



Experience-based Computation:  
**Learning to Optimise**

**Project Number: 766186**

**Project Acronym: ECOLE**

**Project title: Experienced-based Computation: Learning to Optimise**

### **Deliverable D2.4**

**Integrated Software Environment and Manual for Imbalanced Learning,  
Automated Machine Learning, Model-Assisted Robust Optimization, and  
Dynamic Multi-Objective Optimization**

#### **Authors:**

**Sibghat Ullah, Duc Anh Nguyen, Jiawen Kong, Hao Wang, Anna Kononova,  
Wojtek Kowalczyk, and Thomas Bäck – Universiteit Leiden  
Gan Ruan and Xin Yao – University of Birmingham**

**Project Coordinator: Professor Xin Yao, University of Birmingham**

**Beneficiaries: Universiteit Leiden, Honda Research Institute Europe, NEC Laboratories  
Europe**

**H2020 MSCA-ITN**

**Date of the report: 31.09.2021**



## Contents

|   |    |
|---|----|
| Executive summary.....  | 3  |
| Major Achievements.....   | 3  |
| 1. Introduction.....  | 6  |
| 2. Learning from Imbalanced Data .....  | 8  |
| 2.1. On the Performance of Oversampling Techniques for Class Imbalance Problems.....                            | 8  |
| 2.2. Improving Imbalanced Classification by Introducing Additional Attributes .....                             | 9  |
| 3. Automated Machine Learning .....   | 13 |
| 3.1. Automatically Optimize the Class Imbalance Problems.....   | 13 |
| 4. Model-Assisted Robust Optimization.....  | 18 |
| 4.1. Investigation of Modelling Techniques for Robust Optimization .....  | 18 |
| 4.1. Investigation of Dimensionality Reduction Techniques for Efficient SAO .....                               | 19 |
| 4.2. Extending the Moment-Generating Function of Improvement (MGFI) for Robust Bayesian Optimization (RBO)..... | 21 |
| 5. Knowledge Transfer in Dynamic Multi-Objective Optimization.....  | 25 |
| 5.1. When and How to Transfer Knowledge in Dynamic Multi-objective Optimization? .....                          | 25 |
| 5.2. Improving the Efficiency of Knowledge Transfer in Dynamic Multi-objective Optimization.....                | 27 |
| 6. Summary and Outlook .....  | 28 |
| Appendix.....   | 29 |
| 1. User Manual for Section 2 .....  | 29 |
| 2. User Manual for Section 3 .....  | 31 |
| 3. User Manual for Section 4 .....  | 34 |
| 4. User Manual for Section 5 .....  | 38 |
| Bibliography .....  | 41 |

## Executive summary

The objective of WP2.4 is to ready our scientific and software developments for handling imbalanced data sets, automated machine learning, model-assisted robust optimization, and dynamic multi-objective optimization. Eight peer-reviewed papers (please see the publication list in Section 1) have been published for these four research topics. In this report, we briefly describe these publications and focus mainly on the details of software implementations, which will facilitate future utilization of scientific outputs of this project. Note that the detailed information on these eight publications have already been provided in the previous deliverable reports, namely D2.1, D2.2, D2.3, D3.4 and D3.1 respectively. This report focuses on introducing software components, including code repositories, code workflow, and user manuals.

## Major Achievements

Major scientific achievements regarding work package 2.4 are presented below, where we present them with the motivating research questions/practical issues and short answers thereto:

| Research Questions  | Discussion  |
|---|---|
| Over 90 resampling techniques have been developed in the imbalanced learning domain. Do oversampling techniques that take global information of minority samples (new ones) perform better than those which heavily rely on local information of the minority class samples (classical ones)? | From our experimental results, we can conclude that, in most cases, oversampling approaches which consider the minority class distribution (Rapidly Converging Gibbs (RACOG) [1], Wrapper-based Rapidly Converging Gibbs (wRACOG) and Random Walk Oversampling (RWO-Sampling) [2]) perform better and RACOG gave the best performances among the six reviewed oversampling techniques.  |
| Does introducing additional attributes bring improvement to the imbalanced classification performance?  | According to our experimental results, introducing additional attributes can improve the imbalanced classification performance in most cases (6 out of 7 datasets). Furthermore, the proposed idea of introducing additional attributes is simple to implement and can be combined with resampling techniques and other algorithmic-level approaches in the imbalanced learning domain. |
| Existing resampling techniques and classification algorithms have been proved powerful to handle imbalanced datasets. However, how to efficiently choose the best-suited combination of a resampling  | We proposed an automated Combined Algorithm Selection and Hyperparameter (CASH) optimization approach for imbalanced classification problems, which automatically chooses the best set of   |

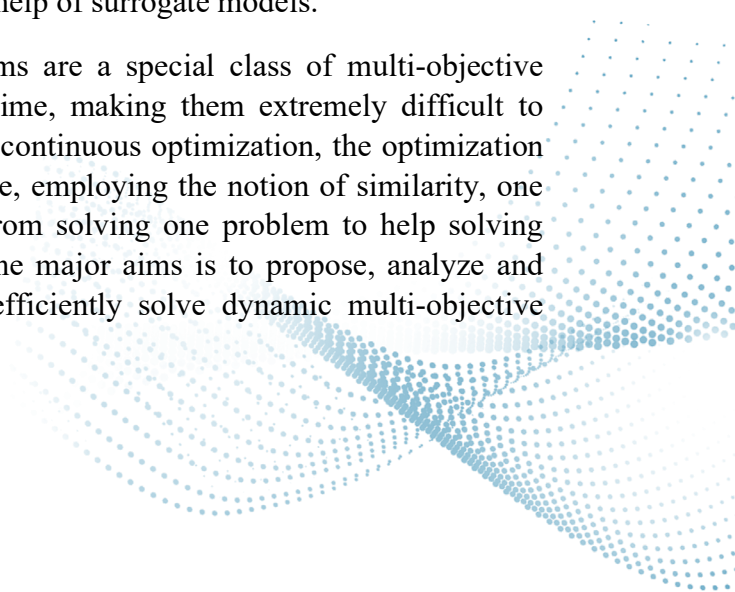
|  |  |
|--|--|
| technique and a classification algorithm for a given problem?  | algorithms, i.e., resampling technique and classification algorithm, together with optimized hyperparameter settings for an arbitrary imbalanced dataset. The numeric results show significantly improved performance with respect to the state-of-the-art techniques in the imbalanced classification domain over 44 examined datasets.   |
| What is the effectiveness of resampling techniques for automatically handling imbalanced classification?   | Our findings indicate that 98% of runs yield the best performance with the help of resampling techniques. Thus, we recommend to use resampling techniques to deal with class imbalance problems.   |
| What is the most efficient optimization approach to deal with the model selection and hyperparameter optimization problems for handling class imbalance problems?  | Our results demonstrate that Bayesian optimization is the best approach for the combined model selection and hyperparameter optimization for this aim.   |
| The co-called "Moment-generating function of the improvement" (MGFI) has been proposed as an acquisition function (AF) for the Bayesian Optimization (BO) algorithm. However, how can it be extended to the robust scenario? | The MGFI can be extended to the robust scenario by: Substituting the minimum observed value of the function (used in nominal BO algorithm) with the robust optimal value predicted by the Kriging model, and by defining the improvement over the robust optimal value predicted by the Kriging model.                                     |
| How does the performance of the robust MGFI compare to that of the baseline?   | The performance of the extended MGFI is competitive to that of the baseline – the robust expected improvement criterion – as it performs superior in 6/12 test cases (see Figure 7).   |
| What is the impact of the initial temperature setting on the performance of the robust MGFI?   | Our findings indicate that the initial temperature setting is problem-dependent and should be configured with hyperparameters optimization in practical scenarios (see Table 8)  |
| When and how to transfer knowledge in dynamic multi-objective optimization (DMOO)?   | By comparing the quality of transferred solutions and those without transfer on a set of benchmark problems with various environmental changes, it is found that the transfer fails on problems with fixed Pareto optimal solution sets, and under small environmental changes. Therefore, it is recommended to avoid transfer under these |

|  |   |
|--|---|
|  | conditions. A mathematical proof demonstrates that the Gaussian kernel function in the existing algorithm is not ideal. Therefore, a linear kernel is proposed to replace the Gaussian one.   |
| How to improve the efficiency of transfer learning in DMOO and whether the improved efficiency will affect the quality of transferred solutions? | Transfer learning is found to be very time-consuming as the 'inner' optimization method in transfer learning is very costly. Two alternative optimization methods can replace the existing 'inner' optimization method to improve the efficiency of the transfer learning. Experimental results show that the greatly enhanced efficiency does not result in huge degeneration on the performance of transfer learning. |



## 1. Introduction

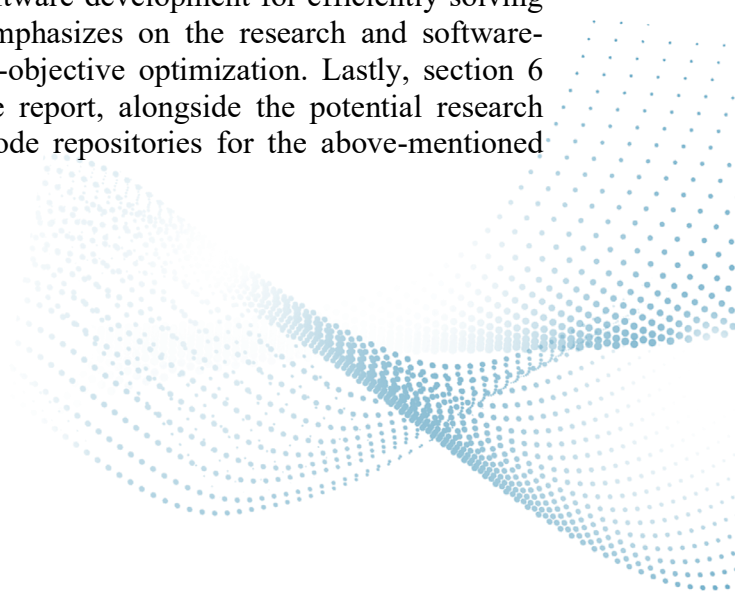
ECOLE aims at shortening the product-development cycle, reducing the resource consumption during the complete process, and creating more balanced and innovative products in engineering and ICT sectors, where the following practical difficulties must be addressed: learning and mining with imbalanced data sets, automatic algorithm configuration and hyperparameter optimization, optimization of expensive-to-evaluate black-box problems in the presence of uncertainty and noise, and dynamic multi-objective optimization.

- 1) Learning and mining in the presence of imbalanced data sets pose additional challenges. Traditional classification techniques, e.g., logistic regression, intrinsically pay more attention to the majority class. In contrast, in practical applications of imbalanced learning, the minority class is of much more importance, e.g., rare-disease identification in electronic healthcare data sets and fraud detection in financial applications. In ECOLE, one of the principal aims is to improve the classification accuracy of the minority class without losing so much the accuracy of the majority class for practical scenarios.
  - 2) The practical applications of machine learning and continuous optimization in the engineering and ICT sectors emphasize efficiency. Hence, it is necessary to automatically learn the best configurations of machine learning and optimization algorithms with considerably less human effort. Therefore, this topic in ECOLE encompasses proposing, evaluating, and comparing techniques to automatically configure the best algorithms and hyperparameters for machine learning applications.
  - 3) When solving real-world optimization problems, a frequently-encountered obstacle is the presence of uncertainties and noise within the system, or model of the system, for which optima are sought. Uncertainties or noise can affect the objective landscape significantly [3]. Therefore, the nominal optimum found by the common optimization algorithms may not be optimal for practical applications where unexpected drift and changes can occur. Uncertainties or noise can also affect the accuracy and convergence speed of optimization algorithms [3] [4], thereby directly affecting the quality of the optimal solution. In ECOLE, one of the major aims is to propose, evaluate, and compare techniques for efficiently solving the noisy black-box problems with the help of surrogate models.
  - 4) Dynamic multi-objective optimization problems are a special class of multi-objective problems where the objectives change over time, making them extremely difficult to optimize directly [5]. In practical scenarios of continuous optimization, the optimization problems are related with each other. Therefore, employing the notion of similarity, one can, in principle, extract knowledge gained from solving one problem to help solving another related problem. In ECOLE, one of the major aims is to propose, analyze and evaluate methods to transfer knowledge to efficiently solve dynamic multi-objective optimization problems.
- 

This report summarizes the implementations of solutions to the issues mentioned above and software development. The following publications are contributing to this report:

- S. Ullah, H. Wang, S. Menzel, B. Sendhoff, and T. Bäck, "An Empirical Comparison of Meta-Modeling Techniques for Robust Design Optimization", in *IEEE Symposium Series on Computational Intelligence*, Xiamen, 2019.
- S. Ullah, et al., "Exploring Dimensionality Reduction Techniques for Efficient Surrogate-Assisted Optimization", in *IEEE Symposium Series on Computational Intelligence*, Canberra, 2020.
- S. Ullah, H. Wang, S. Menzel, B. Sendhoff, and T. Bäck, "A New Acquisition Function for Robust Bayesian Optimization of Unconstrained Problems", in *Genetic and Evolutionary Computation Conference Companion*, Lille, 2021.
- J. Kong, W.J. Kowalczyk, S. Menzel and T. Bäck, "Improving Imbalanced Classification by Anomaly Detection", in *Sixteenth International Conference on Parallel Problem Solving from Nature*, Leiden, 2020.
- J. Kong, et al., "On the Performance of Oversampling Techniques for Class Imbalance Problems", in *24<sup>th</sup> Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Singapore, 2020.
- D. A. Nguyen, et al., "Improved Automated CASH Optimization with Tree Parzen Estimators for Class Imbalance Problems", in *IEEE International Conference on Data Science and Advanced Analytics*, Porto, 2021.
- G. Ruan, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "When and how to transfer knowledge in dynamic multi-objective optimization", in *IEEE Symposium Series on Computational Intelligence*, Xiamen, 2019.
- G. Ruan, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "Computational Study on Effectiveness of Knowledge Transfer in Dynamic Multi-objective Optimization", in *IEEE Congress on Evolutionary Computation*, Glasgow, 2020.

The remainder of this report is organized as following. Section 2 highlights the implementation and software-development for learning and mining with imbalanced data sets in ECOLE. Section 3 presents the same information for automatic algorithm configuration and hyperparameters optimization. Section 4 illustrates the research and software-development for efficiently solving the noisy black-box problems, whereas section 5 emphasizes on the research and software-development for transfer learning for dynamic multi-objective optimization. Lastly, section 6 provides the overall summary and conclusion of the report, alongside the potential research challenges and opportunities. The structure of the code repositories for the above-mentioned publications is provided in the Appendix.





## 2. Learning from Imbalanced Data

The imbalanced classification problem has caught growing attention from both the academic and industrial fields. Technically, any dataset with an unequal class distribution is imbalanced. However, only datasets with a significantly skewed distribution are regarded to be imbalanced in the imbalanced learning domain [6]. This type of data will result in a reduction in the effectiveness of classical machine learning classification algorithms, because these algorithms assume that the distributions of the classes in the dataset are roughly equal [7]. When faced with significantly imbalanced data, these algorithms will be heavily biased towards the majority class and give deceptive accuracy.

In real-world classification problems, there are many applications that suffer from the class-imbalance problem, for instance, fault diagnosis, anomaly detection, medical diagnosis and etc. In these problems, it is much more important to correctly identify the minority samples. The price of misclassifying the minority samples would be a huge loss of money in fault diagnosis, an unqualified product in anomaly detection, and a person's life in medical diagnosis. An application example may also be illustrated in the field of computational design optimization [8], 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. Hence, it is significant to improve the class-imbalance classification.

This section of the report provides a summary of the research and software developed in ECOLE to efficiently improve the class-imbalance classification. Section 2.1 focuses on studying the efficiency of different resampling techniques as well as the relationship between data complexity measures and different resampling techniques. Section 2.2 explains the studies on improving imbalanced classification by adding additional attributes. The structure of the software developed for both studies is shared in the Appendix.

### 2.1. On the Performance of Oversampling Techniques for Class Imbalance Problems

Although over 90 sampling approaches have been developed in the imbalance learning domain, most of the empirical study and application work are still based on the "classical" resampling techniques. In this part of research, several experiments on 19 benchmark datasets are set up to study the efficiency of six powerful oversampling approaches, including both "classical" and new ones. Oversampling techniques that heavily rely on local information of the minority class samples are considered as "classical" resampling techniques (The synthetic minority oversampling technique (SMOTE), The adaptive synthetic (ADASYN), the majority weighted minority oversampling techniques (MWMOTE) [9] [10] [11], while oversampling techniques which take global information of minority samples are considered as "new" resampling techniques (RACOG, wRACOG, RWO-sampling) [1] [2]. In addition, seven data complexity measures are considered for the initial purpose of investigating the relationship between data complexity measures and the choice of resampling techniques. Detailed information has been reported in



deliverables D1.2 and D3.1. We will focus on introducing the implementation of the experiment.

The experimental setup for the study was implemented with the open-source software R. The required libraries include "smotefamily", "imbalance", "ECol", "C50", "dplyr", "pROC", "MLmetrics", "measures" and "e1071". The library "smotefamily" and library "imbalance" are used to implement the resampling techniques, including SMOTE, ADASYN, MWMOTE, RACOG, wRACOG, and RWO-sampling. The library "ECol" is used to calculate the data complexity of the datasets, and the library "C50" is used to perform the classification algorithm "Decision Tree". The other libraries are used to simplify the coding structure and produce the performance measurements.

*Table 1. List of Implemented Oversampling Techniques in R environment.*

| Resampling   | R library   | Usage  |
|--------------|-------------|--|
| SMOTE        | smotefamily | SMOTE(X, target, K = 5, dup_size = 0)  |
| ADASYN       | smotefamily | ADAS(X, target, K=5)   |
| MWMOTE       | imbalance   | mwmote(dataset, numInstances, kNoisy = 5, kMajority = 3, kMinority, threshold = 5, cmax = 2, cclustering = 3, classAttr = "Class") |
| RACOG        | imbalance   | racog(dataset, numInstances, burnin = 100, lag = 20, classAttr = "Class")  |
| wRACOG       | imbalance   | wracog(train, validation, wrapper, slideWin = 10, threshold = 0.02, classAttr = "Class")   |
| RWO-sampling | imbalance   | rwo(dataset, numInstances, classAttr = "Class")  |

## 2.2. Improving Imbalanced Classification by Introducing Additional Attributes

Although the anomaly detection problem can be considered as an extreme case of class imbalance problem, very few studies consider improving class imbalance classification with anomaly detection ideas. Most data-level approaches in the imbalanced learning domain aim to introduce more information to the original dataset by generating synthetic samples. However, in this part of the research, we gain additional information in another way by introducing additional attributes. We propose to introduce the outlier score and four types of samples (safe, borderline, rare, outlier) as additional attributes in order to gain more information on the data characteristics and improve the classification performance.

Four resampling techniques are implemented in our experiment, including two oversampling techniques (SMOTE, ADASYN) and two undersampling techniques – One-Sided Selection (OSS), Neighbourhood Cleaning Rule (NCL) [12] [13]. The detailed information for these resampling techniques has been given in deliverables D1.2 and D3.1.

Table 2. List of Implemented Resampling Techniques in Python

| Resampling | Python packages | Usage   |
|------------|-----------------|---|
| SMOTE      | imblearn        | <code>class imblearn.over_sampling.SMOTE(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None)</code>   |
| ADASYN     | imblearn        | <code>class imblearn.over_sampling.ASADASYN(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None)</code>  |
| OSS        | imblearn        | <code>class imblearn.under_sampling.OneSidedSelection(*, sampling_strategy='auto', random_state=None, n_neighbors=None, n_seeds_S=1, n_jobs=None)</code>              |
| NCL        | imblearn        | <code>class imblearn.under_sampling.NeighbourhoodCleaningRule(*, sampling_strategy='auto', n_neighbors=3, kind_sel='all', threshold_cleaning=0.5, n_jobs=None)</code> |

The first introduced attributes "four types of samples" was first introduced by Napierala and Stefanowski [14]. Either majority or minority samples can be divided into different types: safe, borderline, rare examples, and outliers according to the local characteristics of the samples (reference). Given the number of neighbors  $k$  (odd), the label to a minority example can be assigned through the ratio of the number of its minority neighbors to the total number of neighbors ( $R_{\frac{min}{all}}$ ) according to Table 3. The label for a majority of all examples can be assigned in a similar way.

Table 3: Rules to assign the four types of minority examples.

| Type   | Safe (S)                                      | Borderline (B)  | Rare (R)                                   | Outlier (O)               |
|--|---|---|--|---------------------------|
| Rule   | $\frac{k+1}{2k} < R_{\frac{min}{all}} \leq 1$ | $\frac{k-1}{2k} \leq R_{\frac{min}{all}} \leq \frac{k+1}{2k}$ | $0 < R_{\frac{min}{all}} < \frac{k-1}{2k}$ | $R_{\frac{min}{all}} = 0$ |
| E.G. given the neighbourhood of a fixed size $k = 5$ |   |   |  |                           |
| Rule   | $\frac{3}{5} < R_{\frac{min}{all}} \leq 1$    | $\frac{2}{5} \leq R_{\frac{min}{all}} \leq \frac{3}{5}$       | $0 < R_{\frac{min}{all}} < \frac{2}{5}$    | $R_{\frac{min}{all}} = 0$ |

Local outlier factor (LOF), which indicates the degree of a sample being an outlier, was first introduced by Breunig et al. in 2000 [15]. The LOF of an object depends on its relative degree of isolation from its surrounding neighbors. Several definitions are needed to calculate the LOF and are summarized in the following Algorithm 1.

According to the definition of LOF, a value of approximately 1 indicates that the local density of data point  $X_i$  is similar to its neighbors. A value below 1 indicates that data point  $X_i$  locates in a relatively denser area and does not seem to be an anomaly, while a value significantly larger than 1 indicates that data point  $X_i$  is alienated from other points, which is most likely an outlier.

Algorithm 1: Local outlier factor (LOF) algorithm [15].

---

**Input** :  $\mathbf{X}$  - input data  $\mathbf{X} = (X_1, \dots, X_n)$   
 $n$  - the number of input examples  
 $k$  - the number of neighbours

**Output**: LOF score of every  $X_i$

- 1 initialization;
- 2 calculate the distance  $d(\cdot)$  between every two data points;
- 3 **for**  $i = 1$  **to**  $n$  **do**
- 4     calculate  $k$ -distance( $X_i$ ): the distance between  $X_i$  and its  $k$ th neighbour;
- 5     find out  $k$ -distance neighbourhood  $N_k(X_i)$ : the set of data points whose distance from  $X_i$  is not greater than  $k$ -distance( $X_i$ );
- 6     **for**  $j = 1$  **to**  $n$  **do**
- 7         calculate reachability distance:  

$$reach-dist_k(X_i, X_j) = \max\{k\text{-distance}(X_j), d(X_i, X_j)\};$$
- 8         calculate local reachability density:  

$$lrd_k(X_i) = 1/avg\text{-}reach\text{-}dist_k(X_i)$$

$$= 1/\left(\frac{\sum_{o \in N_k(X_i)} reach\text{-}dist_k(X_i, X_j)}{|N_k(X_i)|}\right);$$

intuitively, the local reachability density of  $X_i$  is the inverse of the average reachability distance based on the  $k$ -nearest neighbours of  $X_i$ ;
- 9         calculate LOF:  

$$LOF_k(X_i) = \frac{\sum_{o \in N_k(X_i)} lrd_k(X_j)}{|N_k(X_i)| \cdot lrd_k(X_i)}$$

$$= \frac{\sum_{o \in N_k(X_i)} \frac{lrd_k(X_j)}{lrd_k(X_i)}}{|N_k(X_i)|}$$

the LOF of  $X_i$  is the average local reachability density of  $X_i$ 's  $k$ -nearest neighbours divided by the local reachability density of  $X_i$ .
- 10     **end**
- 11 **end**

---

[Code example](#) to calculate LOF and visualization

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

print(__doc__)

np.random.seed(42)

# Generate train data
X_inliers = 0.3 * np.random.randn(100, 2)
X_inliers = np.r_[X_inliers + 2, X_inliers - 2]

# Generate some outliers
```

```

X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X_inliers, X_outliers]

n_outliers = len(X_outliers)
ground_truth = np.ones(len(X), dtype=int)
ground_truth[-n_outliers:] = -1

# fit the model for outlier detection (default)
clf = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
# use fit_predict to compute the predicted labels of the training samples
# (when LOF is used for outlier detection, the estimator has no predict,
# decision_function and score_samples methods).
y_pred = clf.fit_predict(X)
n_errors = (y_pred != ground_truth).sum()
X_scores = clf.negative_outlier_factor_

plt.title("Local Outlier Factor (LOF)")
plt.scatter(X[:, 0], X[:, 1], color='k', s=3., label='Data points')
# plot circles with radius proportional to the outlier scores
radius = (X_scores.max() - X_scores) / (X_scores.max() - X_scores.min())
plt.scatter(X[:, 0], X[:, 1], s=1000 * radius, edgecolors='r',
            facecolors='none', label='Outlier scores')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.xlabel("prediction errors: %d" % (n_errors))
legend = plt.legend(loc='upper left')
legend.legendHandles[0]._sizes = [10]
legend.legendHandles[1]._sizes = [20]
plt.show()

```

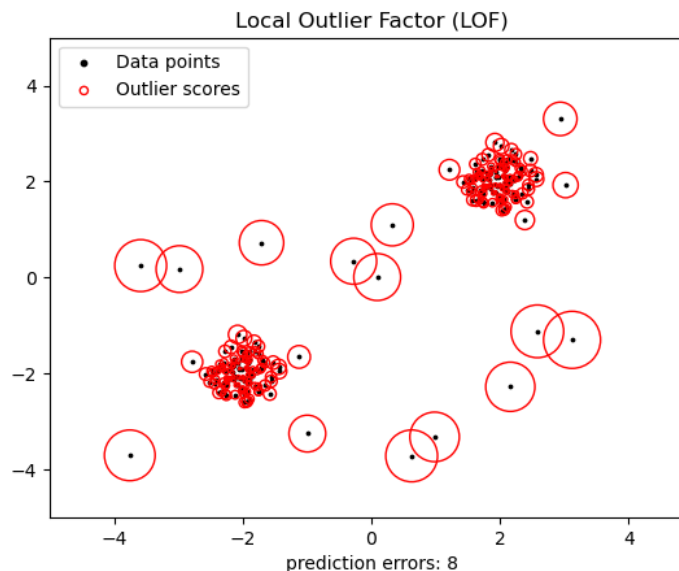


Figure 1. Demonstration of the Local Outlier Factor (LOF) method, in which the outlier score of a data point can be intuitively understood as the inverse of the average distance to its  $k$ -nearest neighbors.

### 3. Automated Machine Learning

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 2. These tasks are complicated and crucially required human efforts: to choose the best fit model and well-suited hyperparameter settings for an application problem. Automated machine learning (AutoML) [16, 17] optimization is an efficient approach to limit human efforts in applying machine learning to real-world problems, making machine learning easy to use and accessible to non-experts.

In ECOLE, many practical approaches have been developed in machine learning domains, e.g., imbalanced classification, robust optimization, and various real-world applications. In order to apply the research achievements of ECOLE in solving real-world problems efficiently, we contemplated an automated configuration approach for above-mentioned tasks, based on the existing works and achievements of ECOLE. This section is dedicated to the software developed for this aim.

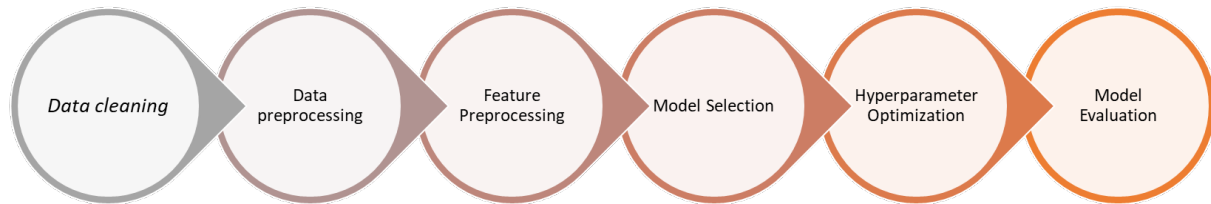


Figure 2. A typical machine learning workflow

#### 3.1. Automatically Optimize the Class Imbalance Problems

The class imbalance is present in many real-world applications. For example, fault detection is a typical example of the imbalanced classification problem in the manufacturing industry since most products are correctly produced, and only a few products are faulty. Therefore, the highly imbalanced nature between the defect and non-defect modules cannot be neglected in software defect prediction.

In this work, we provide a summary of the software developed in ECOLE to efficiently solve the class imbalance problem automatically [18, 19]. Our software comprises three major parts:

1. A search space is defined to represent the feasible search domain. In ECOLE, we use the combination of resampling technique and classification algorithm to handle the class imbalance problem. In this setting, the resampling techniques were used to producing balanced data sets before classifying this dataset by a commonly used classification algorithm. The search space is described in the following:
  - *Classification algorithms*: Our software includes five classification algorithms, i.e., Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbours

(KNN), Decision Tree (DT), and Logistic Regression (LR). All are implemented in the python package scikit-learn, described in Table 5.

- The *resampling algorithms* used in our software can be arranged into three groups, namely "over resampling" (7 techniques [20, 21, 22, 23, 9, 10]), "under resampling" (11 techniques [24, 25, 26, 27, 28, 29, 30, 31, 12, 13]), and "combine resampling" (2 techniques [32, 33]), which are implemented in the python package imbalanced-learn<sup>1</sup> [34]. The detailed information on the resampling techniques used in our software and their hyperparameters are presented in Table 4.
2. A target program is used to assess the selected machine learning pipeline generated by the optimizer on the examined dataset, resulting in a performance evaluation metric (here is geometric mean). The whole computation process consists of the following two phases:
- Static process: For one dataset, categorical variables of the input are converted to integers by the widely-used Label encoder<sup>2</sup>, which is then standardized. Next, a stratified  $k$ -fold cross-validation using  $k=5$  is used.
  - Dynamic process: At each iteration, the optimizer generates a hyperparameter setting in the search space, 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. This process in the gray circle is based on the geometric mean.

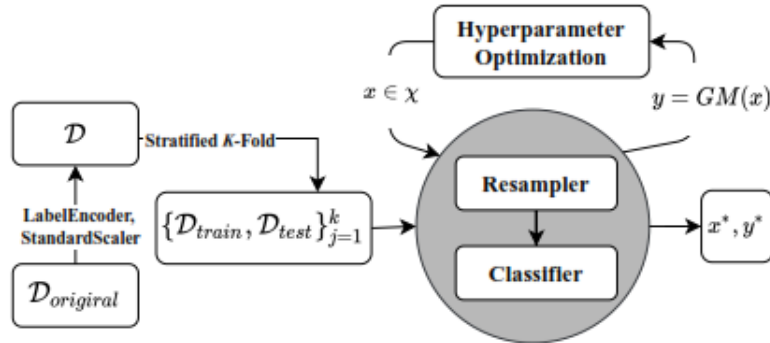


Figure 3. Flowchart of the ECOLE software for class imbalance problems.

3. An optimization algorithm is used to find the best pipeline and optimized hyperparameter settings in the feasible domain defined in the search space. In ECOLE, we applied a special type of Bayesian optimization approach, the Tree Parzen Estimators (or Tree-structured Parzen Estimators), which is implemented in the Python package HyperOpt<sup>3</sup>.

<sup>1</sup> <https://github.com/scikit-learn-contrib/imbalanced-learn> (version 0.7.0)

<sup>2</sup> Label encoder, Standard scaler and Stratified  $k$ -fold cross-validation are implemented in the python library scikit-learn (version 0.23.2)

<sup>3</sup> <https://github.com/hyperopt/hyperopt>



Table 4. Hyperparameters of resampling techniques

| Algorithm                       | Hyperparameter            | Range                      |
|---------------------------------|---------------------------|----------------------------|
| <b>Over-Resampling</b>          |                           |                            |
| SMOTE                           | k_neighbors               | [1, 10]                    |
|                                 | sampling_strategy         | default                    |
| BorderlineSMOTE                 | k_neighbors               | [1, 10]                    |
|                                 | m_neighbors               | [1, 10]                    |
|                                 | sampling_strategy         | default                    |
|                                 | kind                      | [borderline1, borderline2] |
| SMOTENC                         | sampling_strategy         | default                    |
|                                 | k_neighbors               | [1, 10]                    |
| SVM SMOTE                       | sampling_strategy         | default                    |
|                                 | k_neighbors               | [1, 10]                    |
|                                 | m_neighbors               | [1, 10]                    |
|                                 | out_step                  | [0.0, 1.0]                 |
| KMeansSMOTE                     | sampling_strategy         | default                    |
|                                 | k_neighbors               | [1, 10]                    |
|                                 | cluster_balance_threshold | [1e-2, 1]                  |
| ADASYN                          | sampling_strategy         | default                    |
|                                 | n_neighbors               | [1, 10]                    |
| <b>Combine-Resampling</b>       |                           |                            |
| SMOTENN                         | sampling_strategy         | default                    |
| SMOTETomek                      | sampling_strategy         | default                    |
| <b>Under-Resampling</b>         |                           |                            |
| CondensedNearestNeighbour       | n_neighbors               | [1, 50]                    |
|                                 | sampling_strategy         | default                    |
|                                 | n_seeds_S                 | [1, 50]                    |
| EditedNearestNeighbours         | n_neighbors               | [1, 20]                    |
|                                 | sampling_strategy         | default                    |
|                                 | return_indices            | [True, False]              |
|                                 | kind_sel                  | [all, mode]                |
| RepeatedEditedNearestNeighbours | sampling_strategy         | default                    |
|                                 | kind_sel                  | [all, mode]                |
|                                 | n_neighbors               | [1, 20]                    |
| AllKNN                          | n_neighbors               | [1, 20]                    |
|                                 | sampling_strategy         | default                    |
|                                 | kind_sel                  | [all, mode]                |
|                                 | allow_minority            | [True, False]              |

| Algorithm                 | Hyperparameter  | Range  |
|---------------------------|---|--|
| InstanceHardnessThreshold | estimator<br>sampling_strategy<br>cv                            | None, knn, decision-tree,<br>adaboost, gradient-boosting,<br>linear-svm<br><i>default</i><br>[2, 10] |
| OneSidedSelection         | sampling_strategy<br>n_neighbors<br>n_seeds S                   | <i>default</i><br>[1, 20]<br>[1, 20]   |
| RandomUnderSampler        | sampling_strategy<br>replacement                                | <i>default</i><br>[True, False]  |
| TomekLinks                | sampling_strategy   | <i>default</i>   |
| NearMiss                  | sampling_strategy<br>version<br>n_neighbors<br>n_neighbors_ver3 | <i>default</i><br>[1,3]<br>[1, 20]<br>[1, 20]  |
| NeighbourhoodCleaningRule | sampling_strategy<br>threshold_cleaning<br>n_neighbors          | <i>default</i><br>[0.0, 1.0]<br>[1, 20]  |
| ClusterCentroids          | sampling_strategy<br>estimator<br>voting                        | <i>default</i><br>[KMeans, MiniBatchKMeans]<br>[hard, soft]  |

Table 5. Hyperparameters of Classification algorithms

| Algorithm                     | Hyperparameter  | Range  |
|-------------------------------|---|--|
| Support Vector Machines (SVM) | Max_iter<br>Cache_size<br>probability<br>C<br>kernel<br>shrinking<br>gamma<br>gamma_value<br>coef0<br>degree<br>tol | 10000<br>700 (Megabyte)<br>[True, False]<br>[0.5 <sup>5</sup> ,100]<br>[linear, rbf, poly, sigmoid]<br>[true, false]<br>[auto, value, scale]<br>[3.1e-05,8]<br>[-1.0, 1.0]<br>[2, 5]<br>[1e-05, 1e-01] |
| Random Forest (RF)            | n_estimators<br>criterion<br>max_features<br>min_samples_split<br>min_samples_leaf                                  | [1,150]<br>[gini, entropy]<br>[1, sqrt, log2, None]<br>[2, 20]<br>[1, 20]  |

| Algorithm                 | Hyperparameter  | Range   |
|---------------------------|---|---|
|                           | bootstrap   | [True, False]   |
|                           | class_weight  | [balanced, balanced_subsample, None]  |
| K-Nearest Neighbors (KNN) | n_neighbors<br>weights<br>algorithm<br>p  | [1, 51]<br>[uniform, distance]<br>[auto, ball_tree, kd_tree, brute]<br>[0, 20]<br><ul style="list-style-type: none"> <li>p = 0 → metric = chebyshev</li> <li>p = 1 → metric = manhattan</li> <li>p = 2 → metric = euclidean</li> <li>p &gt; 2 → metric = minkowski</li> </ul> |
| Decision Tree (DT)        | criterion<br>max_depth<br>max_features<br>min_samples_split<br>min_samples_leaf | [gini, entropy]<br>[2, 20]<br>[1, sqrt, log2, None]<br>[2, 20]<br>[1, 20]   |
| Logistic Regression (LR)  | C<br>criterion<br>tol<br>l1_ratio<br>(penalty, solver)                          | [1, 150]<br>[0.5 <sup>5</sup> , 100]<br>[1e-05, 1e-01]<br>[1e-09, 1]<br>[(l1, liblinear), (l1, saga), (l2, lbfgs),<br>(l2, newton-cg), (l2, liblinear),<br>(l2, sag), (l2, saga), (elasticnet, saga),<br>(none, newton-cg), (none, lbfgs),<br>(none, sag), (none, saga)]      |



## 4. Model-Assisted Robust Optimization

While solving real-world optimization problems, a frequently-encountered obstacle is the presence of uncertainties and noise within the system, or model of the system, for which optima are sought [4]. Due to various reasons, various types of uncertainties and noise can emerge in optimization problems. Hence, for real-world scenarios, optimization methods are needed which can deal with these uncertainties, and solutions have to be found which are not only optimal in the theoretical sense, but that are also practical in real life. The practice of optimization that accounts for uncertainties and noise is often referred to as Robust Optimization (RO) [35] [3] [36].

In real-world engineering applications, e.g., automobile manufacturing, building construction, and steel production, finding a robust solution – a solution whose performance is not significantly affected by uncertainties – is crucial due to the potentially serious impact in case of a failure. Despite the significance, however, achieving robustness in modern engineering applications is challenging. This can be mainly attributed to the variety of fitness landscapes and high dimensionality, as well as many shapes and forms of uncertainty, which are often unknown in advance. In practice, optimization scenarios in these domains are often treated as black-box problems, which have to be efficiently solved in the face of uncertainties and noise. In ECOLE, RO problems are solved with the help of empirical (statistical) models, which are called Surrogate Models. These models replace the actual (expensive) function evaluations by their predictions, thereby helping to efficiently solve the problem [36].

This section of the report provides a summary of the research and software-development to tackle the expensive to evaluate black-box problem, subject to uncertainty and noise. The first subsection focuses on assessing the suitability of the surrogate models to solve RO problems [37]. This is followed by the details on exploring the dimensionality reduction techniques for efficiently constructing the low dimensional surrogate models [38]. Finally, latest results are reported on extending an infill criterion that underpins Bayesian Optimization (BO) to the robust scenario [39]. The structure of the code repositories for all three studies is included in the Appendix. The overall aim of this section is to provide the fundamental details on the implementation of Surrogate-Assisted Optimization (SAO) in ECOLE.

### 4.1. Investigation of Modelling Techniques for Robust Optimization

Surrogate Models were initially employed to find the nominal solution of expensive to evaluate black-box problems, without taking into account the unexpected drifts and changes in the optimization setup. However, this is unrealistic for many real-world scenarios. As such, a natural question arises on the applicability of the SAO to find solutions that are immune to uncertainty and noise. To answer this question, an empirical study was conducted in ECOLE [5]. The study took into account the impact of some of the most important factors, such as the dimensionality, the type and structure of the uncertainty, the noise level, and the problem landscape.

The experimental setup for this study was implemented with the help of open-source Python frameworks such as "SciPy", "NumPy", "scikit-learn" and "pyDOE". In particular, "scikit-learn" was utilized to implement the modeling techniques [40] [41] such as Kriging/Gaussian Process

Regression (GPR), Regression Trees (RT), Support Vector Machines (SVMs), Quadratic Polynomials (QP), K-Nearest Neighbors (KNN), and Radial Basis Functions (RBFs). The other modules, in particular, "NumPy" was utilized to implement the test problems, such as Ackley, Sphere, and Rastrigin functions. Finally, "pyDOE" was utilized to implement Design of Experiment (DoE) methodologies such as Latin Hypercube Sampling (LHS). The experimental setup for this study is presented in Figure 4 for further clarification.

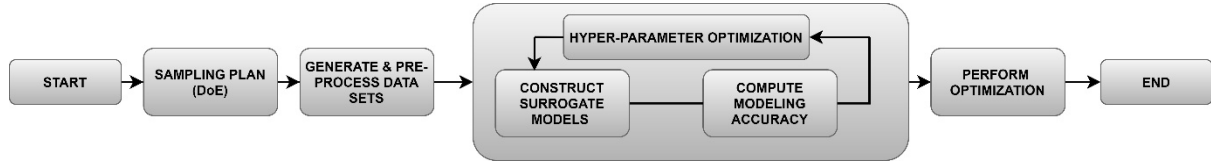


Figure 4. The experimental setup to investigate the suitability of some of the most important modeling techniques for Robust Optimization [37].

Table 6 presents the major open-source Python frameworks utilized in this study. Further details on the study and the experimental setup are provided in ECOLE Deliverable 2.3, titled "Robustness & Uncertainty Modelling in experience-based optimization", whereas the details on the structure of the code are provided in the Appendix.

Table 6. List of major open-source software modules utilized in the experimental setup [37].

| Open-source Software Module                   | Major Purpose  |
|---|--|
| <a href="#">NumPy</a> (Version 1.16.0)        | To implement the test problems.  |
| <a href="#">PyDOE</a> (Version 0.3.8)         | To implement LHS.  |
| <a href="#">Scipy</a> (Version 1.2.1)         | To implement the (global) optimization algorithms, e.g., Sequential Quadratic Programming. |
| <a href="#">Scikit-learn</a> (Version 0.20.0) | To implement the modeling techniques, e.g., Kriging.                                       |

#### 4.1. Investigation of Dimensionality Reduction Techniques for Efficient SAO

Constructing surrogate models of high-dimensional optimization problems is challenging due to the computational complexity involved. The computational complexity can be mainly attributed to two main factors. Firstly, more training data is required to achieve a comparable level of modeling accuracy as the dimensionality increases. Secondly, training-time-complexity often increases rapidly with respect to both, the dimensionality and the number of training data points. Consequently, constructing the surrogate model becomes costlier. Various methodologies have been proposed to deal with the issue of high dimensionality in SAO, including divide-and-conquer, variable screening, and mapping the data space to a lower-dimensional space using dimensionality reduction techniques (DRTs).

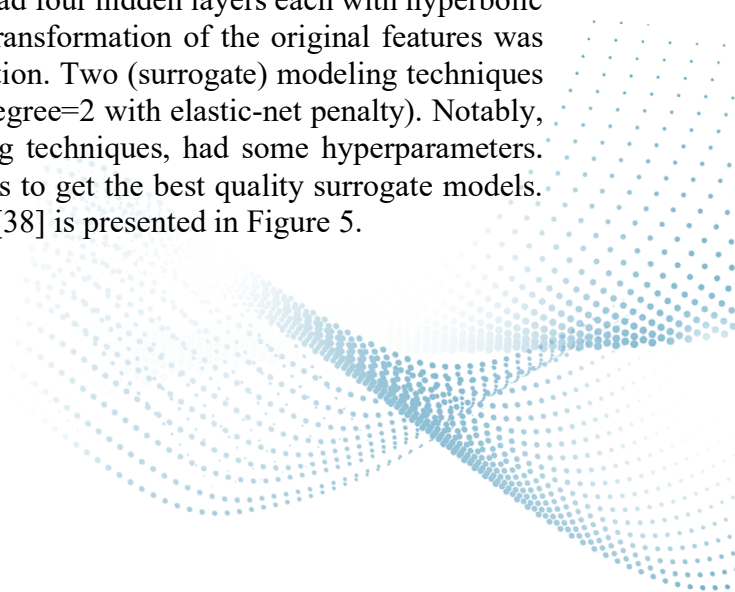
One of the most common DRTs is Principal Component Analysis (PCA) [40] [41]. PCA can be defined as the orthogonal projection of the data onto a lower-dimensional linear space, spanned by the principal components, such that the variance of the projected data is maximized. Various

generalized extensions of PCA have been established in the literature, such as Kernel PCA, Probabilistic PCA, and Bayesian PCA [40]. On the other hand, Autoencoders (AEs) [41] have been introduced as feed-forward neural networks (FFNNs), which are trained to copy their input to their output, so as to learn the useful low dimensional encoding of the data. Like PCA, AEs have also been extended over the years by generalized frameworks such as Sparse Autoencoders, Denoising Autoencoders, Contractive Autoencoders [41] and Variational Autoencoders (VAEs) [42] [43]. Besides PCA and AEs, other important DRTs include Isomap [40], Locally-Linear Embedding [44], Laplacian Eigenmaps [45], Curvilinear component analysis [46], and t-distributed stochastic neighbor embedding [41].

As several DRTs have been employed in the literature, a natural question arises on the suitability of some of the most important DRTs for efficiently constructing the low dimensional surrogate models. To this end, an empirical investigation was conducted in ECOLE [38]. In this study, four of the most important DRTs mentioned above, namely PCA, Kernel PCA, AEs and VAEs respectively, were evaluated and compared with each other. PCA and AEs were chosen due to their historical significance, since both have been employed regularly for dimensionality reduction, lossy data compression, feature learning and data visualization in machine learning [40] [41]. Kernel PCA was incorporated due to the generalized non-linear extension of the classical PCA algorithm. Similarly, VAEs were considered due to the presence of the non-linear stochastic encodings of the data space, which can be utilized for constructing the surrogate models efficiently. This study focused on providing a novel perspective on the applicability of these DRTs in SAO. This was accomplished by performing an experimental study of the corresponding low dimensional surrogate models (LDSMs) on a diverse range of test cases.

The experimental setup for this study was based on ten unconstrained, noiseless, single-objective optimization problems from the continuous benchmark function test-bed known as "Black-Box-Optimization-Benchmarking" (BBOB) [47]. Each of these test functions was evaluated on three different values of dimensionality – 50, 100, and 200, respectively. Note that all test functions mentioned were subject to minimization. Four DRTs were employed – PCA, KPCA, AEs, and VAEs. For each of these techniques, specifying the size of the latent dimensionality was crucial since it could affect the quality of the corresponding LDSM. Therefore, for each distinct value of dimensionality, three different settings of the latent dimensionality were chosen.

In AEs and VAEs, both the encoder and the decoder had four hidden layers each with hyperbolic tangent non-linearity. For PCA and KPCA, a linear transformation of the original features was computed before performing the dimensionality reduction. Two (surrogate) modeling techniques were chosen – Kriging and Polynomial Regression (degree=2 with elastic-net penalty). Notably, both sets of techniques – the DRTs and the modeling techniques, had some hyperparameters. Therefore, it was crucial to tune these hyperparameters to get the best quality surrogate models. The flowchart of the experimental setup for this study [38] is presented in Figure 5.





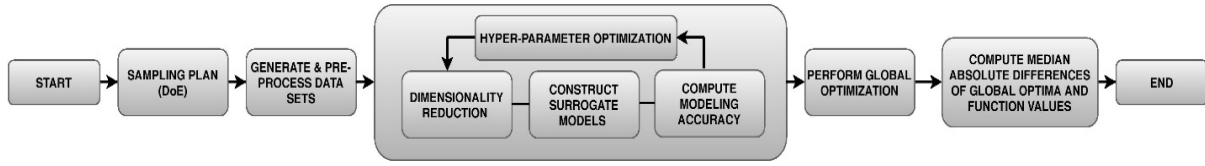


Figure 5. Flowchart of the experimental setup. Each step of the process is shown in grey rectangles. The central rectangles indicate the hyperparameter optimization loop based on the modeling accuracy of the surrogates.

The experimental setup for this study was also carried out with the help of open-source Python frameworks such as "SciPy", "NumPy", "scikit-learn" and "pyDOE". In addition to that, the famous Python framework "PyTorch" was employed for the realization of AEs and VAEs. In the setup, "scikit-learn" was utilized to implement the modeling techniques – Kriging and QP with elastic-net penalty – as well as two DRTs, namely the PCA and Kernel PCA. To implement the test problems, the code provided by BBOB [47] was utilized. Lastly, to implement the hyperparameter optimization (HPO), another open-source Python framework "Hyperopt" was used. Table 7 reports the major open-source Python frameworks utilized in this study, whereas further details on the study and the experimental setup are provided in ECOLE Deliverable 2.1, titled "Experience-based high dimensional and big data assisted optimization". Furthermore, the details on the structure of the code are provided in the Appendix.

Table 7. List of major open-source software modules utilized in the experimental setup in [38].

| Open-source Software Module                   | Major Purpose  |
|---|--|
| <a href="#">PyTorch</a> (Version 1.5.0)       | To implement AEs and VAEs.   |
| <a href="#">PyDOE</a> (Version 0.3.8)         | To implement LHS.  |
| <a href="#">Scipy</a> (Version 1.5.0)         | To implement the global optimization algorithms, e.g., L-BFGS.                       |
| <a href="#">Scikit-learn</a> (Version 0.23.0) | To implement the modelling techniques, e.g., Kriging, as well as PCA and Kernel PCA. |
| <a href="#">BBOB</a>                          | To implement the test problems.  |
| <a href="#">Hyperopt</a> (Version 0.2.4)      | To perform HPO.  |

#### 4.2. Extending the Moment-Generating Function of Improvement (MGFI) for Robust Bayesian Optimization (RBO)

The famous Bayesian Optimization (BO) algorithm [16] has been adapted to efficiently solve RO problems, and is referred to as Robust Bayesian Optimization (RBO) in this report. The performance of the RBO algorithm is greatly determined by the acquisition function (AF), which balances the trade-off of exploration and exploitation. Furthermore, following the intuition of utilizing the higher moment of the improvement in ECOLE, the Moment-Generating Function of the Improvement (MGFI) [17] was extended to efficiently solve problems with uncertainty [7]. The flowchart of the RBO algorithm is presented in Figure 6.

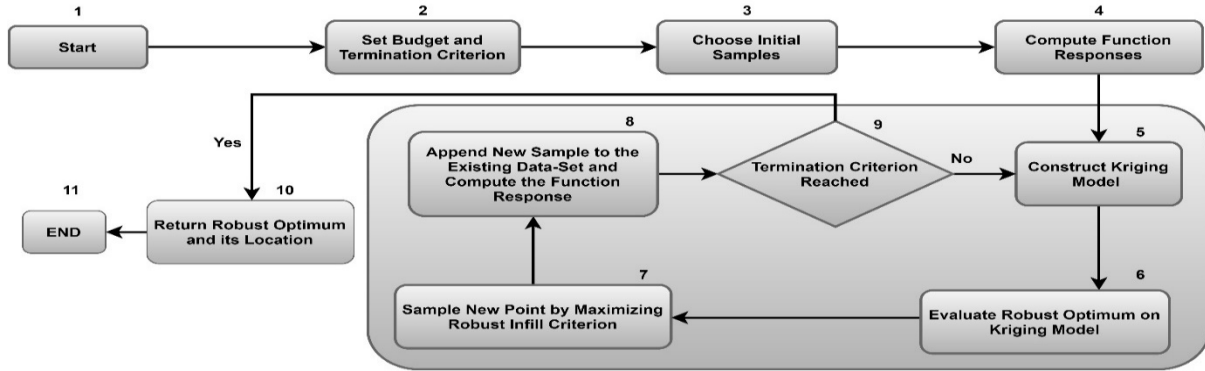


Figure 6. The flowchart of the RBO algorithm. Steps 6 and 7 are adapted to extend the BO algorithm to the robust scenario.

To gauge the ability of the extended MGFI, we evaluated and compared it against the baseline – the robust expected improvement criterion (REIC) proposed in [48]. The comparison [39] involved four test problems in total, all taken from the existing literature. Three of these problems, the so-called "Three", the "Eight" and the "Ten-Dimensional" problems were taken from [48], in addition to a two-dimensional "Branin" function. The comparison also involved three noise levels based on 5, 10 and 20 % deviation in the nominal values of the decision variables, giving rise to a total of 12 test scenarios for comparison. In addition to the baseline comparison, the configuration of the initial temperature was altered to comprehend the role it played in the performance of the extended AF. To this end, only the "Branin" and the "Three-Dimensional" problems were chosen alongside three different configurations for the initial temperature.

Graphs pertaining to the baseline comparison are presented in Figure 7. Note that each column of plots in this figure corresponds to a specific noise level, whereas the rows distinguish between different problem instances. The noise level determines the maximum deviation from the nominal values of the decision variables. Each subplot in this figure presents two curves based on the two AFs. Each of these curves indicates the mean absolute difference (MAD) to the globally robust optimal function value (GROFV) based on 25 independent runs. From Figure 7, we observe that in 6/12 cases, the extended MGFI (denoted as RMGFI in the figure) yielded a better optimal function value, whereas the implementation with the REIC performed superior in 5/12 cases. In particular, the REIC exceeded the RMGFI for all three test scenarios related to the "Branin" function, whereas the RMGFI performed better on the "Three-Dimensional" problem. Additionally, it can be observed that both acquisition functions performed competitively on the third noise level. Note that in the test scenarios of the "Eight" and the "Ten-Dimensional" problems concerning the first noise level, the RMGFI undershoted the GROFV.

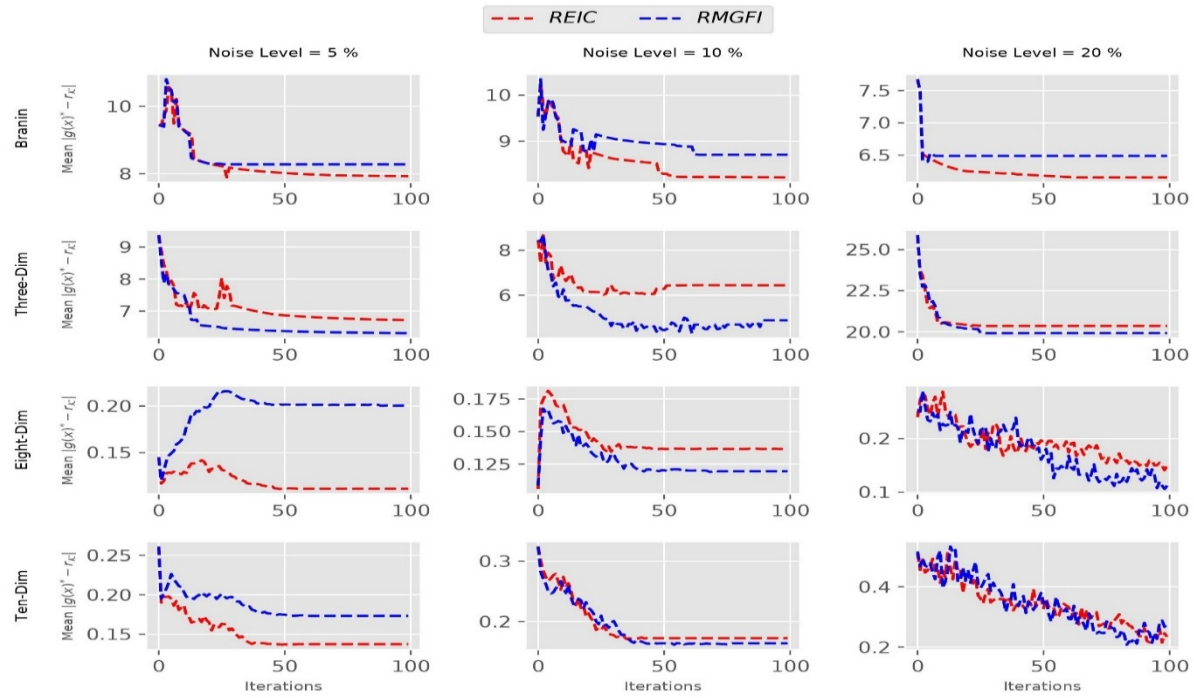


Figure 7. Mean Absolute Difference to the globally robust optimal function value based on the extended RMGFI and the baseline REIC.

Next, the results based on the variation of the initial temperature are presented in Table 8. In this table, the first column reads the optimization problem at hand, whereas the next three columns describe the noise level, initial temperature, and the MAD to the GROFV accompanied with the standard error (SE). An important observation from Table 8 suggests that for the "Branin" function, the best performance was achieved for the low initial temperature. On the other hand, the higher settings for the initial temperature gave rise to a better performance on the "Three-Dimensional" problem.

Table 8. Mean Absolute Difference (MAD) to the globally robust optimal function value based on three different settings of the initial temperature.

| Problem | Noise Level | Initial Temperature | MAD             |
|---------|-------------|---------------------|-----------------|
| Branin  | 5 %         | 1                   | $8.08 \pm 1.44$ |
|         |             | 3                   | $8.87 \pm 1.59$ |
|         |             | 5                   | $8.31 \pm 1.48$ |
|         | 10 %        | 1                   | $7.78 \pm 2.23$ |
|         |             | 3                   | $8.46 \pm 2.22$ |
|         |             | 5                   | $7.81 \pm 2.18$ |
|         | 20 %        | 1                   | $5.88 \pm 0.85$ |
|         |             | 3                   | $6.20 \pm 1.01$ |
|         |             | 5                   | $6.09 \pm 1.01$ |
|         | 5 %         | 1                   | $6.58 \pm 1.09$ |
|         |             | 3                   | $5.66 \pm 0.90$ |
|         |             | 5                   | $5.83 \pm 0.95$ |
|         |             | 1                   | $5.61 \pm 0.79$ |

|                   |      |   |                  |
|-------------------|------|---|------------------|
| Three-Dimensional | 10 % | 3 | $5.27 \pm 0.55$  |
|                   |      | 5 | $5.75 \pm 0.75$  |
|                   | 20 % | 1 | $21.01 \pm 2.02$ |
|                   |      | 3 | $20.15 \pm 1.81$ |
|                   |      | 5 | $19.63 \pm 1.91$ |
|                   |      |   |                  |

The experimental setup for the study was carried out with the help of open-source Python frameworks such as "SciPy", "NumPy", "SMT", "scikit-learn" and "pyDOE2". In particular, "scikit-learn" was utilized to realize the Kriging model. "NumPy" and "SMT" were utilized to implement the test problems. "SciPy" was utilized to implement the global optimization algorithm, i.e., L-BFGS. Finally, "pyDOE2" was utilized to implement the LHS for DoE. Table 9 reports the major open-source Python frameworks utilized in this study, whereas the details on the structure of the software are provided in the Appendix.

*Table 9. List of major open-source software modules utilized in the experimental setup in [39].*

| Open-source Software Module                   | Major Purpose  |
|---|--|
| <a href="#">NumPy</a> (Version 1.20.0)        | To implement the utility functions and test problems.          |
| <a href="#">pyDOE2</a> (Version 1.3.0)        | To implement LHS.  |
| <a href="#">Scipy</a> (Version 1.6.1)         | To implement the global optimization algorithms, e.g., L-BFGS. |
| <a href="#">Scikit-learn</a> (Version 0.24.0) | To implement Kriging.  |
| <a href="#">SMT</a> (Version 1.0.0)           | To implement the test problems.                                |



## 5. Knowledge Transfer in Dynamic Multi-Objective Optimization

Transfer learning [49] is a machine learning method that is able to transfer the knowledge from a source task to a target task. This inherent characteristic of transfer learning makes it intuitive to apply to explore useful experiences that have been obtained in one task (optimization problem) to help solve another related task. Consequently, computational efficiency can be gained while tackling expensive-to-evaluate DMOO problems. The key challenge in DMOO is to constantly trace a changing Pareto optimal front (POF) and/or Pareto optimal set (POS) before the next environmental change occurs [50]. Aiming at this goal, researchers have proposed a method which predicts [50] [51] the good solutions in the next environment after learning the regularity of the environmental changes. In most prediction-based approaches, it is implicitly assumed that the evolution of the solutions used to train and test the prediction model obeys a fixed independent and identical probability distribution. However, this is not always true under dynamic environments in optimization, since the environmental changes may result in different evolution patterns over time. Consequently, the prediction model based on the incorrect assumption may cause inaccurate predictions of optimal solutions. Transfer learning, which does not make this assumption, is a good candidate for solving DMOO problems if it can learn and exploit the relationship among different problems.

So far, there has only been one attempt to introduce transfer learning to solve DMOO problems with evolutionary algorithms (EAs), resulting in a technique referred to as "Transfer learning based dynamic multi-objective optimization algorithms" (Tr-DMOEAs) [52]. Even though experimental studies [52] have shown the superiority of Tr-DMOEAs over the state-of-the-art in addressing DMOO problems, the results also showed that Tr-DMOEAs does not always work well and is time-consuming. It is therefore important to understand why and when transfer learning does not work well. Only after understanding that, we can make some improvements regarding transfer learning in DMOO. In addition, it is also important to understand what is the most time-consuming part in Tr-DMOEAs, after which we are able to take specific actions to increase the computational efficiency of Tr-DMOEAs.

This section of the report provides a summary of the research and software-development in ECOLE to improve the effectiveness and the efficiency of knowledge transfer in DMOO. Section 5.1 aims to answer the research question: when and how to transfer knowledge in DMOO, such that the Tr-DMOEAs algorithm is able to find solutions with better quality than the baseline methods. Section 5.2 illustrates a computational study on the cost and performance of knowledge transfer in DMOO, so as to verify whether two alternatives of the existing optimization method for the 'inner' problem in transfer learning are able to improve the efficiency of Tr-DMOEAs.

### 5.1. When and How to Transfer Knowledge in Dynamic Multi-objective Optimization?

Considering the general process of transfer learning, there are three main components as following:

- What to transfer?
- When to transfer?



- How to transfer?

In terms of the first question, good solutions found for the previous environment are typically aimed to be transferred to the next environment in DMOO. However, there are no straightforward answers to the other two questions. Generally, whenever a new environmental change happens, there is a population that has already been optimized to the previous environment. To check whether transfer learning works, we compare the quality of the following solutions on the new environment:

- Transferred solutions
- Solutions copied from the previously optimized population.

For transfer learning to be considered as successful, the new solution should be of at least similar (and ideally better) quality as opposed to the solution on the new environment.

To gauge the effectiveness of transfer learning, an experimental study is conducted in ECOLE to compare Tr-RMMEDA and RMMEDA where RMMEDA [53] refers to the so-called "Regularity model-based multi-objective estimation of distribution algorithm". In DMOO, the key point is to find the optimal solutions as soon as possible before the next change occurs. In this case, the performance of the Tr-DMOEAs depends on the quality of the generated solutions after each change. Therefore, if the transferred solutions are better than the copied solutions from the previous environment, we consider that transfer learning works well. On the contrary, if the transferred solutions are worse than the copied solutions, transfer learning fails. Experimental results have shown that the transferred solutions are all worse than those from the previous environment on problems with fixed POS and on problems with small changes.

The main idea of Tr-DMOEAs is to find a mapping function to map randomly generated solutions in the objective space of two problems at two consecutive environments by minimizing the distance between those two solution sets in the latent space. However, in the existing works, a Gaussian kernel function is used in the mapping function, which has been mathematically proved to be not ideal. Therefore, transfer learning fails in DMOO due to the Gaussian kernel. We propose an improved version of Tr-DMOEAs, which applies an alternative kernel function that does not have the problem of Gaussian kernel function. Specifically, after each change, transferred solutions and copied solution from the previous environment are firstly combined together. After that, nondominated sorting and crowding distance in Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [54] are used to rank the combined solutions on the new environment. After reviewing present common kernel functions [55] [56] [57], we find that the linear kernel functions overcome the problem presented by the Gaussian kernel function.

The experimental setup for this study was implemented based on the original Tr-DMOEAs under the environment of MATLAB 2018b. The used benchmark functions are taken from [58]. For the parameters of these problems, there are 20 changes. In order to study the effectiveness of Tr-DMOEAs in different dynamics, there are three dynamics with different severity of change (i.e.,  $\tau = 10, 1$  and  $20$ ). They represent the environmental changes are medium, large and small, respectively. Within each change, the population is forced to run 50 generations (i.e.,  $\tau = 50$ ), which enables the population to converge. Inverted Generational Distance (IGD) is used to compare the performance of all solutions sets.



## 5.2. Improving the Efficiency of Knowledge Transfer in Dynamic Multi-objective Optimization

Transfer Component Analysis (TCA) is the main transfer learning method used in Tr-DMOEA. The time complexity of TCA and primal dual interior point has been analyzed in [52]. The major time cost of TCA is spent on the eigenvalue decomposition. This is the case when nonzero eigenvectors are to be extracted. In order to empirically verify the cost of TCA and interior point method, an initial experiment is conducted regarding the computation time of these two parts in a single run. An experimental design is taken to record the computation time of each component in Tr-DMOEA. The findings suggest that the interior point method consumes more computation time. It has been shown that the existing Tr-DMOEA is still extremely time-consuming. It is unclear if it is worthy to consume such long time to use transfer learning in DMOO. In order to figure out the answer of this question, another experimental setup is designed. The main idea behind this setup is to use the computation time of transfer learning to optimize randomly generated solutions. Whenever there is a change, transfer learning is used to get the initial population, while another initial population is randomly generated in the search space. The costs of transfer learning and random generation are recorded, and termed as  $C_{tr}$  and  $C_{ran}$  respectively. The cost is the running time determined based on the stopwatch timer in MATLAB, where the MATLAB command 'tic' and 'toc' starts and ends the timer respectively. Then, the cost  $C_{tr} - C_{ran}$  is used to optimize the randomly generated population. During the optimization, both the transferred population and random one will iterate for the same generations to get two optimized solutions. Experimental results have shown that, given the same cost budget, the quality of random solution is better than that of transferred solutions no matter whether at the first generation after environmental changes or at the last generation after optimization.

Given that the computation time for solving the inner optimization problem in Tr-DMOEA is very large, it is unclear whether the efficiency of solving the inner problem could be enhanced. To explore this, other two popular optimization methods are used here, which are the active set [59] and sequential quadratic programming (SQP) [59] methods. We have further conducted a set of experiments to validate the efficiency and effectiveness of these three Tr-DMOEA variants. The computation time of three different optimization methods in these three Tr-DMOEA variants is only recorded when solving the inner problem, for all test problems with all parameter setting. The findings from this study suggest that for all test problems with picked parameter settings, the interior point method consumes the most time among the three compared methods to solve the inner optimization problem, while the active set method is the most efficient optimization algorithm. This shows that another two inner optimization methods can greatly improve the efficiency of transfer learning in DMOO. In order to verify whether the improved efficiency of Tr-DMOEA affects the solution quality, these three Tr-DMOEA variants are compared on picked benchmark problems. The findings show that Tr-DMOEA with SQP achieves the best optimized solution quality after optimization.

## 6. Summary and Outlook

This deliverable report focuses on the implementation details and software-development regarding the work package 2.4 in ECOLE, namely, “Integrated software environment (Self-Tuning optimization) and manual”. All in all, this deliverable report concentrates on four main issues as following:

- How to improve the classification accuracy of the minority class in imbalanced learning without significantly compromising on the accuracy of the majority class?
- How to automatically perform algorithm configuration and hyperparameters optimization for efficiently applying the common machine learning and optimization algorithms?
- How to efficiently solve the expensive to evaluate black-box problems in the presence of uncertainty and noise?
- How to transfer knowledge for efficiently solving DMOO problems?

To answer the first question, the efficacy of some of the most important oversampling techniques were analyzed in [8]. It was concluded that the oversampling techniques, which considered the minority class distribution helped improving the classification accuracy. Furthermore, it was found that the so-called "F1v" value – a measure for evaluating the overlap between the classes – had a strong negative correlation with the potential area-under-the-curve value in most cases. Lastly, the proposed approach to admit additional attributes helped improving the classification accuracy in imbalanced learning in most cases [60].

For the issue of algorithm configuration and hyperparameters optimization, an approach was proposed in ECOLE [18] to automate the machine learning pipeline (with imbalanced classification problem as a particular use case). The empirical study to compare the results of the proposed approach with the baseline methods indicated the promising nature of the technique as it significantly improved the performance of the classification algorithms over the examined data sets.

For efficiently solving the noisy black-box problems in ECOLE, an empirical study was designed to assess the suitability of the modeling techniques to find the robust solution [37]. The findings of this study indicated the usefulness of Kriging and Response-Surface Models (Polynomials). Furthermore, another study indicated the usefulness of PCA and AEs for efficiently constructing the low dimensional surrogate models [38]. The latest research on this topic focused on the BO algorithm by extending the MGFI to the robust scenario [39]. The potential future research line in this direction aims at providing a novel perspective on the so-called "Computational Cost of Robustness" – the need for additional computational resources to find the robust instead of a nominal solution.

Transferring knowledge to efficiently solve DMOO problems is complex, as our findings demonstrate. The key problem lies in the Gaussian kernel, and our studies recommend the use of linear kernels in lieu of the Gaussian kernel. Our results also demonstrate the superiority of active-set method for solving the inner problem in knowledge transfer. Lastly, an empirical investigation on the running time suggests the expensive nature of interior method.

## Appendix

### 1. User Manual for Section 2

#### Code Workflow for Section 2.1

Input: training samples' features, training samples' class

Output: predicted class label

Software and package: R, R package *smotefamily*, *imbalance* (open source)

Code: available on <https://doi.org/10.5281/zenodo.3855094>

#### Inline configuration :

```
python imb_exp.r --GPU<gpu_id>
```

#### Configuration in the script :

Data set directory : <path>

Data set training split: stratified folds

Classifiers: C5.0 or SVM

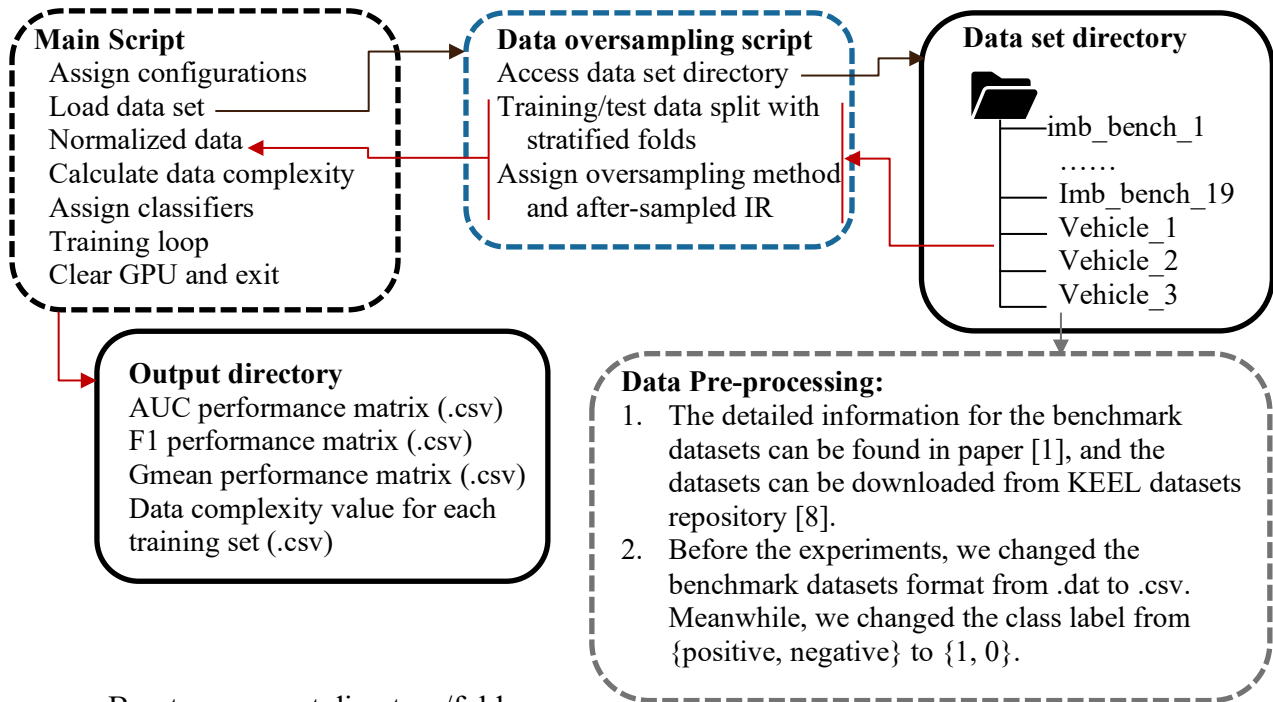
Output directory: <path>

Random seed: <set.seed>

Oversampling methods: <function>

After-sampled IR (imbalance ratio): x in (0.8, 1]

#### Script workflow



———— Box to represent directory/folders

----- Box with this line is the main script

----- Box with this line is the secondary script that the main script is calling

----- Box with this line is a note for the datasets

————> Function in the main script which is calling the secondary script

————> Values that the secondary script return to the main script

————> Notice for the directory/folders

## Code Workflow for Section 2.2

Input: training samples' features, training samples' class

Output: predicted class label

Software and package: Python, R package *imblearn* (open source)

Code: available on <https://doi.org/10.5281/zenodo.5503895>

### Inline configuration :

```
python imb_exp.py --GPU<gpu_id>
```

### Configuration in the script :

Data set directory : <path>

Data set training split: stratified folds

Classifiers: Decision Tree or SVM

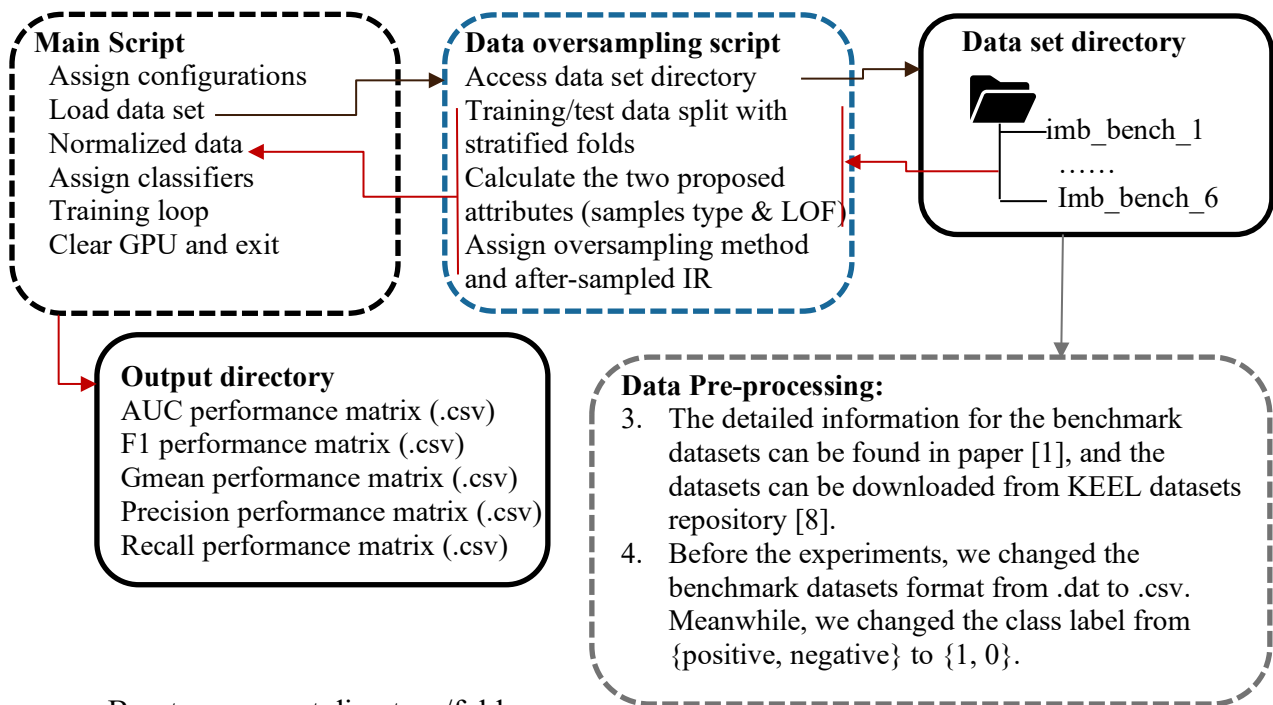
Output directory: <path>

Random seed: <set.seed>

Oversampling methods: <function>

After-sampled IR (imbalance ratio):  $x$  in  $(0.8, 1]$

### Script workflow



———— Box to represent directory/folders

----- Box with this line is the main script

----- Box with this line is the secondary script that the main script is calling

----- Box with this line is a note for the datasets

————> Function in the main script which is calling the secondary script

————> Values that the secondary script return to the main script

————> Notice for the directory/folders

## 2. User Manual for Section 3

Our software is available at <https://doi.org/10.5281/zenodo.5163207>. This manual shows how to use several aspects of our software. It either references to subsection "Configuration and example" where possible or explains certain configurations.

Our software is partly end-to-end software. The whole structure of this software module is provided in Figure 8. This software includes three folders: Code, HPOResults, and Data; whereas the main part of our software is zipped in one script under the *Code* folder, namely 'CASHOptimize.py'. The resampling and classification algorithms and their hyperparameters (as mentioned in Section 3.1) have been defined in the code. However, the predefined search space uses the standard syntax defined by Hyperopt, which can be easily modified. For further information about how to change our predefined configuration, please check the [hyperopt's documentation](#).

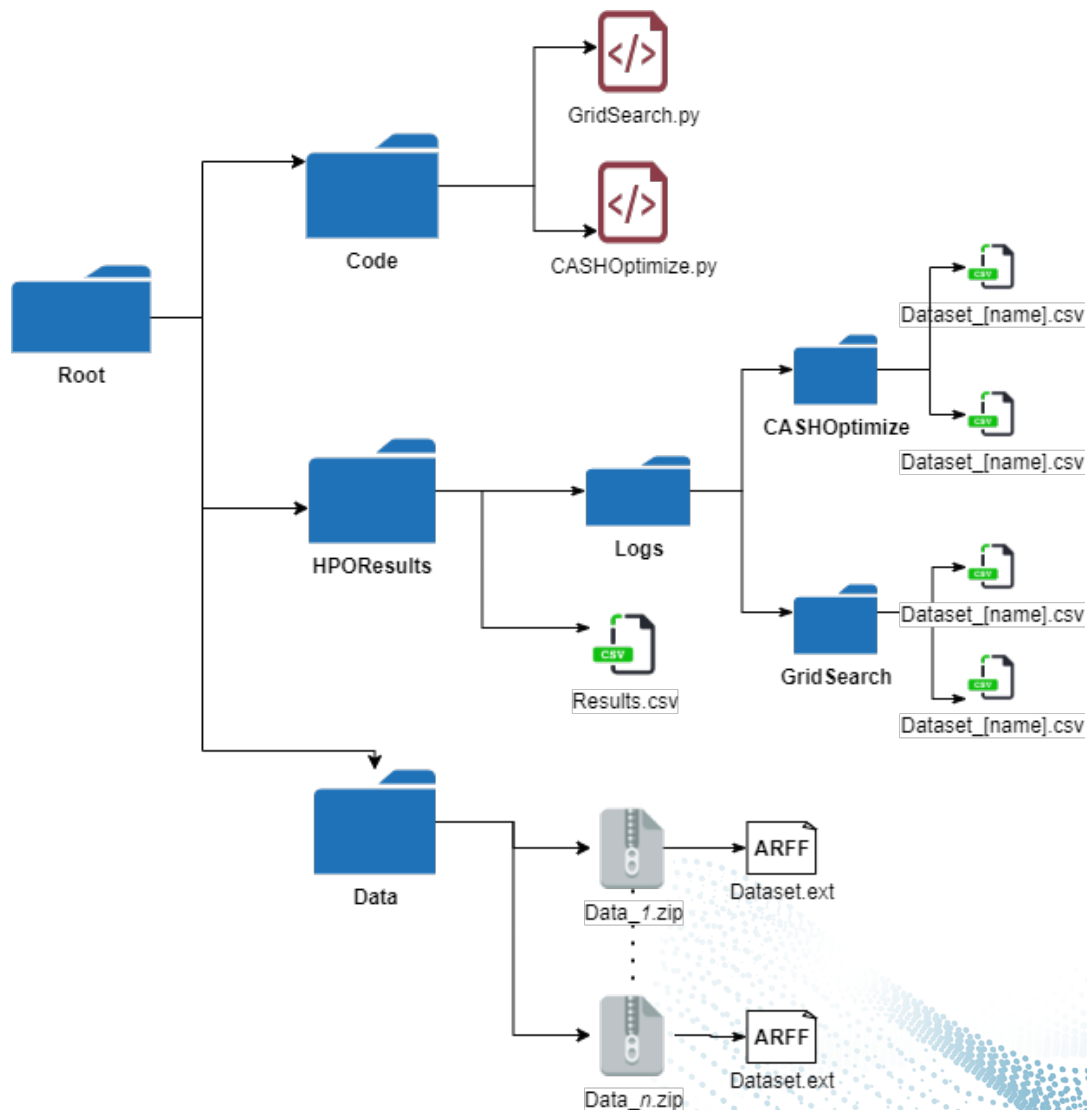


Figure 8. The hierarchical structure of the code repositories for the software module.

## Installation

This software is written in python3 and can be downloaded via :

<https://doi.org/10.5281/zenodo.5163207>

This software requires several packages in the following as build dependencies, which all are available on the PyPi's repositories:

- scikit-learn ( $\geq 0.23.2$ ): to implement the classification algorithms
  - o `pip3 install scikit-learn`
- imbalanced-learn ( $\geq 0.7.0$ ): to implement the resampling algorithms
  - o `pip3 install imbalanced-learn`
- Hyperopt ( $\geq 0.2.5$ ): to perform bayesian optimization.
  - o `Pip3 install hyperopt`

Note that, if installing under anaconda, please use `pip` rather than `pip3`.

## Configuration and example

The following example shows how to fit an arbitrary imbalanced dataset with our software. Our software is end-to-end software. In this work, we provide two independent scripts with similar value inputs:

1. **GridSearch.py** : a python script that will try all combinations of resampling techniques and classification algorithms with the default value of hyperparameters.
  - `RandomState`: is an integer value that uses for reproducibility results, e.g.,  
`RandomState =1`
  - `File`: path to a dataset in the local machine, e.g.,  
`File='../Data/imbalanced.zip'`.
  - `Dataset`: Name of the dataset and its extension, e.g. `Dataset='glass1.dat'`.

Run the provided script with the terminal or command line as python's standard procedure:

`Python GridSearch.py`

2. **CASHOptimize.py**: Our main software is zipped in this python script file. This supports two search strategies are "TPE" and "Randomsearch".
  - `RandomState`: is an integer value that uses for reproducibility results, e.g.,  
`RandomState =1`
  - `File`: path to a dataset in the local machine, e.g.,  
`File='../Data/imbalanced.zip'`.
  - `Dataset`: Name of the dataset and its extension, e.g., `Dataset='glass1.dat'`.
  - `HPOAlg`: uses to identify the search strategy, e.g., `HPOAlg='TPE'`.



Run the provided script with the terminal or command line as python's standard procedure:

```
Python CASHOptimize.py
```

Once the software is finished, the best result (i.e., the best found configuration and its accuracy) will be printed on the screen and recorded in the file 'HPOResults/results.csv'. In addition, the detailed log file caught during the optimize progress is stored under folder 'HPOResults/Logs'.



### 3. User Manual for Section 4

The key-features of the software modules are reported in Table 10. All three software modules are developed in `Python 3.5`, and are compatible with the latest versions. The software module for [38] is only compatible with `PyTorch+cpu`. For extending this software module to the GPU, the code has to be manually tuned.

*Table 10. Key details of all three software modules discussed in the report.*

| <b>Title</b>                          | Investigation of Modelling Techniques for Robust Optimization                        | Investigation of Dimensionality Reduction Techniques for Efficient SAO   | Extending the MGFI for Robust Bayesian Optimization (RBO)                               |
|---------------------------------------|--|--|---|
| <b>Functionality</b>                  | This software module investigates the suitability of the modeling techniques for RO. | This software module evaluates and compares the dimensionality reduction techniques for efficiently constructing the low dimensional models.                 | This software module is used to evaluate and compare the robust MGFI with the baseline. |
| <b>Papers Related to the Software</b> | An Empirical Comparison of Meta-Modeling Techniques for Robust Design Optimization.  | Exploring Dimensionality Reduction Techniques for Efficient Surrogate-Assisted Optimization  | A New Acquisition Function for Robust Bayesian Optimization of Unconstrained Problems   |
| <b>Programming Language</b>           | Python $\geq 3.5$  | Python $\geq 3.5$  | Python $\geq 3.5$   |
| <b>Hardware Requirements</b>          | No Special Hardware Requirements.  | The code is tested for <code>PyTorch+cpu</code> only. For the GPU version, the code has to be manually tuned.  | No Special Hardware Requirements.   |
| <b>Data Sets</b>                      | The software generates the data set it uses to compare the meta-models.              | The software generates the data set it uses to compare the dimensionality reduction techniques for efficiently constructing the low-dimensional meta-models. | The software generates the data set it uses to compare the acquisition functions.       |

The structure of the code repositories for [37] is provided in Figure 9. The repositories do not include the input and output data files, which must be generated by running the code. At the bottom end of the hierarchical structure in this module, there are two Python files, namely "`Generate_Data_Sets.ipynb`" and "`Final_Comparison.ipynb`". These two files are responsible for generating the data set (subject to the noise level and the test problems selected earlier in the hierarchical structure), and performing the comparison of the modelling techniques

for RO. The code<sup>4</sup> part focusing on the hyperparameter optimization is based on the modelling technique chosen. For the creation of graphs and visualizations, a separate repository titled "Results Compilation" can be used. Similar hierarchical structure of the code<sup>5</sup> for [38] is provided in Figure 10. The repositories here also do not include the input and output data files, similar to [37]. The data for the study can be generated using the Python file "Generate\_Data\_Sets.ipynb" by selecting one of the settings of the dimensionality. The Python file "bbobbenchmarks.py" implements the test problems, and is utilized by file "Generate\_Data\_Sets.ipynb". For HPO, and the comparison of the DRTs based on modelling accuracy and the quality of the optimal solution, four Python files in the second hierarchy are utilized. Lastly, the code structure for the implementation of [39] is provided in Figure 11. The code<sup>6</sup> in this case, similar to the previous ones, is based on a hierarchical structure characterizing the most important factors such as the test problem, the noise level, the initial temperature setting and the AF. The Python file "RBO.ipynb", implements the code for the baseline comparison.

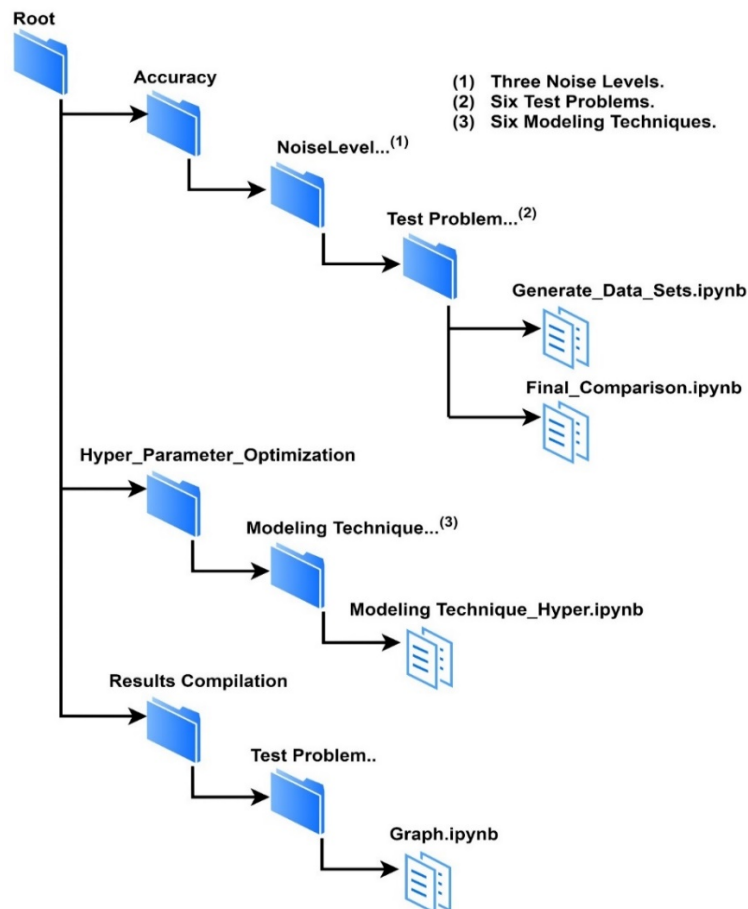


Figure 9. Hierarchical structure of the code repositories in [37]. Three noise levels – small, medium, and high – as well as six test problems and six modelling techniques characterize the organization of these repositories.

<sup>4</sup> The code to reproduce the results in [37] is available at <https://doi.org/10.5281/zenodo.3854910>.

<sup>5</sup> The code to reproduce the results in [38] is available at <https://doi.org/10.5281/zenodo.5500281>.

<sup>6</sup> The code to reproduce the results in [39] is available at <https://doi.org/10.5281/zenodo.5500295>.

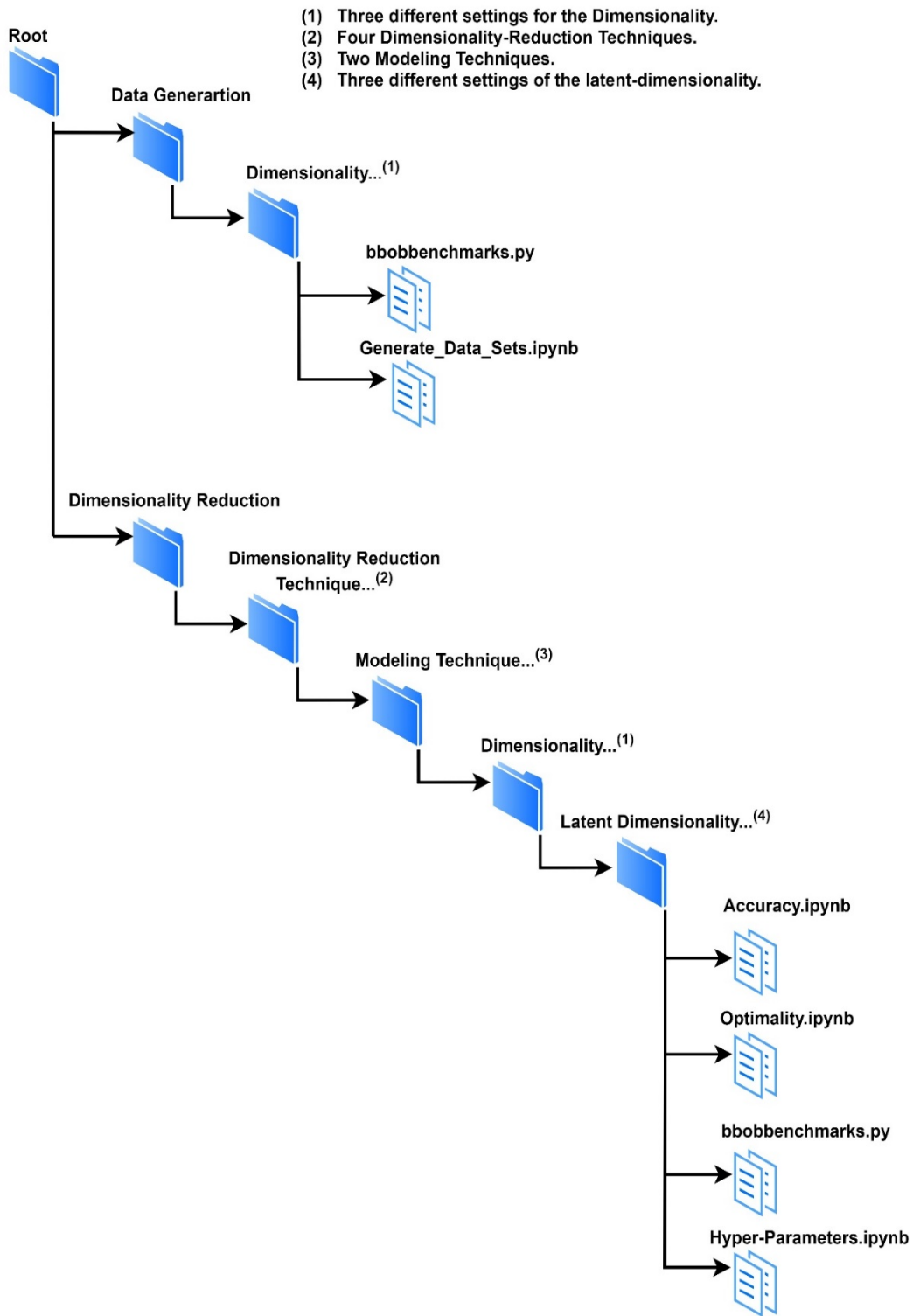


Figure 10. Hierarchical structure of the code repositories in [38]. The code files, i.e., \*.py or \*.ipynb, are specific to the hierarchical structure chosen, e.g., choice of DRT, modeling technique, dimensionality, and the latent size.

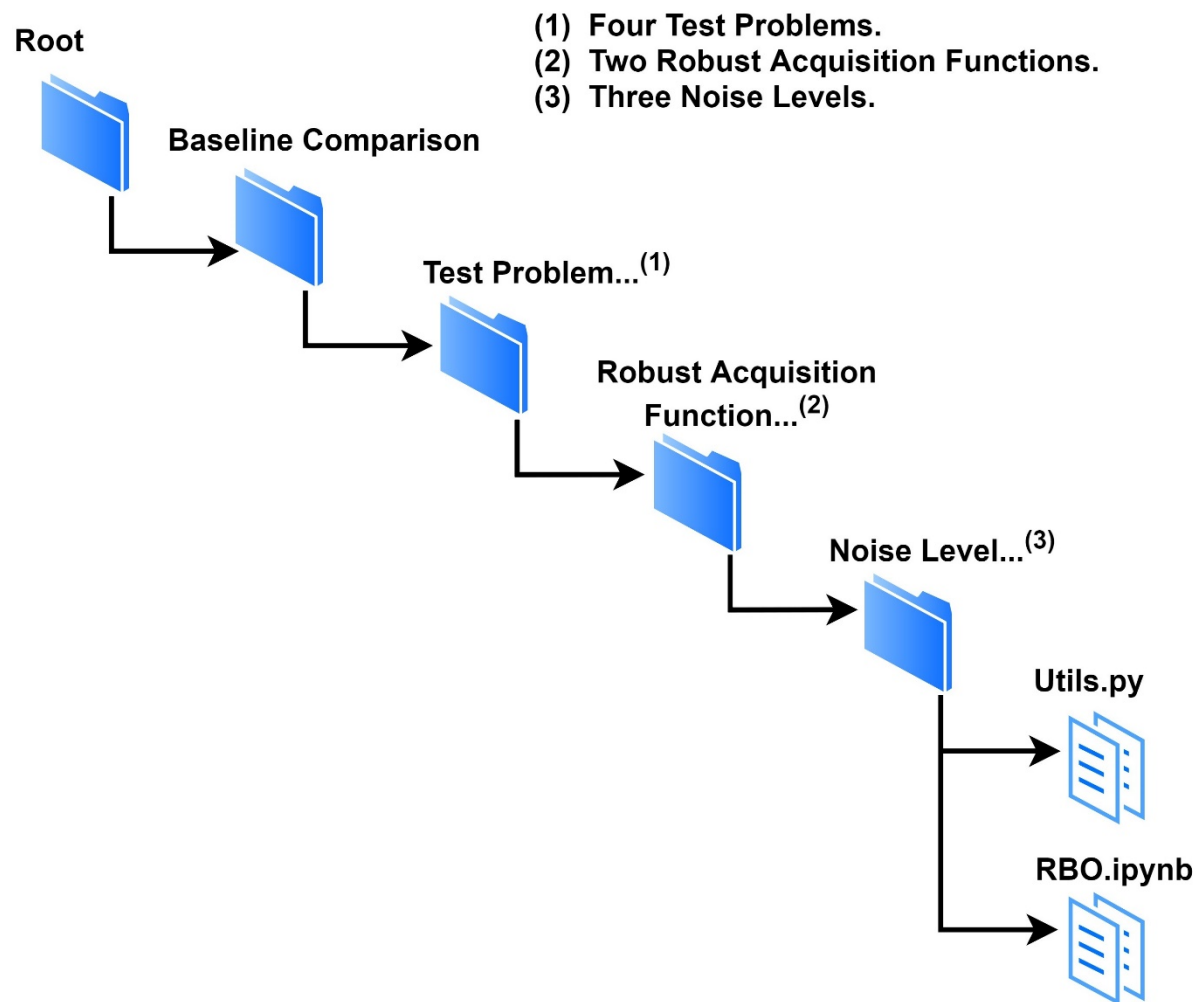


Figure 11. Hierarchical structure of the code repositories in [39]. The code files are specific to the hierarchical structure chosen, e.g., test problem, noise level.

#### 4. User Manual for Section 5

The software modules of the two works in section 5 can be realized via Tr-DMOEAs modules in MATLAB 2018b.

The structure of the code repositories for the work mentioned in section 5.1 [61] is provided in Figure 12. The repositories do not include the input data files. The code is available on <https://doi.org/10.5281/zenodo.5509255>. The used test benchmark functions are defined in the "getFunc.m" file under the folder of "benchmark functions". "IGD.m" is the defined IGD metric to evaluate the performance of found solutions by Tr-DMOEAs on those benchmark functions. Data points that are evenly sampled from the true Pareto front of those benchmark functions are stored in the fold "TruePOF". When running the codes, add the folder "ImTR-DMOEAs" to the MATLAB path, then run the "Main.m" in any Tr- DMOEA (Tr-NSGA-II, Tr-MOPSO and Tr-RMMDA) folder for test. In each folder of Tr- DMOEA, there are three types of codes, which are the "main.m", transfer learning-based codes, like "getKernel.m", "getW.m" and "getNewY.m" and NSGA-II/MOPSO/RMMDA related codes. All parameter settings of the algorithm can be done in the "Main.m" file. The output data files are generated by running the code in the file named "Results" under the folder of each Tr-DMOEAs, i.e., Tr-NSGA-II, Tr-MOPSO and Tr-RMMDA. In order to verify the answer of when and how to transfer in DMO, a folder named "POF-AfTr" under the folder "Results" of Tr-RMMDA is created to store the transferred solutions after changes, as only the running of Tr-RMMDA is used to answer the questions. The "IGD" folder under "Results" of three Tr-DMOEAs store the IGD values of optimized solutions by RM-MEDA, MOPSO and NSGA-II under 20 changes.

The structure of the code repositories for the work mentioned in section 5.2 [62] is provided in Figure 13. The repositories do not include the input data files. The code is available on <https://doi.org/10.5281/zenodo.4289094>. The used test benchmark functions are defined in the "getFunc.m" file under the folder of "benchmark functions". "IGD.m" is the defined IGD metric to evaluate the performance of found solutions by Tr-DMOEAs on those benchmark functions. Data points that are evenly sampled from the true Pareto front of those benchmark functions are stored in the fold "TruePOF". When running the codes, add the folder Root to the MATLAB path, then run the "Main.m" in any Tr- RMMDAs (active-Tr-RMMDA, interior-Tr-RMMDA, SQP-Tr-RMMDA and Time-Tr-RMMDA) folder for test. In each folder of Tr-RMMDA, there are three types of codes, which are the "main.m", transfer learning-based codes, like "getKernel.m", "getW.m" and "getNewY.m" and RMMDA related codes. All parameter settings of the algorithm can be done in the "Main.m" file. The difference of those four Tr-RMMDAs is in the "Main.m" file. active-Tr-RMMDA, interior-Tr-RMMDA, SQP-Tr-RMMDA have the different inner optimisation methods in the "Main.m". As for Time-Tr-RMMDA, the time cost of TCA is recorded to run RMMDA on randomly generated solutions after each change. The output data files are generated by running the code in the file named "Results" under the folder of each Tr-RMMDA. A folder named "POF-AfTr" under the folder "Results" of Tr-RMMDAs is created to store the transferred solutions after changes. In addition, a folder "runtime" under the "Results" is used to record the runtime of Tr-RMMDAs with different inner optimisation methods.



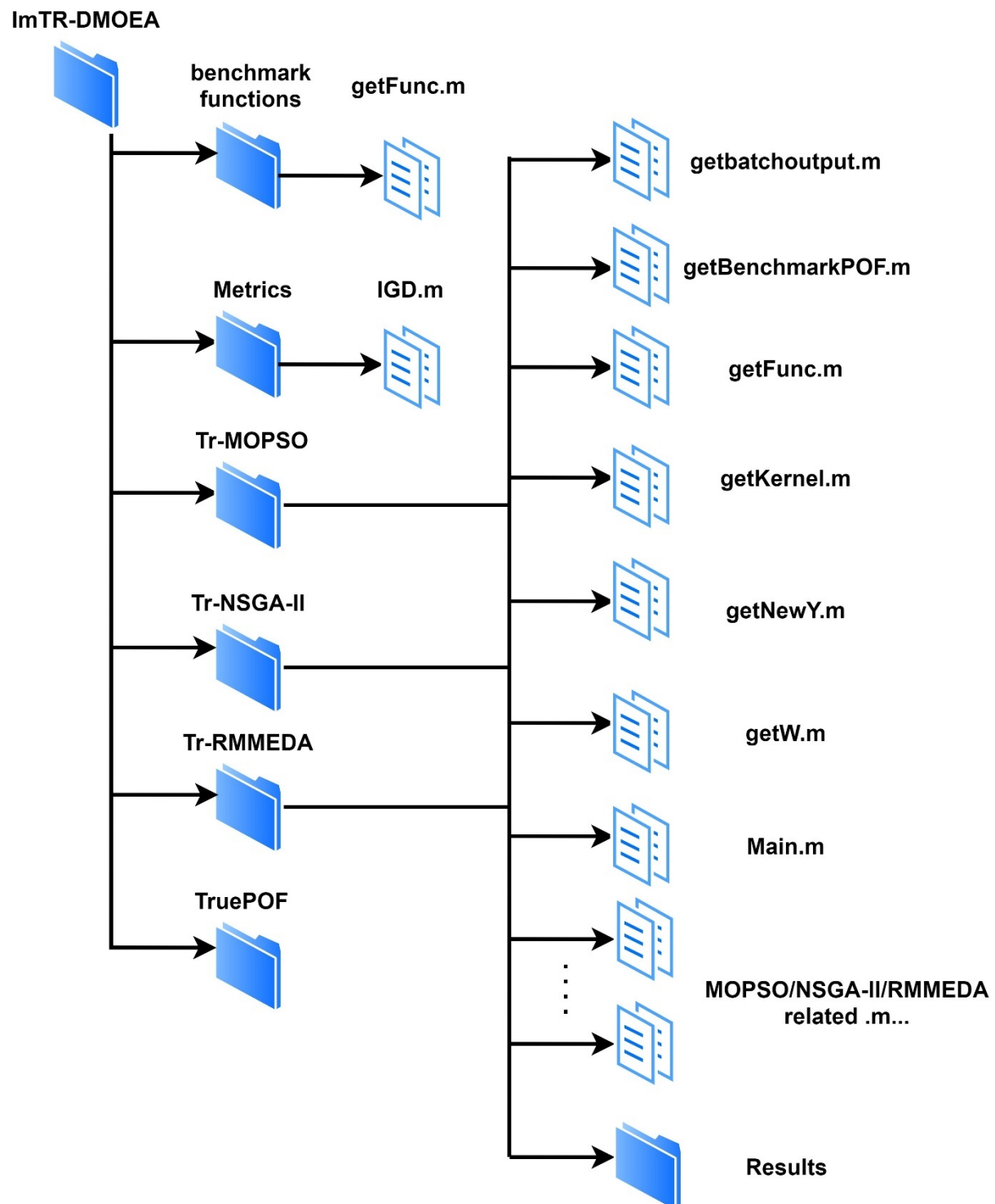


Figure 12. Hierarchical structure of the code repositories in [61].

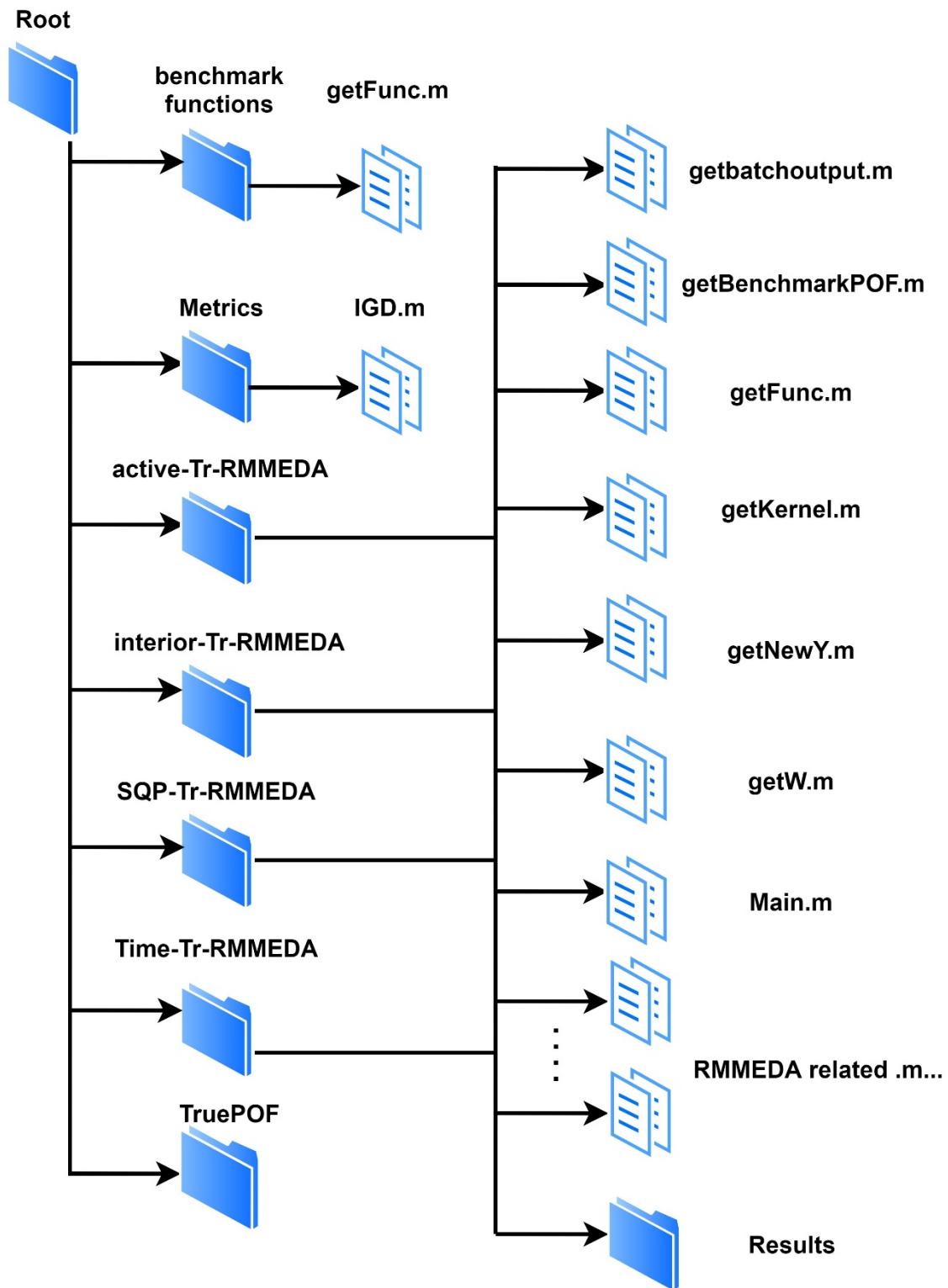


Figure 13. Hierarchical structure of the code repositories in [62].

## Bibliography

- [1] B. Das, N. C. Krishnan and D. J. Cook, "RACOG and wRACOG: Two probabilistic oversampling techniques," *IEEE transactions on knowledge and data engineering*, vol. 27, p. 222–234, 2014.
- [2] H. Zhang and M. Li, "RWO-Sampling: A random walk over-sampling approach to imbalanced data classification," *Information Fusion*, vol. 20, p. 99–116, 2014.
- [3] H.-G. Beyer and B. Sendhoff, "Robust optimization—a comprehensive survey," *Computer methods in applied mechanics and engineering*, vol. 196, no. 33-34, pp. 3190-3218, 2007.
- [4] J. W. Kruisselbrink, *Evolution strategies for robust optimization*, Leiden: Leiden University, 2012.
- [5] M. Farina, K. Deb and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," *IEEE Transactions on evolutionary computation*, vol. 8, p. 425–442, 2004.
- [6] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk and F. Herrera, *Learning from imbalanced data sets*, vol. 11, Springer, 2018.
- [7] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, p. 221–232, 2016.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] S. Barua, M. M. Islam, X. Yao and K. Murase, "MWMOTE—majority weighted minority oversampling technique for imbalanced data set learning," *IEEE Transactions on knowledge and data engineering*, vol. 26, p. 405–425, 2012.
- [12] M. Kubat, S. Matwin and others, "Addressing the curse of imbalanced training sets: one-sided selection," in *ICML*, 1997.
- [13] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," in *Conference on Artificial Intelligence in Medicine in Europe*, 2001.
- [14] K. Napierala and J. Stefanowski, "Types of minority class examples and their influence on learning classifiers from imbalanced data," *Journal of Intelligent Information Systems*, vol. 46, p. 563–597, 2016.
- [15] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, "LOF: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000.
- [16] 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.
- [17] C. Thornton, F. Hutter, H. Hoos and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of classification algorithms," *KDD*, 2012.
- [18] D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. Kononova and T. Bäck, "Improved Automated CASH Optimization with Tree Parzen Estimators for Class Imbalance Problems," in *The 8th IEEE International Conference on Data Science and Advanced Analytics (DSAA2021)*, Porto, Portugal, 2021.
- [19] D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. Kononova and T. Bäck, "Supplementary Material for Improved Automated CASH Optimization with Tree Parzen Estimators for Class Imbalance Problems," <https://doi.org/10.5281/zenodo.5163207>, 2021.
- [20] H. Han, W.-Y. Wang and B.-H. Mao, "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning," 2005.
- [21] F. Last, G. Douzas and F. Bacao, *Oversampling for Imbalanced Learning Based on K-Means and SMOTE*, 2017.
- [22] 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.
- [23] imbalanced-learn, "RandomOverSampler," [Online]. Available: [https://imbalanced-learn.org/stable/generated/imblearn.over\\_sampling.RandomOverSampler.html](https://imbalanced-learn.org/stable/generated/imblearn.over_sampling.RandomOverSampler.html).
- [24] I. Tomek, "Two modifications of CNN," *IEEE Trans. Systems, Man and Cybernetics*, vol. 6, p. 769–772, 1976.
- [25] 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.
- [26] 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.
- [27] I. Tomek, "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, Vols. SMC-6, pp. 448-452, 1976.
- [28] 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.
- [29] 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.
- [30] imbalanced-learn.org, "ClusterCentroids," [Online]. Available: [https://imbalanced-learn.org/stable/generated/imblearn.under\\_sampling.ClusterCentroids.html](https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.ClusterCentroids.html).
- [31] imbalanced-learn.org, "RandomUnderSampler," [Online]. Available: [https://imbalanced-learn.org/stable/generated/imblearn.under\\_sampling.RandomUnderSampler.html](https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.RandomUnderSampler.html).
- [32] G. 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.
- [33] 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.
- [34] 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.
- [35] A. Ben-Tal, L. E. Ghaoui and A. Nemirovski, *Robust optimization*, Princeton University Press, 2009.
- [36] F. Jurecka, *Robust design optimization based on metamodeling techniques*, Munich: Technical University of Munich, 2007.
- [37] S. Ullah, H. Wang, S. Menzel, B. Sendhoff and T. Bäck, "An Empirical Comparison of Meta-Modeling Techniques for Robust Design Optimization," in *IEEE Symposium Series on Computational Intelligence*, Xiamen, 2019.
- [38] S. Ullah, H. Wang, S. Menzel, B. Sendhoff and T. Bäck, "Exploring Dimensionality Reduction Techniques for Efficient Surrogate-Assisted Optimization," in *IEEE Symposium Series on Computational Intelligence*, Canberra, 2020.
- [39] S. Ullah, H. Wang, S. Menzel, B. Sendhoff and T. Bäck, "A New Acquisition Function for Robust Bayesian Optimization of Unconstrained Problems," in *Genetic and Evolutionary Conference Companion*, Lille, 2021.
- [40] C. M. Bishop, *Pattern recognition and machine learning*, springer, 2006.
- [41] I. Goodfellow, A. Courville and Y. Bengio, *Deep learning*. Vol. 1. No. 2., Cambridge: MIT press, 2016.
- [42] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2013.
- [43] D. P. Kingma, D. J. Rezende, S. Mohamed and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in neural information processing systems*, 2014.
- [44] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323-2326, 2000.
- [45] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373-1396, 2003.
- [46] P. Demartines and J. Herault, "Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets," *IEEE Transactions on neural networks*, vol. 8, no. 1, pp. 148-154, 1997.
- [47] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, pp. 114-144, 2021.
- [48] S. ur Rehman, M. Langelaar and F. van Keulen, "Efficient Kriging-based robust optimization of unconstrained problems," *Journal of Computational Science*, vol. 5, no. 6, pp. 872-881, 2014.
- [49] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, p. 1345-1359, 2009.
- [50] G. Ruan, G. Yu, J. Zheng, J. Zou and S. Yang, "The effect of diversity maintenance on



- prediction in dynamic multi-objective optimization," *Applied Soft Computing*, vol. 58, p. 631–647, 2017.
- [51] A. Zhou, Y. Jin and Q. Zhang, "A population prediction strategy for evolutionary dynamic multiobjective optimization," *IEEE transactions on cybernetics*, vol. 44, p. 40–53, 2013.
- [52] M. Jiang, Z. Huang, L. Qiu, W. Huang and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 22, p. 501–514, 2017.
- [53] Q. Zhang, A. Zhou and Y. Jin, "RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, p. 41–63, 2008.
- [54] K. Deb, A. Pratap, S. Agarwal and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, p. 182–197, 2002.
- [55] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, p. 151–175, 2010.
- [56] A. Smola, A. Gretton, L. Song and B. Schölkopf, "A Hilbert space embedding for distributions," in *International Conference on Algorithmic Learning Theory*, 2007.
- [57] J. Shawe-Taylor, N. Cristianini and others, *Kernel methods for pattern analysis*, Cambridge university press, 2004.
- [58] M. Helbig and A. Engelbrecht, "Benchmark functions for cec 2015 special session and competition on dynamic multi-objective optimization," *Dept. Comput. Sci., Univ. Pretoria, Pretoria, South Africa, Rep*, 2015.
- [59] J. Nocedal and S. Wright, *Numerical optimization*, Springer Science & Business Media, 2006.
- [60] J. Kong, W. Kowalczyk, S. Menzel and T. Bäck, "Improving Imbalanced Classification by Anomaly Detection," in *International Conference on Parallel Problem Solving from Nature*, Leiden, 2020.
- [61] G. Ruan, L. L. Minku, S. Menzel, B. Sendhoff and X. Yao, "When and how to transfer knowledge in dynamic multi-objective optimization," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019.
- [62] G. Ruan, L. L. Minku, S. Menzel, B. Sendhoff and X. Yao, "Computational Study on Effectiveness of Knowledge Transfer in Dynamic Multi-objective Optimization," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020.