AutoML with Monte Carlo Tree Search

Herilalaina Rakotoarison, Michèle Sebag

AutoML with Monte Carlo Tree Search

Herilalaina Rakotoarison HERI@LRI.FR and **Michèle Sebag** SEBAG@LRI.FR
TAU, CNRS - INRIA - LRI, University of Paris-Saclay

Abstract

The sensitivity of machine learning (ML) algorithms w.r.t. their hyper-parameters and the difficulty of finding the ML algorithm and hyper-parameter setting best suited to a given dataset has led to the rapidly developing field of automated machine learning (AutoML), at the crossroad of meta-learning and structured optimization. Several international AutoML challenges have been organized since 2015, motivating the development of the Bayesian optimization-based approach `Auto-Sklearn` (Feurer et al., 2015) and the Bandit-based approach Hyperband (Li et al., 2016). In this paper, a new approach, called *Monte Carlo Tree Search for Algorithm Configuration* (`Mosaic`), is presented, fully exploiting the tree structure of the algorithm portfolio and hyper-parameter search space. Experiments (on 133 datasets of the OpenML repository) show that `Mosaic` performances match that of `Auto-Sklearn`.

Keywords: Model selection, Hyper-parameter optimization, Monte Carlo Tree Search, AutoML

1. Introduction

The progress of the machine learning (ML) field is witnessed as an explosion of applications is seen in all fields, from computer vision (Krizhevsky et al., 2012; Karpathy et al., 2014; Ren et al., 2015; He et al., 2016) to recommendation systems (Ricci et al., 2011; Ekstrand et al., 2011). However, the diversity of ML algorithms and their sensitivity w.r.t. their hyper-parameters make it a difficult task to find the approach best suited to the application at hand. This difficulty makes all the more serious the announced shortage of machine learning experts in the next decade. The problem of automatically finding the best setting for an ML problem, referred to as AutoML, has attracted interest since the late 1980s (see Brazdil and Giraud-Carrier (2018) for a survey), with several AutoML international challenges organized in the last decade (Guyon et al., 2015, 2018). These challenges have primed the design and deployment of numerous automated machine learning platforms (AutoMLP in the following) (Feurer et al., 2015; Li et al., 2016; Chen et al., 2018; Olson et al., 2016).

The contribution of the paper is to tackle AutoML as a one-player game, adapting Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006) to the exploration of the complex and structured decision space Λ encompassing all ML options (pre-processing, feature selection, model selection and optimization hyper-parameters). Formally, the proposed approach called *Monte-Carlo Tree Search for AlgorIthm Configuration* (`Mosaic`), tackles the following optimization problem:

$$\text{Find } \lambda^* \in \operatorname*{argmin}_{\lambda \in \Lambda} L(\lambda, D_{train}, D_{valid}), \tag{1}$$

where Λ is the set of all hyper-parameter settings, loss function L measures the learning performance on the validation set D_{valid} of the model learned on the training set D_{train}.

A main difficulty of the above optimization problem lies in the fact that the optimization objective (Eq. 1) usually is a non-convex and expensive one, all the more so when large datasets and/or complex model spaces are considered.

This paper is organized as follows. Section 2 describes and discusses the state-of-the-art of AutoML. For the sake of self containedness, the formal background in Monte Carlo tree search is given in section 3. A detailed overview of `Mosaic` is given in section 4. The experimental validation of the approach is given in section 5 and the paper concludes with some perspectives for further research in section 6.

2. Related Work

The domain of automated machine learning involves several subproblems: i) selection of the ML algorithm most appropriate to the current dataset; ii) optimization of the associated hyper-parameters; iii) optionally, data preprocessing.

The field of hyperparameter optimization is currently dominated by Bayesian optimization methods (Snoek et al., 2015; Hutter et al., 2011; Bergstra et al., 2011). These methods proceed by estimating the conditional probability $p(f|\lambda)$ of the performance f given a hyper-parameter setting λ, and exploit this probabilistic model to select the next most promising candidate λ^*; after evaluating the performance associated to this candidate, the model is updated and the search is iterated. Other approaches handle the expensive optimization problem by considering cheap approximations thereof, e.g. sub-sampling the data or using a surrogate objective (estimating the performance) (Swersky et al., 2014; Li et al., 2016; Klein et al., 2016). Yet another approach is the Bandit-based approach Hyperband (Li et al., 2016), tackling hyperparameter optimization as a resource allocation task.

At the current state-of-the-art in AutoMLPs are `Auto-Sklearn` (Feurer et al., 2015) and `Auto-Weka` (Thornton et al., 2013). They rely on the Bayesian-based approach SMAC. `Auto-Sklearn` involves two extra components: meta-learning components to warm-start the Bayesian optimization procedure, and model ensemble strategy described in Caruana et al. (2004) to build a more robust classifier. Auto-Sklearn involves 15 classifiers, 14 feature preprocessing methods, and 4 data preprocessing methods. Other AutoMLPs based on evolutionary algorithms have recently emerged: TPOT (Olson et al., 2016) and AutoStacker (Chen et al., 2018), with the caveat that they might need many evaluations to yield a good result, thus with a high computational cost.

Interestingly, MCTS has been applied to feature selection (Gaudel and Sebag, 2010): the so-called Fuse (Feature UCT Selection, details of UCT in section 3) extends MCTS to tackle the combinatorial optimization of selecting a best subset of features, cast as a one-player game. Wistuba (2017) also proposed to tune deep neural network with MCTS but only on limited list of parameters. In this work, we would like to extend those approaches to the full autoML problem.

3. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) extends the celebrated Multi-armed Bandit algorithm (Auer, 2002) to tree-structured search spaces.

Multi-armed Bandit Let k arms or options be defined, the i-th arm being associated with an (unknown) probabilistic bounded reward, e.g. a Bernoulli variable of probability μ_i. Based on the arms selected in the past and the associated reward, the goal is to find an arm selection policy such that it maximizes the expected cumulative reward, or equivalently, minimizes the regret (difference with the oracle strategy, always selecting the arm with highest μ_i). The UCB1 algorithm (Auer, 2002) defines such an arm selection policy, selecting in each time step the arm maximizing the following:

$$\text{Select} \quad \arg\max_i \{\mu_i + \sqrt{\frac{2\log n}{n_i}}, \} \tag{2}$$

where μ_i is the sampled average obtained by playing arm i, n_i is the number of times arm i has been chosen, n is the number of iterations so far.
The MCTS algorithm iterates over four phases (Chaslot et al., 2008): selection, expansion, playout and backpropagation:

- **Selection**: In each node of the tree, the child node is selected after a Multi-armed Bandit strategy, e.g. the UCT (Upper Confidence bound applied to Trees) algorithm (Kocsis and Szepesvári, 2006) selects the child node such that it maximizes:

  $$\mu_i + c\sqrt{\frac{\log N}{n_i}}, \tag{3}$$

  where μ_i is the value of node i, N is the number of times the parent node was visited, n_i is the number of times node i is visited, and c is a problem-dependent constant which balances exploration and exploitation.

- **Expansion**: The algorithm adds one or more nodes to the tree. This node corresponds to the first encountered position that was not added in the tree.

- **Playout**: When reaching the limits of the visited tree, a roll-out strategy is used to select the options until reaching a terminal node and computing the associated reward.

- **Backpropagation**: The reward value is propagated back, i.e. it is used to update the value associated to all nodes along the visited path up to the root node.

Remarks. A Multi-armed Bandit problem defines a reinforcement learning (RL) problem with a single state. MCTS relaxes this single-state limitation and is known as an efficient RL approach, at the core of the RL breakthroughs related to the Atari games (Mnih et al., 2013) and the game of Go (Silver et al., 2016).

4. Monte-Carlo Tree Search for Algorithm Configuration

After formalizing the AutoML process as a Markov Decision Process problem, this section gives an overview of the proposed MOSAIC approach and discusses its components.

4.1. AutoML as a Markov Decision Process

In the following, the dataset D at hand is assumed to be fixed. AutoML thus aims to find a 4-tuple $m \in \mathcal{M}$ where m consists of (1) a pre-processing method b, (2) the hyper-parameter setting of b, noted p_b (p_b in $\mathbb{R}^q$, and q depends on b), (3) a learning algorithm a and (4) the hyper-parameter setting p_a of algorithm a (where p_a is in $\mathbb{R}^d$, and d depends on a). The goal is to find the best tuple m^* in the sense of the performance (e.g. predictive accuracy) of the learned model on distribution $\mathcal{D}$ underlying the finite sample D. AutoML problem defines an ill-posed and expensive structured optimization problem, which is tackled as a sequential decision problem for tractability.

In MOSAIC, the order of the choice follows the domain knowledge: the first choice is that of the pre-processing method b; the choice of pre-processing hyper-parameters p_b depends on b, the choice of the learning algorithm a and the choice of a hyper-parameters p_a itself depends on a.

Finally, `Mosaic` follows the standard MCTS procedure, except for two specific issues: handling of continuous hyper-parameter values (section 4.2) and overcoming of the combinatorial nature of the optimization (section 4.3). The MOSAIC strategy is described as follows:

- The tree-path from the root node to an internal node represents a partial solution (incomplete tuple);
- In each non-terminal node, the choice of a child node is conducted using the standard UCB rule in the finite case (see section 4.2 for the continuous case); When reaching the limits of the visited tree, a roll-out random strategy is applied until reaching a terminal node and computing the associated reward (more section 4.3).
- The reward associated to a full tree-path (terminal node, complete solution m) is computed.
- After the evaluation of a terminal node, the reward is backpropagated to update the value in each node of the visited tree path; this value will support the choice among the child nodes in the next tree-walk.

4.2. Continuous hyper-parameter values

As said, a main difficulty of the AutoML optimization problem is its mixed, continuous and discrete, search space. Multi-armed Bandit approaches have been extended to continuous search spaces (see in particular Wang et al. (2009) and Bubeck et al. (2011)), notably exploiting dichotomic divisions of the search space. Another approach has been taken here for two reasons. A first motivation is to enforce the scalability of the approach w.r.t. the dimension of the continuous hyper-parameter space preventing the use of regular decomposition of the search space (exponential in its dimension) along a grid. A second motivation is that, after (Bergstra et al., 2011), the uniform sampling of continuous hyper-parameter space outperforms the grid-based approaches, everything else being equal. Accordingly, the first value selected for a continuous hyper-parameter is uniformly selected in its range. The number of values considered is gradually extended with the number of visits n_i to the choice node, along the Progressive Widening strategy (Couëtoux et al., 2011). Formally, when the

integer value of $\sqrt{n_i}$ is incremented, a new value (uniformly selected in the value range), is considered.

4.3. Details of `Mosaic` Implementation

In its current version, the search space of MOSAIC is built upon the `scikit-learn` machine learning environment (Pedregosa et al., 2011), with 13 data preprocessing methods and 17 classifiers. An extensive list of available data pre-processing methods / machine learning algorithms with their respective hyper-parameters is presented in Appendix A.

`Mosaic` approach relies on two remarks. On the one hand, as said the uniform exploration of the configuration search space is an excellent strategy except for its huge computational cost. On the other hand, as shown by (Li et al., 2016), there is (only) a factor two between the computational resources required by state-of-the-art Bayesian optimization SMAC (Hutter et al., 2011) and TPE (Bergstra et al., 2011), and those of a pure random search needed to reach the same performance level. `Mosaic` accordingly attempts to get the best of both worlds: using the MCTS randomized exploration strategy and search space customized to reflect the domain structure, and its ability to achieve exploitation and intensify the search in the neighborhood of the most promising solutions.

The main limitation of MCTS for AutoML is related to the non-smoothness of the performance landscape. Typically, if a pre-processing algorithm is first evaluated with a poor algorithm (considering the dataset at hand), this pre-processing will not be further considerd for a long time. For this reason, each time a new node is created, k playout ($k = 3$ in the experiments) are launched and the maximum reward (over the k runs) is backpropagated to the ancestor nodes[1].

5. Experimental Results

All AutoMLP approaches can be combined with an ensemble strategy, retaining the best solutions found along the search and averaging them. Focusing on the comparative evaluation of the MCTS and SMAC search strategies, this section will be restricted to the evaluation of the vanilla `Mosaic` and `Auto-Sklearn` (without the ensembling and meta-learning options). The same configuration as in Feurer et al. (2015) is used, considering 133 (binary and multi-class) datasets[2] and averaging the results over 10 independent runs. For each dataset, `Mosaic` and `Auto-Sklearn` are ranked according to their average test performance, and the rank averaged over all datasets of `Mosaic` and `Auto-Sklearn` is finally reported. The performance metric, used to rank two approaches on a fixed dataset, is the *Balanced accuracy*.

Training setup : Hyper-parameters of `Auto-Sklearn` are all set to default. For `Mosaic`, k (the number of evaluations to do when new node is expanded) is set to 3. `Mosaic` searches over a structured space of 173 hyper-parameters instead of 110 for `Auto-Sklearn`. The time budget for each run is set to one hour with six minutes timeout for one hyper-parameter

1. Another strategy, left for further work, is to use specific policy based on extreme value estimates (Achab et al., 2017; Bubeck et al., 2013).
2. Specifically, Feurer et al. (2015) uses 140 datasets but we were unable to get 7 of them.

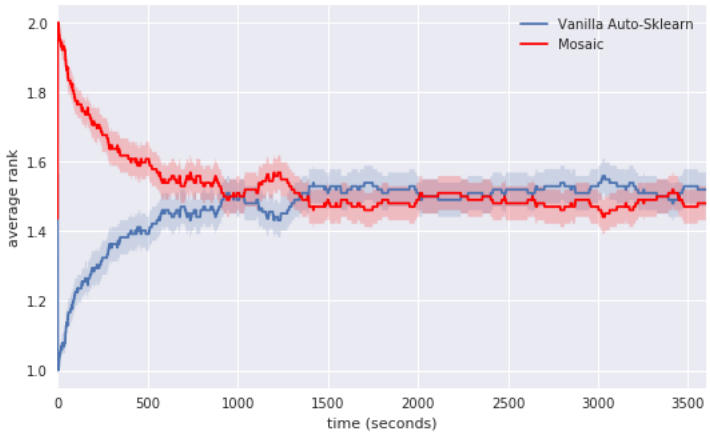

Figure 1: Average rank (lower is better) of `Mosaic` and Vanilla `Auto-Sklearn` across 102 datasets (Datasets on which the performance of both methods differs statistically according the Mann-Whitney rank test with $p = 0.05$).

setting evaluation. The memory is limited to 3GB for each run. All runs are launched on the same CPU model (Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz).

Results A close look at the results show that the performance gap between `Mosaic` and `Auto-Sklearn` is quite small for several datasets (also stated in Li et al. (2016)). For the sake of a fair assessment, we rank the two methods only when their performances are statistically different according to the Mann-Whitney test with $p = 0.05$. We then end up with 102 datasets on which one of the two methods is statistically better than the other. Figure 1 reports the average rank of the two methods showing that `Auto-Sklearn` outperforms `Mosaic` at the beginning. It is due to the Progressive Widening strategy (Section 4.2) used by `Mosaic`. However, `Mosaic` starts gaining performance when the tree search increases and it ends up with a slightly better performance than `Auto-Sklearn`.

6. Discussion and Perspectives

The main contribution of the paper is the `Mosaic` AutoML platform, adapting and extending the Monte-Carlo Tree Search setting to tackle the structured optimization problem of algorithm selection and configuration. The key ingredients of `Mosaic` are: i) the structure of the search space; ii) the ability to handle continuous hyper-parameter ranges. The merits of the approach are demonstrated as it matches the performance of the mature `Auto-Sklearn` which dominates the state of the art in the last 3 years.

Two perspectives for further research will be considered. The first one is concerned with improving the sampling of continuous hyper-parameter values, by taking inspiration from AlphaGo (Silver et al., 2016). A longer term research is to take advantage in the search of the meta-features describing the current dataset, and to capitalize the models learned

from the past datasets, e.g. to achieve a smart initialization of the dataset-dependent hyper-parameter optimization as in `Auto-Sklearn`.

References

Mastane Achab, Stephan Clémençon, Aurélien Garivier, Anne Sabourin, and Claire Vernade. Max k-armed bandit: On the extremehunter algorithm and beyond. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 389–404. Springer, 2017.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002. URL http://www.jmlr.org/papers/v3/auer02a.html.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

Pavel Brazdil and Christophe Giraud-Carrier. Metalearning and algorithm selection: progress, state of the art and introduction to the 2018 special issue. *Machine Learning*, 107(1):1–14, Jan 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5692-y. URL https://doi.org/10.1007/s10994-017-5692-y.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695, 2011.

Sébastien Bubeck, Nicolo Cesa-Bianchi, and Gábor Lugosi. Bandits with heavy tail. *IEEE Transactions on Information Theory*, 59(11):7711–7717, 2013.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. 2008.

Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. 03 2018.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pages 433–445. Springer, 2011.

Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2): 81–173, 2011.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

Romaric Gaudel and Michele Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pages 359–366, 2010.

Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macia, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, et al. Design of the 2015 chalearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

Isabelle Guyon, Wei-Wei Tu, Hugo Jair Escalante, Yuqiang Chen, Guangchuan Shi, Pengcheng Wang, Rong Fang, Beibei Xiao, Jian Liu, and Hai Wang. Automatic machine learning challenge 2018: Towards ai for everyone, 2018. URL https://www.4paradigm.com/competition/pakdd2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908918. URL http://doi.acm.org/10.1145/2908812.2908918.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2009.

Martin Wistuba. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*, 2017.

Appendix A. List of machine learning algorithms and data preprocessing methods with their corresponding hyper-parameters

Methods	Parameters
PCA	n_components, whiten, svd_solver, tol, iterated_power
KernelPCA	n_components, kernel, gamma, degree, coef0, alpha, eigen_solver, tol, max_iter
FastICA	n_components, algorithm, max_iter, tol, whiten, fun
Identity	-
IncrementalPCA	n_components, whiten, batch_size
SelectKBest	score_func, k
SelectPercentile	score_func, percentile
LinearSVC Pre-processing	C, class_weight, max_iter
ExtraTreesClassifier Pre-processing	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, class_weight
FeatureAgglomeration	n_clusters, affinity, linkage
PolynomialFeatures	degree
RBFSampler	gamma, n_components
RandomTreesEmbedding	n_components, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease

Table 1: List of data preprocessing methods and their respective hyper-parameters.

Algorithms	Parameters
LinearDiscriminantAnalysis	solver, shrinkage
QuadraticDiscriminantAnalysis	reg_param
DummyClassifier	-
AdaBoostClassifier	base_estimator, n_estimators, learning_rate, algorithm
ExtraTreesClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, class_weight
RandomForestClassifier	n_estimators, criterion, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, class_weight, bootstrap
GradientBoostingClassifier	loss, learning_rate, n_estimators, max_depth, criterion, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, subsample, max_features, max_leaf_nodes
SGD Classifier	learning rate, penalty, alpha, l1 ratio, loss, epsilon, eta0, power_t, class_weight, max_iter
KNN classifier	K, metric, weights
Perceptron	penalty, alpha, max_iter, tol, shuffle, eta0
RidgeClassifier	alpha, max_iter, class_weight, solver
PassiveAggressiveClassifier	C, max_iter, tol, loss, class_weight
KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric
MLPClassifier	hidden_layer_sizes, activation, solver, alpha, batch_size, learning_rate, learning_rate_init, power_t, max_iter, shuffle, warm_start, momentum, nesterovs_momentum, early_stopping, validation_fraction, beta_1, beta_2, epsilon
SVC	C, max_iter, tol, loss, class_weight, kernel, degree, gamma, coef0
DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_decrease, class_weight
ExtraTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_decrease, class_weight

Table 2: List of machine learning algorithms with their respective hyper-parameters.