iSklearn: automated machine learning with irace

Carlos Vieira*, Adelson de Araújo†, José E. Andrade Júnior* and Leonardo C. T. Bezerra*
*IMD, Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil
Email: {carlosv, dadojunior}@ufrn.edu.br, leobezerra@imd.ufrn.br
†University of Twente, The Netherlands
Email: a.dearaujo@utwente.nl

***Abstract*—Automated algorithm engineering has become an important asset for academia and industry. irace, for instance, is an algorithm configurator (AC) that has successfully designed effective algorithms for optimization problems. The major advantage of irace is combining learning and parallelization, but no fully-functional automated machine learning (AutoML) system powered by irace has yet been proposed. This is rather striking, as some of the most relevant existing AutoML tools are powered by ACs, of which irace is one of the most effective examples.**

In this work, we propose iSklearn, an irace-powered AutoML system. Our proposal improves existing work applying an AC to engineer a machine learning (ML) pipeline. First, our configuration space represents a minimalist pipeline template, demonstrating that simpler pipelines can be competitive with elaborate approaches (e.g. ensembles). Second, our configuration setup improves the application of AC-based AutoML to time series (TS) problems, and is more flexible to fit other applications.

We evaluate iSklearn on three major ML domains, namely computer vision (CV), natural language processing (NLP), and TS. Results prove competitive to AUTOSKLEARN, a state-of-the-art AutoML system also built on scikit-learn. Furthermore, the compositions of the pipelines devised vary with the problem domain and dataset considered, providing further evidence for the need of AutoML tools. We conclude our investigation ablating through the proposed configuration space and setup to understand their impact on the performance of iSklearn.

I. INTRODUCTION

Automated machine learning (AutoML, [1]) is a relevant research effort that seeks to reduce two of the most significant costs in the machine learning (ML) industry. The first is the human resource cost, as the rising demand for ML engineers has led to a shortage of supply, greatly increasing personnel costs. The second, equally important, is the computational/environmental cost, as state-of-the-art ML models leave a significant carbon footprint [2] and require expensive to maintain infrastructures. In an effort to alleviate this cost, cloud computing has become the method-of-choice for companies, specially when deep learning (DL) techniques are required [3].

The best-established AutoML approaches come from the fields of algorithm configuration [4], [5], neuroevolution [6]–[8], and neural architecture search [9]. These techniques have often achieved remarkable results in several different application domains. Indeed, the breakthroughs in the field have led AutoML tools to focus not only on tabular datasets, but also on challenging domains such as computer vision (CV), natural language processing (NLP), and time series (TS) analysis. Though the most striking results in those fields rely on DL, their computational cost and carbon footprint are often prohibitive and so AutoML approaches based on simpler ML pipelines remain an important alternative. Another benefit of simpler pipelines is interpretability, in particular pipeline composition.

This work investigates AutoML in this context of tackling challenging domains using (relatively) simple ML pipelines. To do so, we propose a fully-functional AutoML system dubbed iSklearn, which represents a contribution for at least three reasons. First, iSklearn is powered by irace [10], one of the best-established algorithm configurators that has repeatedly demonstrated its performance in several automated algorithm engineering tasks and/or application domains [11], [12]. In particular, irace bridges the advantages of other configurators already employed in AutoML, namely the parallelization from HyperBand [13] and the model-based learning from SMAC [14]. Yet, so far no AutoML initiative powered by irace could be identified in the literature, likely because irace has been originally proposed for configuring optimization algorithms.

The second way in which our work represents a contribution is related to pipeline simplicity. Specifically, we define the configuration space of iSklearn as a machine learning vanilla template comprising a standard ML pipeline architecture, with a relevant set of algorithmic components available for (i) feature engineering and (ii) prediction. Our template is built on scikit-learn, representing the common options a practitioner has at hand when first working with ML. An instantiation of this template represents a pipeline where all components were jointly selected and configured. Compared to other AutoML tools based on DL or ensembles, the interpretability of the pipelines proposed is greatly improved.

The final contribution of our work concerns the configuration setup, i.e., how to model machine learning datasets as instances of an optimization problem. In more detail, we generalize the approach adopted by other algorithm configurators [4], [5], enabling, for instance, the proper application of ML pipelines to TS analysis datasets. Given the number of application domains that present some form of temporal dependency, this improvement greatly increases the applicability of configurator-based AutoML tools.

We evaluate the effectiveness of the proposed configuration space and setup on different application domains, including CV, NLP, and TS analysis. As expected, the configured pipelines are nearly always more effective than naively selecting an algorithm with its suggested default parameters. More importantly, even if the focus of this work is not benchmarking AutoML approaches, we produce ensembles using AUTOSKLEARN [5] to serve as baseline for our comparison. Remarkably, the differences in performance between models is minimal, and often the

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

simpler pipelines from iSklearn outperform the ensembles from AUTOSKLEARN. Another important observation is how pipeline composition varies not only as a function of the application domain, but also of the dataset being tackled. Effectively, this provides further evidence for the need of AutoML tools that can be employed in academia and industry.

The last part of our investigation ablates the configuration space and setup to understand their individual importance to the effectiveness of iSklearn and produce insights that may indicate relevant future work. To assess configuration space, we use SMAC to engineer pipelines from our template. Since SMAC is also the configurator that powers AUTOSKLEARN, the comparison between pipelines and ensembles helps isolate the impact of the minimalist template we propose. Results show that pipelines configured from iSklearn by SMAC are also competitive to the AUTOSKLEARN-generated ensembles, which confirms the benefits of our proposed configuration space. Finally, to ablate our configuration setup, we assess the impact of configuration budget, maximum cutoff time, and the sampling generalization we propose. Results show that these factors interact with the given domain and dataset, and that adequately adjusting each factor can be helpful. Furthermore, in the context of TS problems, our proposed sampling generalization makes iSklearn results outperform ensembles from AUTOSKLEARN for all datasets considered.

The remainder of this work is structured as follows. In Section II, we briefly discuss background on automated machine learning, algorithm configuration, and irace. Section III describes the configuration space and setup we propose for iSklearn, which we assess in Section IV. The last part of our investigation is given in Section V, where we ablate the configuration space and setup. Finally, we conclude and discuss future work possibilities in Section VI.

II. BACKGROUND

Automated machine learning (AutoML) research spans over a diverse set of fields. In this section, we first highlight the most relevant families of approaches from the literature. Next, we deepen our discussion on algorithm configuration, the field to which our approach belongs. Finally, we detail irace to add context to our proposal in Section III.

A. Automated machine learning

Automated machine learning (AutoML) is a fast-expanding field, largely due to the joint efforts from industry and academia. Yet, its seminal works date from over two decades ago (e.g., [15]), and range from the research on evolutionary algorithms (EAs) to the research on neural networks. Indeed, the more recent, abrupt expansion in AutoML is largely due to the groundbreaking results achieved by deep learning algorithms [3]. Effectively, these results have both (i) drawn the attention of the industry to the efficacy of machine learning, and (ii) demonstrated the challenge in designing and configuring predictors. Since AutoML initiatives stem from diverse research fields, an exhaustive review is beyond the scope of this paper. Below, we focus our discussion on the most important aspects of the main families of approaches from the literature:

Algorithm configuration approaches often model the AutoML task as the CASH problem, i.e., *combined algorithm selection and hyperparameter optimization* [4]. In summary, such approaches attempt to select a predictor from a portfolio while simultaneously configuring their associated hyperparameters. The search is conducted by a configurator, typically a heuristic optimization algorithm. The best known works in this field concern Auto-WEKA [4] and AUTOSKLEARN [5].

Neural architecture search focuses on the design and configuration of neural networks [9]. The AutoML problem is generally modeled as a reinforcement learning problem, where a searching neural network must identify the best target neural network according to a given reward function. Given its focus on neural networks, this research field offers a number of interesting approaches to better address this context. Indeed, a number of recent breakthroughs in deep learning research are directly related to this field.

Neuroevolution [6]–[8] is closely related to the research on both algorithm configuration and neural architecture search. In particular, neuroevolution approaches use EAs to evolve the topology and/or parameters of neural networks. The most emblematic algorithm from this field is likely NEAT [6], and the interest in this topic has been strongly stirred by the industry [7]. More recently, multi-objective neuroevolution has targeted efficacy and efficiency as prototypical, conflicting objectives to be simultaneously optimized [8].

A few important works do not fit our taxonomy. Yet, we believe our brief review is important for the discussion on algorithm configuration we conduct next. More importantly, it highlights the diversity in approaches that have been proposed over the years, and how challenging it would be to properly benchmark them.

B. Algorithm configuration

Algorithm configuration is currently better understood as automated algorithm engineering, a growing field that comprehends different tasks, such as selection, configuration, design, and analysis [11]. The prominent results obtained by these approaches span over a wide range of application domains, such as decision [16], optimization [11], control [17], and, more recently, machine learning [4], [5].

In the context of AutoML, approaches to CASH generally combine a configuration (i) *space* and (ii) *setup*. A configuration space is defined in terms of a meta-description of an algorithmic portfolio, e.g. a template or a grammar. Examples are the templates proposed for Auto-WEKA [4] and AUTOSKLEARN [5], respectively built on WEKA and scikit-learn. In addition, since the predictors and other components of a machine learning pipeline present hyperparameters, a configuration space must also comprise the valid domains for their configuration.

Complementarily, the configuration setup is the definition of an experimental setup to evaluate candidates. In more detail, AutoML approaches powered by configurators search the configuration space by sampling candidate configurations. Navigation of the search space is guided by the performance of these candidates, and hence a proper definition of a configuration setup is critical to the performance of the AutoML approach. The most important factors regarding setup concern the (i) problem samples provided; (ii) performance metric adopted, and; (iii) resource limits allowed. Further discussion on each of these topics is provided in Section III.

Given the role of configurators, it is important to remark the contrast between the large number of configurators proposed in the algorithm configuration literature and the small number of configurators adopted in AutoML research. One likely explanation is their background, since many configurators were proposed in the context of search optimization and their application to machine learning is non-trivial. In general, algorithm configurators can be classified as *model-based* or *model-free* [12]. The former attempt to identify promising regions of the configuration space by modeling the relationship between hyperparameters and performance. This is the case with SMAC [14], which powers Auto-WEKA and AUTOSKLEARN; and irace, not yet applied to AutoML. Alternatively, model-free applications identify promising configurations using stochastic local search or randomized sampling (e.g, HyperBand, [13]).

C. *irace*

irace is an *estimation of distribution algorithm* (EDA), a family of EAs that combine search aspects from both optimization and learning. At each iteration, irace mantains a population of candidates, a dual-nature representation of configurations. Specifically, each candidate c_i alive during a given iteration comprises a set of probability distributions $P_i(\phi_j)$, one distribution for each hyperparameter ϕ_j of the target algorithm. Complementarily, each candidate c_i is evaluated based on a concrete configuration sampled from $P_i(\phi_j)$ when the candidate is first created.

The key idea in EDAs is to evolve these probability distributions, which irace accomplishes by (i) *racing* the concrete configurations alive in a given iteration, and; (ii) updating the probability distributions between iterations based on the surviving candidates. In the racing mechanism, configurations are iteratively run on problem instances and the worst-performing ones are discarded as enough statistical evidence is collected. Through racing, irace is able to promote *sharpening*, i.e., candidate configurations that perform best get discarded last, meaning there is more available evidence by the time irace must decide between configurations that perform similarly well.

An iteration finishes when either a minimum number of surviving candidate configurations or a maximum resource limit is reached. Between iterations, irace produces offspring candidates from the surviving candidates. An offspring candidate presents probability distributions that have been adjusted to better reflect the concrete configuration from its parent, given the good performance of that concrete configuration in the previous iteration. To reduce variability between iterations, offspring candidates are first evaluated on the same problem instances used to evaluate the surviving candidates of the previous iterations. Finally, irace may partially restart the population to prevent premature convergence.

The effectiveness of irace has been repeatedly demonstrated on diverse application domains [11]. Besides its effectiveness, the best feature irace brings is the flexibility in the definition of the configuration space and setup. Concerning the former, irace was one of the first configurators able to couple with numerical and categorical hyperparameters, and with their dependencies. Regarding the configuration setup, irace has been applied to domains as diverse as dynamic and multi-objective optimization. Yet, these effective results were a product of carefully, manually designed setups. This is likely the reason why no application of irace to the context of automated machine learning can be identified in the literature.

D. *Contrasting algorithms*

Though irace is the focus of this work, we provide further discussion on how it compares to the two most relevant algorithm configurators employed in the ML literature, i.e., SMAC [14] and HyperBand [13]. Both racing and sharpening are standard techniques in the algorithm configuration literature [12], and each configurator proposes a different approach to achieve them. SMAC is a sequential model-based approach, i.e., a surrogate model is used to reduce the number of actual evaluations performed. When racing, candidate configurations are evaluated based on the surrogate model, and sharpening is performed by comparing the configuration obtained from the model search with the best current configuration. SMAC and irace are alike in being model-based, as previously discussed. Yet they differ in that SMAC is inherently sequential, whereas irace uses parallelization to a large extent.

HyperBand [13] represents a model-free paradigm, using a massively parallel racing of candidate configurations. Sharpening is promoted by probing configurations with reduced budgets, i.e., training configurations for increasingly longer periods. Since model-based learning is not employed, the candidate configurations in HyperBand are completely independent, which allows for a parallelization level unmatched by model-based configurators. However, sharpening is only effective in HyperBand as long as the dataset investigated presents a strong correlation between performance for varying training budgets, which is not always the case in ML [18].

In a sense, irace represents a compromise paradigm between SMAC and HyperBand, as it uses model-based learning but is still parallelizable. Though other approaches bridging these properties have been proposed recently [19], no fully-functional AutoML tool based on ML pipelines powered by any such configurator can be identified thus far. More strikingly, even a fully-functional HyperBand-based system is not yet available.[1] In the next section, we seek to fill this gap, proposing an AutoML system using irace as configurator.

III. ISKLEARN: A FULLY-FUNCTIONAL AUTOML SYSTEM

As previously discussed, an AutoML approach based on algorithm configuration comprises (i) a configuration space, including a meta-description of a portfolio and valid domains for the hyperparameters of the algorithms that comprise it, and; (ii) an experimental setup to evaluate candidates, enabling the AutoML approach to select/configure an algorithm that is high-performing for the input dataset. In this section, we propose a configuration space and setup, which we will assess on a set of relevant ML datasets in Section IV.

```
S -> Preprocessing Prediction
Preprocessing -> Scaling FE | none
Scaling -> True | False
FE -> Selection | Selection Extraction
      | Extraction Selection | Extraction
Prediction -> Scaling Predictor
```

Algorithm 1: iSklearn template described as a grammar.

TABLE I: Algorithms considered for each template component.

Component	Algorithms	Conditions
Extraction	SVD	sparse datasets
	PCA, ICA, DL	otherwise
Selection	*univariate*, *multivariate*	
multivariate	DT, RF, SVM	classif. & regres.
	LR	regression
Predictor	LR, DT, RF, SVM, kNN, MLP, AB	classif. & regres.
	LR	regression

(LR stands for linear or logistic regression, depending on the task.)

A. Configuration space

The configuration space proposed in this work is modeled as a template, given in Algorithm 1. The template models a standard ML pipeline architecture, comprising two high-level components. The first, `Preprocessing`, represents a feature preprocessing stage, performed over the data prior to fitting the model. In contrast to Auto-WEKA and AUTOSKLEARN, our template offers a single choice of data preparation, namely `Scaling` through standardization. Our rationale for this vanilla version is that data preparation is a fairly important part of the data science process, and that attempting to automatically engineer the whole process would exceed the scope of this work. For this reason, the datasets we later adopt for evaluating pipelines are subject to manual preparation, as we will detail in supplementary material[2].

Besides `Scaling`, the `Preprocessing` component comprises feature engineering (`FE`). Available options are feature `Selection` and `Extraction`, which can be used simultaneously and, if so, in any order. We provide these possibilities as a manual pipeline design typically selects between these choices, but an automated design might benefit from using both. Algorithmic options for these components are given in Table I and further detailed in the supplementary material.

In the case of feature `Extraction`, options are dimensionality reduction algorithms that vary depending on the characteristics of the dataset provided. Options for component `Selection` are organized into groups, namely *univariate* and *multivariate*. Univariate feature selection retrieves a certain percentile of features based on a given scoring function computed between each feature and the target variable. iSklearn provides the most common functions available in scikit-learn, detailed in the supplementary material for brevity. Conversely, multivariate selection fits a feature importance model using a predictor, and retrieves only the most relevant. Table I lists the predictors available for multivariate selection, which we choose due to their balance between efficacy and efficiency when used with their suggested default parameters. Furthermore, multivariate selection can be performed recursively, using the *recursive feature elimination* (RFE) approach.

The second high-level component of iSklearn is `Prediction`, where model fitting is actually performed. For this component, we consider a representative subset of the estimators available in scikit-learn, listed in Table I. Our rationale with this subset is that it represents most families of relevant approaches, such as generalized linear models, trees, manifold learning, neural networks, and ensembles. Options available vary according to the task nature, as iSklearn is able to cope with both classification and regression. Finally, being heuristic algorithms, these predictors present hyperparameters of their own, which we expose for configuration. The details on the hyperparameters exposed for each predictor and their valid domains are given as supplementary material.

B. Configuration setup

As previously discussed, the most important factors comprising a configuration setup concern the (i) problem samples provided; (ii) performance metric adopted, and; (iii) resource limits allowed. Below, we discuss each of these topics, starting from problem sampling, where our contributions lie;

Problem samples. AutoML through algorithm configuration requires three sampling levels. The top level evaluates the generalization of the AutoML approach; following the literature, we adopt holdout for this stage [4], [5]. We refer to this split as *seen* and *test*, since the configurator is never provided test samples. Conversely, the bottom level sampling evaluates the generalization of a candidate configuration on a subset of problem samples. In the literature, Auto-WEKA and AUTOSKLEARN once again adopt holdout. Conversely, we generalize this sampling level and test 5-fold cross-validation instead. Though this change could increase the computational cost to train a single candidate configuration, it improves the application of the AutoML systems to time series (TS) problems. More precisely, walk-forward cross-validation is a better fit for training time series models than holdout.

Finally, the mid-level sampling provides variability to the configuration process, defining what samples are seen by the bottom level sampling when a candidate configuration must be evaluated. An extreme alternative is to provide each candidate evaluation the whole dataset, at a significant computational cost. In addition, the variability between runs from a single configuration on the whole dataset is expected to be reduced for some predictors, even if the dataset is shuffled every time. The other extreme alternative would be to provide candidate configurations as little samples as possible. Though fast, irace would hardly see enough samples to avoid overfitting.

In the literature, Auto-WEKA and AUTOSKLEARN partition *seen* samples into k folds for this mid-level sampling. For clarity, we will dub these folds *meta-folds*. Concretely, every time the configurator must evaluate a candidate, it is provided a single meta-fold, which is then subject to the bottom level sampling. Here, we also split the dataset into k meta-folds, but again we generalize the sampling and provide the configurator p meta-folds at a time. The evaluation is performed using the bottom sampling on each of the p meta-folds independently,

[1] Such a system has been proposed in [20], but it is not publicly available in a fully-functional form.

[2] https://github.com/carlosemv/irace-automl-cec2021

and results are averaged by the configurator. We remark that both our mid-level and bottom level sampling methods may be more expensive than those used by Auto-WEKA and AUTOSKLEARN. Our goal is to provide more samples to the training phase of each individual model, particularly when the dataset considered is not significantly large or there exists some level of class imbalance. Further discussion on how to set k and p is provided as supplementary material.

Performance metric. The choice of performance metric is more related to the problem one wants to address than to the nature of the configurator considered. For regression, typical choices include R^2 or (R)MSE. For classification, configuration for balanced datasets may adopt the traditional accuracy metric, whereas configuration for unbalanced datasets may benefit more from Matthews correlation coefficient (MCC).

Resource limits. iSklearn does not present a priori concerns with memory resources. By contrast, time is a critical factor in two major aspects. The first is the budget provided to iSklearn, which irace uses to compute the maximum number of iterations and the number of candidate configurations to be sampled at each iteration. Also, one must set a cutoff time for the evaluation of a given candidate, or else a very expensive model fitting may compromise the configuration.

Altogether, the configuration space and setup proposed in this section render iSklearn a fully-functional AutoML system, which we evaluate in the next section.

IV. ASSESSING PIPELINES CONFIGURED FROM ISKLEARN

To evaluate our proposal, we take ensembles produced by AUTOSKLEARN as baseline. As discussed, the focus of our investigation is on computer vision (CV), natural language processing (NLP), and time series (TS) prediction problems. The Auto-WEKA and AUTOSKLEARN works used tabular and CV datasets as benchmarks. Among the CV ones, the most relevant are the MNIST and CIFAR-10 classification datasets. However, since the differences in performance between algorithms is generally rather small for MNIST, we replace it with the more challenging Fashion MNIST (FMNIST). We additionally consider SVHN, a house number recognition problem. Regarding the remaining application domains, we included: (i) three NLP classification datasets, namely LMRD for sentiment analysis, and Reuters and AGNews for topic classification, and; (ii) two TS analysis regression datasets, concerning crime incidence prediction [21] in the cities of Boston, MA, EUA, and Natal, RN, Brazil. Further information on these datasets is given in the supplementary material, particularly the details on manual data preparation, sampling, and resource limits for AUTOSKLEARN.

Since irace and SMAC are heuristic algorithms, we report results from 10 runs of both iSklearn and AUTOSKLEARN on each dataset. A single run of iSklearn on a given dataset has a configuration budget of 2 000 experiments. An experiment is defined as evaluating a candidate configuration on a given instance (tuple of p meta-folds). In the assessment done in this section, we set $p = 1$ to isolate the effect of changing the bottom-level sampling; alternative p values will be assessed in Section V-B. Maximum runtime for each experiment is set to 10 minutes. If the evaluation of a configuration exceeds this limit, the configuration is penalized so that irace may discard it at the end of the iteration. After pipelines are configured from iSklearn, we validate them on the test samples and report mean performance and variance over the 10 runs. Following the literature, both configuration and validation are guided by accuracy for classification tasks and R^2 for regression tasks.

We start by analyzing pipeline structures selected by irace for each applicaton domain, given as sunburst plots in Figure 1. FE_1 and FE_2 depict the possibility of using `Selection` and `Extraction` simultaneously – if only one of them is used, it is depicted as FE_1. The two most important insights we observe are the differences in composition between domains and also datasets. In order, composition for NLP, CV, and TS use increasingly more preprocessing components. While pipelines from all domains use `Extraction` a few times, TS pipelines additionally always use `Selection`. Use of different prediction options increases in a different order, with NLP being the domain where the least different alternatives are chosen, and CV the one where the most are. Regarding NLP, 29 out of the 30 pipelines devised adopt LR. This algorithm is also most frequently chosen for TS datasets, followed by SVM and MLP. Finally, we notice that more complex predictors such as ensemble methods are only once selected (AB for SVHN).

We further assess pipeline composition as a function of the given dataset. For brevity, sunburst plots for each dataset are provided as supplementary material. First, a different predictor is mostly selected for each of the CV problems, namely LR for CIFAR-10, SVM for FMNIST, and kNN for SVHN. In addition, preprocessing is increasingly adopted in these datasets, in this order. Concerning TS problems, LR is frequently selected for the Boston dataset, followed by SVM, whereas for the Natal dataset SVM, LR, and MLP are uniformly distributed. Preprocessing patterns for TS problems are little affected by the particularities of each dataset. Finally, in NLP, preprocessing is only adopted more than once for Reuters, mostly through `Extraction`.

The compositions discussed above constitute strong evidence for the benefits of AutoML tools over manual selection and configuration. In addition, the simplicity of our template greatly improves the understanding of the pipelines selected. Finally, some of these observations might suggest the maximum runtime we fixed for each experiment for feasibility affects the composition of the pipelines. This possibility is further explored in Section V-B. Nonetheless, the comparison of final scores for each dataset given in Table II confirms that these pipelines are indeed high-performing w.r.t. the provided configuration space and setup. Accuracy scores given for classification problems, as well as R^2 scores given for regression problems, are to be maximized. The best value per dataset is given in boldface.

As expected, pipelines devised by iSklearn and ensembles devised by AUTOSKLEARN present better performance than default models. The only exception is the Natal dataset, for which LR outperforms both AutoML tools. Comparing iSklearn and AUTOSKLEARN, it is remarkable that pipelines from the former are competitive with the ensembles from the latter. Indeed, the only datasets for which we see a non-negligible gap between the performance of the models devised by the two AutoML approaches are the three CV ones. For CIFAR-

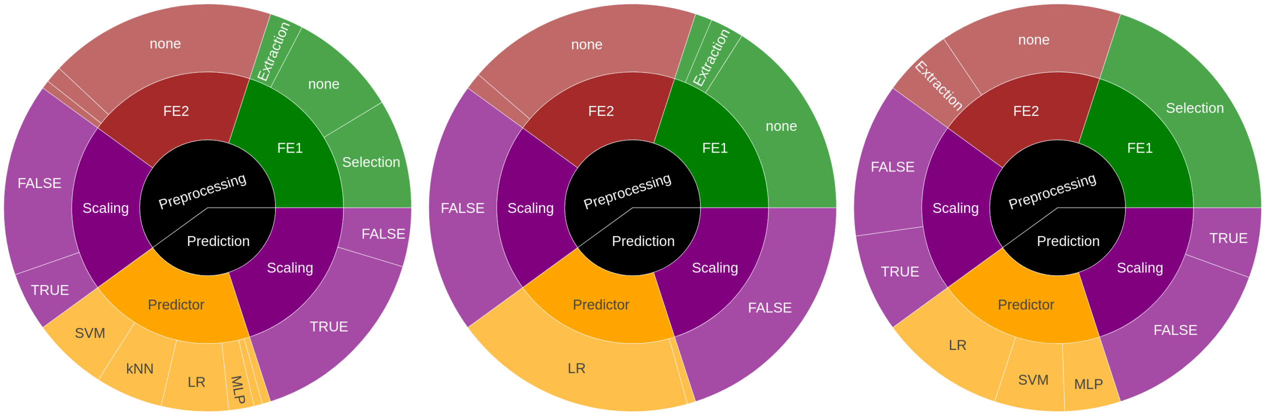


Fig. 1: Pipeline composition for the different domains considered. From left to right: CV, NLP, and TS.

TABLE II: Accuracy (%) and R^2 scores (mean ± stand. deviation) for each dataset. The best algorithm per dataset is highlighted in boldface.

Task	*Domain*	*Dataset*	KNN	DT	RF	AB	MLP	SVM	LR	iSklearn	AutoSklearn
Classification	*CV*	*CIFAR-10*	32.98	26.78±0.27	34.61±0.35	31.08	34.21±0.37	20.44	30.24	43.96±6.95	**50.55**±9.44
		FMNIST	85.54	79.05±0.17	85.44±0.14	54.25	84.87±0.52	10.02	80.12	**88.12**±2.24	79.88±5.48
		SVHN	46.83	42.17±0.22	56.51±0.38	22.78	19.58±0.00	0.00	23.94	**58.02**±15.52	53.25±10.41
	NLP	*LMRD*	67.20	70.23±0.15	73.00±0.64	80.31	86.29±0.09	63.29	88.31	**88.32**±0.19	88.01±0.19
		Reuters	77.74	70.41±0.38	69.32±0.77	47.55±0.14	80.59±0.19	36.20	79.07	81.57±0.35	**82.57**±1.01
		AGNews	90.20	75.97±0.22	84.06±0.41	68.24	91.07±0.19	72.30	91.39	**91.74**±0.07	91.57±0.9
Regression	*TS*	*Natal*	0.94	0.9134±0.0022	0.9538±0.0047	0.9185±0.0089	0.9547±0.0016	0.9389	**0.9575**	0.9538±0.0022	0.9549±0.0087
		Boston	0.9576	0.9428±0.0009	0.9634±0.0004	0.9440±0.0023	0.9728±0.0021	0.8454	0.9743	**0.9751**±0.0001	0.9707±0.0006

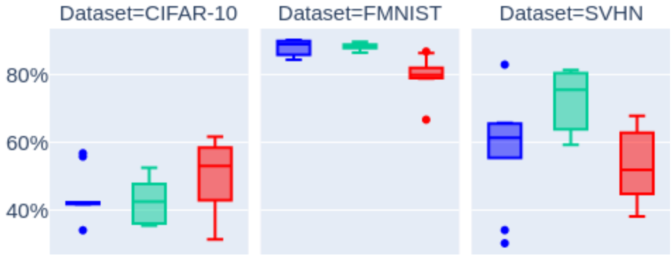


Fig. 2: Accuracy comparison between pipelines configured by irace (blue) and SMAC (green) from iSklearn and ensembles configured by SMAC from AutoSklearn (red).

10, ensembles present higher mean, though at the cost of higher variance as well. For FMNIST and SVHN, it is the pipelines that achieve higher mean, though again at the cost of higher variance for the latter. Altogether, these results evidence the difficulty posed by computer vision problems when deep learning is not adopted. One likely explanation is the computational cost incurred by sample dimensionality, which becomes more critical in runtime-constrained scenarios.

From a practical point of view, most of the pipelines engineered in this section could be deployed into a real-world production setup, due to their simplicity and performance. The only exceptions are CIFAR-10 and SVHN, for which both the pipelines from iSklearn and ensembles from AutoSklearn still require improvements to produce estimators with a reasonable level of effectiveness. In the next section, we investigate the effects of our proposed configuration space and setup in the performance of iSklearn.

V. Ablating iSklearn

Results from the first part of our investigation validated our irace AutoML proposal as a competitive approach in terms of efficacy. We next ablate iSklearn to understand how the proposed configuration space and setup affect its performance. We start with a configuration space analysis, where we assess the benefits of having a minimalist template. Later, we appraise different configuration setups, exploring the idea of the generalized mid-level sampling and providing guidelines for the application of iSklearn to other problems.

A. Comparing configuration spaces

To assess the benefits of having a simpler template in terms of efficacy of the pipelines produced, we use SMAC [14] to configure pipelines from our template. Since SMAC is the configurator powering AutoSklearn, the comparison given in Figure 2 helps us isolate the effects of configurator and templated adopted. We focus this investigation on CV datasets as they were the most challenging for both AutoML approaches.

Boxplots for the different datasets indicate that the performance of pipelines configured from iSklearn by SMAC are more similar in performance to pipelines configured from iSklearn by irace than to the ensembles configured from AutoSklearn by SMAC itself. More repetitions of these experiments would be required to understand if the differences in distributions between irace and SMAC results are consistent, or if the outliers observed are fluctuations in the experiments. Nonetheless, the comparison between pipelines and ensembles confirms that our proposed minimalist template is a contribution not only in terms of simplicity and interpretability, but also as to efficacy. Furthermore, it evidences the generality of the configuration space and setup proposed, as they can be coupled

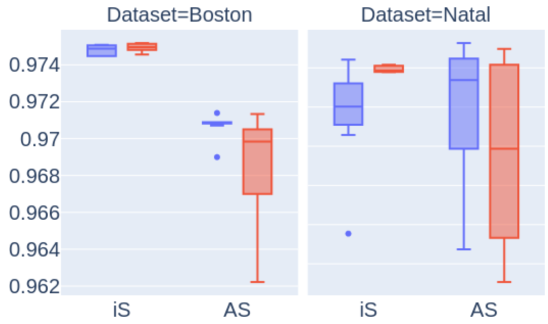


Fig. 3: R^2 comparison of iSklearn and AUTOSKLEARN on TS datasets using different configuration setups. Blue: **regular**; red: **TM**.

with any configurator that supports numerical, categorical, and conditional parameters, such as SMAC.

B. Alternative configuration setups

Results discussed in Section IV indicated the efficacy of the configuration setup adopted, but some relevant questions need to be further investigated. We start with the analysis of the TS datasets. Our goal is to understand the impact of the different sampling generalization approaches we propose, in particular the bottom-level sampling cross-validation approach that should suit TS problems better than the traditional holdout. Figure 3 gives boxplots of experiments where we compare the original setup adopted in Section IV (dubbed **regular**) and a setup where we increase the number of meta-folds to $p = 3$ and total cutoff time to 15min (dubbed **TM**, for *triple meta-fold*). For brevity, a discussion on the balance between the number of meta-folds and total cutoff time is provided as supplementary material. The **regular** setup is given in blue, whereas the **TM** setup is given in red. Since it is not possible to configure the number of meta-folds used for bottom-level sampling in AUTOSKLEARN, results provided for ensembles under **TM** reflect only the increase in total cutoff time.

As previously discussed, the best-performing AutoML tool for the **regular** setup varies as a function of the TS dataset considered. However, under the **TM** setup the pipelines from iSklearn outperform the ensembles from AUTOSKLEARN for all datasets. These results are explained by two factors. First, the performance of iSklearn pipelines is improved by the inclusion of more meta-folds, even if proportionally the cutoff time for the evaluation of each meta-fold is halved. Second, the performance of AUTOSKLEARN ensembles is reduced by the increase in cutoff time. One likely explanation is that the larger training time enables SMAC to select more complex ensembles, which tend to overfit in a time series setup using holdout.

To further investigate the impact of cutoff time and number of meta-folds, we extend our experimental design. Besides those factors, we also investigate the impact of increasing the total number of experiments irace is allowed to perform. Table III depicts the setups we produce from these factors, where **regular** serves as baseline. With **20m**, **5k** and **TM 30m** we independently investigate increased (i) cutoff time, (ii) configuration budget, and (iii) number of meta-folds, respectively. In addition, since a total cutoff time of 30m may be impractical depending on the computational setup available, we investigate with **TM** and **TM 5k** whether halving the total cutoff time would be an option. For instance, the TS analysis above adopted **TM**, where halving the total cutoff time was compensated by the increased number of meta-folds.

TABLE III: Summary of the six configuration setups considered in this section. Regular (**reg.**) stands for the configuration setup assessed in the previous section, used here as baseline.

setup	**reg.**	**20m**	**5k**	**TM 30m**	**TM**	**TM 5k**
p	1	1	1	3	3	3
budget	2000	2000	5000	2000	2000	5000
cutoff	10m	20m	10m	30m	15m	15m

Boxplots given in Figure 4 depict accuracy scores for CV (left column) and NLP (right column) datasets, on which we focus due to the larger improvement margins observed in Section IV. However, margins as large as CIFAR-10 make results for this dataset outliers, and hence the following discussion focuses on the remaining ones. For a given plot, setups are ordered as in Table III. We first remark that the differences in performance among setups is much more significant for CV datasets than for NLP ones. Apart from that, the most important factor observed was cutoff time, as only for Reuters we do not observe improvements under setup **20m**. For this dataset, none of the remaining factors consistently helped, though adopting more samples worsened performance slightly. For the remaining datasets, increasing either the number of experiments (**5k**) or the number of meta-folds (**TM**) improved performance. The only exception was FMNIST, for which increasing the number of meta-folds changed performance to a negligible rate. Finally, reducing the cutoff time did not compensate for the increased number of meta-folds (**TM**), though the possibility of more experiments (**TM 5k**) alleviated the loss in performance. Once again, FMNIST was an exception, since results under **TM 5k** were even better than for **TM 30m**.

VI. CONCLUSION

Automated machine learning (AutoML) is a growing research field both as to the number of tools available and to the results they help achieve. One of its main goals is to bridge non-experts and the specialized knowledge underlying successful ML applications. Another, equally important, is to reduce the computational/environmental cost currently incurred by the ML industry. Overall, AutoML approaches stem from research fields as diverse as algorithm configuration, neuroevolution, and neural architecture search. Yet, the proposal and assessment of these approaches need to be aware of their cost.

In this work, we have conducted an experimental investigation with these two goals in mind, attempting to maximize the number of insights observed from our work while keeping the number of experiments constrained. In this context, we have empirically demonstrated how irace can be used to configure pipelines that outperform the predictors that comprise it, considering several relevant application domains, such as computer vision, natural language processing, and time series analysis. Even if the goal of the paper is not to benchmark AutoML approaches, it is remarkable that the

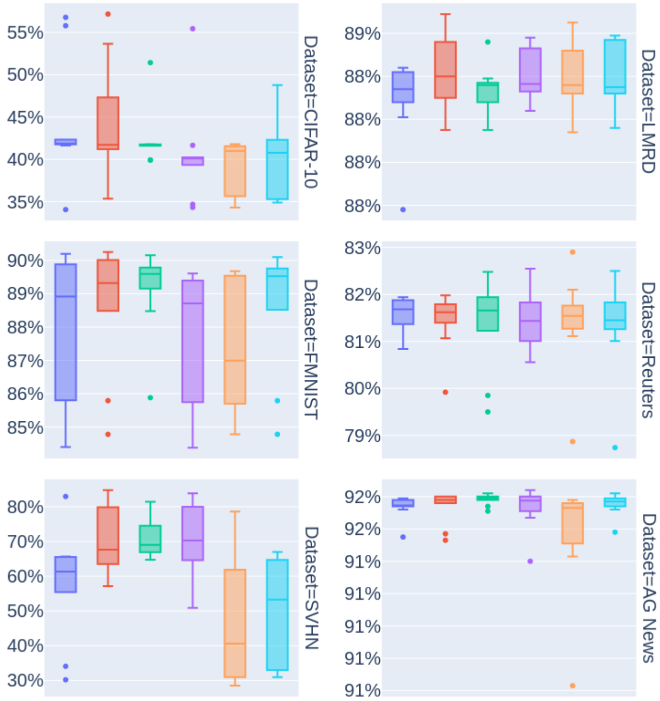


Fig. 4: Accuracy comparison on CV (left column) and NLP (right column) datasets under alternative configuration setups. Within each plot, setups are ordered as in Table III.

pipelines engineered from iSklearn displayed competitive performance w.r.t. more elaborate ensembles produced by the well-known AUTOSKLEARN. Furthermore, our minimalist configuration space proved helpful both as to effectiveness and to interpretability of pipeline composition. Finally, we assessed different configuration setups and discussed the different options a practitioner can employ to further improve efficacy.

Our work aimed to be comprehensive, and so many of the insights we observed deserve future investigation. The first is pushing further the idea of environment-aware AutoML research. Indirectly, this effort has been driven by the desire to bring machine learning models to mobile hardware. One emblematic example is multi-objective neuroevolution, which evolves models to simultaneously optimize these two goals. Concerning algorithm configuration AutoML approaches, irace is an important asset in this task, as it has been effectively employed for multi-objective configuration [11], [12].

A second path for future work is to investigate machine learning-specific setups for irace. Here, we have observed that variations to the setup adopted affect the performance of the pipelines produced, specially for computer vision problems. A likely promising direction is to mimic deep learning training, where models are repeatedly exposed to the same dataset. In the context of iSklearn, this would translate into resuming training when a candidate pipeline needs to be evaluated on the same meta-fold again. The main challenge with this alternative is to balance effectiveness with the costs it incurs.

Finally, while our investigation has focused on AutoML effectiveness, it is also imperative to pursue robustness. More precisely, in this investigation we have sought to include problem datasets from diverse applications domains. Yet, every dataset we have considered was initially subject to some extent of manual data preparation. Other AutoML tools, such as AUTOSKLEARN, aim to automate even this part of the process. Considering the importance of data preparation, its relevance to predicting performance, and the reduced computational cost of this stage, making available a data preparation module to be coupled with iSklearn is an important path of future work.

ACKNOWLEDGMENT

This work is partially supported by UFRN in the context of the Smart Metropolis Project. This research was also supported by NPAD/UFRN.

REFERENCES

[1] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Springer, 2019.

[2] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in *ACL*. ACL, 2019, pp. 3645–3650.

[3] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms,” in *KDD*. ACM, 2013, pp. 847–855.

[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, “Efficient and robust automated machine learning,” in *NeurIPS*, C. Cortes et al., Eds. Curran Associates, Inc., 2015, pp. 2962–2970.

[6] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[7] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” in *ICML*. JMLR.org, 2017, pp. 2902–2911.

[8] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, “NSGA-Net: Neural architecture search using multi-objective genetic algorithm,” in *GECCO*. New York, NY, USA: ACM, 2019, p. 419–427.

[9] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *J. of Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[10] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, “The irace package: Iterated racing for automatic algorithm configuration,” *Oper. Res. Perspec.*, vol. 3, pp. 43–58, 2016.

[11] L. C. T. Bezerra, “A component-wise approach to multi-objective evolutionary algorithms: from flexible frameworks to automatic design,” Ph.D. dissertation, IRIDIA, École polytechnique, ULB, Belgium, 2016.

[12] L. C. Bezerra, M. López-Ibáñez, and T. Stützle, “Automatic configuration of multi-objective optimizers and multi-objective configuration,” in *High-Performance Simulation-Based Optimization*. Springer, 2020, pp. 69–92.

[13] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *The J. of Mach. Learn. Res.*, vol. 18, no. 1, pp. 6765–6816, 2017.

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *LION*. Springer, 2011, pp. 507–523.

[15] E. Ronald and M. Schoenauer, “Genetic lander: An experiment in accurate neuro-genetic control,” in *PPSN*. Springer, 1994, pp. 452–461.

[16] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, “SATenstein: Automatically building local search SAT solvers from components,” *Artif. Intell.*, vol. 232, pp. 20–42, 2016.

[17] K. Hasselmann, F. Robert, and M. Birattari, “Automatic design of communication-based behaviors for robot swarms,” in *ANTS*. Springer, 2018, pp. 16–29.

[18] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, “Nas-bench-101: Towards reproducible neural architecture search,” in *ICML*. PMLR, 2019, pp. 7105–7114.

[19] S. Falkner, A. Klein, and F. Hutter, “BOHB: Robust and efficient hyperparameter optimization at scale,” in *ICML*. PMLR, 2018, pp. 1437–1446.

[20] S. C. N. das Dôres, C. Soares, and D. Ruiz, “Bandit-based automated machine learning,” in *BRACIS*. IEEE, 2018, pp. 121–126.

[21] O. Kounadi, A. Ristea, A. Araujo, and M. Leitner, “A systematic review on spatial crime forecasting,” *Crime Sci.*, vol. 9, pp. 1–22, 2020.

[22] E. R. Girden, *ANOVA: Repeated measures*. Sage, 1992, no. 84.

[23] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[24] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[25] ——, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] G. E. Hinton, "Connectionist learning procedures," in *Machine Learning, Volume III*. Elsevier, 1990, pp. 555–610.

[27] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[28] A. Franzin, L. P. Cáceres, and T. Stützle, "Effect of transformations of numerical parameters in automatic algorithm configuration," *Optimization Letters*, pp. 1–13, 2017.

[29] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[31] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[32] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[33] D. Lewis, "Reuters-21578 text categorization test collection, distribution 1.0," *http://www.daviddlewis.com/resources/testcollections/reuters21578/*, 1997.

[34] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *arXiv preprint arXiv:1509.01626*, 2015.

APPENDIX A
iSKLEARN CONFIGURATION SPACE

Algorithmic options for feature extraction in iSklearn's template (given in Table I) are dimensionality reduction algorithms, and depend on the density of the given dataset: for sparse datasets, *truncated singular value decomposition* (SVD) is the single option; otherwise, either *principal component analysis* (PCA), *independent component analysis* (ICA), or *dictionary learning* (DL) might be used.

As for `Selection`, we have the *univariate* approach, which retrieves a certain percentile of features based on a scoring function: either the ANOVA F-values [22] or the Mutual Information [23] between features and target. This proportion of features retrieved is a hyperparameter itself, and can have any integer value between 1 and 99 (inclusive). Multivariate selection can also be performed recursively, using the *recursive feature elimination* (RFE) approach. As for the *multivariate* approach, a feature importance model is fit using any of the following predictors: *decision trees* (DT, [24]), *random forests* (RF, [25]), and *support vector machines* (SVM), for either classification or regression datasets, and; *linear regression* (LR), for regression datasets only. These were choosen due to their balance between efficacy and efficiency when used with their suggested default parameters, as we do when using these predictors in the context of feature selection, since the configuration of nested models is a complex aspect to be addressed in future work.

For `Prediction`, the second main component in this template, a subset of all estimators in iSklearn are considered. This subset includes the ones already mentioned for multivariate selection (DT, RF, SVM, and LR), as well as *k-nearest neighbors* (kNN), *multilayer perceptron* (MLP, [26]), *AdaBoost* (AB, [27]), and *logistic regression* (LR[3]). These represent most families of ML prediction algorithms commonly used, namely generalized linear models, trees, manifold learning, neural networks, and ensembles. Most of these are available for either classification or regression tasks when possible, as distinguished in Table I.

TABLE IV: Configuration space of the hyperparameters associated with the algorithms comprising iSklearn.

Algorithm	Parameter	Space
KNN	*k*	$\{1, \dots, 100\}$
	weights	$\{\text{uniform}, \text{weighted}\}$
AB	$N_{\text{estimators}}$	$\{2, \dots, 500\}$
	learning rate	$[0.01, 1.0]$
	loss function	$\{\text{linear}, \text{square}, \text{exponential}\}$
DT & RF	*max features*	$[0.01, 1.0]$
	min samples leaf	$[0.01, 0.5]$
	max depth	*none* or $\{2, \dots, 50\}$
	classification criterion	$\{\text{gini}, \text{entropy}\}$
	regression criterion	$\{\text{MSE}, \text{MAE}\}$
RF	$N_{\text{estimators}}$	$\{2, \dots, 1000\}$
SVM	C	$[1e{-}4, 1e5]$
	kernel	{*linear, polynomial, RBF, sigmoid*}
	γ	$[1e{-}5, 10]$
	polynomial degree	$\{1, \dots, 10\}$
MLP	*hidden layers*	{1, 2, 3}
	$nodes_1$, $nodes_2$, $nodes_3$	$\{3, \dots, 500\}$
	activation function	{*identity, logistic, tanh, ReLU*}
	optimizer	{*lbfgs, sgd, adam*}
	L2 penalty	$[1e{-}5, 1e4]$
	initial learning rate	$[1e{-}6, 1]$
	learning rate	{*constant, invscaling, adaptive*}
Logistic Regression	C	$[1e{-}4, 1e5]$
	optimizer	{*newton-cg, lbfgs, liblinear, sag, saga*}
	multi-class	{*ovr, multinomial*}
	maximum iterations	$\{100, \dots, 1000\}$
	penalty	{*l1, l2*}
	dual formulation	{*true, false*}

Nearly all algorithms present in the template described expose associated hyperparameters that need to be configured by irace. Those hyperparameters, as well as their selected domains, are detailed in Table IV, grouped by algorithm. Further information on the hyperparameters of each estimator considered can be found below. For further specifics on these hyperparameters, we refer to the original papers for these algorithms, as well as to the documentation for scikit-learn.

KNN: The number *k* of neighbors is provided as a hyperparameter, as well as whether to use uniform or distance-based weights for each neighbor.

AdaBoost (AB): For both regression and classification, irace must configure the number of base estimators and the learning rate. For regression tasks, three different loss functions are considered, used to update the weights after each iteration.

Decision trees (DT): irace needs to configure (i) the proportion of the features that can be used to build the tree, and (ii) the minimum number of samples required for a leaf node (given as a fraction of the total number of samples). Optionally, irace may configure a maximum depth value

[3] LR stands for Linear Regression in the case of regression tasks, and Logistic Regression in the case of classification tasks

for the tree. The criterion used to measure the quality of a split is also provided as a hyperparameters, with different possible values for classification and regression tasks – gini or entropy for classification, and mean squared error (MSE) or mean absolute error (MAE) for regression.

Random forests (RF): The configuration space for random forests is a superset of the space for decision trees. Besides all of the hyperparameters from DT, irace must also configure the number of estimators (trees) to be used.

Support vector machines (SVM): irace must configure the penalty parameter of the error term (C), the kernel to be used, and its associated hyperparameters γ (in the case of non-linear kernels). Also, for a polynomial kernel, the degree of the polynomial function is configured.

Multilayer perceptron (MLP): irace must configure the number of hidden layers and the number of neurons in each layer. The activation function, solver (or optimizer), and L2 penalty must also be configured. For solvers SGD and Adam, the initial learning rate is also configured and, specifically for SGD, a learning rate schedule is chosen.

logistic regression (LR): For this algorithm, the parameters tuned are: the inverse of regularization strength (C); the optimizer (or solver) used in the optimization problem; whether to fit a binary problem for each label (*ovr*), otherwise the loss minimised is the multinomial loss fit across the entire probability distribution (*multinomial*); maximum number of iterations for the optimizers; the norm used in the penalization; and whether to use dual or prime formulation.

Note that in implementing this configuration space the logarithmic transformation was adopted, being applied to real-valued hyperparameters that present a very large range of values, following relevant findings in this topic [28]. As an example, the L2 penalty hyperparameter in MLPs is then modeled as a surrogate parameter $\alpha \in [-5, 4]$, which is then mapped by iSklearn during candidate evaluation back to L2 $\in [10^{-5}, 10^{4}]$.

Appendix B
Sampling configuration setup

Choosing values for k and p (see discussion on problem sampling in Section III-B) is a manual design stage that needs to account for the characteristics of the particular dataset one wants to address. The choice of k should avoid meta-folds with too little samples, particularly if there is some level of class unbalance or a low sample-per-class ratio. Regarding p, a large value would greatly increase the computational cost for evaluating a single candidate once. Ideally, p should be set to a small value. Yet, if k is small, having a value of p too close to the value of k would mitigate the benefits of the proposed approach, since nearly the same meta-folds would be evaluated every time.

Appendix C
Datasets

Fashion MNIST (FMNIST) [29] is a computer vision classification dataset. It contains a 60000-sample training set and a 10000-sample testing set, each sample being a grey level, 28x28 pixels, centered image of an article of clothing. These can be one of ten different items, constituting the ten possible labels: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

Street View House Numbers (SVHN) [30] is a real-world dataset of house numbers obtained from Google Street View images. It contains images of centered digits, targetting the real-world problem of digit recognition in natural scene images. These images are a little larger than those of FMNIST, with 32x32 pixels. It is also a somewhat larger dataset than FMNIST, with the default split comprising 73257 samples for training and 26032 for testing.

CIFAR-10 [31] is a dataset of 32x32-pixel natural scene images, like SVHN. These depict subjects of 10 different classes, each class having 6000 samples: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The default split comprises 50000 training images and 10000 test images.

LMRD [32] is a NLP dataset used for binary sentiment analysis of online movie reviews collected from IMDB[4]. It consists of 50000 highly polar reviews split evenly into training and testing sets.

Reuters [33] is a collection of documents collected from the Reuters news agency, used for categorical topic classification. It contains 46 possible labels, and the training and testing sets contain 8982 and 2246 samples, respectively.

AGNews [34] is a collection of news articles collected from different sources, and like Reuters is also used here for topic classification, constructed by taking the 4 largest classes from the original corpus. Its train set contains 12000 samples, and its testing set 7600.

Natal is a real-world private dataset of crimes obtained from the state police department in RN, Brazil. It consists of crime counts aggregated monthly in a grid of squares (1km x 1km). A total of 98553 crimes from February 2017 to November 2018 were utilized. The training set consisted of 80% of the samples and 20% were used for testing. Note this dataset is not made publicly available, but was provided under a confidentiality agreement.

Boston is a real-world public dataset of crimes obtained from Boston's open data hub[5]. It consists of crime counts aggregated monthly in a grid of squares (1km x 1km). A total of 301269 crimes from July 2016 to September 2019 were utilized. Training/testing split was the same as that for Natal.

None of the three CV datasets were subject to any special data preparation, except for flattening the images when those were provided as 2-D arrays. Data preparation for NLP data consisted only of feature extraction through the scikit-learn TF-IDF vectorizer, using default parameters. As for the TS datasets, data preparation consisted of generating the input features, which were 24 autoregressive lags (i.e., $\{y_{t-k} \mid k \in \{1, 2, ...24\}\}$), with the regression target being the

[4] http://www.imdb.com

[5] https://data.boston.gov/

crime count in the following month (y_t).

For CV and NLP datasets, their *seen* samples were divided into $k = 20$ meta-folds (see discussion on problem sampling in Section III-B) when the number of samples in that set was equal to, or exceeded, 50000, and into $k = 10$ meta-folds otherwise. This approach was chosen in order to minimize risk of too few samples being provided for proper model training, considering each meta-fold would still undergo 5-fold cross-validation. Under this rule, all three CV datasets were split into 20 meta-folds, and all NLP datasets into 10. For the TS datasets, we chose to fix $k = 10$ even with a larger number of samples, since preliminary experiments showed the time-series cross-validation method strongly disfavored training with the smaller number of samples provided when $k = 20$. See also Appendix B for further considerations on choosing a value for k.

APPENDIX D
AUTOSKLEARN RESOURCE LIMITS

Each run of AUTOSKLEARN was given a memory limit of 12Gb for the machine learning algorithm. A cutoff time was given for training each model equal to the one given for irace to finish an experiment in iSklearn (10 minutes for the regular setup). As iSklearn bases total budget on number of experiments (and not runtime), while AUTOSKLEARN only provides an option for total time limit, the mean time taken by iSklearn over 10 runs of each setup/dataset was used as the time budget for the equivalent AUTOSKLEARN runs.

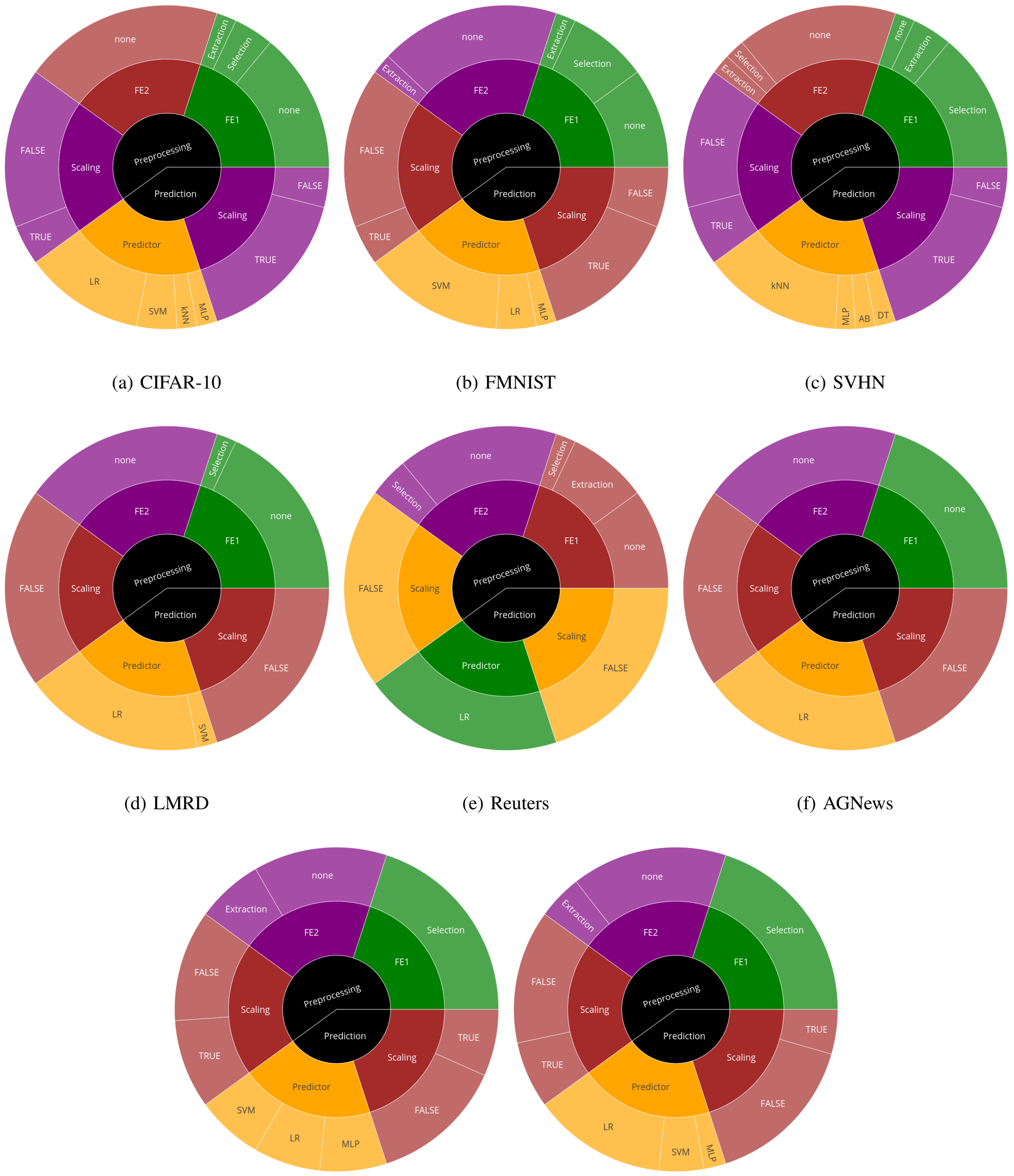
APPENDIX E
ISKLEARN PIPELINE COMPOSITION PER DATASET

Pipeline structures selected by irace for each dataset are given as sunburst plots in Figure 5. First, a different predictor is mostly selected for each of the CV problems: for CIFAR-10 6 out of 10 pipelines used LR, followed by SVM twice, and once each for kNN and MLP; for FMNIST, SVM was predominant, chosen in seven runs, with LR chosen twice and MLP once; and for SVHN, it was kNN, also chosen 7 times, with the remaining three equally divided between MLP, AB, and DT. In addition, preprocessing is increasingly adopted in these datasets, in this same order: while only 3 pipelines used any kind of preprocessing for CIFAR-10, only one did not for SVHN.

Concerning TS problems, LR is frequently selected (6 times) for the Boston dataset, followed by SVM (twice) and MLP (once), whereas for the Natal dataset SVM, LR, and MLP are uniformly distributed. Preprocessing patterns for TS problems are little affected by the particularities of each dataset, with `Selection` being universally adopted and additionally `Extraction` in some cases. Finally, as already discussed in this paper, LR is predominant in the pipelines for the NLP datasets. Furthermore, preprocessing is rarely used here, with only Reuters adopting any kind of preprocessing more than once, mostly through `Extraction`.

APPENDIX F
VARIANCE ANALYSIS FOR REGULAR SETUP

We provide variance analysis for the results shown in Table II through boxplots, given in Figure 6.



(a) CIFAR-10

(b) FMNIST

(c) SVHN

(d) LMRD

(e) Reuters

(f) AGNews

(g) Natal

(h) Boston

Fig. 5: Pipeline composition for each dataset.

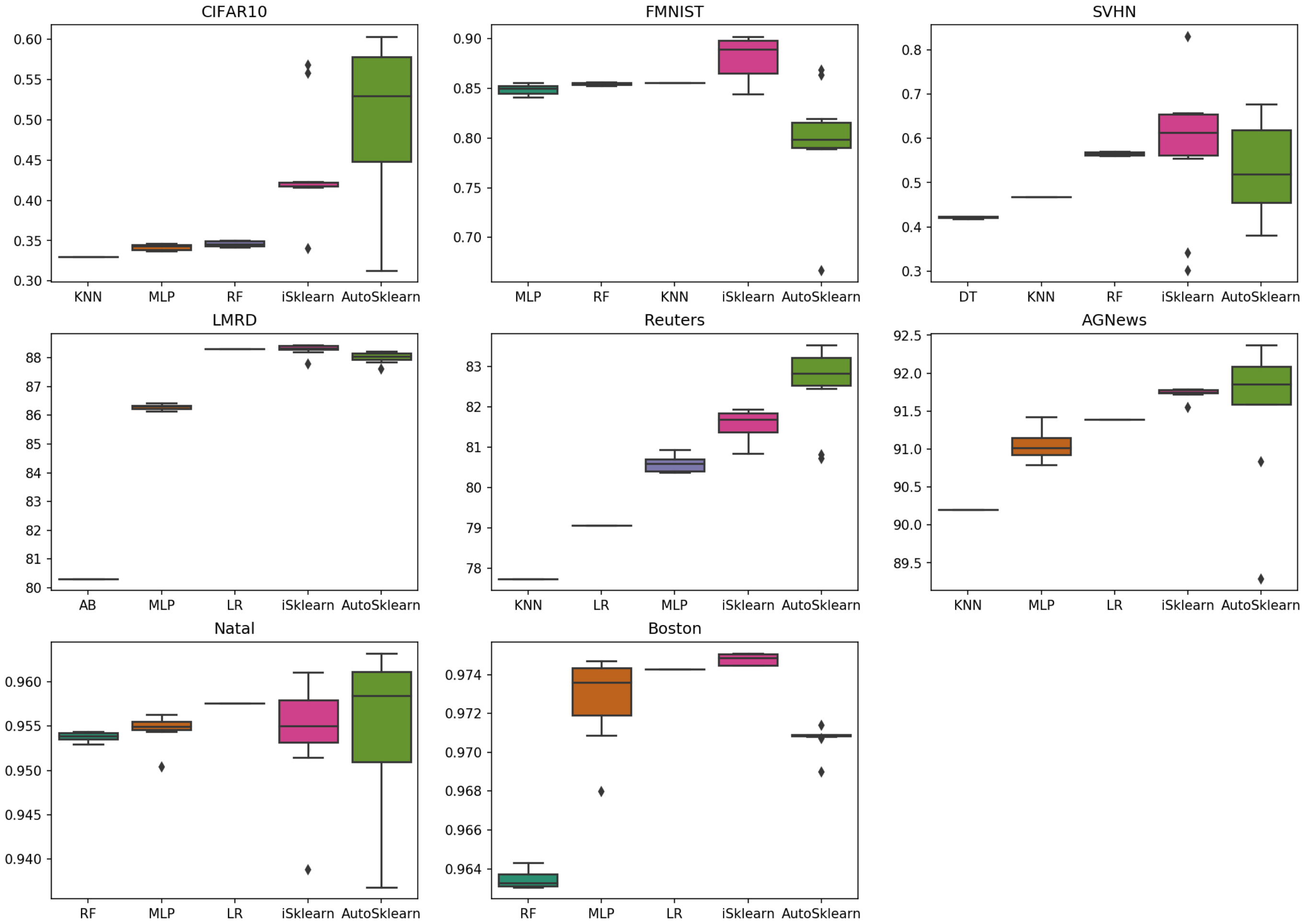


Fig. 6: Accuracy (%) and R^2 score comparison between pipelines configured by iSklearn, ensembles produced by AUTOSKLEARN, as well as the three best-performing default models for each dataset.