An ADMM Based Framework for AutoML Pipeline Configuration

Sijia Liu,† Parikshit Ram,† Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, Alexander Gray
IBM Research AI
†Equal contributions

Abstract

We study the AutoML problem of automatically configuring machine learning pipelines by jointly selecting algorithms and their appropriate hyper-parameters for all steps in supervised learning pipelines. This *black-box* (gradient-free) optimization with *mixed* integer & continuous variables is a challenging problem. We propose a novel AutoML scheme by leveraging the alternating direction method of multipliers (ADMM). The proposed framework is able to (i) decompose the optimization problem into easier sub-problems that have a reduced number of variables and circumvent the challenge of mixed variable categories, and (ii) incorporate black-box constraints alongside the black-box optimization objective. We empirically evaluate the flexibility (in utilizing existing AutoML techniques), effectiveness (against open source AutoML toolkits), and unique capability (of executing AutoML with practically motivated black-box constraints) of our proposed scheme on a collection of binary classification data sets from UCI ML & OpenML repositories. We observe that on an average our framework provides significant gains in comparison to other AutoML frameworks (Auto-sklearn & TPOT), highlighting the practical advantages of this framework.

1 Introduction

Automated machine learning (AutoML) research has received increasing attention. The focus has shifted from hyper-parameter optimization (HPO) for the best configuration of a *single* machine learning (ML) algorithm (Snoek, Larochelle, and Adams 2012), to configuring multiple stages of a ML pipeline (e.g., transformations, feature selection, predictive modeling) (Feurer et al. 2015), to AutoML user experience in the real world (Wang and others 2019). Among the wide-range of research challenges offered by AutoML, we focus on the automatic pipeline configuration problem (that is, joint algorithm selection and HPO), and tackle it from the perspective of mixed continuous-integer black-box nonlinear programming. This problem has two main challenges: (i) the tight coupling between the ML algorithm selection & HPO; and (ii) the black-box nature of optimization objective lacking any explicit functional form and gradients – optimization feedback is only available in the form of function evaluations. We propose a new AutoML framework to address these challenges by leveraging the alternating direction method of multipliers (ADMM). ADMM offers a *two-block* alternating optimization procedure that splits an involved problem (with multiple variables & constraints) into simpler sub-problems (Boyd and others 2011)

Contributions. Starting with a combinatorially large set of algorithm candidates and their collective set of hyper-parameters, we utilize ADMM to decompose the AutoML problem into three problems: (i) HPO with a *small* set of *only continuous* variables & constraints, (ii) a closed-form Euclidean projection onto an integer set, and (iii) a combinatorial problem of algorithm selection. Moreover, we exploit the ADMM framework to handle *any black-box constraints alongside the black-box objective (loss) function* – the above decomposition seamlessly incorporates such constraints while retaining almost the same sub-problems.

Our contributions are: (i) We explicitly model the coupling between hyper-parameters and available algorithms, and exploit the hidden structure in the AutoML problem (Section 3). (ii) We employ ADMM to decompose the problem into a sequence of sub-problems (Section 4.1), which decouple the difficulties in AutoML and can each be solved more efficiently and effectively, demonstrating over $10\times$ speedup and 10% improvement in many cases.(iii) We present the first AutoML framework that explicitly handles general black-box constraints (Section 4.2). (iv) We demonstrate the flexibility and effectiveness of the ADMM-based scheme empirically against popular AutoML toolkits Auto-sklearn (Feurer et al. 2015) & TPOT (Olson and Moore 2016) (Section 5), performing best on 50% of the datasets; Auto-sklearn performed best on 27% and TPOT on 20%.

2 Related work

Black-box optimization in AutoML. Beyond grid-search for HPO, random search is a very competitive baseline because of its simplicity and parallelizability (Bergstra and Bengio 2012). Sequential model-based optimization (SMBO) (Hutter, Hoos, and Leyton-Brown 2011) is a common technique with different 'models' such as Gaussian processes (Snoek, Larochelle, and Adams 2012), random forests (Hutter, Hoos, and Leyton-Brown 2011) and tree-parzen estimators (Bergstra et al. 2011). However, black-box optimization is a time consuming process because the expensive black-box function evaluation involves model training and scoring (on a held-out set). Efficient *multi-fidelity* approximations

of the black-box function based on some budget (training samples/epochs) combined with bandit learning can skip unpromising candidates early via successive halving (Jamieson and Talwalkar 2016; Sabharwal and others 2016) and HyperBand (Li et al. 2018). However, these schemes essentially perform an efficient random search and are well suited for search over discrete spaces or discretized continuous spaces. BOHB (Falkner, Klein, and Hutter 2018) combines SMBO (with TPE) and HyperBand for improved optimization. Meta-learning (Vanschoren 2018) leverages past experiences in the optimization with search space refinements and promising starting points. The collaborative filtering based methods (Yang et al. 2019) are examples of meta-learning, where information from past evaluation on other datasets is utilized to pick pipelines for any new datasets. Compared to the recent works on iterative pipeline construction using tree search (Mohr, Wever, and Hüllermeier 2018; Rakotoarison, Schoenauer, and Sebag 2019), we provide a natural yet formal primal-dual decomposition of autoML pipeline configuration problems.

Toolkits. Auto-WEKA (Thornton et al. 2012; Kotthoff and others 2017) and Auto-sklearn (Feurer et al. 2015) are the main representatives of SBMO-based AutoML. Both apply the general purpose framework SMAC (Sequential Model-based Algorithm Configuration) (Hutter, Hoos, and Leyton-Brown 2011) to find optimal ML pipelines. Both consider a fixed shape of the pipeline with functional modules (preprocessing, transforming, modeling) and automatically select a ML algorithm and its hyper-parameters for each module. Auto-sklearn improves upon Auto-WEKA with two innovations: (i) a meta-learning based preprocessing step that uses 'meta-features' of the dataset to determine good initial pipeline candidates based on past experience to *warm start* the optimization, (ii) an greedy forward-selection ensembling (Caruana et al. 2004) of the pipeline configurations found during the optimization as an independent post-processing step. Hyperopt-sklearn (Komer, Bergstra, and Eliasmith 2014) utilizes TPE as the SMBO. TPOT (Olson and Moore 2016) and ML-Plan (Mohr, Wever, and Hüllermeier 2018) use genetic algorithm and hierarchical task networks planning respectively to optimize over the pipeline shape and the algorithm choices, but require discretization of the hyper-parameter space (which can be inefficient in practice as it leads performance degradation). AlphaD3M (Drori, Krishnamurthy, and others 2018) integrates reinforcement learning with Monte-Carlo tree search (MCTS) for solving AutoML problems but without imposing efficient decomposition over hyperparameters and model selection. AutoStacker (Chen, Wu, and others 2018) focuses on ensembling and cascading to generate complex pipelines and the actual algorithm selection and hyper-parameter optimization happens via random search.

3 An Optimization Perspective to AutoML

We focus on the *joint* algorithm selection and HPO for a *fixed pipeline* – a ML pipeline with a fixed sequence of functional modules (preprocessing $\rightarrow$ missing/categorical handling $\rightarrow$ transformations $\rightarrow$ feature selection $\rightarrow$ modeling) with a set of algorithm choices in each module – termed as the CASH (**c**ombined **a**lgorithm **s**election and **H**PO) problem (Thornton et al. 2012; Feurer et al. 2015) and solved with toolkits such as Auto-WEKA and Auto-sklearn. We extend this formulation by explicitly expressing the combinatorial nature of the algorithm selection with Boolean variables and constraints. We will also briefly discuss how this formulation facilities extension to other flexible pipelines.

Problem statement. For N *functional modules* (e.g., preprocessor, transformer, estimator) with a choice of K_i *algorithms* in each, let $\mathbf{z}_i \in \{0,1\}^{K_i}$ denote the algorithm choice in module i, with the constraint $\mathbf{1}^\top \mathbf{z}_i = \sum_{j=1}^{K_i} z_{ij} = 1$ ensuring that only a single algorithm is chosen from each module. Let $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$. Assuming that categorical hyper-parameters can be encoded as integers (using standard techniques), let $\boldsymbol{\theta}_{ij}$ be the *hyper-parameters* of algorithm j in module i, with $\boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij} \subset \mathbb{R}^{m^c_{ij}}$ as the *continuous* hyper-parameters (constrained to the set $\mathcal{C}_{ij}$) and $\boldsymbol{\theta}^d_{ij} \in \mathcal{D}_{ij} \subset \mathbb{Z}^{m^d_{ij}}$ as the *integer* hyper-parameters (constrained to $\mathcal{D}_{ij}$). Conditional hyper-parameters can be handled with additional constraints $\boldsymbol{\theta}_{ij} \in \mathcal{E}_{ij}$ or by "flattening" the hyper-parameter tree and considering each leaf as a different algorithm. For simplicity of exposition, we assume that the conditional hyper-parameters are flattened into additional algorithm choices. Let $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{ij}, \forall i \in [N], j \in [K_i]\}$, where $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{N}$. Let $f(\mathbf{z}, \boldsymbol{\theta}; \mathcal{A})$ represent some notion of loss of a ML pipeline corresponding to the algorithm choices as per $\mathbf{z}$ with the hyper-parameters $\boldsymbol{\theta}$ on a learning task with data $\mathcal{A}$ (such as the k-fold cross-validation or holdout validation loss). The optimization problem of automatic pipeline configuration is stated as:

$$\min_{\mathbf{z},\boldsymbol{\theta}} f(\mathbf{z}, \boldsymbol{\theta}; \mathcal{A}) \quad \text{subject to} \begin{cases} \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N], \\ \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \boldsymbol{\theta}^d_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i]. \end{cases} \tag{1}$$

We highlight 2 key differences of problem (1) from the conventional CASH formulation: (i) we use explicit Boolean variables $\mathbf{z}$ to encode the algorithm selection, (ii) we differentiate continuous variables/constraints from discrete ones for a possible efficient decomposition between continuous optimization and integer programming. These features better characterize the properties of the problem and thus enable more effective joint optimization. For any given $(\mathbf{z}, \boldsymbol{\theta})$ and data $\mathcal{A}$, the objective (loss) function $f(\mathbf{z}, \boldsymbol{\theta}; \mathcal{A})$ is a black-box function – it does not have an analytic form with respect to $(\mathbf{z}, \boldsymbol{\theta})$ (hence no derivatives). The actual evaluation of f usually involves training, testing and scoring the ML pipeline corresponding to $(\mathbf{z}, \boldsymbol{\theta})$ on some split of the data $\mathcal{A}$.

AutoML with black-box constraints. With the increasing adoption of AutoML (Wang and others 2019), the formulation (1) may not be sufficient. AutoML may need to find ML pipelines with high predictive performance (low loss) that also *explicitly satisfy* application specific *constraints*. Deployment constraints may require the pipeline to have prediction latency or size in memory below some threshold (latency $\leq 10\mu$s, memory ≤ 100MB). Business specific constraints may desire pipelines with low overall classification error and an explicit upper bound on the false positive rate – in a loan default risk application, false positives leads to loan

denials to eligible candidates, which may violate regulatory requirements. In the quest for *fair* AI, regulators may explicitly require the ML pipeline to be above some predefined fairness threshold (Friedler and others 2019). Furthermore, many applications have very domain specific metric(s) with corresponding constraints – custom metrics are common in Kaggle competitions. We incorporate such requirements by extending AutoML formulation (1) to include M *black-box constraints*:

$$g_i\left(\mathbf{z}, \boldsymbol{\theta}; \mathcal{A}\right) \leq \epsilon_i, i \in [M]. \tag{2}$$

These functions have no analytic form with respect to $(\mathbf{z}, \boldsymbol{\theta})$, in constrast to the analytic constraints in problem (1). One approach is to incorporate these constraints into the black-box objective with a penalty function p, where the new objective becomes $f + \sum_i p(g_i, \epsilon_i)$ or $f \cdot \prod_i p(g_i, \epsilon_i)$. However, these schemes are very sensitive to the choice of the penalty function and do not guarantee feasible solutions.

Generalization for more flexible pipelines. We can extend the problem formulation (1) to enable optimization over the ordering of the functional modules. For example, we can choose between 'preprocessor $\rightarrow$ transformer $\rightarrow$ feature selector' OR 'feature selector $\rightarrow$ preprocessor $\rightarrow$ transformer'. The ordering of $T \leq N$ modules can be optimized by introducing T^2 Boolean variables $\mathbf{o} = \{o_{ik} \colon i, k \in [T]\}$, where $o_{ik} = 1$ indicates that module i is placed at position k. The following constraints are then needed: (i) $\sum_{k \in [T]} o_{ik} = 1, \forall i \in [T]$ indicates that module i is placed at a single position, and (ii) $\sum_{i \in [T]} o_{ik} = 1 \forall k \in [T]$ enforces that only one module is placed at position k. These variables can be added to $\mathbf{z}$ in problem (1) ($\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N, \mathbf{o}\}$). The resulting formulation still obeys the generic form of (1), which as will be evident later, can be efficiently solved by an operator splitting framework like ADMM (Boyd and others 2011).

4 ADMM-Based Joint Optimizer

ADMM provides a general effective optimization framework to solve complex problems with mixed variables and multiple constraints (Boyd and others 2011; Liu et al. 2018). We utilize this framework to decompose problem (1) without and with black-box constraints (2) into easier sub-problems.

4.1 Efficient operator splitting for AutoML

In what follows, we focus on solving problem (1) with analytic constraints. The handling of black-box constraints will be elaborated on in the next section. Denoting $\boldsymbol{\theta}^c = \{\boldsymbol{\theta}^c_{ij}, \forall i \in [N], j \in [K_i]\}$ as all the continuous hyper-parameters and $\boldsymbol{\theta}^d$ (defined correspondingly) as all the integer hyper-parameters, we re-write problem (1) as:

$$\min_{\mathbf{z}, \boldsymbol{\theta} = \{\boldsymbol{\theta}^c, \boldsymbol{\theta}^d\}} f\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \boldsymbol{\theta}^d\right\}; \mathcal{A}\right) \quad \text{subject to} \begin{cases} \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N], \\ \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \boldsymbol{\theta}^d_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i]. \end{cases} \tag{3}$$

Introduction of continuous surrogate loss. With $\widetilde{\mathcal{D}}_{ij}$ as the continuous relaxation of the integer space $\mathcal{D}_{ij}$ (if $\mathcal{D}_{ij}$ includes integers ranging from $\{l, \ldots, u\} \subset \mathbb{Z}$, then $\widetilde{\mathcal{D}}_{ij} = [l, u] \subset \mathbb{R}$), and $\widetilde{\boldsymbol{\theta}}^d$ as the continuous surrogates for $\boldsymbol{\theta}^d$ with $\widetilde{\boldsymbol{\theta}}_{ij} \in \widetilde{\mathcal{D}}_{ij}$ (corresponding to $\boldsymbol{\theta}_{ij} \in \mathcal{D}_{ij}$), we utilize a surrogate loss function $\widetilde{f}$ for problem (3) defined solely over the continuous domain with respect to $\boldsymbol{\theta}$:

$$\widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) := f\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \mathcal{P}_{\mathcal{D}}\left(\widetilde{\boldsymbol{\theta}}^d\right)\right\}; \mathcal{A}\right), \tag{4}$$

where $\mathcal{P}_{\mathcal{D}}(\widetilde{\boldsymbol{\theta}}^d) = \{\mathcal{P}_{\mathcal{D}_{ij}}(\widetilde{\boldsymbol{\theta}}^d_{ij}), \forall i \in [N], j \in [K_i]\}$ is the projection of the continuous surrogates onto the integer set. This projection is **necessary** since the black-box function is defined (hence *can only be evaluated*) on the integer sets $\mathcal{D}_{ij}$s. Ergo, problem (3) can be *equivalently* posed as

$$\min_{\mathbf{z}, \boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d, \boldsymbol{\delta}} \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) \quad \text{subject to} \begin{cases} \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N] \\ \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \widetilde{\boldsymbol{\theta}}^d_{ij} \in \widetilde{\mathcal{D}}_{ij}, \forall i \in [N], j \in [K_i] \\ \boldsymbol{\delta}_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i] \\ \widetilde{\boldsymbol{\theta}}^d_{ij} = \boldsymbol{\delta}_{ij}, \forall i \in [N], j \in [K_i], \end{cases} \tag{5}$$

where the equivalence between problems (3) & (5) is established by the equality constraint $\widetilde{\boldsymbol{\theta}}^d_{ij} = \boldsymbol{\delta}_{ij} \in \mathcal{D}_{ij}$, implying $\mathcal{P}_{\mathcal{D}_{ij}}(\widetilde{\boldsymbol{\theta}}^d_{ij}) = \widetilde{\boldsymbol{\theta}}^d_{ij} \in \mathcal{D}_{ij}$ and $\widetilde{f}(\mathbf{z}, \{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\}; \mathcal{A}) = f(\mathbf{z}, \{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\}; \mathcal{A})$. The continuous surrogate loss (4) is key in being able to perform theoretically grounded operator splitting (via ADMM) over mixed continuous/integer variables in the AutoML problem (3).

Operator splitting from ADMM. Using the notation that $I_{\mathcal{X}}(\mathbf{x}) = 0$ if $\mathbf{x} \in \mathcal{X}$ else $+\infty$, and defining the sets $\mathcal{Z} = \{\mathbf{z} \colon \mathbf{z} = \{\mathbf{z}_i\}, \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N]\}$, $\mathcal{C} = \{\boldsymbol{\theta}^c \colon \boldsymbol{\theta}^c = \{\boldsymbol{\theta}^c_{ij}\}, \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \forall i \in [N], j \in [K_i]\}$, $\mathcal{D} = \{\boldsymbol{\delta} \colon \boldsymbol{\delta} = \{\boldsymbol{\delta}_{ij}\}, \boldsymbol{\delta}_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i]\}$ and $\widetilde{\mathcal{D}} = \{\widetilde{\boldsymbol{\theta}}^d \colon \widetilde{\boldsymbol{\theta}}^d = \{\widetilde{\boldsymbol{\theta}}^d_{ij}\}, \widetilde{\boldsymbol{\theta}}^d_{ij} \in \widetilde{\mathcal{D}}_{ij}, \forall i \in [N], j \in [K_i]\}$, we can re-write problem (5) as

$$\min_{\mathbf{z}, \boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d, \boldsymbol{\delta}} \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) + I_{\mathcal{Z}}(\mathbf{z}) + I_{\mathcal{C}}(\boldsymbol{\theta}^c) + I_{\widetilde{\mathcal{D}}}(\widetilde{\boldsymbol{\theta}}^d) + I_{\mathcal{D}}(\boldsymbol{\delta}); \text{ subject to } \widetilde{\boldsymbol{\theta}}^d = \boldsymbol{\delta}. \tag{6}$$

with the corresponding augmented Lagrangian function

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d, \boldsymbol{\delta}, \boldsymbol{\lambda}) := \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) + I_{\mathcal{Z}}(\mathbf{z}) + I_{\mathcal{C}}(\boldsymbol{\theta}^c) + I_{\widetilde{\mathcal{D}}}(\widetilde{\boldsymbol{\theta}}^d) + I_{\mathcal{D}}(\boldsymbol{\delta}) + \boldsymbol{\lambda}^\top\left(\widetilde{\boldsymbol{\theta}}^d - \boldsymbol{\delta}\right) + \frac{\rho}{2}\left\|\widetilde{\boldsymbol{\theta}}^d - \boldsymbol{\delta}\right\|_2^2, \tag{7}$$

where $\boldsymbol{\lambda}$ is the Lagrangian multiplier, and $\rho > 0$ is a penalty parameter for the augmented term.

ADMM (Boyd and others 2011) alternatively minimizes the augmented Lagrangian function (7) over *two* blocks of variables, leading to an efficient operator splitting framework for nonlinear programs with *nonsmooth* objective function and *equality* constraints. Specifically, ADMM solves problem (1) by alternatively minimizing (7) over variables $\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\}$,

and $\{\boldsymbol{\delta}, \mathbf{z}\}$. This can be equivalently converted into 3 sub-problems over variables $\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\}$, $\boldsymbol{\delta}$ and $\mathbf{z}$, respectively. We refer readers to Algorithm 1 for simplified sub-problems and Appendix 1 for detailed derivation[1].

The rationale behind the advantage of ADMM is that it decomposes the AutoML problem into sub-problems with smaller number of variables: This is crucial in black-box optimization where convergence is strongly dependent on the number of variables. For example, the number of black-box evaluations needed for critical point convergence is typically $O(n \sim n^3)$ for n variables (Larson, Menickelly, and Wild 2019). In what follows, we show that the easier sub-problems in Algorithm 1 yield great interpretation of the AutoML problem (1) and suggest efficient solvers in terms of continuous hyper-parameter optimization, integer projection operation, and combinatorial algorithm selection.

Solving $\boldsymbol{\theta}$-min. Problem ($\boldsymbol{\theta}$-min) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d} \widetilde{f}\left(\mathbf{z}^{(t)}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) + \frac{\rho}{2}\left\|\widetilde{\boldsymbol{\theta}}^d - \mathbf{b}\right\|_2^2 \\ &\text{subject to} \begin{cases} \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij} \\ \widetilde{\boldsymbol{\theta}}^d_{ij} \in \widetilde{\mathcal{D}}_{ij}, \end{cases} \forall i \in [N], j \in [K_i], \end{aligned} \tag{8}$$

where both $\boldsymbol{\theta}^c$ and $\widetilde{\boldsymbol{\theta}}^d$ are continuous optimization variables. Since the algorithm selection scheme $\mathbf{z}^{(t)}$ is fixed for this problem, $\widetilde{f}$ in problem (8) only depends on the hyper-parameters of the chosen algorithms – the *active set* of continuous variables $(\boldsymbol{\theta}^c_{ij}, \widetilde{\boldsymbol{\theta}}^d_{ij})$ where ${z_{ij}}^{(t)} = 1$. This splits problem (8) even further into two problems. The *inactive* set problem reduces to the following for all $i \in [N], j \in [K_i]$ such that $z_{ij} = 0$:

$$\min_{\widetilde{\boldsymbol{\theta}}^d_{ij}} \frac{\rho}{2}\|\widetilde{\boldsymbol{\theta}}^d_{ij} - \mathbf{b}_{ij}\|_2^2 \quad \text{subject to} \quad \widetilde{\boldsymbol{\theta}}^d_{ij} \in \widetilde{\mathcal{D}}_{ij}, \tag{9}$$

which is solved by a Euclidean projection of $\mathbf{b}_{ij}$ onto $\widetilde{\mathcal{D}}_{ij}$.

For the *active* set of variables $S = \{(\boldsymbol{\theta}^c_{ij}, \widetilde{\boldsymbol{\theta}}^d_{ij})\colon \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \widetilde{\boldsymbol{\theta}}_{ij} \in \widetilde{\mathcal{D}}_{ij}, z_{ij} = 1, \forall i \in [N], j \in [K_i]\}$, problem (8) reduces to the following black-box optimization with only the *small* active set of *continuous* variables[2]

$$\min_{(\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d) \in S} \widetilde{f}\left(\mathbf{z}^{(t)}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) + \frac{\rho}{2}\left\|\widetilde{\boldsymbol{\theta}}^d - \mathbf{b}\right\|_2^2. \tag{10}$$

The above problem can be solved using Bayesian optimization (Shahriari et al. 2016), direct search (Larson, Menickelly, and Wild 2019), or trust-region based derivative-free optimization (Conn, Scheinberg, and Vicente 2009).

Solving $\boldsymbol{\delta}$-min. According to the definition of $\mathcal{D}$, problem ($\boldsymbol{\delta}$-min) can be rewritten as

$$\min_{\boldsymbol{\delta}} \frac{\rho}{2}\|\boldsymbol{\delta} - \mathbf{a}\|_2^2 \quad \text{subject to } \boldsymbol{\delta}_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i], \tag{11}$$

and solved in closed form by projecting $\mathbf{a}$ onto $\widetilde{\mathcal{D}}$ and then rounding to the nearest integer in $\mathcal{D}$.

Solving z-min. Problem ($\mathbf{z}$-min) rewritten as

$$\begin{aligned} &\min_{\mathbf{z}} \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^{c(t+1)}, \widetilde{\boldsymbol{\theta}}^{d(t+1)}\right\}; \mathcal{A}\right) \\ &\text{subject to } \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N] \end{aligned} \tag{12}$$

is a black-box integer program solved exactly with $\prod_{i=1}^N K_i$ evaluations of $\widetilde{f}$. However, this is generally not feasible. Beyond random sampling, there are a few ways to leverage existing AutoML schemes: (i) *Combinatorial multi-armed bandits.* – Problem (12) can be interpreted through combinatorial bandits as the selection of the optimal N arms (in this case, algorithms) from $\sum_{i=1}^N K_i$ arms based on bandit feedback and can be efficiently solved with Thompson sampling (Durand and Gagné 2014) (ii) *Multi-fidelity approximation of black-box evaluations* – Techniques such as successive halving (Jamieson and Talwalkar 2016; Li et al. 2018) or incremental data allocation (Sabharwal and others 2016) can efficiently search over a discrete set of $\prod_{i=1}^N K_i$ candidates. (iii) *Genetic algorithms* – Genetic programming can perform this discrete black-box optimization starting from a randomly generated population and building the next generation based on the 'fitness' of the pipelines and random 'mutations' and 'crossovers'.

4.2 ADMM with black-box constraints

We next consider problem (3) in the presence of black-box constraints (2). Without loss of generality, we assume that $\epsilon_i \geq 0$ for $i \in [M]$. By introducing scalars $u_i \in [0, \epsilon_i]$, we can reformulate the inequality constraint (2) as the equality constraint together with a box constraint

$$g_i\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \boldsymbol{\theta}^d\right\}; \mathcal{A}\right) - \epsilon_i + u_i,\ u_i \in [0, \epsilon_i],\ i \in [M]. \tag{13}$$

We then introduce a continuous surrogate black-box functions $\widetilde{g}_i$ for $g_i, \forall i \in [M]$ in a similar manner to $\widetilde{f}$ given by (4). Following the reformulation of (3) that lends itself to the application of ADMM, the version with black-box constraints (13) can be equivalently transformed into

$$\begin{aligned} &\min_{\mathbf{z}, \boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d, \boldsymbol{\delta}} \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) \\ &\text{subject to} \begin{cases} \mathbf{z}_i \in \{0,1\}^{K_i}, \mathbf{1}^\top \mathbf{z}_i = 1, \forall i \in [N] \\ \boldsymbol{\theta}^c_{ij} \in \mathcal{C}_{ij}, \widetilde{\boldsymbol{\theta}}^d_{ij} \in \widetilde{\mathcal{D}}_{ij}, \forall i \in [N], j \in [K_i] \\ \boldsymbol{\delta}_{ij} \in \mathcal{D}_{ij}, \forall i \in [N], j \in [K_i] \\ \widetilde{\boldsymbol{\theta}}^d_{ij} = \boldsymbol{\delta}_{ij}, \forall i \in [N], j \in [K_i] \\ u_i \in [0, \epsilon_i], \forall i \in [M] \\ \widetilde{g}_i\left(\mathbf{z}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) - \epsilon_i + u_i = 0, \forall i \in [M]. \end{cases} \end{aligned} \tag{14}$$

Compared to problem (5), the introduction of auxiliary variables $\{u_i\}$ enables ADMM to incorporate *black-box equality* constraints as well as elementary *white-box* constraints. Similar to Algorithm 1, the ADMM solution to problem (14) can be achieved by solving three sub-problems of similar nature, summarized in Algorithm 2 and derived in Appendix 2.

4.3 Implementation and convergence

We highlight that our ADMM based scheme is not a single AutoML algorithm but rather a framework that can be used to

[1] Appendix is available at arXiv (Liu, Ram, and others 2019)

[2] For the AutoML problems we consider in our empirical evaluations, $|\boldsymbol{\theta}| = |\boldsymbol{\theta}^c_{ij}| + |\widetilde{\boldsymbol{\theta}}^d_{ij}| \approx 100$ while the largest possible active set S is less than 15 and typically less than 10.

Algorithm 1 Operator splitting from ADMM to solve problem (5)

$$\left\{\boldsymbol{\theta}^{c(t+1)}, \widetilde{\boldsymbol{\theta}}^{d(t+1)}\right\} = \underset{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d}{\arg\min}\, \widetilde{f}\left(\mathbf{z}^{(t)}, \left\{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d\right\}; \mathcal{A}\right) + I_{\mathcal{C}}(\boldsymbol{\theta}^c) + I_{\widetilde{\mathcal{D}}}(\widetilde{\boldsymbol{\theta}}^d) + (\rho/2)\left\|\widetilde{\boldsymbol{\theta}}^d - \mathbf{b}\right\|_2^2, \quad \mathbf{b} := \boldsymbol{\delta}^{(t)} - \frac{1}{\rho}\boldsymbol{\lambda}^{(t)}, \qquad (\boldsymbol{\theta}\text{-min})$$

$$\boldsymbol{\delta}^{(t+1)} = \underset{\boldsymbol{\delta}}{\arg\min}\, I_{\mathcal{D}}(\boldsymbol{\delta}) + (\rho/2)\left\|\mathbf{a} - \boldsymbol{\delta}\right\|_2^2, \quad \mathbf{a} := \widetilde{\boldsymbol{\theta}}^{d(t+1)} + (1/\rho)\boldsymbol{\lambda}^{(t)}, \qquad (\boldsymbol{\delta}\text{-min})$$

$$\mathbf{z}^{(t+1)} = \underset{\mathbf{z}}{\arg\min}\, \widetilde{f}\left(\mathbf{z}, \left\{\boldsymbol{\theta}^{c(t+1)}, \widetilde{\boldsymbol{\theta}}^{d(t+1)}\right\}; \mathcal{A}\right) + I_{\mathcal{Z}}(\mathbf{z}), \qquad (\mathbf{z}\text{-min})$$

where (t) represents the iteration index, and the Lagrangian multipliers $\boldsymbol{\lambda}$ are updated as $\boldsymbol{\lambda}^{(t+1)} = \boldsymbol{\lambda}^{(t)} + \rho(\widetilde{\boldsymbol{\theta}}^{d(t+1)} - \boldsymbol{\delta}^{(t+1)})$.

Algorithm 2 Operator splitting from ADMM to solve problem (14) **(with black-box constraints)**

$$\left\{\boldsymbol{\theta}^{c(t+1)}, \widetilde{\boldsymbol{\theta}}^{d(t+1)}, \mathbf{u}^{(t+1)}\right\} = \underset{\boldsymbol{\theta}^c, \widetilde{\boldsymbol{\theta}}^d, \mathbf{u}}{\arg\min}\, \widetilde{f} + \frac{\rho}{2}\left\|\widetilde{\boldsymbol{\theta}}^d - \mathbf{b}\right\|_2^2 + I_{\mathcal{C}}(\boldsymbol{\theta}^c) + I_{\widetilde{\mathcal{D}}}(\widetilde{\boldsymbol{\theta}}^d) + I_{\mathcal{U}}(\mathbf{u}) + \frac{\rho}{2}\sum_{i=1}^{M}\left[\widetilde{g}_i + u_i - \epsilon_i + \frac{{\mu_i}^{(t)}}{\rho}\right]^2,$$

$$\boldsymbol{\delta}^{(t+1)} = \underset{\boldsymbol{\delta}}{\arg\min} \quad \frac{\rho}{2}\|\boldsymbol{\delta} - \mathbf{a}\|_2^2 + I_{\mathcal{D}}(\boldsymbol{\delta}),$$

$$\mathbf{z}^{(t+1)} = \underset{\mathbf{z}}{\arg\min}\, \widetilde{f} + I_{\mathcal{Z}}(\mathbf{z}) + \frac{\rho}{2}\sum_{i=1}^{M}\left[\widetilde{g}_i - \epsilon_i + {u_i}^{(t+1)} + \frac{1}{\rho}{\mu_i}^{(t)}\right]^2,$$

where the arguments of $\tilde{f}$ and $\tilde{g}_i$ are omitted for brevity, $\mathbf{a}$ and $\mathbf{b}$ have been defined in Algorithm 1, $\mathcal{U} = \{\mathbf{u}\colon \mathbf{u} = \{u_i\}$, and μ_i is the Lagrangian multiplier corresponding to the equality constraint $\tilde{g}_i - \epsilon_i + u_i = 0$ in (14) and updated as ${\mu_i}^{(t+1)} = {\mu_i}^{(t)} + \rho(\widetilde{g}_i(\mathbf{z}^{(t+1)}, \{\boldsymbol{\theta}^{c(t+1)}, \widetilde{\boldsymbol{\theta}}^{d(t+1)}\}; \mathcal{A}) - \epsilon_i + {u_i}^{(t+1)})$ for $\forall i \in [M]$.

mix and match different existing black-box solvers. This is especially useful since this enables the end-user to plug-in efficient solvers tailored for the sub-problems (HPO & algorithm selection in our case). In addition to the above, the ADMM decomposition allows us to solve simpler sub-problems with a smaller number of optimization variables (a significantly reduced search space since ($\boldsymbol{\theta}$-min) only requires optimization over the active set of continuous variables). Unless specified otherwise, we adopt Bayesian optimization (BO) to solve the HPO ($\boldsymbol{\theta}$-min), e.g., (10). We use customized Thompson sampling to solve the combinatorial multi-armed bandit problem, namely, the ($\mathbf{z}$-min) for algorithm selection. We refer readers to Appendix 3 and 4 for more derivation and implementation details. In Appendix 5, we demonstrate the generalizability of ADMM to different solvers for ($\boldsymbol{\theta}$-min) and ($\mathbf{z}$-min).

The theoretical convergence guarantees of ADMM have been established under certain assumptions, e.g., convexity or smoothness (Boyd and others 2011; Hong and Luo 2017; Liu et al. 2018). Unfortunately, the AutoML problem violates these restricted assumptions. Even for non-ADMM based AutoML pipeline search, there is no theoretical convergence established in the existing baselines to the best of our knowledge. Empirically, we will demonstrate the improved convergence of the proposed scheme against baselines in the following section.

5 Empirical Evaluations

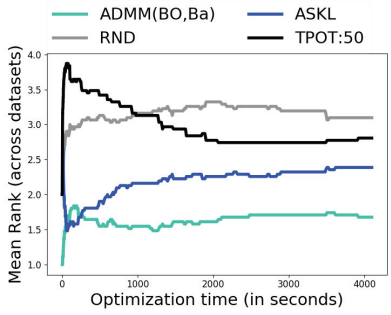
In this evaluation of our proposed framework, we demonstrate three important characteristics: (i) the empirical performance against existing AutoML toolkits, *highlighting the empirical competitiveness of the theoretical formalism*, (ii) the systematic capability to handle black-box constraints, *enabling AutoML to address real-world ML tasks, and* (iii) the flexibility to incorporate various learning procedures and solvers for the sub-problems, *highlighting that our proposed scheme is not a single algorithm but a complete framework* for AutoML pipeline configuration.

Data and black-box objective function. We consider 30 binary classification[3] datasets from the UCI ML (Asuncion and Newman 2007) & OpenML repositories (Bischl and others 2017), and Kaggle. We consider a subset of OpenML100 limited to binary classification and small enough to allow for meaningful amount of optimization for *all baselines* in the allotted 1 hour to ensure that we are evaluating the optimizers and not the initialization heuristics. Dataset details are in Appendix 6. We consider $(1 - \text{AUROC})$ (**a***rea* **u***nder the* **ROC** *curve*) as the black-box objective and evaluate it on a 80-20% train-validation split for all baselines. We consider AUROC since it is a meaningful predictive performance metric regardless of the class imbalance (as opposed to classification error).

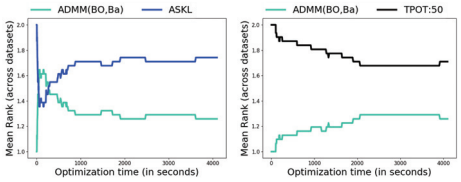
Comparing ADMM to AutoML baselines. Here we evaluate the proposed ADMM framework against widely used AutoML systems Auto-sklearn (Feurer et al. 2015) and TPOT (Olson and Moore 2016). This comparison is limited to black-box optimization with *analytic constraints only* given by (1) since existing AutoML toolkits *cannot handle black-box constraints explicitly*. We consider SMAC based vanilla Auto-sklearn ASKL[4](disabling ensembles and meta-

[3] Our scheme applies to multiclass classification & regression.

learning), random search RND, and TPOT with a population of 50 (instead of the default 100) to ensure that TPOT is able to process multiple generations of the genetic algorithm in the allotted time on all data sets.For ADMM, we utilize BO for ($\boldsymbol{\theta}$-min) and CMAB for ($\mathbf{z}$-min) – ADMM(BO,Ba)[5]. For all optimizers, we use `scikit-learn` algorithms (Pedregosa, Varoquaux, and others 2011). The functional modules and the algorithms (with their hyper-parameters) are presented in Appendix 7 [6]. We maintain parity[7] across the various AutoML baselines by searching over the same set of algorithms (see Appendix 7). For each scheme, the algorithm hyper-parameter ranges are set using Auto-sklearn as the reference[8]. We optimize for 1 hour & generate time vs. incumbent black-box objective curves aggregated over 10 trials. Details on the complete setup are in Appendix 10. The optimization convergence for all 30 datasets are in Appendix 11. At completion, ASKL achieves the lowest mean objective (across trials) in 6/30 datasets, TPOT50 in 8/30, RND in 3/30 and ADMM(BO,Ba) in 15/30, showcasing ADMM's effectiveness.

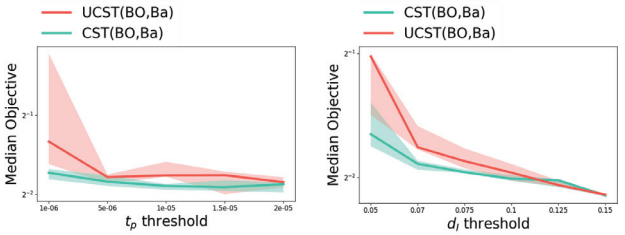


(a) All methods



(b) ASKL vs. ADMM (c) TPOT50 vs. ADMM

Figure 1: Average rank (across 30 datasets) of mean performance across 10 trials – *lower rank is better*.



(a) Varying t_p, $d_I = 0.07$ (b) Varying d_I, $t_p = 10\mu s$

Figure 2: Best objective achieved by any constraint satisfying pipeline from running the optimization for 1 hour seconds with varying thresholds for the two constraints – *lower is better* (*please view in color*). Note the log-scale on the vertical axis.

Figure 1 presents the overall performance of the different AutoML schemes versus optimization time. Here we consider the relative rank of each scheme (with respect to the mean objective over 10 trials) for every timestamp, and average this rank across 30 data sets similar to the comparison in Feurer et al. (2015). With enough time, all schemes outperform random search RND. TPOT50 performs worst in the beginning because of the initial start-up time involved in the genetic algorithm. ASKL and ADMM(BO,Ba) have comparable performance initially. As the optimization continues, ADMM(BO,Ba) significantly outperforms all other baselines. We present the pairwise performance of ADMM with ASKL (figure 1b) & TPOT50 (figure 1c).

AutoML with black-box constraints. To demonstrate the capability of the ADMM framework to incorporate real-world black-box constraints, we consider the recent Home Credit Default Risk Kaggle challenge[9] with the black-box objective of $(1 - \text{AUROC})$, and 2 black-box constraints: (i) (**deployment**) Prediction latency t_p enforcing real-time predictions, (ii) (**fairness**) Maximum pairwise disparate impact d_I (Calders and Verwer 2010) across all loan applicant age groups enforcing fairness across groups (see Appendix 12).

We run a set of experiments for each of the constraints: (i) fixing $d_I = 0.7$, we optimize for each of the thresholds $t_p = \{1, 5, 10, 15, 20\}$ (in μs), and (ii) fixing $t_p = 10\mu s$ and we optimize for each of $d_I = \{0.05, 0.075, 0.1, 0.125, 0.15\}$. Note that the constraints get less restrictive as the thresholds increase. We apply ADMM to the unconstrained problem (UCST) and post-hoc filter constraint satisfying pipelines to demonstrate that these constraints are not trivially satisfied. Then we execute ADMM with these constraints (CST). Using BO for ($\boldsymbol{\theta}$-min) & CMAB for ($\mathbf{z}$-min), we get two variants – UCST(BO,Ba) & CST(BO,Ba). This results in $(5+5) \times 2 = 20$ ADMM executions, each repeated $10\times$.

Figure 2 presents the objective achieved by the optimizer when limited only to constraint satisfying pipelines. Figure 2a presents the effect of relaxing the constraint on t_p while Figure 2b presents the same for the constraint on d_I. As

[4]Meta-learning and ensembling in ASKL are preprocessing and postprocessing steps respectively to the actual black-box optimization and can be applied to any optimizer. We demonstrate this for ADMM in Appendix 8. So we skip these aspects of ASKL here.

[5]In this setup, ADMM has 2 parameters: (i) the penalty ρ on the augmented term, (ii) the loss upper-bound $\hat{f}$ in the CMAB algorithm (Appendix 4). We evaluate the sensitivity of ADMM on these parameters in Appendix 9. The results indicate that ADMM is fairly robust to these parameters, and hence set $\rho = 1$ and $\hat{f} = 0.7$ throughout. We start the ADMM optimization with $\boldsymbol{\lambda}^{(0)} = \mathbf{0}$.

[6]Appendix is available at arXiv (Liu, Ram, and others 2019)

[7]ASKL and ADMM search over the same search space of *fixed pipeline shape & order*. TPOT also searches over different pipeline shapes & orderings because of the nature of its genetic algorithm.

[8]github.com/automl/auto-sklearn/tree/master/autosklearn/pipeline/components

[9]www.kaggle.com/c/home-credit-default-risk

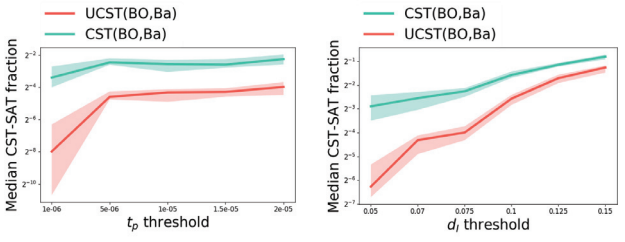
expected, the objective improves as the constraints relax. In both cases, CST outperforms UCST, with UCST approaching CST as the constraints relax. Figure 3 presents the constraint satisfying capability of the optimizer by considering the fraction of constraint-satisfying pipelines found (Figure 3a & 3b for varying t_p & d_I respectively). CST again significantly outperforms UCST, indicating that the constraints are non-trivial to satisfy, and that ADMM is able to effectively incorporate the constraints for improved performance.

Flexibility & benefits from ADMM operator splitting. It is common in ADMM to solve the sub-problems to higher approximation in the initial iterations and to an increasingly lower approximation as ADMM progresses (instead of the same approximation throughout) (Boyd and others 2011). We demonstrate (empirically) that this *adaptive ADMM* produces expected gains in the AutoML problem. Moreover, we show the empirical gains of ADMM from (i) splitting the AutoML problem (1) into smaller sub-problems which are solved in an alternating fashion, & (ii) using different solvers for the differently structured ($\boldsymbol{\theta}$-min) and ($\mathbf{z}$-min).

First we use BO for both ($\boldsymbol{\theta}$-min) and ($\mathbf{z}$-min). For ADMM with a fixed approximation level (*fixed ADMM*), we solve the sub-problems with BO to a fixed number $I = 16, 32, 64, 128$ iterations, denoted by ADMMI(BO,BO) (e.g., ADMM16(BO,BO)). For adaptive ADMM, we start with 16 BO iterations for the sub-problems and progressively increase it with an additive factor $F = 8$ & 16 with every ADMM iteration until 128 denoted by AdADMM-F8(BO,BO) & AdADMM-F16(BO,BO) respectively. We optimize for 1 hour and aggregate over 10 trials.

Figure 4 presents optimization convergence for 1 dataset (fri-c2). We see the expected behavior – fixed ADMM with small I dominate for small time scales but saturate soon; large I require significant start-up time but dominate for larger time scales. Adaptive ADMM ($F = 8$ & 16) is able to match the performance of the best fixed ADMM at every time scale. Please refer to Appendix 13 for additional results.

Next, we illustrate the advantage of ADMM on operator splitting. We consider 2 variants, AdADMM-F16(BO,BO) and AdADMM-F16(BO,Ba), where the latter uses CMAB for ($\mathbf{z}$-min). For comparison, we solve the complete *joint problem* (1) with BO, leading to a Gaussian Process with a large number of variables, denoted as JOPT(BO).



(a) Varying t_p, $d_I = 0.07$ (b) Varying d_I, $t_p = 10\mu s$

Figure 3: Fraction of pipelines found satisfying constraints with optimization for 1 hour with varying thresholds for the 2 constraints – *higher is better*. Note the log-scale on the vertical axis.

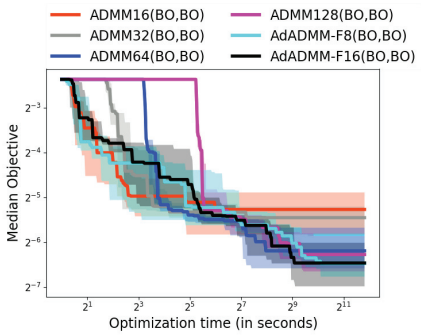


Figure 4: Optimization time (in seconds) vs. median validation performance with the inter-quartile range over 10 trials on fri-c2 dataset – lower is better (*please view in color*). Note the log scale on both axes. See Appendix 13 for additional results.

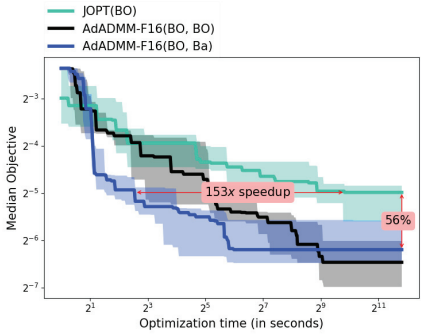


Figure 5: Optimization time vs. median validation performance with the inter-quartile range over 10 trials on fri-c2 dataset – lower is better (*please view in color*). Note the log scale on both axes. See Appendix 14 for additional results.

Figure 5 shows the optimization convergence for 1 dataset (fri-c2). The results indicate that the operator splitting in ADMM provides significant improvements over JOPT(BO), with ADMM reaching the final objective achieved by JOPT with significant speedup, and then further improving upon that final objective significantly. These improvements of ADMM over JPOT on 8 datasets are summarized in Table 1, indicating significant speedup (over $10\times$ in most cases) and further improvement (over 10% in many cases).

Let us use $\mathbf{S}_{\mathrm{Ba}}$ and $\mathbf{S}_{\mathrm{BO}}$ to represent the temporal speedup achieved by AdADMM(BO,Ba) and AdADMM(BO,BO) (eliding "-F16") respectively to reach the best objective of JOPT, and similarly use $\mathbf{I}_{\mathrm{Ba}}$ and $\mathbf{I}_{\mathrm{BO}}$ to represent the objective improvement at the final converged point. Table 1 shows that between AdADMM(BO,BO) and AdADMM(BO,Ba), the latter provides *significantly higher speedups*, but the former provides higher additional improvement in the final objective. This demonstrates ADMM's flexibility, for example, allowing choice between faster or more improved solution.

Dataset	$\mathbf{S}_{\mathrm{Ba}}$	$\mathbf{S}_{\mathrm{BO}}$	$\mathbf{I}_{\mathrm{Ba}}$	$\mathbf{I}_{\mathrm{BO}}$
Bank8FM	$10\times$	$2\times$	0%	5%
CPU small	$4\times$	$5\times$	0%	5%
fri-c2	$153\times$	$25\times$	56%	64%
PC4	$42\times$	$5\times$	8%	13%
Pollen	$25\times$	$7\times$	4%	3%
Puma8NH	$11\times$	$4\times$	1%	1%
Sylvine	$9\times$	$2\times$	9%	26%
Wind	$40\times$	$5\times$	0%	5%

Table 1: Comparing ADMM schemes to JOPT(BO), we list the speedup $\mathbf{S}_{\mathrm{Ba}}$ & $\mathbf{S}_{\mathrm{BO}}$ achieved by AdADMM(BO,Ba) & AdADMM(BO,BO) respectively to reach the best objective of JOPT, and the final objective improvement $\mathbf{I}_{\mathrm{Ba}}$ & $\mathbf{I}_{\mathrm{BO}}$ (respectively) over the JOPT objective. These numbers are generated using the aggregate performance of JOPT and AdADMM over 10 trials.

6 Conclusions

Posing the problem of joint algorithm selection and HPO for automatic pipeline configuration in AutoML as a *formal mixed continuous-integer nonlinear program*, we leverage the ADMM optimization framework to *decompose* this problem into *2 easier sub-problems*: (i) black-box optimization with *a small set of continuous variables*, and (ii) a combinatorial optimization problem involving *only Boolean variables*. These sub-problems can be effectively addressed by existing AutoML techniques, allowing ADMM to solve the overall problem effectively. This scheme also *seamlessly incorporates black-box constraints* alongside the black-box *objective*. We empirically demonstrate the flexibility of the proposed ADMM framework to leverage existing AutoML techniques and its effectiveness against open-source baselines.

References

Asuncion, A., and Newman, D. 2007. UCI ML Repository.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *JMLR* 13(Feb):281–305.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *NeurIPS*.

Bischl, B., et al. 2017. OpenML benchmarking suites and the OpenML100. *arXiv:1708.03731*.

Boyd, S., et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning* 3(1):1–122.

Calders, T., and Verwer, S. 2010. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery* 21(2):277–292.

Caruana, R.; Niculescu-Mizil, A.; Crew, G.; and Ksikes, A. 2004. Ensemble selection from libraries of models. In *ICML*.

Chen, B.; Wu, H.; et al. 2018. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 402–409. ACM.

Conn, A. R.; Scheinberg, K.; and Vicente, L. N. 2009. *Introduction to derivative-free optimization*. SIAM.

Drori, I.; Krishnamurthy, Y.; et al. 2018. Alphad3m: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*.

Durand, A., and Gagné, C. 2014. Thompson sampling for combinatorial bandits and its application to online feature selection. In *AAAI Workshops*.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In *NeurIPS*.

Friedler, S. A., et al. 2019. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 329–338. ACM.

Hong, M., and Luo, Z.-Q. 2017. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming* 162(1):165–199.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In *International Conference on Learning and Intelligent Optimization*. Springer-Verlag.

Jamieson, K., and Talwalkar, A. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*.

Komer, B.; Bergstra, J.; and Eliasmith, C. 2014. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*.

Kotthoff, L., et al. 2017. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *JMLR*.

Larson, J.; Menickelly, M.; and Wild, S. M. 2019. Derivative-free optimization methods. *Acta Numerica* 28:287–404.

Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR* 18(185):1–52.

Liu, S.; Kailkhura, B.; Chen, P.-Y.; Ting, P.; Chang, S.; and Amini, L. 2018. Zeroth-order stochastic variance reduction for nonconvex optimization. In *NeurIPS*.

Liu, S.; Ram, P.; et al. 2019. An ADMM Based Framework for AutoML Pipeline Configuration. *arXiv:1905.00424*.

Mohr, F.; Wever, M.; and Hüllermeier, E. 2018. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning* 107(8-10):1495–1515.

Olson, R. S., and Moore, J. H. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on AutoML*.

Pedregosa, F.; Varoquaux, G.; et al. 2011. Scikit-learn: Machine learning in Python. *JMLR*.

Rakotoarison, H.; Schoenauer, M.; and Sebag, M. 2019. Automated Machine Learning with Monte-Carlo Tree Search. In *IJCAI*.

Sabharwal, A., et al. 2016. Selecting near-optimal learners via incremental data allocation. In *AAAI*.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*.

Thornton, C.; Hoos, H. H.; Hutter, F.; and Leyton-Brown, K. 2012. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *arXiv:1208.3719*.

Vanschoren, J. 2018. Meta-learning: A survey. *arXiv:1810.03548*.

Wang, D., et al. 2019. Human-AI Collaboration in data science: Exploring data scientists' perceptions of automated AI. *arXiv preprint arXiv:1909.02309*.

Yang, C.; Akimoto, Y.; Kim, D. W.; and Udell, M. 2019. OBOE: Collaborative filtering for AutoML model selection. In *KDD*.