AutoML for Multi-Label Classification

Authors:-**Kushal Saraf**, Department of Computer Science, NMIMS Computer Science,

Abstract

Automated Machine Learning (AutoML), which includes the choice, fusion, and parameterization of machine learning algorithms as its basic components, makes it possible to manipulate numerical data effectively and to customize machine learning pipelines. Broadly, AutoML approaches consist of two major components: a search space model and an optimizer that navigates through the space. Recent methods have demonstrated significant success in supervised learning domains, particularly in single-label classification (SLC). Initial attempts have also been made to extend these approaches to multi-label classification (MLC). While the space of potential pipelines is already extensive in SLC, the complexity of the search space becomes even more intricate in MLC.

One might question whether the optimizers well-suited for SLC can effectively handle this heightened complexity in MLC, and to what extent they adapt. Firstly, we review existing approaches. Secondly, we enhance these approaches by incorporating optimizers previously unexplored for MLC. Thirdly, we establish a benchmarking framework that ensures a fair and systematic comparison. Finally, we undertake a comprehensive exploratory study, evaluating the methods across a range of MLC problems. Notably, we observe that a syntax-based best-first search exhibits commendable performance in comparison to other optimizers.

INTRODUCTION

AutoML Brief

The recent emergence of automated machine learning has played a crucial role in democratizing the field of artificial intelligence [1].

AutoML refers to a set of tools and techniques that streamline the process of developing machine learning models by automating tasks such as selecting optimal network architectures, pre-processing methods, and hyperparameter optimization. Using a simple graphical interface, users can build highly accurate machine learning models without the need for writing complex code. Automated machine learning has become increasingly important in the field of artificial intelligence as it seeks to automate various tasks involved in machine learning and enable non-experts to use machine learning effectively. Automated machine learning is a technological advancement that holds the potential to democratize the field of artificial intelligence.

The recent emergence of automated machine learning has been hailed as a major step in the democratization of AI. AutoML allows for the automation of key tasks in model development, such as selecting optimal network architectures, pre-processing methods, and hyperparameter optimization.

By utilizing a simple graphical interface, users can build highly accurate machine-learning models without the need for extensive coding. Automated machine learning seeks to automate tasks involved in machine learning, enabling non-experts to use this technology more widely. Automated machine learning has emerged as a significant development in the field of artificial intelligence, representing a major step toward the democratization of AI. Automated machine learning is a recent development that has had a significant impact on democratizing the field of artificial intelligence. AutoML refers to a set of tools and techniques that automate various aspects of machine learning model development, including the selection of network architectures, pre-processing methods, and hyperparameter optimization.

The development of automated machine learning has been a major milestone in the democratization of artificial intelligence. AutoML allows for the automation of time-consuming and tedious tasks involved in building machine learning models, such as hyperparameter tuning, feature selection, and model selection

[2]. Automated machine learning, or AutoML, has emerged as a transformative advancement that streamlines model development by automating the selection of optimal network architectures, pre-processing methods, and hyperparameter optimization. Automated machine learning, or AutoML, has been instrumental in democratizing artificial intelligence by automating various tasks involved in machine learning and enabling non-experts to utilize this technology effectively. AutoML has the potential to make AI more accessible to a wider range of users, including those without extensive coding knowledge or expertise in machine learning.
The recent emergence of automated machine learning, also known as AutoML, has been a significant step in the democratization of artificial intelligence [1]. AutoML refers to a set of techniques and tools that automate the manual and time-consuming aspects of building machine learning models [2].
The building of models is streamlined by automated machine learning, or AutoML, which automates the selection of the best netoptimization, pre-processing techniques, and hyperparameter optimisation.

AutoML has the potential to make machine learning more accessible and user-friendly by removing the barriers of technical expertise and reducing the time and effort required for model development [1].
In the rapidly evolving field of artificial intelligence, automated machine learning has emerged as a pivotal advancement that plays a crucial role in democratizing.

AUTOMATED machine intelligence (AutoML) is commonly understood as the task of automating the process of creating a "machine learning pipeline" tailored specifically to a given problem, particularly to a dataset on which a predictive model needs to be inferred. This involves the selection, combination, and parameterization of machine learning (ML) algorithms as the foundational elements of the pipeline. This core feature constitutes the primary advantage offered by an AutoML framework, enabling the construction of a robust pipeline that can be effectively trained on the dataset. This stands in contrast to "basic" ML algorithms, such as feedforward neural networks or support vector machines, which address specific prediction tasks. In this context, AutoML can be seen as addressing a "learning to learn" task.

For conventional problem categories like single-label classification (SLC) and regression, numerous approaches have been proposed and extensively evaluated in recent years, yielding promising results in various exploratory studies.

However, the practical application of AutoML frameworks presents challenges and can lead to misinterpretations. This is primarily due to the intricate nature of AutoML, comprising multiple components including a search space model and an optimization mechanism. Consequently, understanding the specific factor responsible for improved performance can be elusive. Diverse frameworks, for instance [1] and [2], often employ varying search space scopes, complicating the attribution of performance enhancements solely to a single aspect. Although optimizing the search space can improve the representation of an AutoML framework, it can hinder result interpretation when novel approaches are introduced to address the search task. This leads to uncertainties about whether the improved performance arises from the enhanced search space representation or the newly proposed search mechanism.

Advancing beyond standard single-goal prediction tasks, initial endeavors have been made to extend AutoML to multi-label classification (MLC) tasks [3]. This is particularly relevant for the common MLC problem [4], [5], [6], [7], [8]. While the space of candidate pipelines is already vast in SLC, the complexity of the search space significantly increases in MLC. This is primarily due to the incorporation of more intricate learning algorithms for MLC, often serving as meta-algorithms in conjunction with existing SLC learning algorithms (such as individual per label). An illustrative example of a potential multi-label classifier configuration is depicted in Figure 1. Notably, the MLC search space encompasses the SLC search space, demonstrating various opportunities for configuration. Furthermore, evaluating candidate solutions is notably more time-consuming for MLC compared to SLC due to the elevated inherent complexity.

Given these challenges, one may question whether existing enhancement techniques designed for probing candidate pipelines, primarily developed for SLC, are capable of scaling to the heightened complexity of MLC search spaces and how they compare to one another. Addressing this inquiry systematically, this paper makes the following contributions:

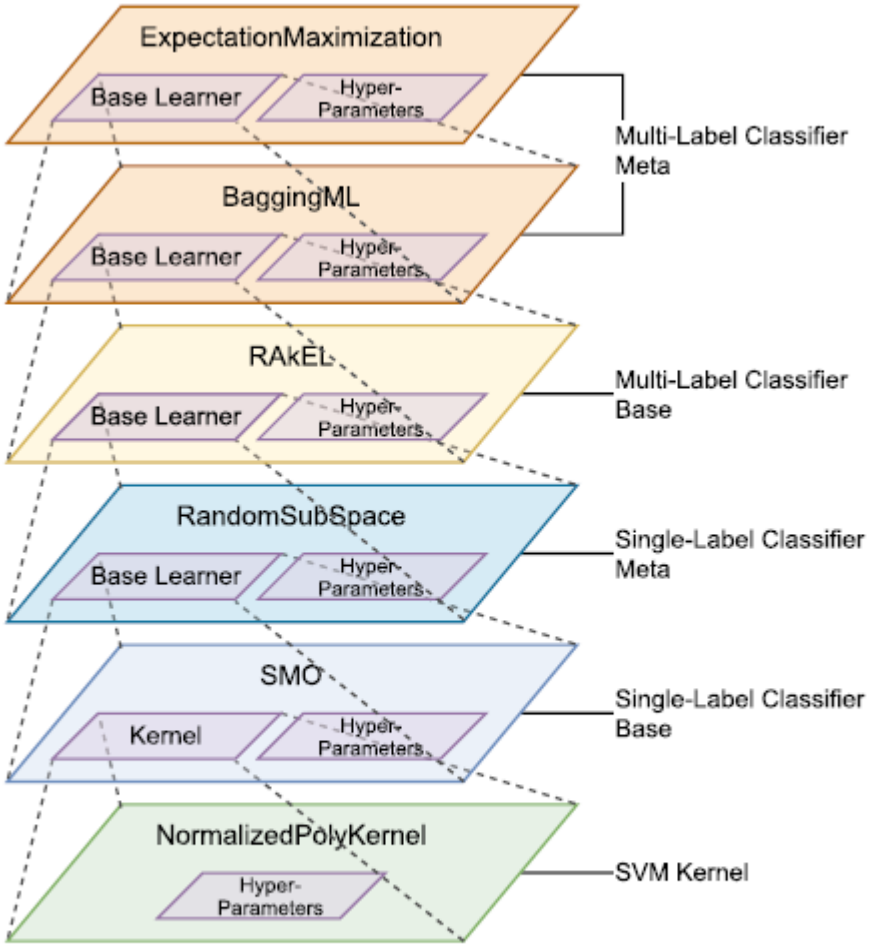


Fig. 1. Hierarchical representation of a multi-label classifier's structure being recursively configured with base learners and finally a kernel for the support vector machine (SMO, short for Sequential Minimal Optimization).

To begin, we will provide an overview of the cutting-edge methods, examining various systemic approaches about their relevance to the Multi-Label Classification (MLC) issue. We will then present an outline of the existing approaches to Automated Machine Learning (AutoML) for MLC, primarily characterized by their search space specifications (Section 4).

Next, we will further expand upon these methodologies by introducing enhancement techniques that have been pursued for MLC so far. This will include Bayesian optimization, evolutionary algorithms, and combinations of these methods (see Section 5).

In the third part, we will propose a benchmarking framework designed to facilitate a fair and systematic comparison (Section 6). Our framework ensures that all optimization strategies adhere to the same runtime constraints, operate within similar search space models, and utilize a consistent evaluation procedure for solution candidates.

Moving on, using this framework, we will conduct an extensive experimental survey, evaluating these methods on a set of MLC problems (Section 7). Our trials reveal that all approaches seem to grapple with the immense size of the search space. However, a syntax-based best-first search approach emerges as the top performer within the considered MLC search space, surpassing the other optimization methods.

Before delving into the main contributions of the paper as outlined above, we will provide a brief introduction to Automated Machine Learning (Section 2) and Multi-Label Classification (Section 3).

Automated Machine Learning

Despite the relatively short history of Automated Machine Learning, a diverse array of methods has been proposed to address the issue of combined algorithm selection and hyperparameter optimization (abbreviated as "AutoML" or "Money"). This concept was initially introduced in [9] and can be formally described as follows:

Let A = {A(1), A(2), ..., A(n)} represents a set of algorithms, and L(1), L(2), ..., L(n) denote their corresponding hyperparameter spaces. Further, let the training (validation) and test data from a dataset space D be represented as Dtrain = (Xtrain, Ytrain) ∈ D and Dtest = (Xtest, Ytest) ∈ D, along with a target loss function L to be minimized. The objective is to find an algorithm A' along with an appropriate hyperparameter configuration that generalizes well beyond the training data:

$$A' = \text{again } A(i) \in A,\ \theta \in L(i)\ E[L(Ytest, A(j)(Xtest))],$$

where θ denotes the hyperparameters for algorithm A(j), and j ranges from 1 to n.

In practice, however, the test loss is unavailable and is approximated using the average validation loss. For this purpose, the training data is split into training data (D0train) used for training and validation data (Dval = (Xval, Yval)) for assessing the performance of solution candidates:

$$A' = \text{argmin } A(i) \in A,\ \theta \in L(i)\ E[L(Yval, A(j)(Xval))],$$

where θ represents the hyperparameters for algorithm A(j).

The estimated performance is then utilized to guide the search for the best solution to the AutoML problem.

Initial approaches reduced the AutoML problem to a Hyperparameter Optimization (HPO) problem. This involved treating the choice of an algorithm and another hyperparameter as a binary factor (set to 1 if the specific algorithm is chosen) and connecting them with the hyperparameters of the chosen algorithms into a single hyperparameter vector. While this reduction made the problem amenable to established HPO tools such as SMAC [10], Hyperband [11], and their combination BOHB [12], it came with the potential downside of losing essential information due to "flattening" the search space.

Multi-Label Classification

In addition to the aforementioned tools, several other interesting methods have emerged in recent years. These include brain engineering search [21], techniques focusing on stacking [22], [23], utilizing reinforcement learning [24], and harnessing the power of random search for parallelization [25].

However, due to the rapid development in this field, it has become challenging to keep track of the overall progress and understand the strengths and weaknesses of various analyzers and comprehensive AutoML tools. Notably, newly proposed tools are often evaluated on different datasets and compared to a more or less randomly selected subset of existing tools as baselines. This approach complicates the global understanding of the diverse AutoML tools and their performances. Another challenge arises from the fact that new AutoML approaches are frequently composed of multiple components: optimization method, search space, and evaluation methodology (including data splitting for training, validation, and testing, as well as performance metrics) for assessing solution candidates. Consequently, performance improvements or differences cannot be attributed to a single specific modification. Despite initial efforts in this direction [26], [27], a comprehensive large-scale comparison of core optimization methods operating on a shared search space of reasonable size remains an open issue, particularly in the context of multi-label classification (MLC).

MULTI-Label Classification

Multi-label classification represents a distinct type of multi-target prediction [3], where the targets comprise parallel factors encoding the "relevance" or "irrelevance" of a specific aspect (denoted by a label) for a data instance (an observation). In MLC, the primary task is to learn set-valued function mapping instances to subsets of potentially relevant class labels. Essentially, MLC can be regarded as an extension of standard multi-class classification, where an instance is assigned to exactly one class. For instance, consider the problem of image labeling, where an image may be labeled with multiple class names such as "Sun," "Oceanside," and "Yacht" simultaneously. For a more comprehensive overview of multi-label classification, refer to the survey articles [28] and [29].

Problem Formulation

To formalize the MLC problem, let X denote an instance space and L = {l1, ..., lm} represent a finite set of m class labels. An instance x ∈ X is (non-deterministically) associated with a subset of class labels L ∈ L. This subset L is often referred to as the set of relevant labels. The concept of a set of relevant labels L can be represented by a binary vector y = (y1, ..., ym), where yi = 1 if li ∈ L and yi = 0 otherwise. The set of all possible label combinations is denoted as Y = {0, 1}^m.

Formally, a multi-label classifier h is a mapping h: X → Y. Given an input instance x ∈ X, the classifier produces a prediction in the form of a vector:
h(x) = (h1(x), h2(x), ..., hm(x))

The task of training a multi-label classifier from data can be expressed as follows: Given a finite set of observations Dtrain = {(xi, yi)}^N_i=1, where each observation xi ∈ X is associated with a label vector yi ∈ Y, the goal is to learn a classifier h: X → Y that generalizes beyond the provided observations by minimizing a specific loss function.

Loss Functions

A variety of loss functions have been proposed for multi-label classification, many of which are adaptations or variations of losses commonly used for single-label classification. These loss functions can generally be categorized into three main types: instance-wise, label-wise, and overall label matrix (flattened to a single vector), also known as micro-averaging. While instance-wise loss functions compute a loss for each test instance and then aggregate (average) over instances, label-wise loss functions compute a (binary classification) loss for each label and then aggregate the respective values across labels. To elaborate further, let Dtest = {(xi, yi)}^S_i=1 be a test set of size S and H = (h(x1), ..., h(xS)) ∈ Y^S. A loss function can be defined as L: Y^S → {0, 1}. Below, we outline three different ways to extend the F-measure to multi-label classification using instance-wise, macro-averaging, and micro-averaging loss functions commonly employed in the literature.

Since the number of relevant labels is typically small (resulting in a sparse label matrix), the F-measure (which is a measure of precision and recall) has been adapted to the MLC context in various ways. One approach involves computing the F-measure for the predicted label vector of each instance in the test set, and then aggregating across instances; this is known as the instance-wise F-measure:
FI(Ytest, H) = (1 / S) * Σ^S_i=1 [2 * Σ^m_j=1 y_i,j * h_j(xi) / (Σ^m_j=1 (y_i,j + h_j(xi)))]

Similarly, it can be defined in a label-wise manner as the macro-averaging F-measure:
FL(Ytest, H) = (1 / m) * Σ^m_j=1 [2 * Σ^S_i=1 y_i,j * h_j(xi) / (Σ^S_i=1 (y_i,j + h_j(xi)))]

Finally, a third variation applies the F-measure through micro-averaging:
Fm(Ytest, H) = (1 / m) * (1 / S) * Σ^m_j=1 [2 * Σ^S_i=1 y_i,j * h_j(xi) / (Σ^S_i=1 (y_i,j + h_j(xi)))]

The F-measure, being the harmonic mean of precision and recall, demands both a high true positive rate and a high true negative rate for optimal performance. Unlike other commonly used MLC loss functions such as Hamming loss, the F-measure addresses the class imbalance problem and avoids a strong bias toward negative predictions. However, the specific variation used affects how mistakes in predictions are considered, making certain classifiers more suitable for one version over another.

THE MULTI-LABEL SEARCH SPACE

Building on single-label classification algorithms as a starting point, multi-label classifiers have been developed in two primary ways. First, the multi-label problem can be transformed into one or more single-label problems that can be addressed using existing algorithms. Second, existing learning algorithms can be adapted to handle multi-label classification [30]. In the latter case, the algorithm is extended to support multiple labels within its framework. A common example is the extension of decision tree learning from standard classification to multi-label classification [31].

Design of Multi-Label Classifiers

On one hand, the design of adapted learners, such as neural networks with multiple output units (one for each label), multi-target trees, or k-nearest neighbor learners, remains consistent with previous approaches and doesn't impose any specific challenges due to the multi-label classification context.

On the other hand, transformation methods typically reduce the original multi-label classification problem to a set of binary or multi-class classification problems, which can then be addressed using established techniques such as random forests, SVMs, logistic regression, etc. For instance, Binary Relevance (BR) transforms the problem into a set of binary classification problems, one for each label [32]. These binary problems involve predicting the presence of the corresponding label independently of the other labels. While BR might seem like a straightforward and effective solution for multi-label classification, it is often criticized for not capturing dependencies and statistical relationships between class labels. Indeed, leveraging such relationships to improve predictive performance is a primary motivation for many multi-label learning algorithms.

For instance, consider the scenario where the class label "Yacht" may be positively associated with the class label "Ocean." If the former is positive (indicating the presence of a yacht in an image), then the latter is likely to be positive as well. Therefore, while predictions like (0,0), (0,1), and (1,1) may seem plausible, a multi-label classifier should be more hesitant to predict (1,0). As an example of a slightly more complex transformation method, let's discuss classifier chains [33]. In the classifier chain (CC) approach, predictive models are trained sequentially, with each model in the chain predicting a label conditionally on the preceding labels. The chain arranges labels in a specific order. For instance, starting with a model ^ys(1) = h1(x), CC trains a second model ^ys(2) = h2(x, ^ys(1)), a third model ^ys(3) = h3(x, ^ys(1), ^ys(2)), and so on.

In the CC example above, the prediction for the label "Ocean" depends on the predicted presence of "Yacht" as well as on the event data x. Hence, label dependencies can be partially captured. However, a theoretical limitation of CC is that the label information used as additional features by classifiers is available only during training, not at prediction time. Therefore, since the true label information ys(1) is unavailable, h2 will predict ^ys(2) = h2(x, ^ys(1)), where ^ys(1) is the estimate from h1. Similarly, h3 will predict ^ys(3) = h3(x, ^ys(1), ^ys(2)), and so on. This introduces a form of inherent noise and potentially leads to error propagation along the chain [34].

Transformation methods can be considered meta-learning techniques, initiated with a base learner such as a binary classifier in BR or a classifier chain in CC. As previously mentioned, constructing a multi-label classification algorithm can thus become quite intricate (refer to Fig. 1), requiring the user or ML practitioner to make numerous decisions, such as choosing from over 70 algorithms and configuring up to 25

hyperparameters simultaneously. Additionally, empirical studies suggest that for optimizing the performance of transformation methods, the choice of the base learner is indeed crucial [35], [36].

Beyond the selection and configuration of base learners, one could also consider specifying the meta-learner itself, thereby further increasing the number of hyperparameters. A straightforward example is the stages in classifier chains, which are known to impact performance positively [37]. Alternatively, instead of selecting a single base learner for each label, an individual base learner could be selected and tuned for each label separately. Such a label-wise configuration may indeed be beneficial, as demonstrated in the case of BR [36]. However, this would significantly increase the complexity of the search space by several magnitudes. Consequently, we focus on the simpler task of recursively selecting the base learners and tuning their hyperparameters.

Representation of Search Space

The search space for multi-label classification, as depicted in Fig. 2, encompasses five different categories of algorithms: meta and base algorithms for multi-label classification, as well as meta and base algorithms for single-label classification. Additionally, segments that can be applied to an SVM classifier (illustrated in the figure by the Sequential Minimal Optimization algorithm; SMO) are included. Specifically, the search space contains the following algorithms:

MEKA Meta Algorithms: Meta Multi-Label Binary Relevance (Meta MBR), SubsetMapper (SM), Random Subspace Multi-Label (RSS), MLC Bayesian Model and Search Dependency (MLCBMD), Bagging Multi-Label (BML), Bagging Multi-Label with Duplicates (BMLdup), Ensemble Multi-Label (EML), Expectation Maximization (EM), Chain Model (CM)

MEKA Base Algorithms: Binary Relevance (BR), Binary Relevance with Quantization (BRq), Classifier Chains (CC), Classifier Chains with Quantization (CCq), Binary Classifier Chains (BCC), Probabilistic Classifier Chains (PCC), Multi-label Classifier Chains (MCC), Probabilistic Multi-label Classifier Chains (PMCC), Classifier Tree (CT), Chain of Decision Nodes (CDN), Chain of Decision Tables (CDT), Filtered Workflows (FW), Random Trees (RT), Label Combination (LC), Pruned Sets (PS), Pruned Sets with Threshold (PSt), Random k-label sets (RAkEL), Random k-label sets with Dynamic (RAkELd), Back-Propagation Neural Network (BPNN), Hierarchical Adaptive Stacking Ensemble Learning (HASEL), Majority Labelsets (MLS), Dynamic Back-Propagation Neural Network (DBPNN)

WEKA Meta Algorithms: AdaBoostM1 (ABM1), Vote (V), Stacking (S), Local Weighted Learning (LWL), Random SubSpace (RSS), Bagging (B), Random Committee (RC), Attribute Selected Classifier (ASC), Additive Regression (AR), Classification via Regression (CVR), Logit Boost (LB), Multi-Class Classifier (MCC)

WEKA Base Algorithms: J48, M5P, M5Rules (M5R), Voted Perceptron (VP), Simple Linear Regression (SLR), Simple Logistic (SL), Naive Bayes Multinomial (NBM), LMT, Decision Stump (DS), Random Forest (RF), Random Tree (RT), Decision Table (DT), JRip (JR), OneR (OR), PART, ZeroR (ZR), IBk, KStar (KS), Multilayer Perceptron (MP), Sequential Minimal Optimization (SMO), Logistic (L), Naive Bayes (NB), Bayes Network (BN), REPTree (REPT)

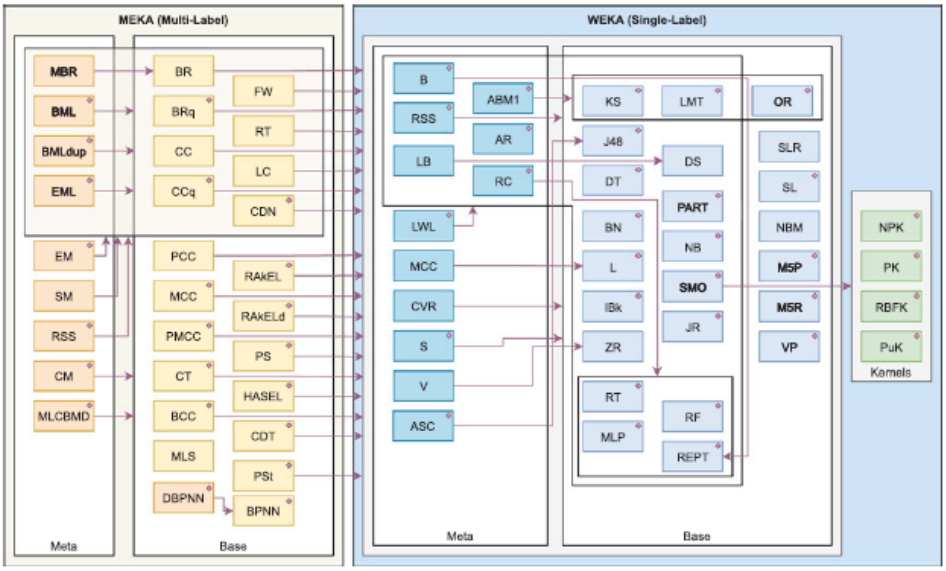


Fig. 2. Overview of the search space showing classification algorithms from MEKA for multi-label and WEKA for single-label classification. An arc pointing to a box frame means an arc to every classifier contained in this frame. Purple diamonds indicate whether the respective classifier exposes hyper-parameters to be tuned.

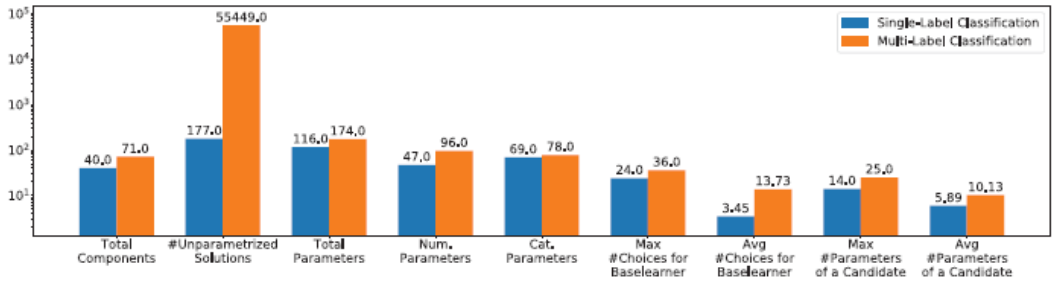


Fig. 3. Comparison of statistics regarding characteristics of the multi-label classification search space and the subsumed search space for single-label classification. Note that the there is a substantial increase in the number of unparameterized solution candidates, i.e., the number of distinct classifier configurations ignoring hyper-parameter configuration. Moreover, the maximum number of hyper-parameters that are optimized simultaneously for a single configuration is almost double the amount.

The diagram in Figure 2 illustrates the search space layout, indicating that most algorithms typically require the specification of a base algorithm. This base algorithm can be of the same type as the specific algorithm or of the next type in the given list. In the figure, this requirement is represented by an arc pointing to either a particular algorithm or a box containing multiple algorithms. The latter case simplifies the visualization by drawing a single arc from the respective algorithm to each algorithm within the box. Algorithms with adjustable hyperparameters are denoted by purple diamonds.

Figure 2 provides a concise representation of the entire search space. At first glance, it seems that the extension from single-label to multi-label classification merely doubles the complexity, as the available algorithms only increase by a factor of two. However, the true complexity arises from the need to recursively configure base algorithms. For instance, a base algorithm of one method might require a base algorithm of another method to be configured in turn. Therefore, the shortcut arcs pointing from an algorithm to a box represent the majority of the complexity.

Figure 3 presents a comparison of various statistics regarding the search spaces for single-label and multi-label classification, both for individual and multi-label cases. Although the number of algorithms (components) and the quantity of specified hyperparameters in the search space increase only slightly, the total size of the search space dramatically escalates from 177 unparameterized configuration candidates to over 55,000. However, it's not just the sheer number of distinct algorithm choices that magnifies the challenges of AutoML (Automated Machine Learning) tasks, but also the potential maximum number of parameters that a single configuration candidate can involve. In the extreme case, a single configuration candidate may encompass up to 25 hyperparameters, in contrast to the 14 hyperparameters in the case of single-label classification.

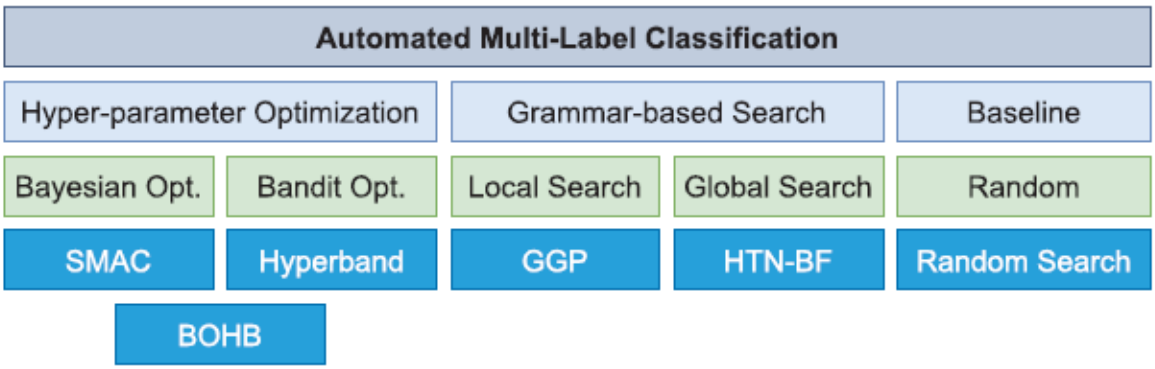


Fig. 4. Ontology showing the considered optimization techniques proposed for automating machine learning.

In comparison to single-label classification, the average number of hyperparameters also increases from 5.89 to 10.13.

Considering the above, the multi-label classification search space contains significantly more configuration candidates when compared to single-label classification. Additionally, due to the increased number of hyperparameters that need to be optimized for a single candidate, the hyperparameter optimization complexity becomes even more intricate.

Improvement Techniques

The field of AutoML for standard classification and regression has introduced several strategies for exploring the vast space of configuration candidates. However, for multi-label classification, only a few of these methods have been considered so far. These include genetic algorithms, language-based genetic programming, hierarchical task network planning, and a classifier-specific approach based on neural architecture search. This discussion focuses on techniques for traditional AutoML that address the challenge of combined algorithm selection and hyperparameter optimization.

The following sections provide a brief overview of various optimization approaches, covering both hyperparameter optimization and language-based search. Additionally, we explore how these methods can be adapted to automate multi-label classification and discuss whether this adaptation has been proactively explored in the existing literature. For a more comprehensive understanding of these individual approaches, interested readers can refer to research papers on traditional AutoML.

Figure 4 provides an overview of the considered optimization techniques, and Table 1 discusses their application in both standard AutoML and AutoML for multi-label classification.

Reduction to Hyperparameter Optimization

A straightforward approach to tackle the AutoML problem is to reduce it to the problem of instance-specific hyperparameter optimization. In this approach, a hyperparameter space L is defined over various hyperparameters, a dataset space D is given, and a quality measure u: $L \times D \rightarrow \mathbb{R}$ quantifies how well a specific hyperparameter configuration performs on a given dataset. Given a dataset D from D, the objective is to find the best hyperparameter configuration $\Theta\_D \in L$, as defined by:

$\Theta\_D = \text{argmax}\_{\Theta \in L} u(\Theta, D)$.

In the context of AutoML, the quality measure u often corresponds to a scoring or loss function such as the F-measure or Hamming loss.

This reduction from the AutoML problem to hyperparameter optimization is achieved by encoding the choice of each algorithm and its components through binary parameters. These binary parameters indicate the selection of a specific algorithm or component. These binary parameters, along with the original hyperparameters of each possible algorithm and component, form a hyperparameter vector. Many tools also require a set of constraints that specify which hyperparameters are associated with which algorithms and components. This information can be utilized to organize the vector into trees where only relevant hyperparameters are considered.

Bayesian Optimization

Bayesian Optimization (BO) is a prominent technique in hyperparameter optimization and serves as the foundation for early approaches to AutoML. BO involves an iterative process of building/updating a surrogate model based on observations of the quality measure u. This surrogate model is then utilized, along with an acquisition function, to select the next configuration to evaluate. This process is repeated until a stopping criterion is met, such as wall-clock time or a specified number of evaluations of u.

For AutoML tasks, Tree Parzen Estimators or Random Forests are often used as surrogate models. Gaussian Processes are also a common choice, but they might not scale well with the high-dimensional search space of the AutoML problem. The choice of surrogate model depends on various factors, including the search space's structure, noise in the quality measure, and others.

The surrogate model ^u is combined with an acquisition function to determine which hyperparameter configuration should be evaluated next with u. To be efficient, this choice should provide as much informative exploration of the search space as possible. Acquisition functions balance exploration and exploitation to guide the search toward promising candidates, considering both the predicted values and uncertainty estimates. While various acquisition functions exist, including entropy search, information gain, and...
anticipated improvement (EI) [57], [58]. Among various acquisition functions, we focus on EI due to its common usage in the AutoML field. The main idea behind EI is to explore candidates that offer improvements in comparison to the best configuration observed so far. Formally, EI can be expressed in terms of a hyperparameter configuration Θ_D and the best hyperparameter configuration Θ_best observed up to that point:

$$EI(\Theta\_D) = E[\max(u(\Theta\_best, D) - u(\Theta\_D, D) , 0)].$$

Note that the expectation is taken because u(Θ_best, D) is a random variable with an unknown outcome when calculating EI(Θ_D). Using this definition, the EI acquisition function selects the configuration that maximizes EI.

Bayesian Optimization (BO) has been employed as an optimization technique in several AutoML tools for standard classification and regression tasks. However, to the best of our knowledge, BO has not been utilized for AutoMLC (AutoML for multi-label classification) tasks before.

Hyperband

Another category of methods for hyperparameter optimization revolves around formalizing the problem as a multi-armed bandit (MAB) problem, which is a sequential stochastic dynamic problem. In the MAB framework, an agent selects one "arm" at a time from a set of options and receives a reward signal that reflects the quality of that choice. The goal is to maximize a performance metric, such as cumulative regret, which measures the difference between the total rewards that could have been obtained by always choosing the best arm and the rewards obtained during the exploration phase.

Hyperparameter optimization can be framed as an MAB problem by considering each possible hyperparameter configuration (or AI pipeline in the context of AutoML) as an arm. The rewards obtained when pulling an arm correspond to the evaluation of the corresponding configurations within a given budget, such as time.

A simple approach to identifying a promising arm (configuration) in this setting is to allocate a total budget of B evenly to each of the K arms, meaning each arm is allocated a budget of B/K. However, this approach can allocate a significant portion of the budget to non-optimal arms.

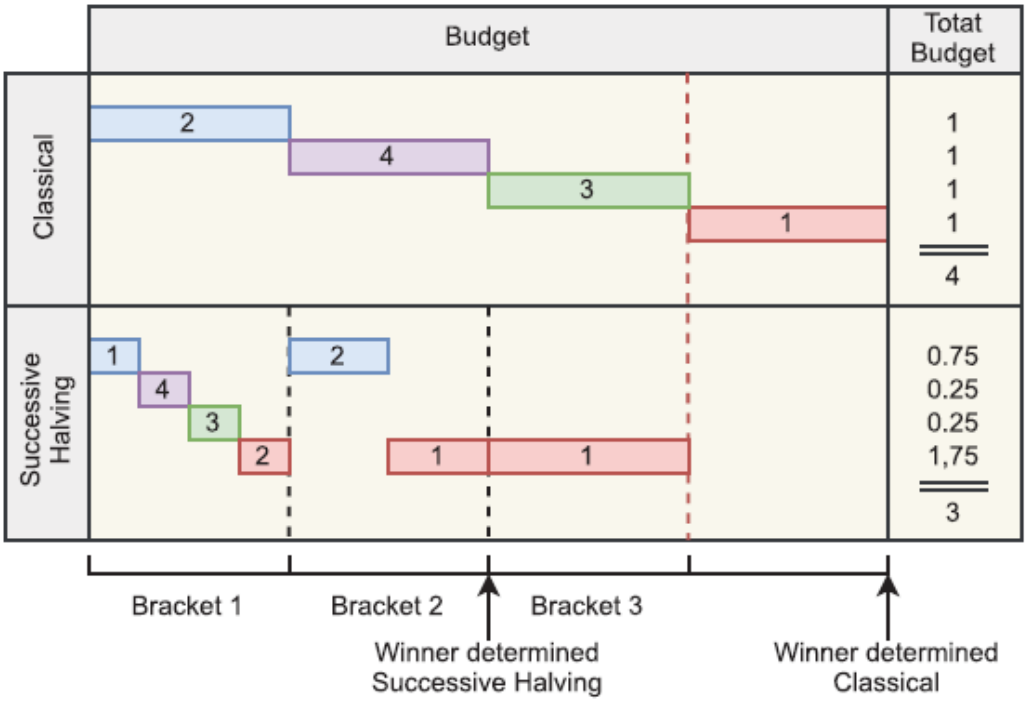


Fig. 5. Comparison of the classical approach (top) and successive halving (SH) (bottom) to identify the best performing configuration out of 4 candidates. Numbers within colored rectangles indicate the rank of a configuration. Within each bracket, the current set of configurations is evaluated on a portion of the totally assignable budget and after each bracket the worse half drops out. After bracket 2, SH already identified the winner configuration (red). The right column summarizes the total budget spent per configuration.

Progressive halving, a technique to mitigate this issue, divides the time steps into N phases, allocates the budget evenly across the phases, and reduces the number of arms to be pulled at the end of each phase. Based on the rewards obtained, the top half of the arms are retained and advanced to the next phase. This process results in selecting a single best arm after approximately log2(K) - 1 phase.

The success of this approach depends on the assumption that discarding arms based on low-budget evaluations effectively eliminates bad configurations but does not eliminate potentially good configurations that may perform better on larger budgets. Hyperband extends this approach by iteratively invoking the progressive halving procedure with different sizes of initial arm sets (K) to finally return the best configuration found in this process.

Hyperband has been applied to AutoML for classification tasks, yet to our knowledge, its application to AutoMLC tasks has not been extensively explored.
has been used to tackle the AutoMLC problem so far, which will be done in this work for the first time.

BOHB, or Bayesian Optimisation and Hyperband

An obvious weakness of Hyperband is its random sampling of configurations at the beginning of each iteration, which is addressed by an approach combining the idea of Hyperband with Bayesian Optimization, called BOHB [12]. More specifically, it replaces the random sampling procedure of Hyperband with BO-based sampling. TPE models are constructed for different budgets B based on observed configuration performances. In each iteration, the majority of configurations are iteratively sampled using these models,

while the remaining configurations are sampled at random for reasons of convergence. As one is eventually interested in the performance of a configuration evaluated on the maximum budget, BOHB always queries the model associated with the largest budget available.

BOHB can be instantiated to solve AutoML problems in the same way as SMAC and Hyperband, namely by reducing the AutoML problem to a problem of hyperparameter optimization. Once again, to the best of our knowledge, this work is the first one to apply BOHB to tackling the AutoMLC problem, although it has been used in the context of AutoML for classification before [46].

Genetic Algorithms

Genetic algorithms (GAs) are quite popular and frequently used as a tool for black-box optimization. The basic idea is to maintain a population of candidate solutions and to refine these candidates iteratively by applying randomized operators (e.g., mutation and crossover inspired by biological evolution) to maximize a given fitness function. Each candidate solution is encoded by a fixed-size binary or real-valued vector of so-called genes, also referred to as a genetic representation.

Applying genetic algorithms to the problem of AutoML thus requires a proper genetic representation, which can be obtained by encoding every hyperparameter by a single gene (using integers for categorical or integer hyperparameters, and reals for any other numeric hyperparameters). However, such an encoding is difficult to handle for standard GAs, because most of the genes are "inactive" in the sense of not belonging to the currently selected algorithm(s). This also hinders the exchange of parts of the current solution. Alternatively, messy GAs can be used but the mutual exchange of individuals remains difficult [60]. These issues may explain why standard GAs have not been considered very much in the AutoML literature.

To the best of our knowledge, only a simple GA called GAAutoMLC has been used for the problem of automating multi-label classification [4]. However, only a very small selection of algorithms has been considered in this work, which is mostly due to the chosen genetic representation. To compress the genetic representation, the genes for hyperparameters were shared among different algorithms. More specifically, the number of genes for hyperparameters was chosen according to the method exposing the highest number of hyperparameters. The values encoded in the genes are then interpreted concerning the selected method, and the remaining information is ignored.

Later on, a detailed ablation study [5] revealed that a grammar-based genetic programming approach can outperform such a simple genetic algorithm for the same search space. These findings can be attributed to the more suitable genetic representation. Furthermore, the genetic programming approach is even more flexible and allows for a larger portfolio of algorithms. Because of these results, we exclude GA-AutoMLC from our study.

Grammar-Based Search

Grammar-based search approaches have emerged as another line of research for designing AutoML tools (cf. [2], [5], [16], [17]). In contrast to reduction techniques representing the optimization space by a (flat) vector of hyperparameters combined with additional conditions, grammar-based formalisms allow for modeling the hierarchical structure of machine learning pipelines and classifiers more naturally. This hierarchical structure is particularly prominent in the case of multi-label classifiers, which usually employ single-label classifiers as base learners. Yet, it is also inherent to single-label classifiers, as shown by examples like a bagged ensemble of support vector machines, which in turn require a kernel function to be specified.

In the following, we describe two representatives of grammar-based approaches, first an evolutionary approach for evolving tree-shaped structures called grammar-based genetic programming (Section 5.2.1), and a technique from the field of AI planning dubbed hierarchical task network (HTN) planning (Section 5.2.2).

Grammar-Based Genetic Programming

Just like genetic algorithms, grammar-based genetic programming (GGP) algorithms belong to the family of evolutionary algorithms. Yet, in contrast to standard GAs, GGPs make use of grammar to describe the correct syntax of individuals. This syntax is used to generate an initial population of valid individuals and is also provided to genetic operators that are specifically crafted for GGP. Another difference to standard GAs is the genetic representation. Instead of representing individuals in terms of fixed-length vectors of genes, they are described in the form of trees describing derivations of the grammar, which makes the entire approach more flexible concerning more complex structures and larger portfolios of algorithms. Furthermore, the size of such a tree does not necessarily need to be fixed or bounded. For a more comprehensive description of grammar-based genetic programming, we refer the interested reader to [48].

Due to their appealing properties, GGPs have been used to tackle the AutoML problem in various ways [2], [15], [16]. All these approaches have in common that the search space is described by context-free grammar, hierarchically structuring the space, and having algorithm names and hyperparameter values as terminals. Prominent examples of applying GGP to AutoML for single-label classification or regression are TPOT [2], RECIPE [16], and GAMA [15].

Even more interestingly, GGP provides the basis of an AutoML tool for multi-label classification called Auto-MEKAGGP [5]. However, from a methodological point of view, nothing has been implemented in Auto-MEKAGGP that could be considered as specific for MLC.

HTN Planning and Best-First Search

The basic idea of Hierarchical Task Network planning [49], a technique from the field of automated planning, is to hierarchically structure the space of possible solutions based on logical language and specific operators. To this end, HTN planning describes the search space in terms of complex tasks, primitive tasks, and methods that specify how complex tasks are refined again into complex tasks or primitive tasks. Simple tasks are seen as atomic and often represent something that can be "executed," whereas larger jobs are not.
can be viewed as a composition of simpler tasks and thus need to be decomposed recursively. Intuitively, HTN planning mimics the way a machine learning expert approaches a multi-label classification task, decomposing it into smaller and simpler tasks such as selecting classifiers, and base learners, and eventually tuning the hyper-parameters [61]. A "ground" solution, also referred to as a plan, is obtained once all complex tasks are fully refined and only primitive tasks are left.

The idea is similar to derivations in context-free grammars, where complex tasks are non-terminal symbols and primitive tasks are terminals. In contrast to context-free grammars, primitive tasks do not only work in a generative manner but can also modify a (logical) state, a concept featured in HTN.

HTN problems are typically solved by a reduction to a graph search problem that can be approached with standard algorithms, e.g., depth-first search. A typical translation of the HTN problem into a graph is to select the first complex task of a list and to define one successor for each applicable method that can be used to refine the task; this is called forward decomposition [49]. As a consequence, the shape of the resulting search graph is a tree. While leaf nodes of the tree represent plans, an inner node represents a prefix of a plan. Hence, the root node is an empty plan.

Data mining and machine learning have been automated using HTN planning, which maps simple tasks to several algorithm configurations and hyper-parameter settings and builds an abstract structure over these choices using complex tasks [17], [62]. The graph in Fig. 6 sketches an excerpt from such a search graph for the automated multi-label classification problem. In [17], a best-first search is applied to the resulting search graph. As a heuristic, the proposed best-first search assigns scores to inner nodes by randomly drawing several path completions to leaf nodes to obtain fully specified pipelines that can be evaluated as usual, e.g., applying cross-validation. The score of the inner node is determined by the best completion to bound the true optimum that can be found in the respective sub-tree (assuming the objective function to be

minimized). By configuring the number of random completions drawn for assessing the quality of an inner node in terms of an approximate score, we can trade off the degree of exploitation and the degree of exploration of the search.

In analogy to AutoML for single-label classification, we can instantiate HTN planning combined with a best-first search for the MLC setting. Extending the search space, tuning the search and the evaluation strategy to the specifics of the MLC search space, extensions of [17] have been proposed in [6], [7].

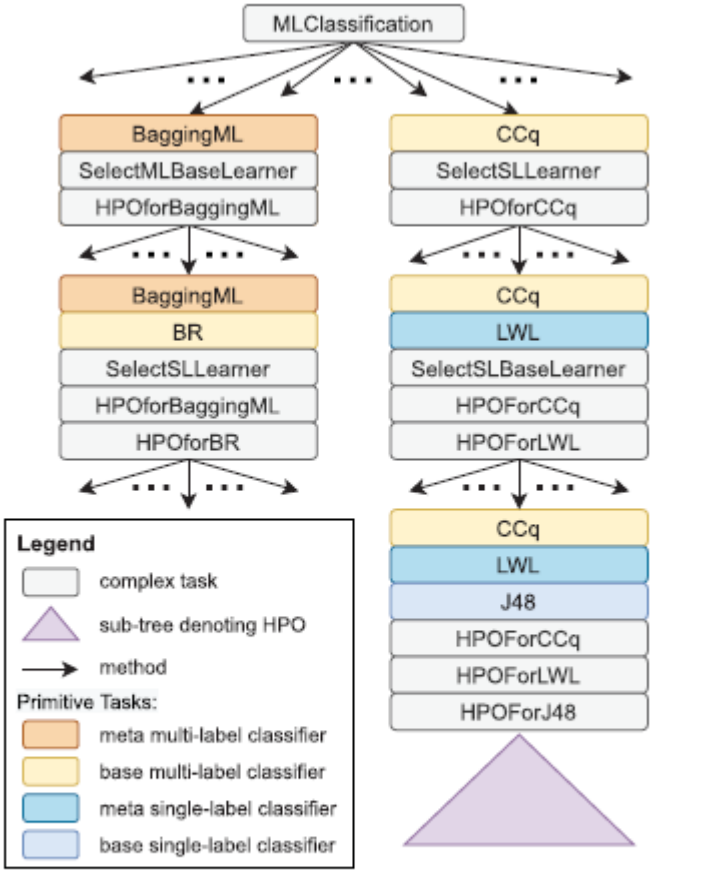


Fig. 6. Sketch of a search tree induced via HTN planning for automated multi-label classification. Primitive tasks are additionally distinguished by color according to their role within a multi-label classifier. Note that the indicated refinements are of exemplary character. Further options as well as sub-trees are only hinted at.

AUTOMLC BENCHMARK

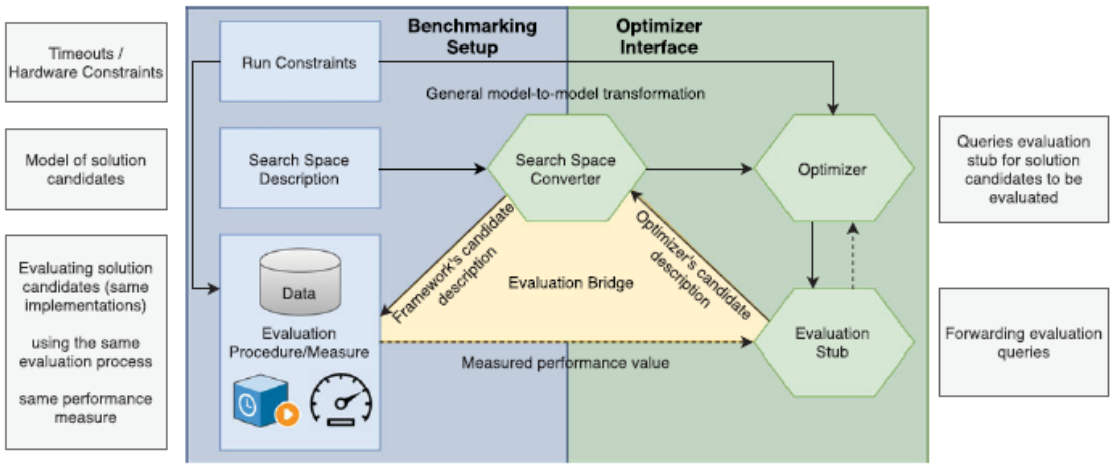
In empirical AutoML studies, multiple components are often changed at a time without carrying out ablation studies. For example, different optimizers with different search spaces are compared, sometimes even with different candidate evaluation methods. One quite frequent example is to propose a new optimization technique together with a different search space, while not changing the search space for the baseline methods considered for comparison. In such cases, the results of the studies are difficult to interpret. Regardless of whether the newly proposed method is superior, competitive, or inferior to the baselines, it is not clear whether this finding should be attributed to the change of the search space or the optimization method.

The general issue has already been acknowledged in the literature [26], where AutoML tools are evaluated within consistent hardware and timeout environments as well as optimized for the same target loss function. However, the compared AutoML methods are considered a black box, and the design of the search space is considered a part thereof. As a consequence, the latter differs from approach to approach. Therefore, it is unknown whether performance differences between AutoML methods can be attributed to the optimization techniques or the search space definition...

Note that the meaning of the last option enormously affects the issue's intricacy. Indeed, even little changes might rearrange the issue a ton or, going against the norm, make it a lot harder. Expanding the pursuit space by a solitary ensembling calculation, containing an erratic rundown of base students, may increment the size of the inquiry space from limited to boundless. Similarly, eliminating a solitary calculation from the pursuit space can prompt a huge disentanglement of the enhancement task, obviously, additionally suggesting that the best calculation for a specific errand is at this point not accessible. The topic of which enhancer might perform best in which setting is accordingly still an open inquiry.

In [39], the creators endeavor to respond to the inquiry by considering different streamlining agents for a similar pursuit space and indeed, even a similar inward assessment system. Be that as it may, the approach taken in [39] is restricted in a few respects:

- It is confined to streamlining agents accessible in Python, while the benchmark proposed here highlights cross-stage capacities.
- The inquiry space just thinks about a level arrangement of calculations to be picked, i.e., the streamlining agents are permitted to pick out of 13 unique classifiers and enact hyper-boundaries to be advanced by this decision. Even though there is an idea of boundaries being designed in various leveled manners on account of SVMs, the pursuit space definition has no idea for refining base students, e.g., of troupes.
- Besides, the runtime of the analyzers is by implication restricted using the number of assessments, which in turn is limited by a limit of 10 minutes for every assessment. Nonetheless, the limit on the number of assessments pointlessly punishes improvement systems that like to inspect applicants widely with an extremely short runtime. While the quantity of assessments is a legitimate means to guarantee equivalence in the domain of black-box capability improvement, the arrangement competitors in AutoML are every so often excessively assorted. In our trial assessment, we give experimental proof for the high difference in the assessment times for various arrangement applicants.



A typical benchmark, all things considered, is attractive since AutoML studies are costly concerning time and computational assets. With each recently proposed technique, the relating concentrates over and over executing different other techniques and baselines. This is important, first because exploratory arrangements, i.e., time requirements, allowed equipment assets, target capabilities, and datasets, are changed, and second, there is no normal benchmark guaranteeing the similarity of test results. Besides, normal benchmarks are valuable to smooth out research, guaranteeing equivalence of the assessments of new strategies to previously existing ones and in a perfect world authorize detachment of worries.

As the line of examination on AutoML for multi-mark orders is still in its early stages, we propose a brought-together system for benchmarking strategies and augmentations for AutoML in the issue space of MLC to guarantee likeness across various streamlining agents (across various stages) and to keep away from superfluous re-assessments of currently distributed techniques in what's in store. Besides, it shapes a reason for future examination on both refining the MLC search space and refining enhancement procedures to adapt to the more perplexing pursuit space. An outline of the system is portrayed in Fig. 7. The critical elements of the system are shared run requirements, a model-to-model change for search space depictions, and a shared (cross-stage) execution assessment technique. The structure is coordinated into two sections. To start with, the benchmarking arrangement nt (a blue piece of the figure) contains the specialized particulars, i.e., the

worldwide run limitations, search space portrayal, and the presentation assessment technique. Second, the connection point of the enhancer(a green piece of the figure), which is answerable for deciphering the arrangement data into an organization reasonable by the particular enhancer and giving a stub that can be called to question the exhibition assessment methodology.

By the way, aside from its substantial launch, nothing of the system is task-explicit (in regards to multi-mark order). In this manner, the benchmark structure could on a basic level...
be utilized to accomplish the likeness of various enhancement methods for some other AutoML tasks, as well. For instance, the benchmark could be utilized to research the abilities of various streamlining agents to look for standard order machine learning pipelines including (different) pre-handling calculations. In any case, as we center around computerized AI for multi-mark order here, this sort of examination is out of the extent of this paper and left for future work.

Benchmarking Arrangement

The benchmarking except for every one of the boundaries significant to an AutoML benchmark, except the enhancer that is utilized to investigate the space of potential arrangement applicants. All the more definitively, the benchmarking arrangement characterizes the whole trial arrangement, remembering imperatives for the run characterizing the level of parallelization and the breaks. The system takes into account characterizing breaks for both the whole AutoML process and the assessment of a solitary competitor freely.

The centerpiece of the benchmarking arrangement is the pursuit space portrayal, which indicates every likely arrangement that might be tried by the calculations. Our benchmarking climate accompanies its own (JSON-based) language to depict an inquiry space, which is not difficult to peruse and alter, and which takes into account displaying search spaces keeping up with various leveled structures. In this model, each calculation is viewed as a product part with given and required interfaces. The points of interaction are simply names and have no practical determination. For instance, a double significance student gives an interface MultiLabelClassifier and requires a connection point BaseLearner, which thus can be given, for the model, by an SVM. For each part, one can characterize a bunch of boundaries with their spaces and conditions among them, e.g., "if worth of x ¼ 3, the area of conceivable values for y becomes [0,1]".

To make this search space portrayal justifiable to the different enhancers, a hunt space converter should be composed for each enhancer to be viewed as in the benchmark. Every enhancement apparatus acknowledges some type of search space depiction, however, the substantial organizations unequivocally fluctuate among the different streamlining agents. For this paper, we carried out such converters for the thought about analyzers to arrange the right contributions for these analyzers.

Second, the run limitations contain breaks and computational assets. All the more unequivocally, one characterizes the in general break for the pursuit cycle, the break for single assessments, and what's more, requirements on memory and central processor use. Unnecessary to say, the substantial decision of breaks can be pretty much gainful for an enhancer. In any case, since similar imperatives apply to all analyzers, this effect ought not to be excessively enormous.

The third and last piece of the benchmarking arrangement concerns the assessment methodology, and consequently, additionally, the presentation measure, which fills in as the objective misfortune to be streamlined. Sharing this piece of the benchmarking arrangement across the unique analyzers guarantees that there is no benefit regarding assessment speed, which could mutilate the general exhibition. Ordinarily, to guarantee this sort of decency, the quantity of permitted assessments is restricted. Our methodology ensures something very similar level of reasonableness likewise for whatever settings.

As well as guaranteeing reasonableness and likeness, a benefit of decoupling the benchmarking arrangement from the analyzer is to create meta-learning approaches free of a substantial enhancer. For instance, a substitute for surveying the exhibition of an answer up-and-comer can be utilized by substituting the assessment methodology. Along these lines, the substitute can be tried in blend with any streamlining agent carried out inside the structure. Moreover, the system takes into consideration task-explicit

transformations of the inquiry space, e.g., by expecting which calculations will probably be excessively tedious for a picked assessment break and barring these calculations right all along. Just the diminished pursuit space is then given to the analyzer.

Analyzer Point of Interaction

The streamlining agent point of interaction is liable for associating an enhancer to the remainder of the benchmarking framework. More explicitly, this mostly concerns setting the hyper-boundaries of the analyzer and changing over the hunt space portrayal from the system's configuration into the particular arrangement of the streamlining agent. Notwithstanding the actual enhancer, the streamlining agent interface contains an assessment stub spanning between the streamlining agent and the assessment methodology that is important for the benchmarking arrangement. The assessment stub takes assessment demands from the streamlining agent and advances them to the assessment system. If the assessment of the particular arrangement applicant is fruitful, the assessment stub will take care of the outcome esteem back to the enhancer. The enhancer and the assessment stub are skeptical about the misfortune capability used to compute the bring esteem back. Notwithstanding, on account of an ineffective assessment, the assessment method gives criticism in regards to the reason and separates between crashed assessments and those with a break.

The third part of the enhancer connection point is a planning from the structure's inquiry space portrayal design into the particular organization of the analyzer. Via naturally creating search space depictions, just the model-to-model change should be right, which rearranges support and takes into account thinking about various pursuit spaces in a predictable manner across different enhancers.

Trial Assessment

The exploratory assessment examines the presentation of the streamlining procedures for AutoML presented previously in the issue area of multi-name arrangement. We research the versatility of the streamlining agents alone concerning the expanded search space intricacy, coming about because of the more profound progressive designs of multi-name classifiers and the more extensive competitor assessments. To this end, we apply the benchmarking structure as proposed in Segment 6, ensuring that all enhancers are working on a similar hunt space and stick to similar imperatives with regard to equipment assets and breaks.

Trial Arrangement

In our trial assessment, we do all tests in the proposed benchmarking system considering the following advancement techniques:

- Bayesian advancement (SMAC)
- Crook enhancement (Hyperband; HB)
- Bayesian Streamlining and Hyperband (BOHB)
- Syntax-based hereditary programming (GGP)
HTN Planning and Best-First Search (HTN-BF)

In addition, as a preliminary benchmark, we conducted a random search that sampled algorithm selections uniformly at random (including recursive choices for various algorithms) and subsequently selected hyperparameters for the chosen algorithms randomly from their respective hyperparameter domains.

All runs were executed on nodes equipped with 8 CPU cores (Intel Xeon E5-2670) and 32 GB of RAM, with an overall time limit of 24 hours and a limit of 30 minutes for evaluating a single classifier. For evaluating a solution candidate's performance, we employed 5 randomly generated train/validation splits with 70% training and 30% validation data from the "training" data provided for the AutoML run. Moreover, we used three different performance metrics as the target objectives: instance-wise F-measure (FI), label-wise F-measure (FL), and micro-averaged F-measure (Fm).

The best-first search was configured with the default settings proposed in [17], which involved testing 3 random path completions for assessing the quality of a node, resulting in a relatively greedy search behavior. As for SMAC, we used its parallelized version, with mostly the default configuration. Additionally, we enabled multi-fidelity optimization by allowing Hyperband and BOHB to determine how many train and validation splits are used for evaluating the performance of a solution candidate. For this purpose, they were configured to choose budgets b ranging from 1 to 5,000 (to also consider thorough exploration as the budget limits also determine the number of candidates that are explored), which was translated to db=1000 training and validation splits.

The syntax-based genetic programming approach was configured to operate with a population size of 15, following the default configuration of Auto-MEKAGGP. The probabilities for applying crossover and mutation for recombination of individuals were set to 0.9 and 0.1, respectively. Each new generation retains the best individual from the previous generation.

Unlike Auto-MEKAGGP, our implementation of syntax-based genetic programming does not perform any reshuffling of train and validation splits but solely utilizes the performance evaluation method provided by the benchmarking framework as a fitness measure. Additionally, the algorithm was used in an anytime setting, meaning it can return a solution as soon as the first successful candidate evaluation was completed and continues the evolution for as long as time is available. Train and test splits are determined via 10-fold cross-validation, resulting in 10 train and test splits for each dataset. A list of the datasets used for benchmarking along with some descriptive statistics is provided in Table 2. The descriptive statistics include the number of instances (#I), the number of labels (#L), the label-to-instance ratio (L2IR), the unique label combinations (ULC), and the average number of labels assigned to an example (also known as label cardinality).

In total, we conducted 720 runs for each method, except for the random search, which we executed only for 240 runs to reduce computational costs. As the random search does not make any decisions based on candidate solutions seen so far, we only needed one run for all three objective losses combined. Each of the methods was executed with 8 parallel workers. Adding up to a total of 3,840 experiments over 24 hours, the experimental assessment contains data equivalent to around 84 CPU years (¼ 3; 840 24h 8 cores¼ 737; 280 CPUh).

To define the search space, we considered the multi-label classifiers provided by MEKA [63], a multi-label classification extension of the well-known WEKA [64] machine learning library. Both libraries are implemented in Java, which is one of the reasons why our benchmarking framework is also implemented in Java. For the global schema of the search space, we used the AILibs2 format of the project HASCO and the detailed description of MEKA and WEKA provided in [65]. The source code for the benchmarking framework and the experiments is publicly available on GitHub [3].

Analysis of Hypothesis Performance

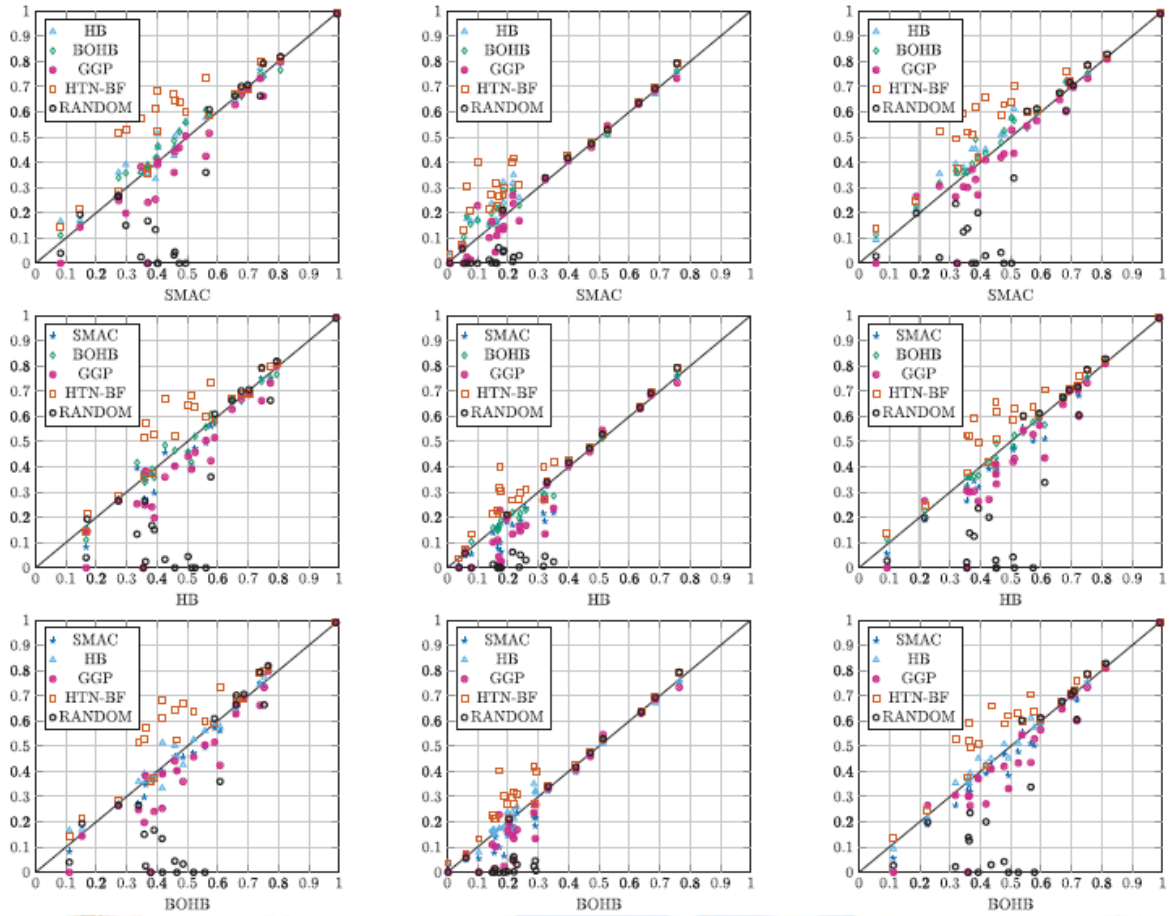
The test performances for each of the methods and datasets across 10 train and test splits and the three performance metrics (instance-wise, label-wise, and micro-averaged F-Measure) are presented in Table 3. Initially, it's evident that HTN-BF performs favorably in the majority of cases and tends to outperform the other methods on a wide range of datasets. To gain a better and more comprehensive understanding, we have also visualized the results in scatter plots (Fig. 8), where we compare the performance of one method against all others for each of the performance metrics. Each point in this plot represents the overall performance of one method against another compared method for one of the datasets. The performance of the one method is plotted on the x-axis, and that of the compared method is on the y-axis. The hypothesis performance of the considered method improves from left to right, and the performance of the compared methods trends upwards.
A tie in the hypothesis performance is noted whenever a point lies on the diagonal line. If a point lies below (above) the diagonal, it indicates that the considered method performs better (worse).

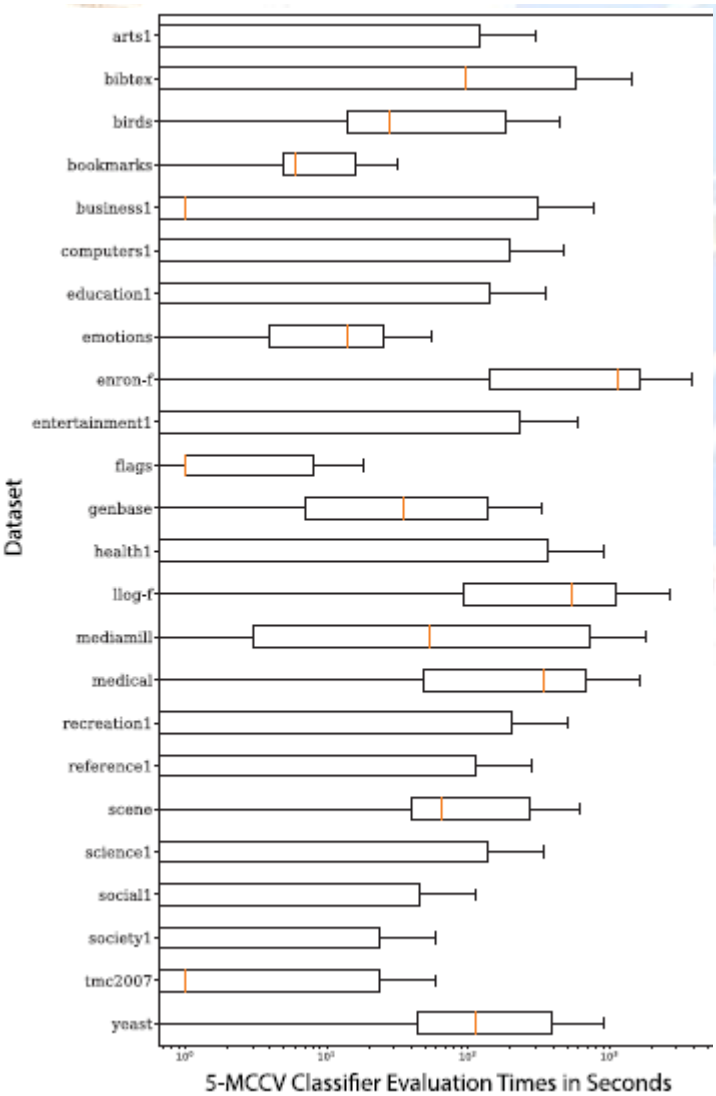
These plots clearly demonstrate that HTN-BF generally outperforms other methods and yields (in most cases, only slightly) inferior results on a few datasets only. The few instances where another algorithm exhibits better performance are not even particularly significant. While the advantage of HTN-BF is evident for all performance metrics, it is especially pronounced in the case of label-wise F-Measure optimization. This metric seems somewhat challenging to optimize using AutoML approaches, given that the scores are generally quite low. However, HTN-BF manages to achieve scores that improve by up to a factor of three compared to SMAC and Hyperband (let alone Random Search, which is substantially ineffective). Additionally, we can observe that SMAC falls in the middle ground, whereas HB and BOHB generally perform better than the other methods (except for HTN-BF). In contrast, GGP typically performs inferior to the other compared methods, with most of its points positioned above the diagonal line.

In Table 3, we can observe that the advantage of HTN-BF is often quite substantial. For each dataset, we report the mean result of each algorithm along with its standard deviation. The algorithm with the highest mean score is highlighted in bold, and we underline those results that are not significantly worse from a statistical perspective (according to a Wilcoxon signed-rank test with a significance level of 0.05) for the same dataset. As indicated by the relatively low standard deviations and confirmed by the significance test, the results are not simply due to chance. Thus, the superiority of HTN-BF appears to be reliable. Despite HTN-BF achieving improvements over other methods by factors on certain datasets, the statistical difference is less pronounced for the label-wise F-Measure. For the other two performance metrics, the vast majority of favorable entries are also statistically significant.

The random search approach manages to produce improved solutions compared to the other optimizers on some datasets even after 24 hours of runtime. Furthermore, for two of the three metrics, achieving a better average rank is even comparable to GGP, approaching SMAC and GGP for the label-wise F-Measure. However, random search does not offer a consistently viable alternative, as it also produces poor results on many datasets. The highly fluctuating performance can be attributed to the fact that random search first selects one component from the set of all possible unparameterized classifiers, which inherently biases towards more complex classifier structures (i.e., a higher propensity for including meta classifiers for multi-label classification and single-label base learners), as those represent a larger portion of the set.

In the radial pie charts of Fig. 11, we present the relative frequency of an algorithm being selected by each respective optimizer across all runs. The layers of the radial pie charts represent the five different component types, reading from outside to inside: meta multi-label, base multi-label, meta single-label, base single-label, and component algorithms. For clarity, only algorithms with a share of at least 0.05 are shown, while algorithms below this threshold are grouped together under the label "Others". If no algorithm has been selected for a particular layer, this is indicated by a "/". It's important to note that meta strategies are not necessarily expected to be selected instead of base multi-label algorithms, as the latter are also intended to be included in any solution. This figure clearly highlights that SMAC, HB, and BOHB tend to select somewhat similar configurations, which aligns with their comparable performance in various settings. However, SMAC's and HB's choices differ more from each other than any of them differ from BOHB. Another interesting observation is that the bias of random search towards more complex classifier structures is evident and clearly discernible.


HB
BOHB
GGP
HTN-BF
RANDOM
SMAC
SMAC
BOHB
GGP
HTN-BF
RANDOM
HB
SMAC
HB
GGP
HTN-BF
RANDOM
BOHB


arts1
bibtex
birds
bookmarks
business1
computers1
education1
emotions
enron-f
entertainment1
flags
genbase
health1
llog-f
mediamill
medical
recreation1
reference1
scene
science1
social1
society1
tmc2007
yeast
Dataset
5-MCCV Classifier Evaluation Times in Seconds

this bias from another strategy. On one hand, this bias enables random search to achieve the best performances on some of the datasets. On the other hand, classifier assessments are more prone to timeouts, as more complex classifiers usually require significantly more assessment time, leading to the poor results mentioned earlier. Finally, GGP and HTN-BF favor simpler configurations that barely incorporate meta-algorithms at all. However, the configurations selected by GGP and HTN-BF differ considerably, especially in the set of selected multi-label base algorithms, where HTN is more diverse than GGP.

Methods that rely on a reduction to hyper-parameter optimization are generally inferior to HTN-BF but perform better on a few datasets. Overall, though, it is evident that HB and BOHB compare well with SMAC, which we attribute to the feature of multi-fidelity optimization. As HB and BOHB are allowed to evaluate single runs of the Monte Carlo cross-validation (MCCV), they can use extra time to explore a more diverse range of classifiers and then focus more on the promising candidates. In the anytime average rank plots in Fig. 10, we can observe that these methods typically perform superior initially, but HTN-BF overtakes them after one hour (first vertical dashed line). While HB and BOHB compete head-to-head, SMAC is generally inferior, particularly for label-wise F-Measure (FL) and micro-averaged F-Measure (Fm). However, in the case of instance-wise F-Measure (FI), SMAC manages to perform competitively with BOHB.

GGP and Random Search quickly drop to the lower positions, which can be attributed to sampling the (current) incumbent at random leading to complex models that take longer to evaluate or might even timeout, and any method is considered to have a score of 0 as long as no incumbent was found.

Language-based genetic programming performs the worst overall. After 12 hours (third vertical dashed line), it significantly falls behind in terms of average ranks compared to all other methods, already performing worse than Random Search for FI and Fm. However, the poor performance can be attributed to the configuration of the evaluated GGP approach, which was designed with a population of only 15 individuals, as advised in [5]. While this seems to be a reasonable value for moderate runtimes of up to 6 hours (second vertical dashed line), it hampers a thorough exploration of the search space as conducted by other methods. Additionally, we only use a straightforward version of GGP that doesn't utilize more advanced features, such as model restarting.

Another interesting insight is regarding the "stability" of the approaches. We can see that the standard deviation is smaller for HTN-BF than for all other algorithms, both on average and at the extremes. In other words, HTN-BF consistently produces good results across various cases. Consequently, HTN-BF is expected to generate improved results than SMAC or HB in almost all scenarios, not just on average. Furthermore, results obtained from other methods can also perform significantly worse than the mean performance, indicating a certain risk when being applied in practice. Finally, even if HTN-BF plays a significantly dominant role, it's worth emphasizing that all other methods still yield important insights into the AutoML optimization process and should not be disregarded.

Interpretation of Results

The main conclusion that can be drawn from the results is that greedily pursuing candidate configurations is effective in the multi-label classification scenario. Among all the compared algorithms, HTN-BF along with GGP is clearly the most greedy algorithm; its only exploration lies in the number of samples drawn for each node evaluation. However, even though GGP can also be considered greedy due to its local search behavior, it heavily depends on its initialization and gets stuck in local optima easily. Given HTN-BF's exceptional overall performance, it can be inferred that greediness is preferable over exploration for this setting, characterized by an extremely large search space.

This observation also seems to have a natural explanation in the lengthy assessment times in multi-label classification, as demonstrated in the box plots in Fig. 9. There simply isn't enough time for exhaustive exploration, and being stuck in a local optimum, even if given sufficient exploration initially, is a much smaller gamble here than not optimizing at all. However, upon deeper investigation, it's not entirely clear whether the advantage of HTN-BF is due to the search behavior or due to the appropriate model for specifying the search space. Generally, it might be that the advantage comes from using a language-based approach to model the search space instead of converting the space to a hyper-parameter optimization vector, while the (greedy) algorithm used to navigate that space has a less significant impact.

This suspicion seems to be confirmed by the fact that the random search, being the least greedy algorithm, sometimes performs well too. Among all instances where HTN-BF is not optimal, random search has the highest chance of winning. For these specific datasets, this is either due to the fact that the more sophisticated methods tend to focus on flatter classifiers, resulting in simpler classifier architectures, or that the data encountered so far doesn't seem to favor any approach that exploits it. However, the results of the random search are often very poor as well, as it frequently encounters timeouts and cannot find any

reasonable configuration. For instance, the scores on datasets like media mill, social1, society1, and tmc2007 are 0, compared to values ranging from 0.3 to 0.6 for other algorithms. Therefore, a random search is certainly not a viable alternative.

Note that in cases where the score is 0, classifiers are returned that are quick to evaluate but perform poorly. These configurations often use a majority classifier as a base learner for the transformation methods, which, due to the unique label distribution, typically scores 0.

Overall, all methods seem to struggle with the vast size of the search space. While greediness still seems to be the most efficient approach to deal with this challenge, similar to other methods, it tends to overlook classifiers that are structurally more complex. As demonstrated by the results of the random search, which is biased towards such methods, excluding the more complex methods would come at the cost of excluding the optimal solution for certain tasks. Nevertheless, in order to elevate the performance of the obtained configurations, either the methods need to be further adapted to operate more effectively in the MLC search space, or the problem itself needs to be modified so that the methods can better cope with it.

For the latter option, one approach is to implement meta-learning techniques to progressively prune parts of the search space, i.e., in an instance-wise manner, which are expected to be irrelevant for the final configuration. For example, this could involve using approaches such as extreme algorithm selection (XAS), which has proven useful in settings with a large number of different algorithms. Thus, the methods could focus on the more promising candidates as suggested by the XAS model. Another option is to incorporate safeguards for the evaluation of configuration candidates to prevent timeouts, thereby allowing time to be spent on regions that are not eliminated from the effective solution space. In any case, the observation that either method needs to be adapted to better fit the MLC setting or that the search space needs to be changed to better suit the methods already developed for SLC aligns well with the philosophy according to which classifiers for MLC have been developed in the literature up to this point.

Conclusion

In this study, we examined existing optimization approaches for automating multi-label classification and expanded other commonly used single-label AutoML methods to the domain of multi-label classification. Additionally, we introduced a benchmarking framework for multi-label classification, which allows for controlled algorithm comparisons by ensuring that all methods operate within the same computational and time constraints and that they explore the same search space, using the same performance evaluation of candidate configurations.

Our extensive analysis revealed that directly reducing the AutoML problem to hyper-parameter optimization does not scale well to the multi-label classification problem. Therefore, to effectively apply these techniques, more effort is needed to address the challenges posed by the very large search space and the complex hierarchical structures of multi-label classifiers.

Contrary to the norm, an eager global search approach based on hierarchical task network planning shows promising results, indicating its potential to effectively handle the hierarchical structures inherent in the search space model. However, all the examined AutoML approaches share a common tendency to focus on classifiers with flatter structures than others. As a result, more complex classifiers with better potential performance are not adequately explored. To address this issue, we propose two interesting research directions that align with the historical development of classifiers for multi-label classification: either adapting the methods to the specific characteristics of the multi-label classification search space, or transforming the original AutoML problem for multi-label classification into a form that is more amenable to the existing approaches.

In conclusion, this study sheds light on the challenges and opportunities presented by automating multi-label classification using AutoML techniques. It emphasizes the importance of considering the unique characteristics of the multi-label classification problem when applying and developing AutoML methods, ultimately aiming to improve the performance of automated multi-label classification systems.

REFERENCES

1. Balaji, A., & Allen, A. (2018). Benchmarking automatic machine learning frameworks. arXiv preprint arXiv:1808.06492.

2. Chen, B., Wu, H., Mo, W., Chattopadhyay, I., & Lipson, H. (2018). Autostacker: A compositional evolutionary learning system. In Proc. Genetic Evol. Comput. Conf. (pp. 402-409).

3. Drori, I., Sharma, A., & Kapoor, A. (2018). AlphaD3M: Machine learning pipeline synthesis. In Proc. Autom. Mach. Learn. Workshop ICML.

4. Erickson, N., Mueller, R., Shirkov, A., Goyal, S., Xie, L., Smola, A. J., & Li, M. (2020). AutoGluon-tabular: Robust and accurate AutoML for structured data. arXiv preprint arXiv:2003.06505.

5. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In Proc. 28th Int. Conf. Neural Inf. Process. Syst. (pp. 2755-2763).

6. Gijsbers, P., & Vanschoren, J. (2019). GAMA: Genetic automated machine learning assistant. Journal of Open Source Software, 4(33), 1132.

7. He, X., Zhao, K., & Chu, X. (2019). AutoML: A survey of the state-of-the-art. arXiv preprint arXiv:1908.00709.

8. Komer, B., Bergstra, J., & Eliasmith, C. (2019). Hyperopt-sklearn. In Automated Machine Learning (pp. 23-43). Springer, Berlin, Germany.

9. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(1), 6765-6816.

10. Mohr, F., Wever, M., & Hüllermeier, E. (2018). ML-plan: Automated machine learning via hierarchical planning. Machine Learning, 107(8-10), 1495-1515.