ECOLE
Experience-based Computation:
Learning to Optimise

Project Number: 766186
Project Acronym: ECOLE
Project title: Experienced-based Computation: Learning to Optimize

Deliverable D2.2
Integrated model selection and hyperparameter optimization

Authors:
Duc Anh Nguyen, Anna V. Kononova, Thomas Bäck – Leiden University

Project Coordinator: Professor Xin Yao, University of Birmingham
Beneficiaries: Universiteit Leiden, Honda Research Institute, NEC Laboratories

H2020 MSCA-ITN
Date of the report: 31.03.2021

Contents

Acknowledgment

This deliverable was initially named "Constrained and Multi-Criteria optimization". However, deliverable 1.2, "Multi-Criteria Optimization Focusing On Learning For Adaptive Feature Selection And Constraints Prediction" has been designed to include this topic. Due to the fact that the model selection and hyperparameter optimization problem are common problems for all research lines in our project, we changed this deliverable topic into "Integrated model selection and hyperparameter optimization".

Executive Summary

This document provides a concise report on the research invested and the scientific contributions made regarding the work package 2.2 in ECOLE. This work package deals with the model selection and hyperparameter optimization problems, which are commonly faced in most research lines in the ECOLE project. A comprehensive computational investigation into three well-known optimization approaches (i.e., Bayesian optimization, grid search, and random search) in the context of automatic model selection and hyperparameter optimization, to find out which approach yields the best performance. In this comparison, the findings from ECOLE demonstrate that Bayesian optimization always achieves the highest performance in all tested cases. In fact, it outperforms Random search and Grid search on the default hyperparameters on 91 % and 89 % of the tested datasets, respectively, while equal performance is found on the remaining cases. Additionally, compared to the most recent work in the literature (in Section 5), it can improve by 7.3% in terms of the geometric mean (GM) performance measure across all data sets, with 95% less function evaluations.

Major Achievements

Major scientific achievements regarding the work package 2.2 are presented below. In particular, short answers to some of the most important research questions are described:

Research Questions	Discussion
Can hyperparameter optimization improve the performance of machine learning problems?	For the imbalanced classification problem, our finding indicates that Bayesian optimization and Random search outperform grid search on the default hyperparameter approach in 39 and 30 of 44 tested cases, respectively (Section 5). For other machine learning problems (e.g., regression, other classification problems), further research has to be invested.
What is the most efficient optimization approach to deal with the model selection and hyperparameter optimization problems?	Our results demonstrate that Bayesian optimization is the best approach for the combined model selection and hyperparameter optimization (Section 2 and Section 5).

1. Introduction

The research aims of ECOLE include shortening the product cycle, reducing the resource consumption during the complete process, and creating more balanced and innovative products. Instead of just developing technologies to solve a given optimization problem, it will take a bold step forward and optimize automatically across problems. Referring to knowledge, skill, and practice derived from problem-solving processes in time, the experience of optimizing one product or process will be learned and transferred automatically to solve other optimization problems. Work package 2.2 focuses on model selection and hyperparameter optimization problems. This WP issue is important because: In the context of applying the existing research achievements of ECOLE in solving real-world problems efficiently, it is needed to automatically optimize the optimization problems based on the existing works and achievements of ECOLE with less human effort.

This report summarizes the work and research invested in work package 2.2, which deals with the model selection and hyperparameter optimization problems. In this report, therefore, we summarize the following scientific findings and research outcomes of the work package 2.2:

- A literature study on the combination of model selection and hyperparameter optimization problems is summarized in Section 2. Two commonly used approaches, i.e., sequentially and integrated approaches, are delineated.
- A literature study on the presence of the hyperparameter optimization problem is summarized in Section 3. We concisely summarize the state-of-the-art methodologies based on the existing work [1, 2, 3, 4, 5, 6].
- A literature study on the combined model/algorithm selection and hyperparameter optimization (CASH) is provided in Section 4. We introduce the CASH approach based on the existing works [7, 8, 9, 10, 11, 12].
- An empirical investigation of the CASH optimization is given in Section 5. Our contribution in this section is to solve the CASH problem efficiently for the class imbalance classification problem.

The following publication of the ECOLE project is contributing to this report:

- D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. V. Kononova, and T. Bäck, "Comparison of Automated CASH Optimization Approaches for Class Imbalance Problems", 2021, to appear.

2. Background

In the context of applying machine learning in many real-world applications, researchers have to make several high-level decisions: choose a machine learning model such as a learning algorithm (i.e., classification or regression algorithm), different preprocessing techniques (data preprocessing, feature preprocessing), and select a well-suited configuration to their problem, as depicted in Figure 1. This kind of task is typically divided into two separate problems: algorithm selection and hyperparameter optimization.

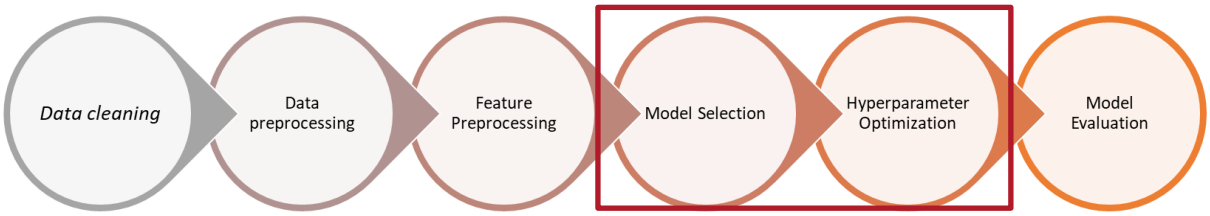

Figure 1. A typical machine learning workflow

Usually, these tasks are addressed separately and sequentially, where the practitioner can choose to handle either task first. Commonly, practitioners proceed by tuning the hyperparameters for each modeling algorithm separately and then choosing the best model. However, this approach is considerably more expensive than the combined approach due to the number of models to optimize. Alternatively, the practitioner can select a suitable model by training all models with their default hyperparameters or based on their experience, and then further tune the hyperparameters only for the best model. Nevertheless, this approach might get stuck in a local optimum of the initially chosen model, based on the default hyperparameter setting.

On the other hand, instead of sequentially solving these problems, the integrated approach aims to optimize both problems at the same time by combining them into a single problem. That is commonly referred to as the Combined Algorithm Selection and Hyperparameter optimization (CASH) or Full Model Selection (FMS) [13, 7] approach. Approaches for tackling the CASH problem have been widely proposed in the machine learning domain, especially in the context of automated machine learning (AutoML), e.g., Auto-Weka [8, 7], Auto-Sklearn [9, 10], TPOT [12], HyperOpt-Sklearn [11]. In addition, [14] demonstrated that the CASH approach is competitive with the sequential approach and requires less computational effort. Hence, in this report, we introduce CASH in the context of optimizing the machine learning pipeline for the classification problem.

3. Hyperparameter optimization

Hyperparameter optimization (HPO) has become increasingly important in the machine learning and optimization community. In the context of optimization, HPO is generally viewed as a black-box optimization problem, which aims to find the global optimum *x* of hyperparameters, with respect to a loss function *f*, namely,

$$x^* = arg\min_{x \in \chi} f(x),$$

where χ stands for the search space of hyperparameters. In the following paragraphs, we briefly introduce two state-of-the-art HPO algorithms: Random search and Bayesian optimization.

3.1. Random search and Grid search

Grid search is the most basic HPO algorithm. We are given a set of hyperparameters, each of which is defined over a (finite) set of values. By enumerating all combinations of these sets, we have a list of all candidates. Grid search evaluates all of these candidates and chooses the best configuration among them. The number of function evaluations is precisely the number of configurations. Usually, the practitioner gives a limited computational budget, i.e., number of function evaluations, for HPO, which is typically much smaller than the number of possible configurations to evaluate - such setup is not fully compatible with grid search. Unlike grid search, which assesses all configurations, random search [1] evaluates only a subset of configurations at random until the given budget runs out and returns the best of the sampled configurations.

3.2. Bayesian optimization

As the HPO task is typically time-consuming, it is preferable to devise/choose an optimizer that would deliver a good hyperparameter setting with a relatively smaller computational budget. Built upon surrogate models, Bayesian Optimization (BO, also called Sequential Model-Based Optimization (SMBO)) [15] is designed for this scenario. Generally speaking, BO iteratively updates a surrogate model **M** that aims to learn the probability distribution of the loss value conditioned on hyperparameter *x*, i.e., *P(f|x)*, from the historical information - the evaluated hyperparameter and the corresponding loss values. We subsequently choose a new candidate hyperparameter by optimizing the so-called acquisition function (e.g., Expected Improvement), which is defined over the surrogate model **M** and often balances the exploration and exploitation of the search. Many variants have been proposed for BO, including the Sequential Model-based Algorithm Configuration (SMAC) [5], Sequential Parameter Optimisation (SPO) [2], Mixed-Integer Parallel Efficient Global Optimization (MIPEGO) [6], and Tree-structured Parzen Estimator (TPE) [3, 16]. They differ mostly in the initial sampling method, the probabilistic model, and the acquisition function. Common choices for the probabilistic model are Random Forests (RF) [17], Gaussian Process regression (GPR) [18], and TPE. As for the acquisition function, Expected Improvement (EI) [19], Probability of Improvement (PI) [20], and Upper Confidence Bound (UCB) [21] are more frequently applied among many other alternatives. Determining the sample size and strategy for the initial sampling is still a daunting task, and the well-known Latin Hypercube Sampling (LHS) [22] strategy is generally perceived as a

statistically more robust option, compared to others like simple random sampling and random sequences of low-discrepancies. The schematic procedure of SMBO is depicted in Figure 2. The procedure starts from the initialization step: a set of configurations is sampled and evaluated with the objective function, then the configurations and their corresponding loss values are used to build a probabilistic model. Next, based on optimizing the chosen acquisition function, a new configuration is generated and evaluated by the objective function. Finally, the new tuple is added to the historical tuning data (samples) for the new loop.

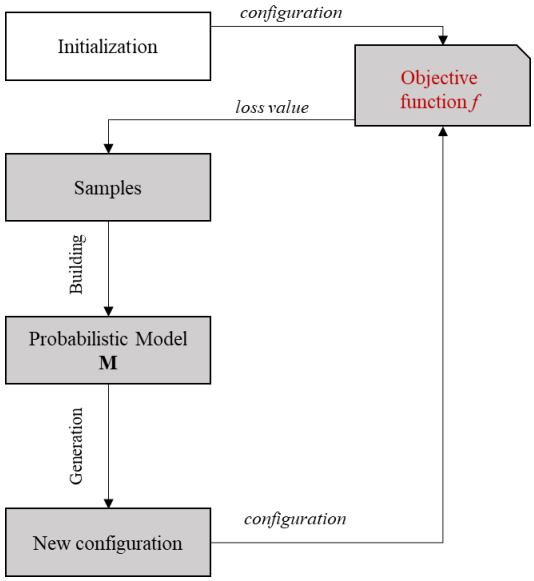

Figure 2. Schematic procedure of Bayesian optimization.

4. The combined algorithm selection and hyperparameter optimization problem

The CASH problem was defined by Thornton et al. in Auto-Weka [7], an automated machine learning framework, where it aims to identify the best machine learning model (e.g., preprocessing techniques, classification/regression algorithm) and their hyperparameters that minimize the loss value of an arbitrary real-valued objective function *f*.

Given a set of algorithms $\mathcal{A} = \{A^1, \dots, A^n\}$ and their hyperparameter spaces $\chi = \{\chi^1, \dots, \chi^n\}$, let $\mathcal{D}_{train}^{(j)}$ and $\mathcal{D}_{valid}^{(j)}$, for $j = 1, \dots, k$, denote training and validation sets, generated by applying *k*-fold cross-validation on dataset $\mathcal{D}$. Then the CASH problem is defined as:

$$A^*, x^* = \arg \min_{A^{(i)} \in \mathcal{A}, x \in \chi^{(i)}} \frac{1}{k} \sum_{j=1}^{k} f\left(A_x^{(i)}, \mathcal{D}_{train}^{(j)}, \mathcal{D}_{valid}^{(j)}\right).$$

Here, $f\left(\mathrm{A}_{\mathrm{x}}^{(\mathrm{i})}, \mathcal{D}_{train}^{(j)}, \mathcal{D}_{valid}^{(j)}\right)$ denote the loss achieved by the learning algorithm $\mathrm{A}^{(i)}$ and its corresponding hyperparameters $x \in \chi^{(i)}$ when trained and evaluated on $\mathcal{D}_{train}^{(j)}$, $\mathcal{D}_{valid}^{(j)}$.
Note that most HPO methods in practice can handle the CASH problem by modeling the algorithms' choice as a categorical hyperparameter. Each algorithm is mapped to its locally dependent hyperparameters by the so-called conditional parameter.

5. Empirical comparison of CASH optimization for the class imbalance problem.

5.1. Class imbalance problem.

Due to the fact that class imbalance is present in many real-world applications, the imbalanced classification has brought along much attention from researchers and practitioners. In the manufacturing industry, fault detection on the products is a typical example of the imbalanced classification problem, since most products are correctly produced, and only a few products are faulty ones [23]. In software defect prediction, the highly imbalanced nature between the defect and non-defect modules cannot be neglected. Several studies have shown that proper class imbalance learning methods can benefit software defect prediction and improve the performance of detecting possible failures [24, 25]. Due to recent developments in data storage and management, it is possible for practitioners from industry and engineering to collect a large amount of data in order to extract knowledge and acquire hidden insights. An application example may be illustrated in the field of computational design optimization [26], where product parameters are modified to generate digital prototypes, and the performances are usually evaluated through numerical simulations, which often require minutes to hours of computation time. Here, some parameter variations (minority number of designs) would result in effective and producible geometric shapes, but the given constraints are violated in the final step of optimization. In this case, performing proper imbalanced classification algorithms on the design parameters could save computation time. A detailed discussion of the class imbalance classification problem is provided in another ECOLE deliverable, namely the 3.1- Semi-supervised learning for class imbalance problems.

The resampling techniques are designed to handle the class imbalance scenario by producing balanced data sets. The resampling algorithms used in our experiments can be arranged into three groups, namely "over resampling" (7 techniques [27, 28, 29, 30, 31, 32]), "under resampling" (11 techniques [33, 34, 35, 36, 37, 38, 39, 40, 41, 42]), and "combine resampling" (2 techniques [43, 44]), which are implemented in the python package **imbalanced-learn**[1] [45]. The under-sampling technique balances the class distribution by removing samples from the majority class, while oversampling balances class distribution by adding more copies of the minority class samples. The combined resampling indicates a combination of both over-resampling and under-resampling techniques. The list of these resampling techniques and their hyperparameters is provided in Table II. Their detailed information is provided in Sections 2, 3.1, 3.2 of deliverable 3.1.

In the class imbalance domain, it is widely known that *accuracy rate* is a deceptive estimate of performance [46, 47] . Instead of *accuracy rate*, other metrics such as the area under the receiver operating characteristic (ROC) curve (AUC), F-measure (FM), or geometric mean (GM) are commonly used to measure performance [48]. For comparison with previous studies [49, 50], we use GM as the performance evaluation metric, i.e.:

$$\mathrm{GM} = \sqrt{\mathrm{TP}_{\mathrm{rate}} \cdot \mathrm{TN}_{\mathrm{rate}}},$$

[1] https://github.com/scikit-learn-contrib/imbalanced-learn (version 0.7.0)

where $TP_{rate} = \frac{TP}{TP+FN}$ and $TN_{rate} = \frac{TN}{FP+TN}$ are the true positive and true negative rate, respectively, with TP, TN, FN and FP denoting the number of true positive, true negative, false negative and false positive samples.

5.1.1. Imbalanced datasets

Technically, any dataset with an unequal class distribution is imbalanced, in which the number of samples of one class is much lower than the one of the other classes. Here, the one or more underrepresented classes are called minority class(es), and the other class(es) are called majority classes. However, only datasets with a significantly skewed distribution are regarded to be imbalanced in the imbalanced learning domain [23]. The 44 benchmark imbalanced datasets from the KEEL collection [51] are given in Table I, which includes the number of negative and positive samples, as well as their imbalance ratio (IR),i.e., the ratio of the number of majority class samples to the number of minority class samples.

Table I. Imbalanced data used. The number of positive and negative classes and the imbalance ratio (IR) of the KEEL Datasets are arranged by IR value

Dataset	# Negative	# Positive	#Attributes	#IR
glass1	138	76	9	1.82
ecoli-0_vs_1	77	143	7	1.86
wisconsin	444	239	9	1.86
pima	500	268	8	1.87
iris0	100	50	4	2
glass0	144	70	9	2.06
yeast1	1055	429	8	2.46
haberman	225	81	3	2.78
vehicle2	628	218	18	2.88
vehicle1	629	217	18	2.9
vehicle3	634	212	18	2.99
glass-0-1-2-3_vs_4-5-6	163	51	9	3.2
vehicle0	647	199	18	3.25
ecoli1	259	77	7	3.36
new-thyroid1	180	35	5	5.14
new-thyroid2	180	35	5	5.14
ecoli2	284	52	7	5.46
segment0	1979	329	19	6.02
glass6	185	29	9	6.38
yeast3	1321	163	8	8.1
ecoli3	301	35	7	8.6

Dataset	# Negative	# Positive	#Attributes	#IR
page-blocks0	4913	559	10	8.79
yeast-2_vs_4	463	51	8	9.08
yeast-0-5-6-7-9_vs_4	477	51	8	9.35
vowel0	898	90	13	9.98
glass-0-1-6_vs_2	175	17	9	10.29
glass2	197	17	9	11.59
shuttle-c0-vs-c4	1706	123	9	13.87
yeast-1_vs_7	429	30	7	14.3
glass4	201	13	9	15.46
ecoli4	316	20	7	15.8
page-blocks-1-3_vs_4	444	28	10	15.86
abalone9-18	689	42	8	16.4
glass-0-1-6_vs_5	175	9	9	19.44
shuttle-c2-vs-c4	123	6	9	20.5
yeast-1-4-5-8_vs_7	663	30	8	22.1
glass5	205	9	9	22.78
yeast-2_vs_8	462	20	8	23.1
yeast4	1433	51	8	28.1
yeast-1-2-8-9_vs_7	917	30	8	30.57
yeast5	1440	44	8	32.73
ecoli-0-1-3-7_vs_2-6	274	7	7	39.14
yeast6	1449	35	8	41.4
abalone19	4142	32	8	129.44

5.2. Optimization methods

In the ECOLE project, we tested the random search and a Bayesian optimization variant, so-called TPE, which are implemented in the Python package **HyperOpt**[2] (version 0.2.4) as HPO algorithms.

The HPO algorithms are deployed to optimize the hyperparameters of five classification algorithms, i.e., Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Decision Tree (DT), and Logistic Regression (LR), which are described in Table III, and 20 resampling techniques plus one option not to use resampling technique, leading up to 64 hyperparameters in total (detailed information on these hyperparameters are provided in Table II). Moreover, to study the HPO algorithms' effectiveness, we evaluated all possible combinations of classification and resampling algorithms, using the default hyperparameter settings for each algorithm. On each dataset, the combination with the highest GM is reported. Thus, there are 5 x 21 = 105 combinations in total. Evaluating these combinations one by one is named "Grid-Def" here.

[2] https://github.com/hyperopt/hyperopt

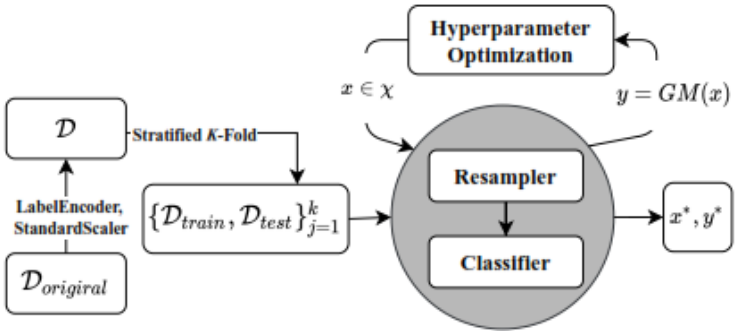

Figure 3. Flowchart of the experimental setup.

For clarification, the workflow of the experimental procedure is provided in Figure 3, which consists of the following two phases:

- Preprocessing: For one dataset input is encoded by Label encoder[3] to encode any categorical data to a number for the input dataset. Then the encoded data is preprocessed by Standard Scaler[3] to have zero mean and a standard deviation of one. Next, Stratified *k*-fold cross-validation using *k* =5 is used.
- Hyperparameter optimization: All parameters of HPO are initialized, taking values from the provided input, including search space χ, objective function in the grey circle, and *k* folds of the examined dataset. Then, the algorithm optimizes the search space χ until the number of function evaluations reaches 500. At each iteration, HPO generates a hyperparameter setting $x \in \chi$, which includes the selected resampler and classifier, with the selected resampler being applied to each fold to make it balanced and applied the classifier to the balanced fold. The hyperparameter optimization process in the gray circle is based on the geometric mean.

Table II. Hyperparameters of resampling techniques

Algorithm	**Hyperparameter**	**Range**
Over-Resampling		
SMOTE	k_neighbors sampling_strategy	[1, 10] *default*
BorderlineSMOTE	k_neighbors m_neighbors sampling_strategy kind	[1, 10] [1, 10] *default* [borderline1, borderline2]
SMOTENC	sampling_strategy	*default*

[3] Label encoder, Standard scaler and Stratified *k*-fold cross-validation are implemented in the python library scikit-learn (version 0.23.2)

Algorithm	Hyperparameter	Range
	k_neighbors	[1, 10]
SVMSMOTE	sampling_strategy k_neighbors m_neighbors out_step	*default* [1, 10] [1, 10] [0.0, 1.0]
KMeansSMOTE	sampling_strategy k_neighbors cluster_balance_threshold	*default* [1, 10] [1e-2, 1]
ADASYN	sampling_strategy n_neighbors	*default* [1, 10]
Combine-Resampling		
SMOTENN	sampling_strategy	*default*
SMOTETomek	sampling_strategy	*default*
Under-Resampling		
CondensedNearestNeighbour	n_neighbors sampling_strategy n_seeds_S	[1, 50] *default* [1, 50]
EditedNearestNeighbours	n_neighbors sampling_strategy return_indices kind_sel	[1, 20] *default* [True, False] [all, mode]
RepeatedEditedNearestNeighbours	sampling_strategy kind_sel n_neighbors	*default* [all, mode] [1, 20]
AllKNN	n_neighbors sampling_strategy kind_sel allow_minority	[1, 20] *default* [all, mode] [True, False]
InstanceHardnessThreshold	estimator sampling_strategy cv	None, knn, decision-tree, adaboost, gradient-boosting, linear-svm *default* [2, 10]
OneSidedSelection	sampling_strategy n_neighbors n_seeds_S	*default* [1, 20] [1, 20]
RandomUnderSampler	sampling_strategy replacement	*default* [True, False]

Algorithm	Hyperparameter	Range
TomekLinks	sampling_strategy	*default*
NearMiss	sampling_strategy version n_neighbors n_neighbors _ver3	*default* [1,3] [1, 20] [1, 20]
NeighbourhoodCleaningRule	sampling_strategy threshold_cleaning n_neighbors	*default* [0.0, 1.0] [1, 20]
ClusterCentroids	sampling_strategy estimator voting	*default* [KMeans, MiniBatchKMeans] [hard, soft]

Table III. Hyperparameters of Classification algorithms

Algorithm	Hyperparameter	Range
Support Vector Machines (SVM)	Max_iter Cache_size probability C kernel shrinking gamma gamma_value coef0 degree tol	10000 700 (Megabyte) [True, False] [0.5^5,100] [linear, rbf, poly, sigmoid] [true, false] [auto, value, scale] [3.1e-05,8] [-1.0, 1.0] [2, 5] [1e-05, 1e-01]
Random Forest (RF)	n_estimators criterion max_features min_samples_split min_samples_leaf bootstrap class_weight	[1,150] [gini, entropy] [1, sqrt, log2, None] [2, 20] [1, 20] [True, False] [balanced, balanced_subsample, None]
K-Nearest Neighbors (KNN)	n_neighbors weights algorithm	[1, 51] [uniform, distance] [auto, ball_tree, kd_tree, brute]

Algorithm	Hyperparameter	Range
	p	[0, 20] • p = 0 → metric = chebyshev • p = 1 → metric = manhattan • p = 2 → metric = euclidean • p > 2 → metric = minkowski
Decision Tree (DT)	criterion max_depth max_features min_samples_split min_samples_leaf	[gini, entropy] [2, 20] [1, sqrt, log2, None] [2, 20] [1, 20]
Logistic Regression (LR)	C criterion tol l1_ratio (penalty, solver)	[1, 150] $[0.5^5,100]$ [1e-05, 1e-01] [1e-09, 1] [(l1, liblinear), (l1, saga), (l2, lbfgs), (l2, newton-cg), (l2, liblinear), (l2, sag), (l2, saga), (elasticnet, saga), (none, newton-cg), (none, lbfgs), (none, sag), (none, saga)]

Additionally, since the used algorithms, i.e., classifiers and resamplers, in the objective function are stochastic, we fixed ten random seeds for HPO, classifiers, and resamplers in 10 different runs to make the objective function deterministic, i.e., to assure the objective function always returns the same value for identical input values.

5.3. Experimental Results

The experimental results are presented in Table IV to illustrate the performance differences between the three integrated optimization approaches used, i.e., TPE, Random search (RS) and Grid-Def (Grid), and to compare them against Evolutionary Under-Sampling (EUS) methods [49]. In this table, our results are presented in the corresponding columns on the left side (not shaded). The results from [49] are presented on the right side (grey shaded) for EUS, EUS-windowing (EUSW), the clustering-based surrogate model for EUS (EUSC), and the hybrid surrogate model for EUS (EUSHC). In both groups, the highest performance for each dataset is highlighted in bold. In the experimental results, the methods performing significantly worse than the best according to the Wilcoxon signed-rank test with $\alpha = 0.05$ are underlined. A value labeled with * indicates that the result obtained in ECOLE outperforms those from [49] for the corresponding dataset. Additionally, an extra column to the right summarizes the method that achieves the highest GM for the corresponding dataset. The results show the following findings:

- HPO approaches exhibit better performance compared to the Grid-Def approach, which uses default hyperparameters. Moreover, according to the results of the Wilcoxon signed-rank test, TPE is always the best method found: it significantly outperforms the Grid-Def in 32/44 datasets, while it significantly outperforms RS in 26/44 tested cases.

- Overall, TPE shows the highest GM in most of the datasets, 41/44. Other compared methods win on different datasets, e.g., EUSC and EUS achieve the highest GM on "glass-0-1-2-3_vs_4-5-6" and "vowel0", respectively. All approaches get the maximum GM on the dataset "iris0".
- Comparing ECOLE experimental results and the best results from [49], it can be concluded that TPE wins for 41/44 datasets, RS wins for 38/44 datasets, and Grid-Def wins for 37/44 datasets. This is a high-impact result of ECOLE, since the number of function evaluations used in this experiment is much smaller than in [49]: 500 function evaluations for TPE and RS, 105 function evaluations for Grid-Def vs. 10000 function evaluations for each method in [49]. A possible explanation for this might be that [49] employs a simple KNN rule with *k=1* as the mere classifier, while more complicated classification algorithms are used in the ECOLE. More precisely, according to our experimental results, KNN only wins in 11 % (TPE), 13 % (RS), and 9 % (Grid-Def) of all cases.

Table IV. Average geometric mean (rounded to 4 decimals) over 10 repetitions for the 44 datasets, ordered by increasing IR value.

Dataset	**IR**	**Our experimental results**			**Evolutionary algorithms – results from [49]**				**Overall Winner**
		TPE	**RS**	**Grid**	**EUS**	**EUSW**	**EUSC**	**EUSHC**	
glass1	1.82	***0.7989**	0.7763	0.7793	0.7773	0.7010	**0.7941**	0.7367	TPE
ecoli-0_vs_1	1.86	***0.9864**	***0.9864**	***0.9864**	0.9583	0.9312	0.9581	**0.9615**	TPE | RS | Grid
wisconsin	1.86	***0.9814**	*0.9807	*0.9788	**0.9690**	0.9652	0.9600	0.9590	TPE
pima	1.87	***0.7711**	*0.7651	*0.7599	0.6943	0.6749	0.6957	**0.7145**	TPE
iris0	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	-
glass0	2.06	***0.8749**	*0.8588	*0.8719	0.8009	0.6176	**0.8047**	0.6595	TPE
yeast1	2.46	***0.7324**	*0.7304	*0.7183	0.6533	0.6501	**0.6600**	**0.6600**	TPE
haberman	2.78	***0.7025**	*0.6926	*0.6678	0.5475	**0.5635**	0.5521	0.5497	TPE
vehicle2	2.88	***0.9915**	*0.9874	*0.9895	0.9259	0.9175	**0.9265**	0.9173	TPE
vehicle1	2.9	***0.8658**	*0.8429	*0.8333	0.6729	0.6624	0.6512	**0.6926**	TPE
vehicle3	2.99	***0.8482**	*0.8231	*0.8108	**0.7280**	0.7142	0.7165	0.7204	TPE
glass-0-1-2-3_vs_4-5-6	3.2	**0.9559**	0.9505	0.9483	0.9525	0.9385	**0.9647**	0.9546	EUSC
vehicle0	3.25	***0.9863**	*0.9809	*0.9766	**0.9164**	0.9027	0.9103	0.9016	TPE
ecoli1	3.36	***0.9047**	*0.8966	*0.8999	**0.8634**	0.8306	0.8554	0.8424	TPE
new-thyroid1	5.14	***0.9969**	*0.9966	*0.9944	**0.9882**	0.9809	0.9859	0.9653	TPE
new-thyroid2	5.14	***0.9978**	*0.9966	*0.9910	**0.9865**	0.9773	0.9831	0.9746	TPE
ecoli2	5.46	***0.9361**	*0.9337	***0.9361**	0.9000	0.8663	**0.9034**	0.8772	TPE | Grid
segment0	6.02	***0.9993**	*0.9990	*0.9965	**0.9881**	0.9870	0.9876	0.9858	TPE
glass6	6.38	***0.9524**	*0.9516	*0.9381	0.8889	0.9071	**0.9156**	0.9054	TPE
yeast3	8.1	***0.9428**	*0.9395	*0.9290	0.8728	0.8740	**0.8752**	0.8550	TPE
ecoli3	8.6	***0.9061**	*0.9023	*0.9044	0.8348	0.8153	**0.8500**	0.8097	TPE
page-blocks0	8.79	***0.9456**	*0.9422	*0.9401	**0.9117**	0.9038	0.9096	0.9085	TPE

Dataset	IR	Our experimental results			Evolutionary algorithms – results from [49]				Overall Winner
		TPE	RS	Grid	EUS	EUSW	EUSC	EUSHC	
yeast-2_vs_4	9.08	***0.9559**	*0.9474	*0.9401	0.9042	0.8774	**0.9156**	0.8930	TPE
yeast-0-5-6-7-9_vs_4	9.35	***0.8212**	*0.8063	*0.7938	0.7685	0.7663	**0.7901**	0.7535	TPE
vowel0	9.98	**0.9581**	0.9483	0.9427	**0.9897**	0.9719	0.9877	0.9831	EUS
glass-0-1-6_vs_2	10.29	***0.8498**	*0.8216	*0.7904	0.6383	0.6164	**0.6651**	0.5815	TPE
glass2	11.59	***0.8516**	*0.8271	*0.7903	0.7194	0.6525	**0.7262**	0.6173	TPE
shuttle-c0-vs-c4	13.87	***1**	***1**	***1**	0.9960	**0.9968**	0.9960	0.9960	TPE | RS | Grid
yeast-1_vs_7	14.3	***0.8028**	*0.7926	*0.7979	**0.7176**	0.7079	0.7068	0.6669	TPE
glass4	15.46	***0.9323**	*0.9244	*0.9318	**0.8700**	0.8513	0.8613	0.8531	TPE
ecoli4	15.8	***0.9727**	0.9551	0.9415	0.8984	0.9362	0.8857	**0.9645**	TPE
page-blocks-1-3_vs_4	15.86	***0.9925**	*0.9877	*0.9884	**0.9674**	0.9399	0.9471	0.9294	TPE
abalone9-18	16.4	***0.8889**	*0.8752	*0.8536	**0.7269**	0.6772	0.7224	0.6559	TPE
glass-0-1-6_vs_5	19.44	***0.9567**	*0.9530	0.9304	0.9214	0.9151	0.9160	**0.9501**	TPE
shuttle-c2-vs-c4	20.5	***1**	***1**	***1**	**0.9577**	0.6449	0.9414	0.7365	TPE | RS | Grid
yeast-1-4-5-8_vs_7	22.1	***0.7035**	*0.6874	*0.6650	0.6569	0.6088	**0.6604**	0.6149	TPE
glass5	22.78	***0.9637**	0.9555	0.9438	0.8105	0.9076	**0.9600**	0.9103	TPE
yeast-2_vs_8	23.1	***0.8231**	*0.8031	*0.7945	**0.7931**	0.7496	0.7656	0.7668	TPE
yeast4	28.1	***0.8803**	*0.8664	*0.8585	0.8050	0.7799	**0.8288**	0.7970	TPE
yeast-1-2-8-9_vs_7	30.57	***0.7459**	*0.7402	*0.7289	**0.6721**	0.6078	0.6704	0.6500	TPE
yeast5	32.73	***0.9803**	*0.9790	*0.9788	0.9634	0.9494	0.9455	**0.9653**	TPE
ecoli-0-1-3-7_vs_2-6	39.14	***0.9095**	*0.8770	*0.9091	0.6700	**0.7048**	0.6625	0.6865	TPE
yeast6	41.4	***0.8972**	*0.8905	*0.8840	**0.8357**	0.8080	0.8034	0.8031	TPE
abalone19	129.44	***0.7967**	*0.7942	*0.7579	0.6258	0.6061	**0.7214**	0.6556	TPE

Figure 4 provides insights into the tuning behavior of these two approaches on the dataset "abalone9-18". Two sub-figures show the observed GM values of TPE (top) and RS (bottom) over 500 function evaluations. In each sub-figure, the scatter plots show the sequence of observed values vs. the number of function evaluations, the red line shows the current best value, and the black vertical bars indicate the infeasible configurations where GM returns a zero if an invalid configuration is used. The stacked histogram plot next to the scatter plot shows the distribution of all observed values. Five last stacked histogram plots to the right indicate the distributions for each classification algorithms, namely SVM, RF, KNN, LR, and DT. In the Figure 4 we observe the following:

- Configurations generated by TPE can avoid the undefined infeasible area better than RS. In this run, the number of errors occurring in the TPE and RS runs are 14 and 22, respectively. According to a summary over all datasets and repetitions, the number of infeasible configurations of TPE and RS are 4.4% and 5.9%, respectively.

- Apart from zero values, the GM values of TPE are mostly in the range from 0.8 to 0.9, while the GM values of RS are distributed in the range from 0.6 to 0.7. This is because the TPE generates new configurations based on historical information so far, while RS does not.

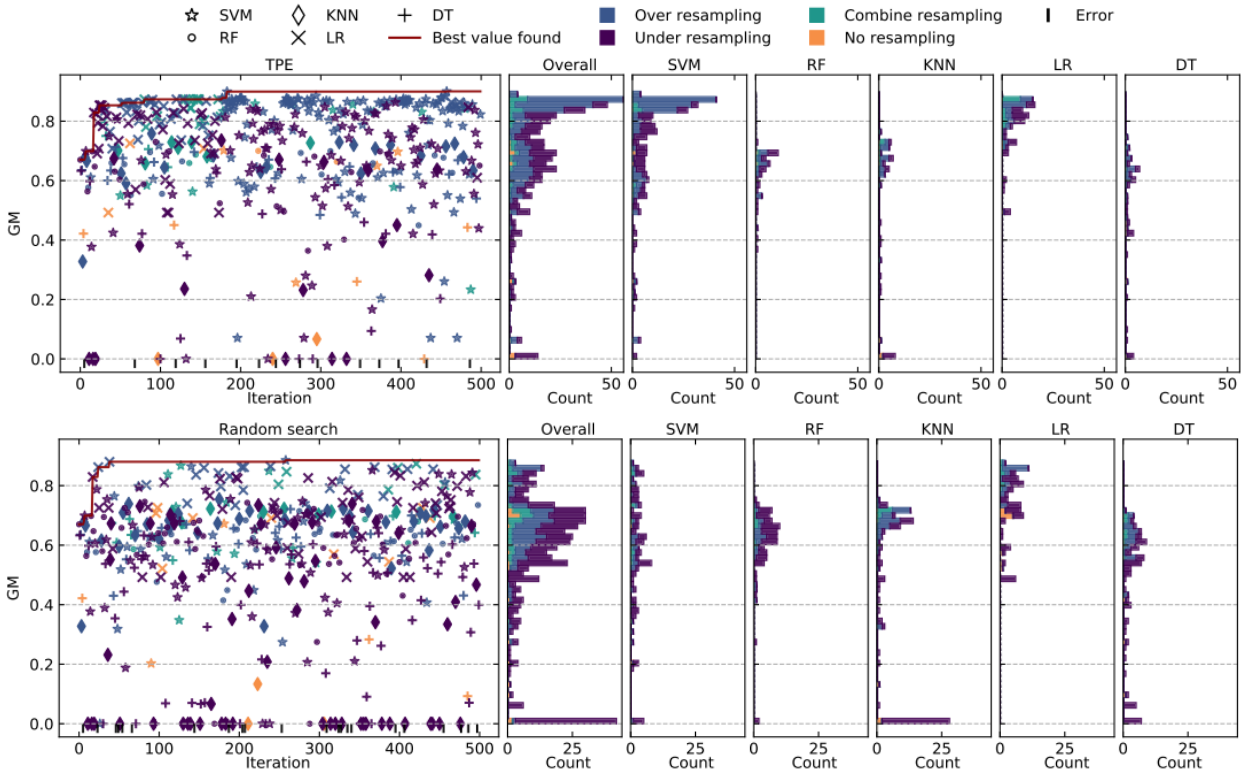

Figure 4. Illustration of the hyperparameter tuning process on dataset "abalone9-18".

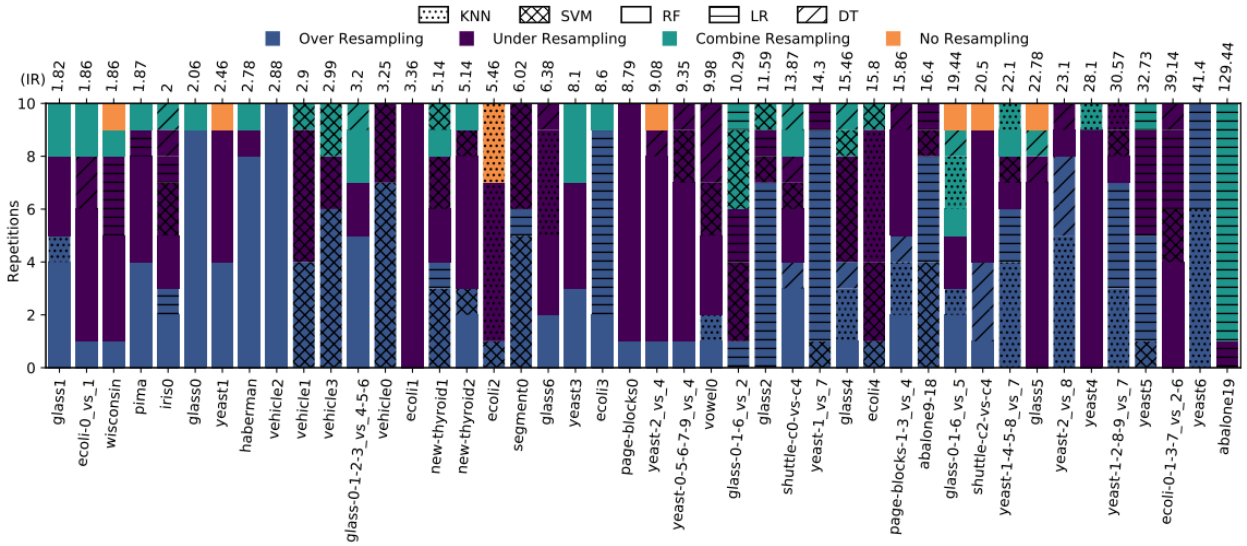

Figure 5. The resulting combination of classifier and resampler choices for an optimization process using TPE across 10 repetitions on 44 datasets. Figure best viewed in color.

According to TPE's highest results, Figure 5 shows the final combination of choices of classification algorithms and resampling approaches once the optimization run is over. There is no dominant algorithm over many datasets in the plot, but different datasets benefit from different classification algorithms. For example, "glass0", "yeast1", "yeast3", "haberman", "vehicle2", "ecoli1" and "page-blocks0" achieve the best results with SVM, "vehicle0", "vehicle1", "vehicle3" with KNN, whereas "abalone9-18" always results in LR. Besides, 98% of runs yield the best performance by using resampling techniques. Particularly, Over resampling, Under resampling, and Combine resampling obtain 182, 199, 50 wins over 44 x 10 = 440 runs. Additionally, no classifier/resampler combination is providing the best classification performance over all datasets. Specifically, RF and SVM obtain 206 and 84 wins, while other algorithms (LR, KNN, DT) find the best performance in 73, 48, 29 runs, respectively.

6. Summary and Outlook

This deliverable report focuses on the research invested and scientific achievements regarding the work package 2.2. in ECOLE, namely, integrated model selection and hyperparameter optimization. Section 2 of this report highlights the common approaches for handling the model selection and hyperparameter optimization problem in the machine learning domain. Section 3 of this report shares an overview of three commonly used optimization methods: Grid search, Random search, and Bayesian optimization. Section 4 introduces the combined algorithm selection and hyperparameter optimization problem. Section 5 shares the experimental results and key results from our research.

As these project results show, when compared to other approaches, Bayesian optimization (BO) produces the best results. In fact, BO outperforms Random search and Grid search on the default hyperparameters on 91 % and 89 % of the tested datasets, respectively, while equal performance is found on the remaining cases. Hence, we recommend using the BO approach for handling CASH optimization for class imbalance problems.

Another major achievement, compared to the best results from [49], ECOLE approach improves classification performance by 7.3% in terms of GM across all datasets, with 95% less function evaluations.

There are several next steps for extending this work in the final phase of ECOLE. First, we intend to apply more Bayesian optimization variants such as SMAC, SPO, and MIP-EGO, in order to study the performances between variants in this domain. Additionally, instead of applying hyperparameter tuning on the level of an individual dataset, we are interested in studying the behavior of HPO approaches in tuning on a set of datasets. Next, besides Bayesian optimization, we will extend this research with other state-of-the-art HPO approaches such as irace [52] and Hyperband [53]. Finally, the outcomes of this work will be applied at Tata steel[4] and HRI-EU[5], in order to minimize the human effort in solving many real-world problems such as fault detection problems and computational design optimization problems.

[4] Tata Steel Netherlands Technology B.V., Velsen-Noord, The Netherlands
[5] Honda Research Institute Europe GmbH, Offenbach/Main, Germany

Bibliography

[1] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281-305, 3 2012.

[2] T. Bartz-Beielstein, C. Lasarczyk and M. Preuss, "Sequential parameter optimization," *2005 IEEE Congress on Evolutionary Computation, IEEE CEC 2005. Proceedings*, vol. 1, pp. 773-780, 1 2005.

[3] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, "Algorithms for Hyper-parameter Optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, USA, 2011.

[4] S. Falkner, A. Klein and F. Hutter, "BOHB: Robust and Efficient Hyperparameter Optimization at Scale," *ArXiv*, vol. abs/1807.01774, 2018.

[5] F. Hutter, H. H. Hoos and K. Leyton-Brown, "Sequential Model-based Optimization for General Algorithm Configuration," in *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, Berlin, 2011.

[6] H. Wang, B. van Stein, M. T. M. Emmerich and T. Bäck, "A new acquisition function for Bayesian optimization based on the moment-generating function," *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 507-512, 2017.

[7] C. Thornton, F. Hutter, H. Hoos and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of," *KDD*, 2012.

[8] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," *Journal of Machine Learning Research*, vol. 18, pp. 1-5, 2017.

[9] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, Cambridge, 2015.

[10] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer and F. Hutter, *Auto-Sklearn 2.0: The Next Generation*, 2020.

[11] B. Komer, J. Bergstra and C. Eliasmith, "Hyperopt-Sklearn," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff and J. Vanschoren, Eds., Cham, Springer International Publishing, 2019, p. 97–111.

[12] R. S. Olson and J. H. Moore, "TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff and J. Vanschoren, Eds., Cham, Springer International Publishing, 2019, p. 151–160.

[13] F. Hutter, L. Kotthoff and J. Vanschoren, Eds., Automatic Machine Learning: Methods, Systems, Challenges, Springer, 2018.

[14] D. Vermetten, H. Wang, C. Doerr and T. Bäck, "Integrated vs. Sequential Approaches for Selecting and Tuning CMA-ES Variants," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, New York, NY, USA, 2020.

[15] D. R. Jones, M. Schonlau and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *J. of Global Optimization,* vol. 13, p. 455–492, 12 1998.

[16] J. Bergstra, D. Yamins and D. D. Cox, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," in *ICML*, 2013.

[17] L. Breiman, "Random Forests," *Mach. Learn.,* vol. 45, p. 5–32, 10 2001.

[18] C. E. Rasmussen, "Gaussian Processes in Machine Learning," in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg and G. Rätsch, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, p. 63–71.

[19] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, Berlin, 1975.

[20] D. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *J. of Global Optimization,* vol. 21, pp. 345-383, 2001.

[21] P. Auer, "Using Confidence Bounds for Exploitation-Exploration Trade-offs.," *Journal of Machine Learning Research,* vol. 3, pp. 397-422, 2002.

[22] M. D. McKay, R. J. Beckman and W. J. Conover, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics,* vol. 21, p. 239–245, 1979.

[23] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk and F. Herrera, Learning from imbalanced data sets, Springer, 2018.

[24] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability,* vol. 62, p. 434–443, 2013.

[25] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014.

[26] J. Kong, T. Rios, W. Kowalczyk, S. Menzel and T. Bäck, "On the Performance of Oversampling Techniques for Class Imbalance Problems," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020.

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research,* vol. 16, p. 321–357, 2002.

[28] H. Han, W.-Y. Wang and B.-H. Mao, "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning," 2005.

[29] H. He, Y. Bai, E. A. Garcia and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008.

[30] F. Last, G. Douzas and F. Bacao, *Oversampling for Imbalanced Learning Based on K-Means and SMOTE*, 2017.

[31] H. M. Nguyen, E. W. Cooper and K. Kamei, "Borderline Over-Sampling for Imbalanced Data Classification," *Int. J. Knowl. Eng. Soft Data Paradigm.*, vol. 3, p. 4–21, 4 2011.

[32] imbalanced-learn, "RandomOverSampler," [Online]. Available: https://imbalanced-learn.org/stable/generated/imblearn.over_sampling.RandomOverSampler.html.

[33] I. Tomek, "Two modifications of CNN," *IEEE Trans. Systems, Man and Cybernetics,* vol. 6, p. 769–772, 1976.

[34] M. Kubat, "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection," *Fourteenth International Conference on Machine Learning,* 6 2000.

[35] K. Gowda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (Corresp.)," *IEEE Transactions on Information Theory,* vol. 25, pp. 488-490, 7 1979.

[36] D. L. Wilson, "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Transactions on Systems, Man, and Cybernetics,* Vols. SMC-2, pp. 408-421, 7 1972.

[37] I. Tomek, "An Experiment with the Edited Nearest-Neighbor Rule}," *IEEE Transactions on Systems, Man, and Cybernetics,* Vols. SMC-6, pp. 448-452, 1976.

[38] M. R. Smith, T. Martinez and C. Giraud-Carrier, "An Instance Level Analysis of Data Complexity," *Mach. Learn.,* vol. 95, p. 225–256, 5 2014.

[39] J. Ziang, "KNN approach to unbalanced data distributions: a case study involving information extraction," *Proc. Int'l. Conf. Machine Learning1 (ICML'03), Workshop Learning from Imbalanced Data Sets,* 2003.

[40] J. Laurikkala, "Improving Identification of Difficult Small Classes by Balancing Class Distribution," 2001.

[41] imbalanced-learn.org, "ClusterCentroids," [Online]. Available: https://imbalanced-learn.org/stable/generated/ imblearn.under_sampling.ClusterCentroids.html.

[42] imbalanced-learn.org, "RandomUnderSampler," [Online]. Available: https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.RandomUnderSampler.html.

[43] G. E. A. P. A. Batista, R. C. Prati and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter,* vol. 6, p. 20–29, 2004.

[44] G. Batista, A. Bazzan and M.-C. Monard, "Balancing Training Data for Automated Annotation of Keywords: a Case Study.," *the Proc. Of Workshop on Bioinformatics,* pp. 10-18, 1 2003.

[45] G. Lemaître, F. Nogueira and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," *Journal of Machine Learning Research,* vol. 18, pp. 1-5, 2017.

[46] J. Kong, W. Kowalczyk, N. D. Anh, S. Menzel and T. Bäck, "Hyperparameter optimisation for improving classification under class imbalance," in *IEEE Symposium Series on Computational Intelli-gence (SSCI),* 2019.

[47] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering,* vol. 17, pp. 299--310, 2005.

[48] V. López, A. Fernández, S. García, V. Palade and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information sciences,* vol. 250, pp. 113--141, 2013.

[49] L. H. Le, D. Landa-Silva, M. Galar, S. Garcia and I. Triguero, "A Hybrid Surrogate Model for Evolutionary Undersampling in Imbalanced Classification," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020.

[50] L. H. Le, D. Landa-Silva, M. Galar, S. Garcia and I. Triguero, "EUSC: A clustering-based surrogate model to accelerate evolutionary undersampling in imbalanced classification," *Applied Soft Computing*, vol. 101, 2021.

[51] J. Alcalá-Fdez, L. Sánchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas and others, "KEEL: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, p. 307–318, 2009.

[52] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle and M. Birattari, "The irace Package: Iterated Racing for Automatic Algorithm Configuration," *Operations Research Perspectives*, vol. 3, 1 2011.

[53] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, "Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization," *J. Mach. Learn. Res.*, vol. 18, p. 6765–6816, 1 2017.

Appendix A. Additional plots

The distribution of GM over 10 repetitions for 44 datasets is visualized in Figure 6. Each box plot represents 10 repetitions. The horizontal-inner line shows the median. The whisker's ends show the lowest and the highest observed values (here, the whisker's scale is taken as 1.5). The ends of the color-box show the first and third quartiles, respectively. The dots in color represent outliers, and black-dots show the observed values. Apart from datasets "iris0", "shuttle-c0-vs-c4", "ecoli-0_vs_1", and "shuttle-c2-vs-c4", where all approaches get equal GM, we observe the following:

- The BO approach's median values are higher than those of the Grid-Def approach in 38/40 cases, except datasets "glass4" and "ecoli2". The most significant improvement is recorded in dataset "glass2".
- In the family of two HPO approaches, median and whiskers values of TPE are higher than those of Random search on 39/40 datasets, except the dataset "glass6".

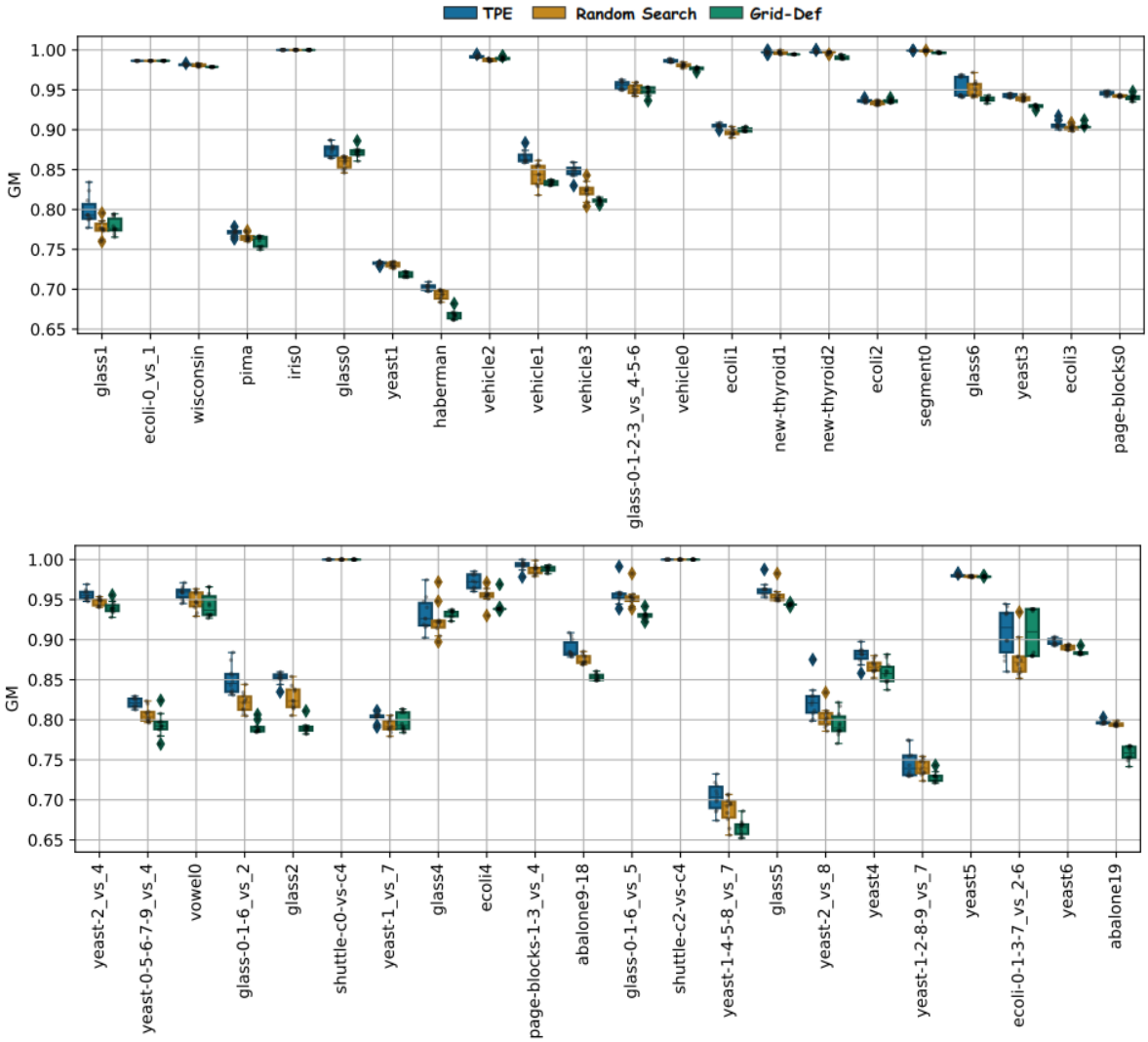

Figure 6. Box plots showing the distribution of Geometric Mean (GM) for 44 data sets. The black-dots inside boxes show the observed values