsenic-male-lung	2	5	559	Real
947	3810	arsenic-male-bladder	2	5	559	Real
950	3813	arsenic-female-lung	2	5	559	Real
949	3812	arsenic-female-bladder	2	5	559	Real
1061	3911	ar4	2	30	107	Real
1059	3909	ar1	2	30	121	Real
748	3614	analcatdata_wildcat	2	6	163	Real
724	3590	analcatdata_vineyard	2	4	468	Real
728	3594	analcatdata_supreme	2	8	4052	Real
921	3784	analcatdata_seropositive	2	4	132	Real
771	3637	analcatdata_michiganacc	2	4	108	Real
450	3542	analcatdata_lawsuit	2	5	264	Real
1025	3887	analcatdata_germangss	2	6	400	Real

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
1014	3877	analcatdata_dmft	2	5	797	Real
461	3552	analcatdata_creditscore	2	7	100	Real
875	3739	analcatdata_chlamydia	2	4	100	Real
444	3538	analcatdata_boxing2	2	4	132	Artificial
448	3540	analcatdata_boxing1	2	4	120	Artificial
970	3833	analcatdata_authorship	2	71	841	Real
764	3630	analcatdata_apnea3	2	4	450	Artificial
765	3631	analcatdata_apnea2	2	4	475	Artificial
767	3633	analcatdata_apnea1	2	4	475	Artificial
734	3600	ailerons	2	41	13750	Real
1556	9901	acute-inflammations	2	7	120	Real
720	3586	abalone	2	9	4177	Real
337	3496	SPECTF	2	45	349	Real
336	3495	SPECT	2	23	267	Real
160	228	RandomRBF_50_1E-4	5	11	1000000	Artificial
159	227	RandomRBF_50_1E-3	5	11	1000000	Artificial
158	226	RandomRBF_10_1E-4	5	11	1000000	Artificial
157	225	RandomRBF_10_1E-3	5	11	1000000	Artificial
156	224	RandomRBF_0_0	5	11	1000000	Artificial
1100	3937	PopularKids	3	11	478	Real
1444	7559	PizzaCutter3	2	38	1043	Artificial
1443	7558	PizzaCutter1	2	38	661	Artificial
1453	10099	PieChart3	2	38	1077	Artificial
1452	10098	PieChart2	2	37	745	Artificial
1451	10097	PieChart1	2	38	705	Artificial
1442	7556	MegaWatt1	2	38	253	Real
1113	3947	KDDCup99	23	42	494020	Real

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances	Real or Artificial
976	3839	JapaneseVowels	2	15	9961	Real
1446	7557	CostaMadre1	2	38	296	Artificial
1447	10096	CastMetal1	2	38	327	Artificial
272	2267	BNG(zoo)	7	17	1000000	Artificial
271	2266	BNG(waveform-5000)	3	41	1000000	Artificial
141	210	BNG(vehicle,nominal,1000000)	4	19	1000000	Artificial
268	2263	BNG(vehicle)	4	19	1000000	Artificial
264	2259	BNG(sonar)	2	61	1000000	Artificial
130	156	BNG(segment)	7	20	1000000	Artificial
261	2256	BNG(pendigits)	10	17	1000000	Artificial
125	151	BNG(page-blocks,nominal,295245)	5	11	295245	Artificial
259	2254	BNG(page-blocks)	5	11	295245	Artificial
123	149	BNG(optdigits)	10	65	1000000	Artificial
254	2145	BNG(mfeat-zernike)	10	48	1000000	Artificial
252	2143	BNG(mfeat-karhunen)	10	65	1000000	Artificial
250	2141	BNG(mfeat-fourier)	10	77	1000000	Artificial
249	2140	BNG(lymph)	4	19	1000000	Artificial
74	136	BNG(letter,nominal,1000000)	26	17	1000000	Artificial
72	134	BNG(kr-vs-kp)	2	37	1000000	Artificial
146	215	BNG(ionosphere)	2	35	1000000	Artificial
267	2262	BNG(heart-statlog)	2	14	1000000	Artificial
265	2260	BNG(glass)	7	10	137781	Artificial
119	145	BNG(cmc,nominal,55296)	3	10	55296	Artificial
727	3593	2dplanes	2	11	40768	Artificial

APPENDIX C – METAML DATASETS DIVERSITY

In Figure 26, we illustrate the use of Uniform Manifold approximation and Projection for Dimension Reduction (UMAP) (MCINNES; HEALY; MELVILLE, 2018) on the 290 datasets used in building MetaML, configured with k = 3 neighbors, to visualize the datasets across different types of meta-features in a two-dimensional space. The distribution indicates diversity, reinforcing the representativeness of the selected datasets.

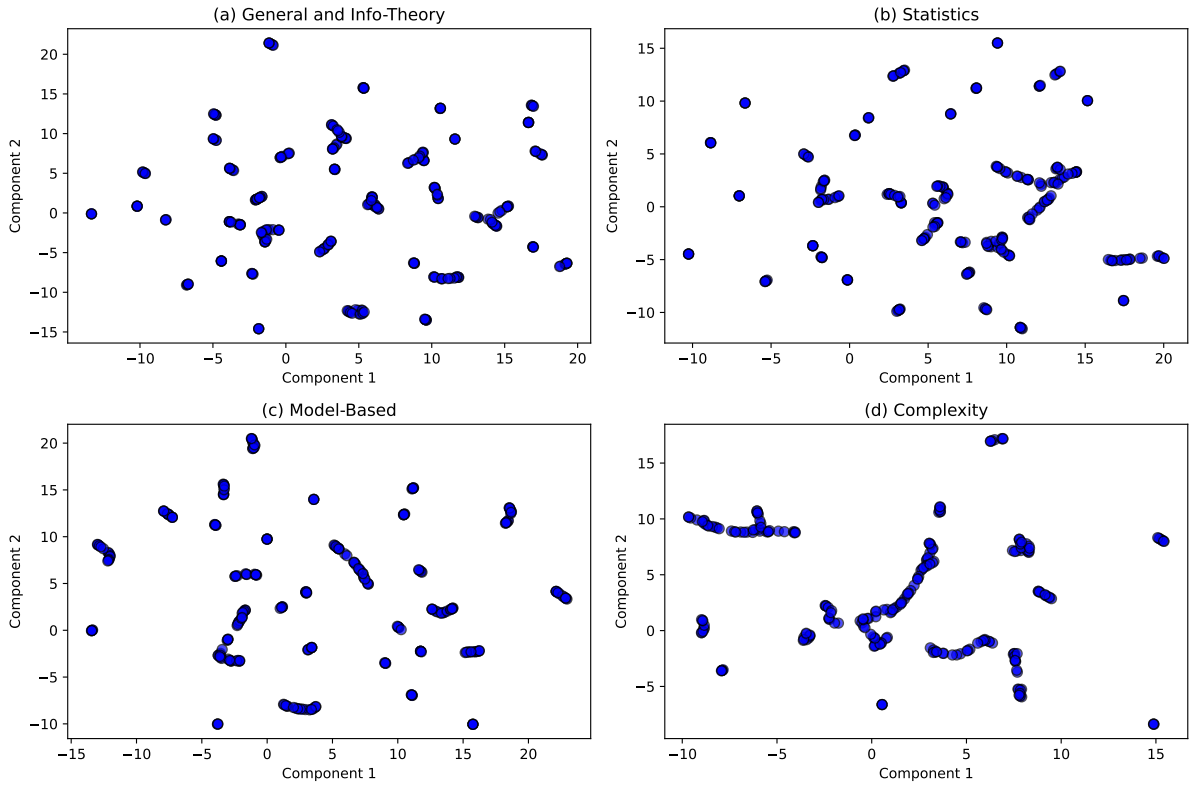


Figure 26 – Representation of the meta-features space with UMAP.

Complementing the analyses, the Figure 27 illustrates some meta-features of the complexity measure only. The complexity measure describes the difficulty of the problems. We observed a heterogeneous distribution of these measures covering different scenarios.

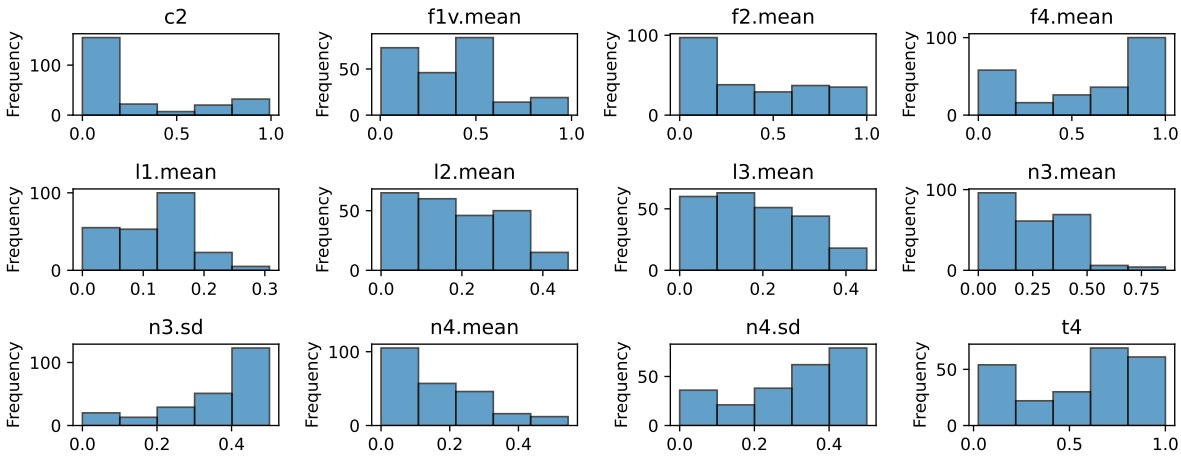


Figure 27 – Distribution of 12 complexity measures used in the construction of the meta-dataset.

APPENDIX D – META-FEATURES

Table 15 shows the eighty-six meta-features used to represent each dataset.

Table 15 – Datasets used to construct the meta-dataset.

Meta-feature	Description	Group
Dimensionality	Number of attributes divided by the number of instances.	simple
NumberOfBinaryFeatures	Number of binary attributes.	simple
NumberOfClasses	Number of distinct values of the target attribute (if it is nominal).	simple
NumberOfFeatures	Number of attributes (columns) of the dataset.	simple
NumberOfInstances	Number of instances (rows) of the dataset.	simple
NumberOfInstancesWithMissingValues	Number of instances with at least one value missing.	simple
NumberOfMissingValues	Number of missing values in the dataset.	simple
NumberOfNumericFeatures	Number of numeric attributes.	simple
NumberOfSymbolicFeatures	Number of nominal attributes.	simple
PercentageOfBinaryFeatures	Percentage of binary attributes.	simple
PercentageOfInstancesWithMissingValues	Percentage of instances having missing values.	simple
PercentageOfMissingValues	Percentage of missing values.	simple
PercentageOfNumericFeatures	Percentage of numeric attributes.	simple
PercentageOfSymbolicFeatures	Percentage of nominal attributes.	simple
ClassEntropy	Entropy of the target attribute values.	info-theory
AutoCorrelation	Average class difference between consecutive instances.	statistical
Quartile1KurtosisOfNumericAtts	First quartile of kurtosis among attributes of the numeric type.	statistical

Meta-feature	Description	Group
Quartile1MeansOfNumericAtts	First quartile of means among attributes of the numeric type.	statistical
Quartile1SkewnessOfNumericAtts	First quartile of skewness among attributes of the numeric type.	statistical
Quartile1StdDevOfNumericAtts	First quartile of standard deviation of attributes of the numeric type.	statistical
Quartile2KurtosisOfNumericAtts	Second quartile (Median) of kurtosis among attributes of the numeric type.	statistical
Quartile2MeansOfNumericAtts	Second quartile (Median) of means among attributes of the numeric type.	statistical
Quartile2SkewnessOfNumericAtts	Second quartile (Median) of skewness among attributes of the numeric type.	statistical
Quartile2StdDevOfNumericAtts	Second quartile (Median) of standard deviation of attributes of the numeric type.	statistical
Quartile3KurtosisOfNumericAtts	Third quartile of kurtosis among attributes of the numeric type.	statistical
Quartile3MeansOfNumericAtts	Third quartile of means among attributes of the numeric type.	statistical
Quartile3SkewnessOfNumericAtts	Third quartile of skewness among attributes of the numeric type.	statistical
Quartile3StdDevOfNumericAtts	Third quartile of standard deviation of attributes of the numeric type.	statistical
StdvNominalAttDistinctValues	Standard deviation of the number of distinct values among attributes of the nominal type.	statistical
MajorityClassPercentage	Percentage of instances belonging to the most frequent class.	statistical
MajorityClassSize	Number of instances belonging to the most frequent class.	statistical

Meta-feature	Description	Group
MaxKurtosisOfNumericAtts	Maximum kurtosis among attributes of the numeric type.	statistical
MaxMeansOfNumericAtts	Maximum of means among attributes of the numeric type.	statistical
MaxNominalAttDistinctValues	The maximum number of distinct values among attributes of the nominal type.	statistical
MaxSkewnessOfNumericAtts	Maximum skewness among attributes of the numeric type.	statistical
MaxStdDevOfNumericAtts	Maximum standard deviation of attributes of the numeric type.	statistical
MeanKurtosisOfNumericAtts	Mean kurtosis among attributes of the numeric type.	statistical
MeanMeansOfNumericAtts	Mean of means among attributes of the numeric type.	statistical
MeanNominalAttDistinctValues	Average number of distinct values among the attributes of the nominal type.	statistical
MeanSkewnessOfNumericAtts	Mean skewness among attributes of the numeric type.	statistical
MeanStdDevOfNumericAtts	Mean standard deviation of attributes of the numeric type.	statistical
MinKurtosisOfNumericAtts	Minimum kurtosis among attributes of the numeric type.	statistical
MinMeansOfNumericAtts	Minimum of means among attributes of the numeric type.	statistical
MinNominalAttDistinctValues	The minimal number of distinct values among attributes of the nominal type.	statistical
MinSkewnessOfNumericAtts	Minimum skewness among attributes of the numeric type.	statistical

Meta-feature	Description	Group
MinStdDevOfNumericAtts	Minimum standard deviation of attributes of the numeric type.	statistical
MinorityClassPercentage	Percentage of instances belonging to the least frequent class.	statistical
MinorityClassSize	Number of instances belonging to the least frequent class.	statistical
leaves	Number of leaf nodes in the DT model.	model-based
leaves_branch.mean	Size of branches in the DT model.	model-based
leaves_branch.sd	Size of branches in the DT model.	model-based
leaves_corrob.mean	Leaves corroboration of the DT model.	model-based
leaves_corrob.sd	Leaves corroboration of the DT model.	model-based
leaves_homo.mean	DT model Homogeneity for every leaf node.	model-based
leaves_homo.sd	DT model Homogeneity for every leaf node.	model-based
leaves_per_class.mean	Proportion of leaves per class in DT model.	model-based
leaves_per_class.sd	Proportion of leaves per class in DT model.	model-based
nodes	Number of non-leaf nodes in DT model.	model-based
nodes_per_attr	Ratio of nodes per number of attributes in DT model.	model-based
nodes_per_inst	Ratio of non-leaf nodes per number of instances in DT model.	model-based
nodes_per_level.mean	Ratio of number of nodes per tree level in DT model.	model-based
nodes_per_level.sd	Ratio of number of nodes per tree level in DT model.	model-based
nodes_repeated.mean	Number of repeated nodes in DT model.	model-based

Meta-feature	Description	Group
nodes_repeated.sd	Number of repeated nodes in DT model.	model-based
tree_depth.mean	Depth of every node in the DT model.	model-based
tree_depth.sd	Depth of every node in the DT model.	model-based
tree_imbalance.mean	Tree imbalance for each leaf node.	model-based
tree_imbalance.sd	Tree imbalance for each leaf node.	model-based
tree_shape.mean	Tree shape for every leaf node.	model-based
tree_shape.sd	Tree shape for every leaf node.	model-based
var_importance.mean	Features importance of the DT model for each attribute.	model-based
var_importance.sd	Features importance of the DT model for each attribute.	model-based
c1	Entropy of class proportions.	complexity
c2	Imbalance ratio.	complexity
f1.mean	Maximum Fisher's discriminant ratio.	complexity
f1.sd	Maximum Fisher's discriminant ratio.	complexity
f1v.mean	Directional-vector maximum Fisher's discriminant ratio.	complexity
f2.mean	Volume of the overlapping region.	complexity
f3.mean	Feature maximum individual efficiency.	complexity
f4.mean	Collective feature efficiency.	complexity
l1.mean	Sum of error distance by linear programming.	complexity
l2.mean	OVO subsets error rate of linear classifier.	complexity
l3.mean	Non-Linearity of a linear classifier.	complexity
t2	Average number of features per dimension.	complexity
t3	Average number of PCA dimensions per points.	complexity

Meta-feature	Description	Group
t4	Ratio of the PCA dimension to the original dimension.	complexity

APPENDIX E – DATASETS USED IN COMPARATIVE EXPERIMENTS

Table 16 shows the 152 datasets that were used in the experiments for comparative analysis of AutoML methods.

Table 16 – Datasets that were used in the comparative analysis. DID - Data ID, TID - Task ID.

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
965	3828	zoo	2	17	101
753	3619	wisconsin	2	33	194
187	2382	wine	3	14	178
1511	9988	wholesale-customers	2	9	440
1546	9919	volcanoes-e5	5	4	1112
1545	9918	volcanoes-e4	5	4	1252
1544	9917	volcanoes-e3	5	4	1277
1543	9916	volcanoes-e2	5	4	1080
1540	9922	volcanoes-d3	5	4	9285
1539	9921	volcanoes-d2	5	4	9172
1537	9933	volcanoes-c1	5	4	28626
1534	9930	volcanoes-b4	5	4	10190
1533	9929	volcanoes-b3	5	4	10386
1527	10103	volcanoes-a1	5	4	3252
736	3602	visualizing_environmental	2	4	111
719	3585	veteran	2	8	137
994	3857	vehicle	2	19	846
1508	9944	user-knowledge	5	6	403
788	3653	triazines	2	61	186
885	3748	transplant	2	4	131
1506	9969	thoracic-surgery	2	17	470
1115	3949	teachingAssistant	3	7	151

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
955	3818	tae	2	6	151
902	3765	sleuth_case2002	2	7	147
958	3821	segment	2	20	2310
1498	9961	sa-heart	2	10	462
1520	9938	robot-failures-lp5	5	91	164
996	3859	prnn_fglass	2	10	214
1567	9890	poker-hand	10	11	1025009
1490	9953	planning-relax	2	13	182
1167	4001	pc1_req	2	9	320
1488	9951	parkinsons	2	23	195
784	3649	newton_hema	2	4	140
880	3744	mu284	2	11	284
164	2373	molecular-biology_promoters	2	58	106
995	3858	mfeat-zernike	2	48	2000
962	3825	mfeat-morphological	2	7	2000
1020	3883	mfeat-karhunen	2	65	2000
971	3834	mfeat-fourier	2	77	2000
733	3599	machine_cpu	2	7	209
10	10	lymph	4	19	148
941	3804	lowbwt	2	10	189
1045	3898	kc1-top5	2	95	145
1066	3916	kc1-binary	2	95	145
1048	3901	jEdit_4.2_4.3	2	9	369
1073	3921	jEdit_4.0_4.2	2	9	274
969	3832	iris	2	5	150
843	3708	house_8L	2	9	22784
1513	9948	heart-switzerland	5	13	123

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
53	52	heart-statlog	2	14	270
1565	9894	heart-h	5	14	294
1026	3888	grub-damage	2	9	155
1005	3868	glass	2	10	214
714	3580	fruitfly	2	5	125
805	3670	fri_c4_500_50	2	51	500
838	3703	fri_c4_500_25	2	26	500
855	3720	fri_c4_500_10	2	11	500
918	3781	fri_c4_250_50	2	51	250
749	3615	fri_c3_500_5	2	6	500
896	3759	fri_c3_500_25	2	26	500
873	3737	fri_c3_250_50	2	51	250
744	3610	fri_c3_250_5	2	6	250
792	3657	fri_c2_500_5	2	6	500
879	3743	fri_c2_500_25	2	26	500
869	3733	fri_c2_500_10	2	11	500
877	3741	fri_c2_250_50	2	51	250
766	3632	fri_c1_500_50	2	51	500
870	3734	fri_c1_500_5	2	6	500
779	3645	fri_c1_500_25	2	26	500
824	3689	fri_c1_500_10	2	11	500
888	3751	fri_c0_500_50	2	51	500
926	3789	fri_c0_500_25	2	26	500
943	3806	fri_c0_500_10	2	11	500
732	3598	fri_c0_250_50	2	51	250
1012	3875	flags	2	29	194
1473	9984	fertility	2	10	100

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
1011	3874	ecoli	2	8	336
931	3794	disclosure_z	2	4	662
795	3660	disclosure_x_tampered	2	4	662
827	3692	disclosure_x_noise	2	4	662
774	3640	disclosure_x_bias	2	4	662
818	3683	diggle_table_a2	2	9	310
1075	3923	datatrieve	2	9	130
796	3661	cpu	2	8	209
150	218	covertype	7	55	581012
890	3753	cloud	2	8	108
900	3763	chscase_census6	2	7	400
906	3769	chscase_census5	2	8	400
907	3770	chscase_census4	2	8	400
908	3771	chscase_census3	2	8	400
909	3772	chscase_census2	2	8	400
991	3854	car	2	7	1728
23499	52945	breast-cancer-dropped	2	10	277
853	3718	boston	2	14	506
778	3644	bodyfat	2	15	252
1463	10094	blogger	2	6	100
1121	3955	badges2	2	11	294
463	3554	backache	2	32	180
745	3611	auto_price	2	16	159
1554	9903	autoUniv-au7-500	5	13	500
1551	9907	autoUniv-au6-400	8	41	400
1555	9902	autoUniv-au6-1000	8	41	1000
756	3622	autoPrice	2	16	159

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
1061	3911	ar4	2	30	107
1059	3909	ar1	2	30	121
748	3614	analcatdata_wildcat	2	6	163
724	3590	analcatdata_vineyard	2	4	468
921	3784	analcatdata_seropositive	2	4	132
771	3637	analcatdata_michiganacc	2	4	108
450	3542	analcatdata_lawsuit	2	5	264
1025	3887	analcatdata_germangss	2	6	400
461	3552	analcatdata_creditscore	2	7	100
875	3739	analcatdata_chlamydia	2	4	100
444	3538	analcatdata_boxing2	2	4	132
448	3540	analcatdata_boxing1	2	4	120
764	3630	analcatdata_apnea3	2	4	450
765	3631	analcatdata_apnea2	2	4	475
767	3633	analcatdata_apnea1	2	4	475
1556	9901	acute-inflammations	2	7	120
337	3496	SPECTF	2	45	349
336	3495	SPECT	2	23	267
160	228	RandomRBF_50_1E-4	5	11	1000000
159	227	RandomRBF_50_1E-3	5	11	1000000
158	226	RandomRBF_10_1E-4	5	11	1000000
157	225	RandomRBF_10_1E-3	5	11	1000000
156	224	RandomRBF_0_0	5	11	1000000
1100	3937	PopularKids	3	11	478
1444	7559	PizzaCutter3	2	38	1043
1443	7558	PizzaCutter1	2	38	661
1453	10099	PieChart3	2	38	1077

DID	TID	Dataset	Nº of class	Nº of attributes	Nº of instances
1452	10098	PieChart2	2	37	745
1451	10097	PieChart1	2	38	705
1442	7556	MegaWatt1	2	38	253
1113	3947	KDDCup99	23	42	494020
1446	7557	CostaMadre1	2	38	296
1447	10096	CastMetal1	2	38	327
272	2267	BNG(zoo)	7	17	1000000
271	2266	BNG(waveform-5000)	3	41	1000000
268	2263	BNG(vehicle)	4	19	1000000
264	2259	BNG(sonar)	2	61	1000000
130	156	BNG(segment)	7	20	1000000
261	2256	BNG(pendigits)	10	17	1000000
259	2254	BNG(page-blocks)	5	11	295245
123	149	BNG(optdigits)	10	65	1000000
254	2145	BNG(mfeat-zernike)	10	48	1000000
252	2143	BNG(mfeat-karhunen)	10	65	1000000
250	2141	BNG(mfeat-fourier)	10	77	1000000
249	2140	BNG(lymph)	4	19	1000000
72	134	BNG(kr-vs-kp)	2	37	1000000
146	215	BNG(ionosphere)	2	35	1000000
267	2262	BNG(heart-statlog)	2	14	1000000
265	2260	BNG(glass)	7	10	137781

APPENDIX F – COMPARATIVE PERFORMANCE OF META-MODELS: MULTI-LABEL VS. SINGLE-LABEL APPROACHES

An analysis was conducted comparing our multi-label approach (PCC in the zero-shot setting) with traditional single-label models (RF, SVC, and DT). Table 17 compares MetaML using three different multi-label approaches (LP, CC, and PCC) versus three single-label approaches (RF, DT, and SVC). The statistical significance was evaluated using the Wilcoxon signed-rank test, and p-values were adjusted using a Bonferroni correction to control for family-wise Type I error, with a corrected significance level of $\alpha = 0.004$, indicating cases where there is a significant difference between the compared method and MetaML's performance.

Table 17 – Wins, ties of losses of MetaML using three different multi-label approaches versus single label approaches. The p-values, resulting from a Wilcoxon signed rank test that are lower than $\alpha = 0.004$ are highlighted in bold.

	Win/tie/loss	**p-value**
MetaML-LP		
MetaML-RF	31/208/51	0.038
MetaML-SVC	54/168/68	0.161
MetaML-DT	52/189/49	0.437
MetaML-CC		
MetaML-RF	31/189/70	**5.54e-05**
MetaML-SVC	50/152/88	**0.000**
MetaML-DT	43/182/65	0.083
MetaML-PCC zero-shot		
MetaML-RF	33/186/71	**3.65e-05**
MetaML-SVC	52/148/90	**0.000**
MetaML-DT	45/179/66	0.062

The results indicate no statistically significant difference between the multi-label models and MetaML-DT. However, significant differences were observed when comparing RF and SVC with multi-label models CC and PCC. These results suggest that single-label approaches, particularly RF and SVC, achieved slightly better performance than the multi-label. Despite the somewhat better performance, we emphasize that the core motivation for adopting multi-label meta-models, particularly PCC, lies in their ability to capture the interdependence of pipeline steps.

APPENDIX G – PERFORMANCE ANALYSIS: REAL VS. ARTIFICIAL DATASETS

In Table 18, we present the mean performance, mean rank, and Win/Tie/Loss analyses of MetaML 3-shot. The results are analyzed separately for the 73 real datasets (48.03%) and the 79 artificial datasets (51.97%), considering the 152 datasets used for comparison with the state of the art.

Table 18 – A base-level analysis of the mean performance (accuracy) of the pipelines recommended by the AutoML methods, Win/tie/loss of the MetaML 3-shot in real (a) and artificial (b) datasets against the others, mean ranking, p-value of the Wilcoxon signed rank test with $\alpha = 5.56\text{e-}04$.

	Mean acc.	Win/tie/loss	Mean rank	p-value
Auto-WEKA	0.830	51/4/18	6.178	0.002
AutoSklearn 1.0	0.832	37/10/26	5.136	0.062
AutoSklearn 2.0	0.847	33/9/31	**3.863**	0.488
AutoGluon	0.841	48/7/18	6.363	0.005
H2O AutoML	0.813	53/1/19	6.815	5.728e-05
Naive AutoML	0.830	36/14/23	5.089	0.060
TPOT	0.819	35/12/26	5.260	0.125
FLAML-Zero	0.770	55/11/7	7.856	4.275e-10
TabPFN	0.835	31/17/25	4.212	0.607
MetaML 3-shot	**0.845**	-	4.226	-

(a) Real

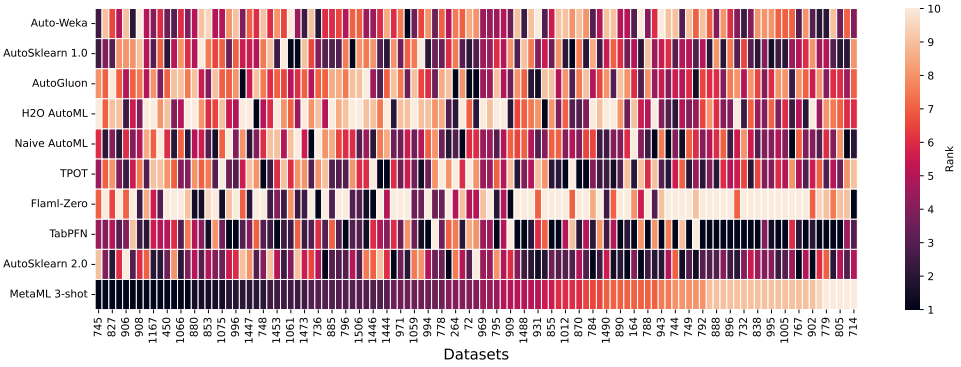
	Mean acc.	Win/tie/loss	Mean rank	p-value
Auto-WEKA	0.663	49/7/23	7.018	0.000
AutoSklearn 1.0	0.721	33/5/41	5.373	0.820
AutoSklearn 2.0	0.741	29/7/43	4.588	0.161
AutoGluon	**0.819**	26/2/51	4.943	5.18e-05
H2O AutoML	0.766	39/1/39	6.240	0.873
Naive AutoML	0.794	30/8/41	4.740	0.044
TPOT	0.679	41/3/35	5.860	0.109
FLAML-Zero	0.749	52/12/15	6.468	9.23e-08
TabPFN	0.704	40/5/34	**4.373**	0.399
MetaML 3-shot	0.799	-	5.392	-

(b) Artificial

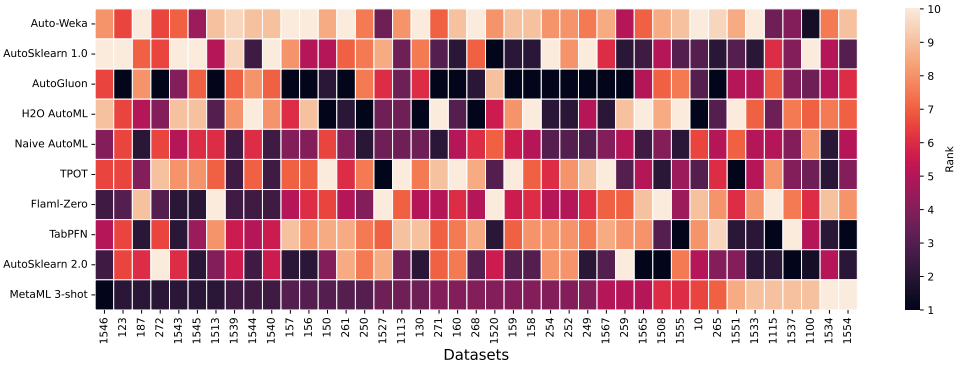
Considering the real datasets, MetaML 3-shot achieved the highest mean accuracy (0.845). Although it did not obtain the best ranking, its average rank (4.226) was very close to that of AutoSklearn 2.0 (3.863) and TabPFN (4.212). We observed that the AutoML methods had a superior performance, with more wins than losses in the real data, with FLAML-Zero standing out as the one with the worst performance. In the artificial datasets, AutoGluon achieved the best average accuracy, followed by MetaML 3-shot, while TabPFN obtained the best average ranking. The method that presented the worst performance was Auto-WEKA. In general, all methods performed better on real datasets.

APPENDIX H – COMPARISON OF AUTOMLS CONSIDERING DATASETS CLASSES AND INSTANCES

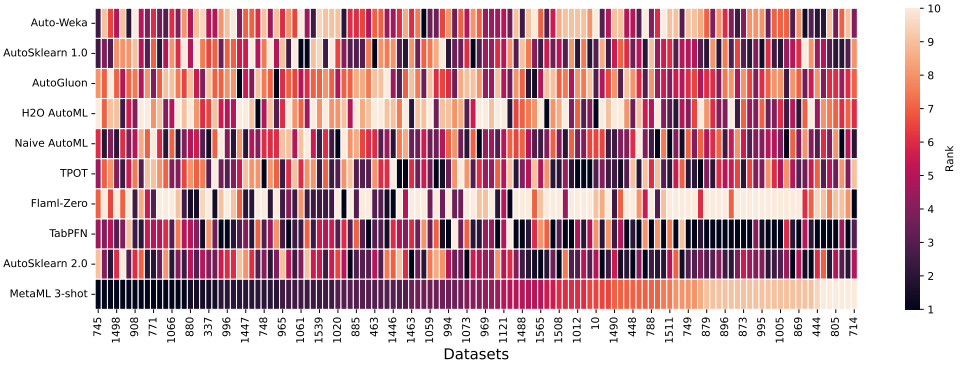
An analysis was performed considering the characteristics of the datasets, such as more classes, fewer classes, more instances, and fewer instances, comparing MetaML and AutoML methods per dataset in terms of accuracy, illustrated in a heatmap in Figure 28.



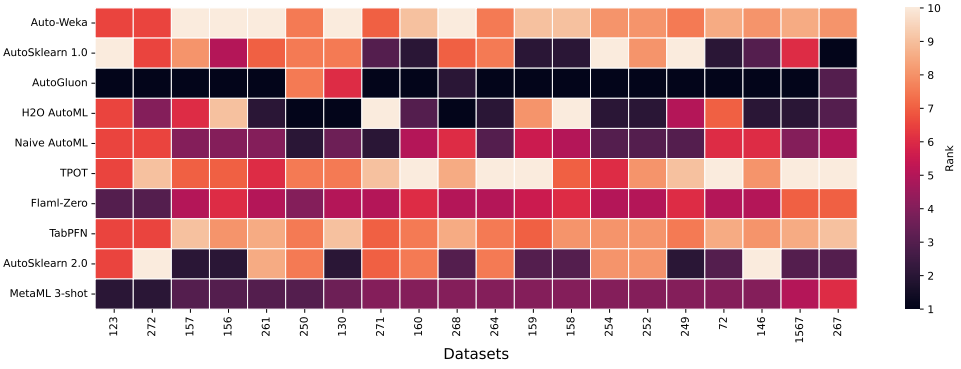
(a) Binary datasets.



(b) Multiclass datasets.



(c) Datasets with less than $10\,000$ instances.



(d) Datasets with more than $1\,000\,000$ instances.

Figure 28 – Accuracy comparison of AutoMLs considering specific dataset characteristics.

The key findings from these heatmaps are highlighted below:

- For binary classification (a) and datasets with up to 10,000 instances (c), AutoML methods and MetaML 3-shot generally achieve better ranks (indicated by darker colors), with few cases of low performance (indicated by lighter shades in the heatmaps).
- For multiclass tasks (b) and datasets with more than 1 million instances (d), the performance of several methods, such as Auto-Weka, AutoSklearn 1.0, TPOT, TabPFN, and H2O AutoML, drops significantly, as seen by the prevalence of lighter shades (worse ranks).
- AutoGluon stands out for maintaining strong performance even in challenging scenarios (multiclass and large datasets), showing a behavior similar to MetaML 3-shot.
- MetaML 3-shot consistently performs well across all scenarios, including binary, multiclass, small, and large datasets, evident from the consistent presence of darker regions in all heatmaps.
- Although TabPFN performs well in binary tasks (a), it underperforms in multiclass (b) and large-scale (d) datasets compared to MetaML 3-shot.

APPENDIX I – EXAMPLES OF PIPELINES RECOMMENDED BY METAML AND NAIVE AUTOML

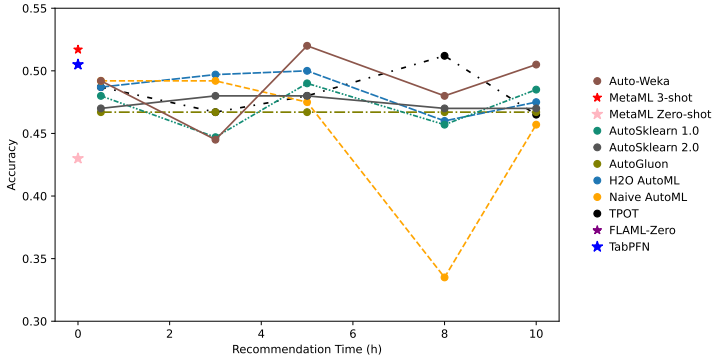
Figure 29 showcases the recommendations for four datasets by MetaML and Naive AutoML. The pipelines are represented by a tuple of blocks. We observe that each method presents distinct search spaces and number of pipeline blocks, with Naive AutoML also including hyperparameter settings.

Figure 29 – Representation of the pipelines.

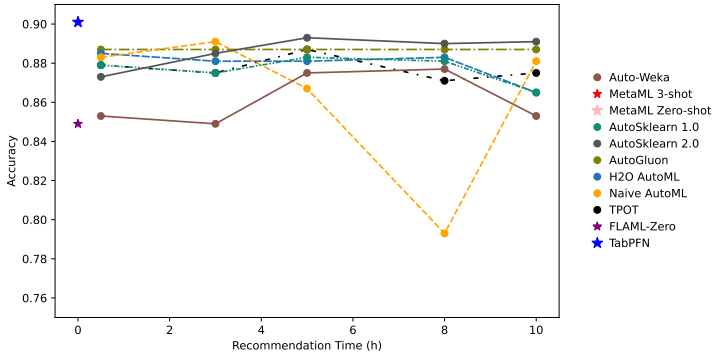
Furthermore, on the vehicle (data ID: 994) and wholesale customer (data ID: 1511) datasets, MetaML recommends Random Forest (RF), while Naive AutoML applies Power Transform (PT) followed by Extremely Randomized Trees (ET). This highlights how the Naive AutoML naive approach, where each step pipeline is independent, can introduce redundancies. For example, preprocessing steps like PT (a scaling technique) do not improve the performance of tree-based algorithms.

APPENDIX J – COMPARISON OF AUTOML METHODS USING DIFFERENT TIME BUDGETS

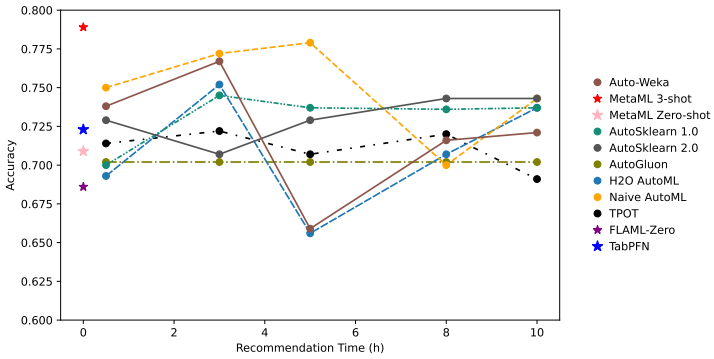
Figure 30 presents a comparison (base-level) of AutoML methods versus the MetaML's in datasets of different sizes and time budgets.



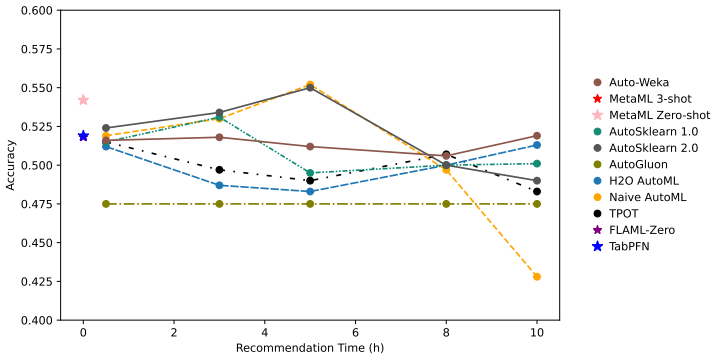
(a) chscase_census3 dataset with 400 instances.



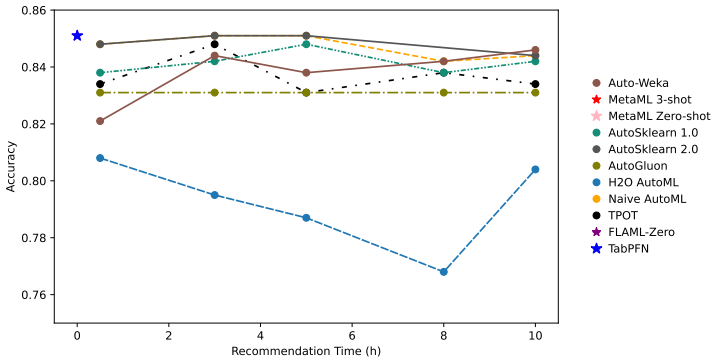
(b) boston dataset with 506 instances.



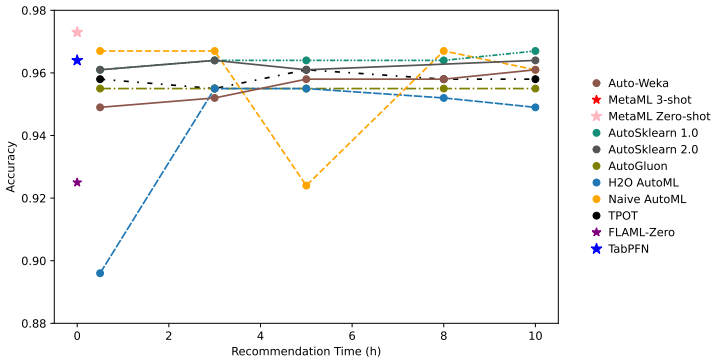
(a) veteran dataset with 137 instances.



(b) disclosure_x_noise dataset with 662 instances.



(c) thoracic-surgery dataset with 470 instances.



(d) ecoli dataset with 336 instances.

Figure 30 – Comparison of AutoML methods using different recommendation time budgets.

APPENDIX K – META-FEATURES

Table 20 shows the eighty-one meta-features used to represent each dataset.

Table 19 – Meta-features used to construct the meta-dataset MetaMl 2.0.

Meta-feature	Description	Group
attr_conc.mean	Concentration coef. of each pair of distinct attributes.	info-theory
attr_conc.sd	Concentration coef. of each pair of distinct attributes.	info-theory
attr_ent.mean	Shannon's entropy for each predictive attribute.	info-theory
attr_ent.sd	Shannon's entropy for each predictive attribute.	info-theory
class_conc.mean	Concentration coefficient between each attribute and class.	info-theory
class_conc.sd	Concentration coefficient between each attribute and class.	info-theory
class_ent	Target attribute Shannon's entropy.	info-theory
eq_num_attr	Number of attributes equivalent for a predictive task.	info-theory
joint_ent.mean	Joint entropy between each attribute and class.	info-theory
joint_ent.sd	Joint entropy between each attribute and class.	info-theory
mut_inf.mean	Mutual information between each attribute and target.	info-theory
mut_inf.sd	Mutual information between each attribute and target.	info-theory
ns_ratio	Noisiness of attributes.	info-theory
leaves	Number of leaf nodes in the DT model.	model-based

Meta-feature	Description	Group
leaves_branch.mean	Size of branches in the DT model.	model-based
leaves_branch.sd	Size of branches in the DT model.	model-based
leaves_corrob.mean	Leaves corroboration of the DT model.	model-based
leaves_corrob.sd	Leaves corroboration of the DT model.	model-based
leaves_homo.mean	DT model Homogeneity for every leaf node.	model-based
leaves_homo.sd	DT model Homogeneity for every leaf node.	model-based
leaves_per_class.mean	Proportion of leaves per class in DT model.	model-based
leaves_per_class.sd	Proportion of leaves per class in DT model.	model-based
nodes	Number of non-leaf nodes in DT model.	model-based
nodes_per_attr	Ratio of nodes per number of attributes in DT model.	model-based
nodes_per_inst	Ratio of non-leaf nodes per number of instances in DT model.	model-based
nodes_per_level.mean	Ratio of number of nodes per tree level in DT model.	model-based
nodes_per_level.sd	Ratio of number of nodes per tree level in DT model.	model-based
nodes_repeated.mean	Number of repeated nodes in DT model.	model-based
nodes_repeated.sd	Number of repeated nodes in DT model.	model-based
tree_depth.mean	Depth of every node in the DT model.	model-based
tree_depth.sd	Depth of every node in the DT model.	model-based
tree_imbalance.mean	Tree imbalance for each leaf node.	model-based
tree_imbalance.sd	Tree imbalance for each leaf node.	model-based
tree_shape.mean	Tree shape for every leaf node.	model-based
tree_shape.sd	Tree shape for every leaf node.	model-based

Meta-feature	Description	Group
var_importance.mean	Features importance of the DT model for each attribute.	model-based
var_importance.sd	Features importance of the DT model for each attribute.	model-based
cov.mean	Absolute value of the covariance of distinct dataset attribute pairs.	statistical
cov.sd	Absolute value of the covariance of distinct dataset attribute pairs.	statistical
eigenvalues.mean	Eigenvalues of covariance matrix from dataset.	statistical
eigenvalues.sd	Eigenvalues of covariance matrix from dataset.	statistical
gravity	Distance between minority and majority classes center of mass.	statistical
iq_range.mean	Interquartile range (IQR) of each attribute.	statistical
iq_range.sd	Interquartile range (IQR) of each attribute.	statistical
mad.mean	Median Absolute Deviation (MAD) adjusted by a factor.	statistical
mad.sd	Median Absolute Deviation (MAD) adjusted by a factor.	statistical
nr_cor_attr	Number of distinct highly correlated pair of attributes.	statistical
nr_norm	Number of attributes normally distributed based in a given method.	statistical
nr_outliers	Number of attributes with at least one outlier value.	statistical
sparsity.mean	Sparsity metric for each attribute.	statistical
sparsity.sd	Sparsity metric for each attribute.	statistical

Meta-feature	Description	Group
t_mean.mean	Trimmed mean of each attribute.	statistical
t_mean.sd	Trimmed mean of each attribute.	statistical
attr_to_inst	Ratio between the number of attributes.	simple
cat_to_num	Ratio between the number of categoric and numeric features.	simple
freq_class.mean	Relative frequency of each distinct class.	simple
freq_class.sd	Relative frequency of each distinct class.	simple
inst_to_attr	Ratio between the number of instances and attributes.	simple
nr_attr	Total number of attributes.	simple
nr_bin	Number of binary attributes.	simple
nr_cat	Number of categorical attributes.	simple
nr_class	Number of distinct classes.	simple
nr_inst	Number of instances (rows) in the dataset.	simple
nr_num	Number of numeric features.	simple
c1	Entropy of class proportions.	complexity
c2	Imbalance ratio.	complexity
cls_coef	Clustering coefficient.	complexity
density	Average density of the network.	complexity
f1v.mean	Maximum Fisher's discriminant ratio.	complexity
f3.mean	Feature maximum individual efficiency.	complexity
f4.mean	Collective feature efficiency.	complexity
l1.mean	Sum of error distance by linear programming.	complexity
l2.mean	OVO subsets error rate of linear classifier.	complexity
l3.mean	Non-Linearity of a linear classifier.	complexity

Meta-feature	Description	Group
n3.mean	Mean error rate of the nearest neighbor classifier.	complexity
n3.sd	Error rate of the nearest neighbor classifier.	complexity
n4.mean	Non-linearity of the k-NN Classifier.	complexity
n4.sd	Compute the non-linearity of the k-NN Classifier.	complexity
t2	Average number of features per dimension.	complexity
t3	Average number of PCA dimensions per points.	complexity
t4	Ratio of the PCA dimension to the original dimension.	complexity

APPENDIX L – LIST OF EXCLUDED META-FEATURES

The list of the 49 meta-features excluded for having more than 10% missing values is presented in Table 20

Table 20 – Meta-features excluded.

Meta-feature	Description	Group
range.mean	Range (max - min) of each attribute.	statistical
range.sd	Range (max - min) of each attribute.	statistical
mean.mean	Mean value of each attribute.	statistical
mean.sd	Mean value of each attribute.	statistical
min.mean	Minimum value from each attribute.	statistical
min.sd	Minimum value from each attribute.	statistical
roy_root	Roy's largest root.	statistical
median.mean	Median value from each attribute.	statistical
median.sd	Median value from each attribute.	statistical
nr_disc	Number of canonical correlation between each attribute and class.	statistical
max.sd	Maximum value from each attribute.	statistical
max.mean	Maximum value from each attribute.	statistical
w_lambda	Wilks' Lambda value.	statistical
lh_trace	Lawley-Hotelling trace.	statistical
sd.mean	Standard deviation of each attribute.	statistical
can_cor.sd	Canonical correlations of data.	statistical
can_cor.mean	Canonical correlations of data.	statistical
sd.sd	Standard deviation of each attribute.	statistical
sd_ratio	Statistical test for homogeneity of covariances.	statistical
var.mean	Variance of each attribute.	statistical
var.sd	Variance of each attribute.	statistical

Meta-feature	Description	Group
p_trace	Pillai's trace.	statistical
g_mean.mean	Geometric mean of each attribute.	statistical
g_mean.sd	Geometric mean of each attribute.	statistical
h_mean.sd	Harmonic mean of each attribute.	statistical
h_mean.mean	Harmonic mean of each attribute.	statistical
cor.mean	Absolute value of the correlation of distinct dataset column pairs.	statistical
cor.sd	Absolute value of the correlation of distinct dataset column pairs.	statistical
kurtosis.sd	kurtosis of each attribute.	statistical
kurtosis.mean	kurtosis of each attribute.	statistical
skewness.sd	Skewness for each attribute.	statistical
skewness.mean	Skewness for each attribute.	statistical
num_to_cat	Number of numerical and categorical features.	simple
f1v.sd	Maximum Fisher's discriminant ratio.	complexity
l3.sd	Non-Linearity of a linear classifier.	complexity
l1.sd	Sum of error distance by linear programming.	complexity
l2.sd	OVO subsets error rate of linear classifier.	complexity
f4.sd	Collective feature efficiency.	complexity
f3.sd	Feature maximum individual efficiency.	complexity
f2.mean	Volume of the overlapping region.	complexity
n2.sd	Ratio of intra and extra class nearest neighbor distance.	complexity
n2.mean	Ratio of intra and extra class nearest neighbor distance.	complexity
f1.sd	Maximum Fisher's discriminant ratio.	complexity

Meta-feature	Description	Group
f1.mean	Maximum Fisher's discriminant ratio.	complexity
f2.sd	Volume of the overlapping region.	complexity
n1	Fraction of borderline points.	complexity
lsc	Local set average cardinality.	complexity
hubs.sd	Hub score.	complexity
hubs.mean	Hub score.	complexity