Improved Automated CASH Optimization with Tree Parzen Estimators for Class Imbalance Problems

Duc Anh Nguyen*, Jiawen Kong*, Hao Wang*, Stefan Menzel§,
Bernhard Sendhoff§, Anna V. Kononova*, and Thomas Bäck*
*Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
Email: {d.a.nguyen,j.kong,h.wang,a.kononova,t.h.w.baeck}@liacs.leidenuniv.nl
§Honda Research Institute Europe GmbH (HRI-EU), Offenbach/Main, Germany
Email: {stefan.menzel,bernhard.sendhoff}@honda-ri.de

***Abstract*—The imbalanced classification problem is very relevant in both academic and industrial applications. The task of finding the best machine learning model to use for a specific imbalanced dataset is complicated due to a large number of existing algorithms, each with its own hyperparameters. The Combined Algorithm Selection and Hyperparameter optimization (CASH) has been introduced to tackle both aspects at the same time. However, CASH has not been studied in detail in the class imbalance domain, where the best combination of resampling technique and classification algorithm is searched for, together with their optimized hyperparameters. Thus, we target the CASH problem for imbalanced classification. We experiment with a search space of 5 classification algorithms, 21 resampling approaches and 64 relevant hyperparameters in total. Moreover, we investigate performance of 2 well-known optimization approaches: Random search and Tree Parzen Estimators approach which is a kind of Bayesian optimization. For comparison, we also perform grid search on all combinations of resampling techniques and classification algorithms with their default hyperparameters. Our experimental results show that a Bayesian optimization approach outperforms the other approaches for CASH in this application domain.**

***Keywords*—hyperparameter optimization, algorithm selection, imbalanced classification, CASH optimization**

I. Introduction

The imbalanced classification problem has caught growing attention from both academic and industrial fields. Technically, any dataset with an unequal class distribution is imbalanced. However, only datasets with a significantly skewed distribution are traditionally regarded as imbalanced in the imbalanced learning domain [1]. Academic researchers aim to propose novel algorithms to handle imbalanced classification problems in different scenarios, e.g., resampling techniques and algorithm-level approaches, while industrial researchers focus on improving imbalanced classification performances for specific real-life applications, e.g., fault diagnosis or anomaly detection [2], [3].

The combination of resampling techniques and classification algorithms is the most commonly used approach for handling imbalanced data [4]. However, the No Free Lunch (NFL) theorem [5] prescribes that there is no universally best algorithm for all problems. This leads to a challenge for an imbalanced classification problem on how to choose the best model (i.e., a combination of a resampling method) and a classifier (the so-called model selection problem or algorithm selection problem [6]). Besides, both the resampling techniques and classification algorithms have their hyperparameters that need to be tuned in order to achieve better performance [7].

Therefore, two tasks have to be taken into account in this paper: model selection (MS) and hyperparameter optimization (HPO). Usually, these tasks are addressed separately and sequentially [8], [9], 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 due to a high number of combinations operations.Alternatively, the practitioner can select a suitable model, by training all models with their default hyperparameters or based on some experience, and then tune the hyperparameters only for the best model. This approach might get stuck in a local optimum of the model that has been initially chosen, based on the default hyperparameter setting. On the other hand, instead of sequentially solving these problems, they can be combined into a single problem and solved at the same time. Such approach is commonly referred to as the Combined Algorithm Selection and Hyperparameter optimization (CASH) or Full Model Selection (FMS) [10] 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 [10], [11] and Auto-Sklearn [11], [12], TPOT [13], HyperOpt-Sklearn [14]. In addition, [8] demonstrated that the CASH approach is competitive with the sequential approach and requires less computational effort. However, the CASH approach has not been studied yet in detail in the context of learning from imbalanced data.

Hence, in this work, we introduce CASH in the context of optimizing the machine learning pipeline of combined classification algorithms and resampling techniques for the class imbalance problem. We are particularly interested in studying which optimization approach for handling the CASH problem yields the best classification performance. We experiment with two well-known approaches, Random search and Tree Parzen Estimators approach (TPE) which is a kind of Bayesian

optimization, on a search space of 64 hyperparameters[1] of 5 classification algorithms (Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Decision Tree (DT) and Logistic Regression (LR)) and 21 choices of resampling techniques on a range of 44 imbalanced datasets.

Furthermore, we experiment with dropping the hyperparameter tuning and carrying out only the MS part, as sometimes done by practitioners. Our results suggest the inferiority of such approach and demonstrate that applying CASH optimization gives better performance, for all test cases considered. Moreover, we observe that the Bayesian optimization approach produces better results than Random search. Hence, we recommend using this approach for handling the CASH problem for the class-imbalanced classification problem.

The remainder of this paper is organized as follows. In Section II, the relevant background knowledge on imbalance classification and hyperparameter optimization are provided, and Section III lays out the experimental setup. Experimental results are discussed in Section IV. Finally, the paper is concluded, and further work is outlined in Section V.

II. Background

In this section, we first provide a brief introduction to imbalanced classification (Section II-A), hyperparameter optimization approaches (Section II-B), and the CASH problem (Section II-C) studied in this paper.

A. Imbalanced Classification

The main problem in imbalanced classification is that the number of samples of one class is much lower than of the other classes [1]. Here, the one or more classes being underrepresented are called minority class(es) and the other class(es) are called majority classes.

It has been shown that both the data-level (resampling) approaches and algorithm-level approaches are efficient in handling class-imbalance problems [16]. The data-level approaches focus on producing balanced datasets based on the unbalanced original data, while the algorithmic-level approaches concentrate on adjusting classification algorithms to make them appropriate for imbalanced datasets. In the imbalanced learning domain, resampling techniques can be further divided into three groups: under-resampling, over-resampling, and combine-resampling. Under-resampling balances the class distribution by removing majority samples, e.g., TomekLinks [17], while over-resampling balances the class distribution via producing synthetic minority samples, e.g., SMOTE [18]. The combine-resampling integrates both, removing the majority samples and creating synthetic minority samples in order to balance the class distribution, e.g., SMOTETomek [19].

Due to recent developments in data storage and management, it became 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 [20], 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, applying proper imbalanced classification algorithms to the design parameters could save computation time.

In the class imbalance domain, it is widely known that *accuracy* is a deceptive estimate of performance [7], [21]. Instead of *accuracy*, other metrics such as the area under the receiver operating characteristic (ROC) curve, F-measure, or geometric mean (GM) are commonly used to measure performance [22]. For comparison with previous studies [23], [24], we use GM as the performance evaluation metric, i.e.:

$$GM = \sqrt{TP_{rate} \cdot TN_{rate}}\ , \tag{1}$$

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.

B. Hyperparameter Optimization

Hyperparameter optimization (HPO) has become increasingly important in the community of machine learning and optimization [25], [26]. In the context of optimization, HPO is generally viewed as a black-box optimization problem, which is aimed at finding the global optimum x^* of the hyperparameters, with respect to some real-valued loss function f, namely,

$$x^* = \arg\min_{x \in \chi} f(x)\ , \tag{2}$$

where χ stands for the search space of hyperparameters. In the following paragraphs, we briefly introduce the HPO algorithms chosen in this study.

1) Grid search and Random search: Grid search is the most basic HPO algorithm. Given a set of hyperparameters, each of which has a (finite) set of values, we enumerate all combinations of these sets and create a list of all candidates. Grid search evaluates each of these candidates and chooses the best configuration among them – the number of function evaluations is precisely the number of configurations. However, practitioners are usually restricted by a limited computational budget, i.e., number of function evaluations, for HPO. Such limited budget is typically much smaller than the number of possible configurations to evaluate. Thus, limited budget restricts the applicability of grid search.

Unlike grid search which assesses all configurations, random search [27] evaluates only a subset of available candidate configurations at random until the given budget runs out and returns the best of the sampled configurations.

[1]Detailed information on these hyperparameters are provided in [15] (Section I)

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 relative small computational budget. Built upon surrogate models, Bayesian Optimization (BO) [28] is designed for such 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, i.e, the evaluated hyperparameters and the corresponding loss function values. The new candidate hyperparameter is chosen by optimizing the so-called acquisition function [29], 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) [30], Sequential Parameter Optimisation (SPO) [31], Mixed-Integer Parallel Efficient Global Optimization (MIPEGO) [32], and Tree-structured Parzen Estimator (TPE) [33], [34]. 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) [35], Gaussian process regression (GPR) [36], and TPE. As for the acquisition function, Expected Improvement (EI), Probability of Improvement (PI) [37], and Upper Confidence Bound (UCB) [38] are more frequently applied among many other alternatives. The initial sampling is also an essential consideration with conventional techniques including random, random sequences of low discrepancies, and Latin Hypercube Design (LHS) [39]. However, for current study we opted the random sampling. In this study, we use TPE as the BO approach and EI for the acquisition function.

C. The Combined Algorithm Selection and Hyperparameter Optimisation (CASH) Problem

The CASH problem [10] aims to identify the best machine learning model, e.g., preprocessing techniques, classification/regression algorithm and their hyperparameters, for minimizing the loss value of an arbitrary real-valued objective function f.

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

$$A^*, x^* = \underset{A^{(i)} \in \mathcal{A}, x \in \chi^{(i)}}{\arg\min} \frac{1}{K} \sum_{j=1}^{K} f(A^{(i)}_x, \mathcal{D}^{(j)}_{train}, \mathcal{D}^{(j)}_{valid}). \quad (3)$$

Here, $f(A^{(i)}_x, \mathcal{D}^{(j)}_{train}, \mathcal{D}^{(j)}_{valid})$ denotes the loss achieved by the learning algorithm $A^{(i)}$ and its corresponding hyperparameters $x \in \chi^{(i)}$ when trained and evaluated on $\mathcal{D}^{(j)}_{train}, \mathcal{D}^{(j)}_{valid}$.

Note that most HPO methods in practice can handle the CASH problem by modeling the choice of algorithms as a categorical hyperparameter. Each algorithm is mapped to its locally dependent hyperparameters by the so-called conditional parameter.

III. Experimental Setup

In this section, we briefly introduce the datasets (Section III-A), resampling techniques (Section III-B) and hyperoptimization approaches (Section III-C) chosen in this work. Finally, we specify the experimental procedure (Section III-D).

A. Datasets

For this study, 44 binary class imbalanced datasets from the KEEL repository [40] are used. Their *Imbalance Ratio* (IR), i.e., the ratio of the number of majority class instances to that of minority class instances, ranges here from 1.82 to 129.44. Figure 1 shows 44 examined datasets presenting the relation of the imbalance ratio (#IR) on the x-axis and the number of samples (#samples) on the y-axis; meanwhile colour of the symbols denotes the number of attributes for each dataset. The full list of datasets is given in [15] (Section II) of the supplementary materials file.

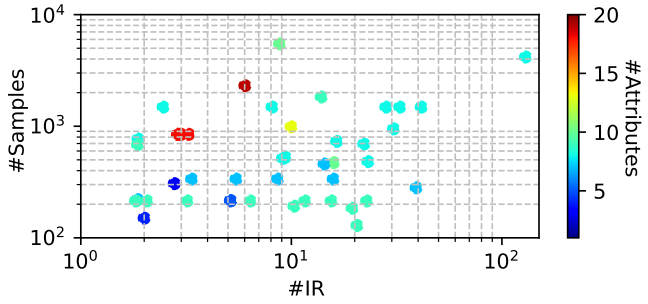

Fig. 1. Overview of the characteristics of the datasets. The scatter plot the Imbalance Ratio (#IR) and the number of samples (#samples) for all 44 datasets on a logarithmic scale. The color indicates the number of attributes.

B. Resampling Algorithms

The resampling algorithms are designed to handle the class imbalance scenario by producing balanced datasets. The resampling algorithms used in our experiments can be arranged into three groups:

1) *Over-resampling (7 algorithms)*: In the imbalanced learning domain, SMOTE is the most famous resampling technique and it generates synthetic samples based on the random interpolation between the chosen minority samples and their K-nearest neighbors. Various SMOTE-based extensions have been proposed to give further improvement on the SMOTE basis. For example, ADASYN [41] focuses on the harder-to-learn sample and BorderlineSMOTE [42] emphasizes the borderline samples. Other over-resampling approaches considered in this paper are KMeansSMOTE [43], SMOTENC [18], SVMSMOTE [44] and RandomOverSampler [45].
2) *Under-resampling (11 algorithms)*: In a binary classification problem, a Tomek link is defined as a pair

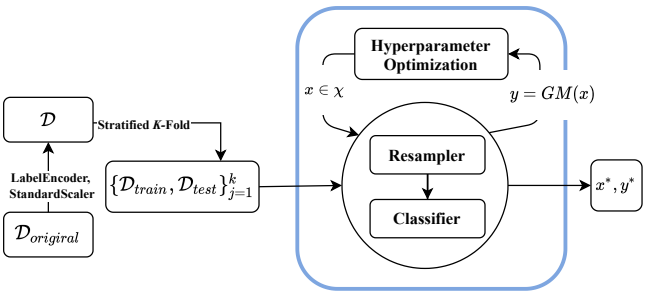

Fig. 2. Flowchart of the experimental setup.

of samples from different classes which are the nearest neighbors for each other [17]. The undersampling method TomekLinks removes the Tomek links in the dataset in order to produce a clear decision boundary. OneSidedSelection [46] first removes noisy and borderline majority samples, then removes the safe majority samples which have limited contribution for building the decision boundary with the CondensedNearestNeighbour Rule [47]. Other under-resampling methods considered in this paper are CondensedNearestNeighbour, EditedNearestNeighbours [48], RepeatedEditedNearestNeighbours [49], AllKNN [49], InstanceHardnessThreshold [50], NearMiss [51], NeighbourhoodCleaningRule [52], ClusterCentroids [53], and RandomUnderSampler [54].

The family of evolutionary under-resampling techniques (EUS) have been proved powerful to handle instance reduction [55]. An EUS algorithm tries to optimize the selected samples in the majority class by performing a binary search guided by an evolutionary algorithm [56]. Results of the EUS and the most recent research studies in this family consist of EUS-Windowing (EUSW) [57], clustering-based surrogate model for EUS (EUSC) [23] and hybrid surrogate model for EUS (EUSHC) [24] are also compared with our approach in the followed section.

3) *Combine-resampling (2 algorithms)*: SMOTETomek first oversamples the minority class using SMOTE, after which the Tomek links for the after-sampled samples are removed. Similar to SMOTETL, SMOTEENN first oversamples the minority class with SMOTE. After that, the Wilson's Edited Nearest Neighbors (ENN) is used to remove the sample which has a different class from at least two of its three nearest neighbors [58].

The setup also allows a "no resampling" option. The resampling algorithms are implemented in the Python package **imbalanced-learn**[2]

C. Hyperparameter Optimization

In this study, random search and TPE as implemented in the Python package **HyperOpt**[3] are used as HPO algorithms. Based on the initial experiments, we set the number of iterations of HPO to 500, after which the algorithms have shown no significant improvements.

Moreover, to study the effectiveness of the HPO algorithms, we evaluated all possible combinations of classification and resampling algorithms with their default hyperparameter settings. On each dataset, we report the combination with the highest GM. As mentioned in Section I, an experimental study with 5 classification algorithms and 21 resampling techniques is conducted, thus making $5 \times 21 = 105$ combinations in total. Evaluating these combinations one by one is referred to as "Grid-Def" here (grid search HPO algorithm).

D. Implementation details

Algorithm 1: Experimental setup

Input: χ: Search space, r: Random seed, K: Number of folds , m: Number of iterations , f: Objective function (*see Algorithm* 2)
Output: x^*: the best configuration, y^*: GM achieved by x^*
Data: dataset $\mathbf{D}$

1 $\mathbf{D} \leftarrow$ DATAPREPROCESS($\mathbf{D}$)
/* DATAPREPROCESS includes LABELENCODER, STANDARDSCALER */
2 $\{\mathcal{D}_{train}, \mathcal{D}_{test}\}_{j=1}^{K} \leftarrow$ STRATIFIEDK-FOLD($\mathbf{D}, K, r$)
3 HPO $\leftarrow$ HPO.INIT($\chi, f, r, m, \{\mathcal{D}_{train}, \mathcal{D}_{test}\}_{j=1}^{K}$)
/* initialize optimizer */
4 $x^*, y^* \leftarrow$ HPO.OPTIMIZE()

The overall structure of our implementation is summarized in Fig. 2. The process begins with data preprocessing on the input dataset. The 5-fold cross-validation is applied at this step to overcome the over-fitting problem. The outcome is fed into the second phase consisting of resampling and classification processes. Our contribution is emphasized at this phase, where the hyperparameters of the resampler and classifier are tuned. This block is highlighted in the blue rounded rectangle. The complete pseudo-code of this flowchart is elaborated in Algorithm 1.

Algorithm 1 consists of the following two steps:

- Preprocessing (line 1-2): We need to apply data preprocessing since machine learning models require input and output data to be numeric. Thus, we use Label encoder[4] to encode any categorical data to a number for the input dataset. Then, we apply Standard Scaler[4] on the encoded dataset to have zero mean and a standard deviation of one (line 1). Next, Stratified K-fold cross-validation[4] using $K = 5$, commonly used in the literature, is used.
- Hyperparameter optimization (line 3-4): All parameters of HPO are initialized (line 3), taking values from the provided input including search space χ, random seed r, number of iterations m, objective function f and K folds of the examined dataset. Then, the algorithm optimizes the search space χ until the number of function evaluations reaches 500.

[2]https://github.com/scikit-learn-contrib/imbalanced-learn (version 0.7.0)
[3]https://github.com/hyperopt/hyperopt (version 0.2.5)

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

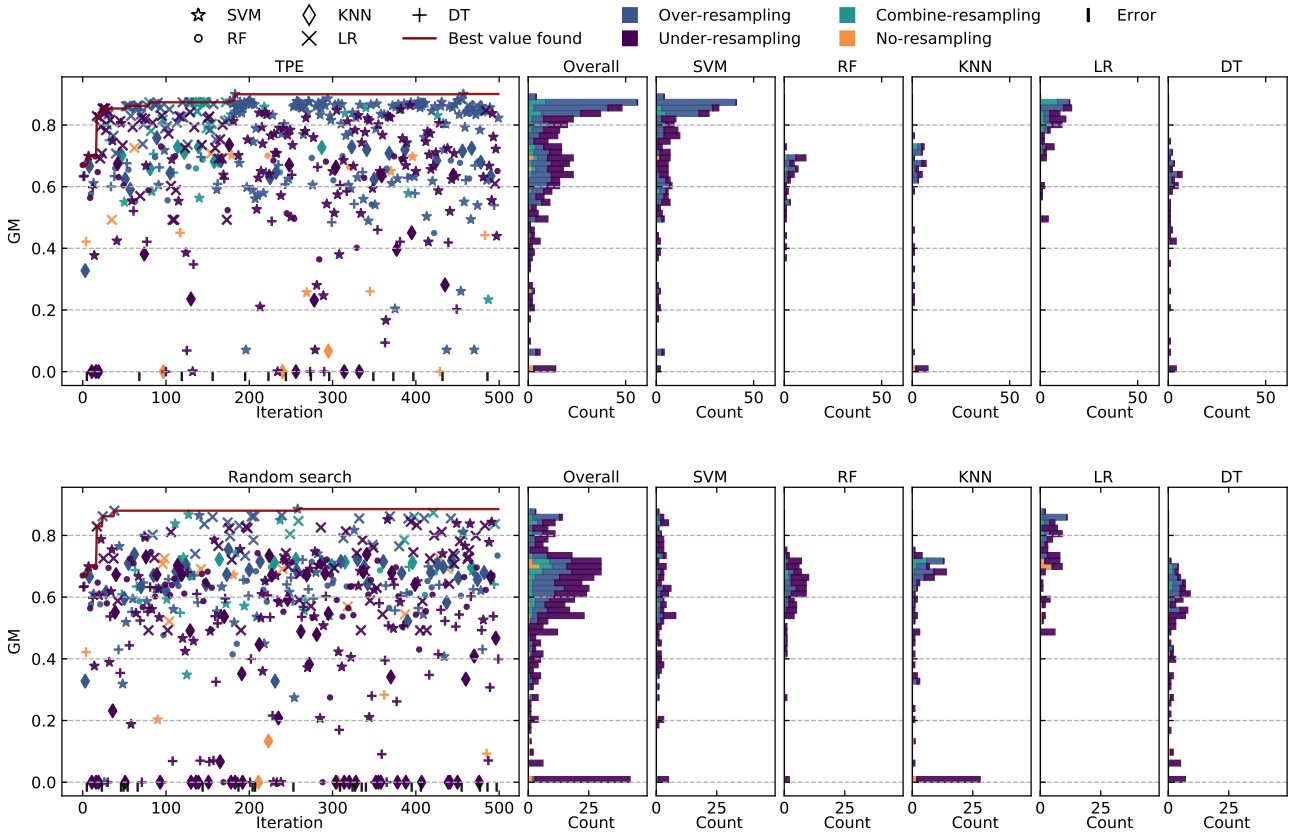

Fig. 3. Illustration of the hyperparameter tuning process on dataset "abalone9-18". Scatter plots show the sequence of observed values vs the number of function evaluations, the red line shows the current best value, and the black vertical bars indicate the infeasible configurations where GM returns 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 of the classification algorithms, namely SVM, RF, KNN, LR and DT.

Algorithm 2: Objective function

Input: Hyperparameter configuration $x \in \chi$ generated by HPO; r: Random seed

/* $x = \underbrace{x_{re_0}, x_{re_1}, \ldots, x_{re_q}}_{\text{RESAMPLER}}, \underbrace{x_{cls_0}, x_{cls_1}, \ldots, x_{cls_p}}_{\text{CLASSIFIER}}$ */

Data: $\{\mathcal{D}_{train}, \mathcal{D}_{test}\}_{j=1}^{K}$

1 **foreach** $\{\mathcal{D}_{train}^{(j)}, \mathcal{D}_{test}^{(j)}\} \in \{\mathcal{D}_{train}, \mathcal{D}_{test}\}_{j=1}^{K}$ **do**

/* Build resampler and classifier models */

2 RESAMPLER $\leftarrow$ Build RESAMPLER x_{re_0} with the hyperparameters $\{x_{re_1}, \ldots, x_{re_q}\}$ and random seed r

3 CLASSIFIER $\leftarrow$ Build CLASSIFIER x_{cls_0} with the hyperparameters $x_{cls_1}, \ldots, x_{cls_p}$ and random seed r

4 $\mathcal{D}_{train}^{(j)} \leftarrow$ RESAMPLER$(\mathcal{D}_{train}^{(j)})$

5 $y_j \leftarrow$ CLASSIFIER.LEARN$(\mathcal{D}_{train}^{(j)})$.EVALUATE$(\mathcal{D}_{test}^{(j)})$

6 **return** $y \leftarrow \frac{1}{K}\sum_{j=1}^{K} y_j$

The computation of the objective function is presented in Algorithm 2. It elaborates further steps shown in the circle in Fig. 2. The input is a parameter setting generated by HPO consisting of random seed r and hyperparameter configuration x. The configuration x consists of two parts: the choice of resampler represented by x_{re_0}, and classifier denoted by x_{cls_0}, together with their corresponding hyperparameter settings $\{x_{re_1}, \ldots, x_{re_q}\}$ and $\{x_{cls_1}, \ldots, x_{cls_p}\}$.

For a fold of the examined dataset, the computation of an evaluation has the following steps:

- Step 1 (line 2-3): Resampler and classifier are initialized, using values of the configuration x and random seed r.
- Step 2 (line 4-5): The selected resampler is applied to the fold, followed by the classifier, which is applied to the balanced result from the resampler. Then, the GM on each validation fold is calculated.

The final value of the objective function is an average GM of K folds (line 6). Additionally, since the used algorithms, i.e., classifiers and resamplers, in the objective function are stochastic, we fixed 10 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. The reported result of each dataset for an individual HPO approach is averaged over 10 executions.

IV. Results and Discussion

In this section, we report the results and discuss our insights. The experimental results are summarised in Table I to illustrate

TABLE I

AVERAGE GEOMETRIC MEAN (ROUNDED TO 4 DECIMALS) OVER 10 REPETITIONS FOR THE 44 DATASETS, ORDERED BY INCREASING IR VALUE. THE LEFT PART SHOWS OUR EXPERIMENTAL RESULTS, I.E., TPE, RANDOM SEARCH (RS), GRID-DEF (GRID), AND THE RIGHT PART (GREY SHADED) THOSE OBTAINED BY EVOLUTIONARY ALGORITHMS ACCORDING TO [23], I.E., EUS, EUS-WINDOWING, EUSC, EUSHC, RESPECTIVELY. BOLDFACE HIGHLIGHTS THE BEST PERFORMING METHOD PER DATASET ON EACH GROUP AND UNDERLINE INDICATES RESULTS THAT ARE SIGNIFICANTLY DIFFERENT FROM THE BEST METHOD IN THAT GROUP ACCORDING TO A WILCOXON SIGNED-RANK TEST ($p < 0.05$). A VALUE LABELED WITH * INDICATES THAT OUR EXPERIMENTAL RESULT OUTPERFORMS THE RESULTS FROM [23] FOR THE CORRESPONDING DATASET.

Dataset	**#IR**	**Our experimental results**			**Evolutionary algorithms** [23]				**Overall**
		TPE	**RS**	**Grid**	**EUS**	**EUSW**	**EUSC**	**EUSHC**	**Winner**
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 ǀ RS ǀ 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	2.00	__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	2.90	*__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	3.20	__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 ǀ 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	8.10	*__0.9428__	*0.9395	*0.9290	0.8728	0.8740	__0.8752__	0.8550	TPE
ecoli3	8.60	*__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
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 ǀ RS ǀ Grid
yeast-1_vs_7	14.30	*__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	15.80	*__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	16.40	*__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	20.50	*__1__	*__1__	*__1__	__0.9577__	0.6449	0.9414	0.7365	TPE ǀ RS ǀ Grid
yeast-1-4-5-8_vs_7	22.10	*__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	23.10	*__0.8231__	*0.8031	*0.7945	__0.7931__	0.7496	0.7656	0.7668	TPE
yeast4	28.10	*__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	41.40	*__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

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 the state-of-the-art Evolutionary under-resampling (EUS) methods [23]. In this table, our results are presented in the corresponding columns on the left side (not shaded) and the results from [23] are presented on the right side (grey shaded) for EUS, EUSW, EUSC and EUSHC, respectively. In both groups, the highest performance for each dataset is highlighted in bold. In our 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 our result outperforms those from [23] 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 allow the following insights:

- HPO approaches exhibit better performance compared

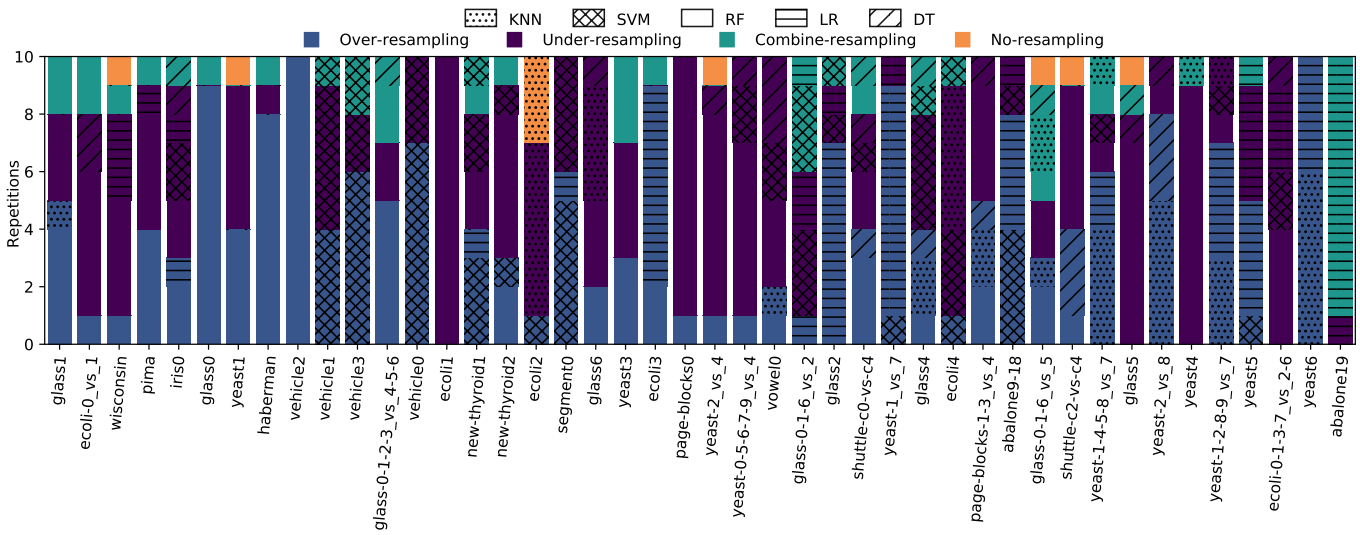

Fig. 4. Resulting combination of classifier and resampler choices for an optimization process using TPE across 10 repetitions on 44 datasets. Figure best viewed in color.

to the Grid-Def approach which uses static 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[5].

- Overall, TPE shows the highest GM for 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 dataset "iris0".
- Furthermore, based on our experimental results, we conclude that TPE wins over the methods from [23] on $41/44$ datasets, RS – on $38/44$ datasets and Grid-Def – on $37/44$ datasets. This is surprising since the number of function evaluations used in our experiment is much smaller than in [23]: 500 function evaluations for TPE and RS, 105 function evaluations for Grid-Def vs 10.000 function evaluations for each method in [23]. A possible explanation for this might be that [23] employs a simple KNN rule with $k = 1$ as the mere classifier, while more complicated classification algorithms are used in our study. More precisely, according to our experimental results, tuned KNN wins only in 11% (TPE), 13% (RS), and 9% (Grid-Def) of all cases.

To investigate the tuning behavior of the methods, we plot single runs of TPE and RS on the dataset "abalone9-18"[6] in Fig. 3. The scatter plots on the left show the observed GM values over 500 function evaluations. The scatter plots on the left show the observed GM values over 500 function evaluations. Six last stacked histogram plots describe the distribution of the observed values, in which the first plot shows all observed values, and the five last plots indicate the distributions for each of the classification algorithms, i.e. SVM, RF, KNN, LR and DT. We conclude that:

- Configurations generated by TPE can avoid area of infeasible parameters better than RS[7]. In this run, the number of errors occurring in the TPE and RS runs are 14 and 22, respectively. Based on all datasets and repetitions, the number of infeasible configurations encountered by 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 of RS are distributed in the range from 0.6 to 0.7.

Based on the highest results obtained by TPE, Fig. 4 shows the final combination of choices of classification algorithms and resampling approaches once the optimization run is over. Clearly, no dominant algorithm exists over many datasets 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 some resampling technique. Particularly, over-resampling, under-resampling and combine-resampling obtain 182, 199, 50 wins over $44 \times 10 = 440$ runs. Additionally, there is no classifier/resampler combination 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 and 29 runs, respectively.

[5]Full details of our experimental results are provided in [15] (Section V) of the supplementary materials file

[6]Note that, due to the page limitation, the plots for this dataset over 10 repetitions can be found in Section IV of the supplementary material [15]

[7]Evaluations with infeasible combinations of parameters are marked on the figure as black dashes with $GM = 0$.

V. Conclusions and Future Work

In this paper, we applied a special type of Bayesian Optimization approach, the Tree Parzen Estimators to optimize the combined algorithm selection and hyperparameter optimization problem to improve the performance of classification algorithms for the class imbalance problems. In other words, we proposed an automated CASH optimization approach for imbalanced classification problems. Our approach automatically chooses the best set of algorithms, i.e., resampling technique and classification algorithm, together with their optimized hyperparameter settings for an arbitrary imbalanced dataset. The numeric results show significantly improved performance with respect to the state-of-the-art techniques in the imbalanced classification domain over 44 examined datasets.

Four main conclusions can be drawn from our experimental results:

1) Use of HPO clearly improves classification performance compared to using static default parameters.
2) TPE outperforms Random search on 91% of the tested datasets, while equal performance is found on the remaining cases.
3) Overall, the TPE approach produces the best results among other competitors in various scenarios. Hence, we recommend using TPE for handling CASH optimization in imbalanced classification problems.
4) Another finding is that 98% of runs yield the best performance with the help of resampling techniques. Thus we recommend researchers to use resampling to deal with class imbalanced problems.

There are several interesting research directions for extending this work. Firstly, we intend to apply other Bayesian optimization variants such as SMAC, SPO, and MIPEGO, in order to study the performance between variants in this domain. Secondly, the scope of this study was limited in terms of classification problems; therefore, our future research might extend our research for regression problems. Thirdly, in addition to GM, other commonly used performance evaluation metrics in this domain will be investigated in our future work, including the Area Under the ROC Curve (AUC), F-measure, and recall. Fourthly, the penalty-based methods, e.g., penalized-SVM, themselves can efficiently handle imbalanced datasets in several cases. Thus, we plan to study their effectiveness in the context of CASH optimization. Additionally, instead of applying hyperparameter tuning on the level of an individual dataset, we are interested in studying the behavior of HPO approaches when tuning for a set of datasets. Finally, besides Bayesian optimization, we will extend our research with other state-of-the-art HPO approaches such as iRace [59] and Hyperband [60] for the class-imbalanced problem.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186 (ECOLE).

References

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

[2] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[3] 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*, pp. 1–10, 2014.

[4] V. López, A. Fernández, J. G. Moreno-Torres, and F. Herrera, "Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6585–6608, 2012.

[5] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, April 1997.

[6] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," 2018.

[7] J. Kong, W. Kowalczyk, D. A. Nguyen, S. Menzel, and T. Bäck, "Hyperparameter optimisation for improving classification under class imbalance," in *IEEE Symposium Series on Computational Intelligence*, pp. 3072–3078, 2019.

[8] 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*, GECCO '20, (New York, NY, USA), p. 903–912, Association for Computing Machinery, 2020.

[9] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *J. Mach. Learn. Res.*, vol. 15, p. 3133–3181, Jan. 2014.

[10] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," *KDD*, 08 2012.

[11] 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, no. 25, pp. 1–5, 2017.

[12] 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, MA, USA), p. 2755–2763, MIT Press, 2015.

[13] R. S. Olson and J. H. Moore, *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, pp. 151–160. Cham: Springer International Publishing, 2019.

[14] B. Komer, J. Bergstra, and C. Eliasmith, *Hyperopt-Sklearn*, pp. 97–111. Cham: Springer International Publishing, 2019.

[15] D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. V. Kononova, and T. Bäck, "Supplementary material for improved automated cash optimization with tree parzen estimators for class imbalance problems." https://github.com/ECOLE-ITN/CASH4IMBALANCE, 2021.

[16] V. Ganganwar, "An overview of classification algorithms for imbalanced datasets," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 4, pp. 42–47, 2012.

[17] I. Tomek, "Two modifications of cnn," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, pp. 769–772, Nov 1976.

[18] 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, pp. 321–357, 2002.

[19] 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, 01 2003.

[20] 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*, pp. 84–96, Springer, 2020.

[21] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[22] 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.

[23] H. L. 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)*, pp. 1–8, 2020.

[24] H. L. 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 Comp.*, vol. 101, p. 107033, 2021.

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

[26] H. H. Hoos, *Automated Algorithm Configuration and Parameter Tuning*, pp. 37–71. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[27] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, 03 2012.

[28] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, Dec. 1998.

[29] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference Novosibirsk* (G. I. Marchuk, ed.), pp. 400–404, 1975.

[30] 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*, pp. 507–523, 2011.

[31] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss, "Sequential parameter optimization," *2005 IEEE Congress on Evolutionary Computation*, vol. 1, pp. 773–780, 01 2005.

[32] H. Wang, B. v. 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.

[33] 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*, NIPS'11, pp. 2546–2554, 2011.

[34] 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.

[35] L. Breiman, "Random forests," *Machine Learning*, vol. 45, p. 5–32, Oct. 2001.

[36] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[37] D. Jones, "A taxonomy of global optimization methods based on response surfaces," *J. of Global Optimization*, vol. 21, pp. 345–383, 12 2001.

[38] P. Auer, "Using confidence bounds for exploitation-exploration tradeoffs.," *Journal of Machine Learning Research*, vol. 3, pp. 397–422, 01 2002.

[39] 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, no. 2, pp. 239–245, 1979.

[40] 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, *et al.*, "Keel: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009.

[41] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1322–1328, IEEE, 2008.

[42] H. Han, W. Wang, and B. H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," vol. 3644, pp. 878–887, 09 2005.

[43] F. Last, G. Douzas, and F. Bacao, "Oversampling for imbalanced learning based on k-means and smote," 2017.

[45] Imbalanced-learn, "Random over sampler." https://imbalanced-learn.org/stable/generated/ imblearn. over_sampling. RandomOverSampler.html. Accessed: 2020-08-30.

[46] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *In Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186, Morgan Kaufmann, 1997.

[47] 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, July 1979.

[48] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-2, no. 3, pp. 408–421, 1972.

[49] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, pp. 448–452, June 1976.

[50] M. R. Smith, T. Martinez, and C. Giraud-Carrier, "An instance level analysis of data complexity," *Mach. Learn.*, vol. 95, p. 225–256, May 2014.

[51] 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.

[52] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," pp. 63–66, 06 2001.

[53] Imbalanced-learn, "Cluster centroids." https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.ClusterCentroids.html. Accessed: 2020-08-30.

[54] Imbalanced-learn, "Randomunder sampler." https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.RandomUnderSampler.html. Accessed: 2020-08-30.

[55] S. García and F. Herrera, "Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy," *Evolutionary Computation*, vol. 17, no. 3, pp. 275–306, 2009.

[56] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. USA: Oxford University Press, Inc., 1996.

[57] J. Bacardit, D. Goldberg, M. Butz, X. Llora, and J. M. Garrell, "Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy," vol. 3242, pp. 1021–1031, 09 2004.

[58] G. E. 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, no. 1, pp. 20–29, 2004.

[59] 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, 01 2011.

[60] 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, pp. 6765–6816, Jan. 2017.