Universiteit
Leiden
The Netherlands

Efficient tuning of automated machine learning pipelines

Nguyen, D.A.

Citation
Nguyen, D. A. (2024, October 9). *Efficient tuning of automated machine learning pipelines*. Retrieved from https://hdl.handle.net/1887/4094132

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4094132

Note: To cite this publication please use the final published version (if applicable).

CHAPTER 6

On the use of AutoML optimization in real-world applications

Accurate classification of multiple classes is crucial in industrial applications, especially in identifying surface defects in the steel industry. Quality of surface of steel products is among the most significant contributors to their overall quality. Therefore, it is of vital importance to detect and classify various surface defects correctly. While established quality control measures implemented at various production stages successfully warrant against the high number of defects, they complicate further defect detection due to the high imbalance in the occurrence of defects vs defect-free cases. The situation is further complicated by a wide range of possible types of surface defects, with a heavily imbalanced distribution among them. In addition, setting appropriate hyperparameters of new classification methods to obtain a stable and accurate classification performance is far from straightforward given their strong interdependence. A hyperparameter optimizer is typically applied to identify the best Machine Learning (ML) model by evaluating its performance based on standard metrics such as accuracy rate, recall, precision, etc. However, some classes are more important in many real-world applications. Thus, to accommodate the latter, we propose an approach for penalizing existing classification performance metrics with a user-defined class importance matrix. We demonstrate the proposed approach on a highly imbalanced instance of multi-class classification of steel surface defects. We solve the Combined Algorithm Selection and hyperparameter optimization (CASH) problem to identify the best ML model. Such optimization is done by means of a competitive Bayesian optimization method in a search space of 21 resampling techniques and 5 classification algorithms (and their corresponding hyperparameter settings) for three commonly used multiple-class classification techniques (Multi-class direct classification, One vs. One and One vs. Rest). The results of our experiments show that the proposed approach

improves the performance significantly on the considered classification problem compared to the current classification system at TATA Steel (TATA).

The remainder of this chapter is organized as follows. The motivation, introduction and problem formulation are provided in Section 6.1. In Section 6.2, the relevant background knowledge on imbalance classification and hyperparameter optimization are provided, and Section 6.3 lays out the experimental setup. Additionally, experimental results are discussed in Section 6.3.3. Finally, the chapter is concluded in Section 6.4.

6.1 Introduction

The appearance of surface of a steel product is one of the significant quality aspects [49]. While established quality control measures already implemented at various production stages successfully warrant against the high number of defects in the resulting products, they complicate further defect detection due to the high imbalance in the occurrence of defects vs defect-free cases. The situation is further complicated by a wide range of different types of surface defects, with a heavily imbalanced distribution among these defect kinds. Additionally, setting appropriate hyperparameters of new classifiers to obtain a stable and accurate classification performance is far from straightforward given their strong interdependence. To maximize the classification performance, practitioners need to find a fine-tuned ML pipeline out of an extensive portfolio made up of a range of suitable algorithms with their complex hyperparameter settings. The practical surface defects classification problem faces two main challenges: (i) unequal/imbalanced distribution of defects across classes, (ii) unequal importance between classes (some imperfections are more severe than others).

The imbalanced classification problem can be solved by applying a well-performing combination of a resampling technique and a classification algorithm [74]. Finding such a well-performing combination of methods and the setting of their hyperparameters falls within the problem domain Combined Algorithm Selection and hyperparameter (CASH) optimization problems which can be solved effectively [47] via the Bayesian optimization [153].

Inside the optimization, the assessment method is critical in evaluating classification performance to choose the suitable classification model for the given problem. The performance metrics usually assume that all classes are equally important. However, users might want to stipulate preferences over classes. To

illustrate this, Figs. 6.1a, 6.1b show two classification outcomes that are indistinguishable from standard performance metrics. In practice, the practitioner will prefer the first outcome if `class 2` is more important than `class 1`. Therefore, the existing classification performance metrics are not able to evaluate and rank ML models for unequal class importance values. Consequently, the *automatic machine learning approaches* (such as Hyperparameter Optimization, CASH optimization, and AutoML optimization), which are mainly focused on selecting the ML model with the best predictive performance, *are not able to solve the problem efficiently in case of unequal class importance values.*

To solve that unequal importance classes problem, the assessment method has to be adjusted to reward correct predictions of important classes while penalizing incorrect predictions of those classes. Since almost all performance metrics are built on the confusion metric [221], we propose a novel approach that adjusts the confusion matrix by combining the confusion matrix with a user-defined penalty matrix (see Figure 6.1c), which contains different weights for predictions over classes. The general performance metric is then computed based on the penalized confusion matrix, potentially helping the optimizer solve the unequal importance classes efficiently.

Based on the formulation of the target objective function (Equation 1.2) given in Section 1.1, let $\alpha(\hat{y}_i, y_i)$ denote a penalty value of $\hat{y}_i$ and y_i, that can be extracted

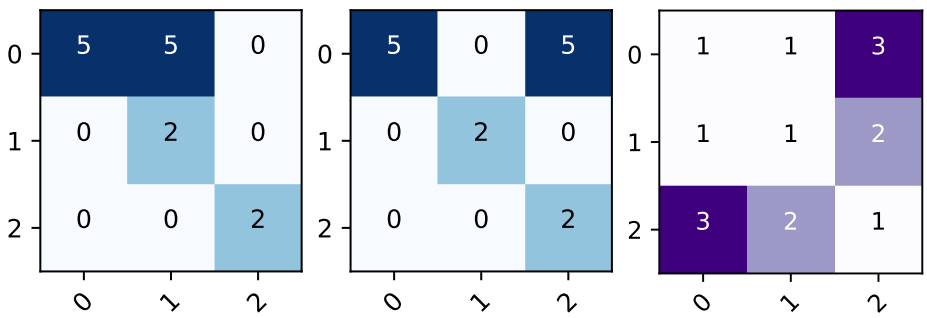


Figure 6.1: Examples of two confusion matrices with misclassifications in `class 0` (left and middle figures, where true and predicted labels are shown vertically and horizontally, respectively). A corresponding example of proposed penalty matrix (right figure) indicating that `class 2` is more important than `class 0` and `class 1`. Numbers in the penalty matrix indicate the misclassification severity weights per predicted-true label pair (e.g. a sample of `class 2` misclassified as `class 0` will be multiplied by 3).

from the pre-defined penalty matrix $\mathbf{P}_{\mathrm{M}\times\mathrm{M}}$ of M classes (see Figure 6.1c for illustration, for example, $\alpha(0,2)=3$). Adding $\alpha(\hat{y}_i, y_i)$ to the Equation 1.2, to punish wrong or reward correct prediction on class i. Then, the Equation 1.2 to compute the overall performance of the ML model p when trained on $\mathcal{D}_{\text{train}}$ of n samples and evaluated on $\mathcal{D}_{\text{valid}}$ of $(m-n)$ samples, can be adapted to:

$$f(p, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \mathbf{P}_{\mathrm{M}\times\mathrm{M}}) = \frac{1}{m-n}\sum_{j=1}^{m-n} R(\hat{y}_{n+j}, y_{n+j}, \alpha(\hat{y}_{n+j}, y_{n+j})) \tag{6.1}$$

where $\hat{y}$ is the predicted class, y is the true class and R indicates a measure metric.

Referring back to the discussion on CASH approach in Chapter 1 (Section 1.1.2) and search space for class imbalance problem in Chapter 5 (Section 5.2.2), the ML model p is structured as $p = \{(\mathcal{A}_{res}, \lambda_{res}, \mathcal{A}_{cls}, \lambda_{cls}) | \mathcal{A}_{res} \in \lambda^0_{res}, \mathcal{A}_{cls} \in \lambda^0_{cls}, \lambda_{res} \in \{\Lambda^1_{res}, \ldots, \Lambda^{n_r}_{res}\}, \lambda_{cls} \in \{\Lambda^1_{cls}, \ldots, \Lambda^{n_c}_{cls}\}\}$. Hence, the CASH optimization can be defined as:

$$p^* = \arg\max_{p\in\Lambda} f(p, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \mathbf{P}_{\mathrm{M}\times\mathrm{M}}) \,, \tag{6.2}$$

where $f(p, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \mathbf{P}_{\mathrm{M}\times\mathrm{M}})$ denotes the penalized performance accuracy of the ML pipeline p when trained on $\mathcal{D}_{\text{train}}$, evaluated on $\mathcal{D}_{\text{valid}}$, and penalized by the penalty matrix $\mathbf{P}_{\mathrm{M}\times\mathrm{M}}$.

6.2 Background

In this section, we review some background knowledge. We first provide a brief introduction of multi-class classification approaches (Section 6.2.1), the commonly used performance metric in the field of multi-classes imbalanced learning (Section 6.2.2) is presented.

6.2.1 Multi-Class Imbalance Learning

Most studies on the classification problem are devoted to the two-class classification scenario. However, a significant number of real-world applications contain more than two classes, for instance, image classification, protein classification, and medical diagnosis. The increasing number of classes poses new challenges for learning from multi-class imbalanced problems. First of all, more decision boundaries need to be defined during the multi-class classification process. Another challenging issue is that the imbalance among classes becomes more complicated as there will be multi-majority and multi-minority classes [222]. The data complexity, an

important cause of the degradation in binary case [220], is more sophisticated. Several solutions designed for imbalanced binary classification are extended to multi-class scenarios.

Class decomposition is an intuitive method to deal with multi-class classification problems [223]. After transforming the multi-class problem into multiple subsets, the existing approaches for handling the binary scenarios can be applied directly. Unlike the class decomposition approaches, *multi-class direct classification* [224] aims to solve the multi-class problem directly without reducing the problem to multiple binary classification tasks. This section first reviews two commonly used decomposition strategies: One vs. Rest (OvR) and One vs. One (OvO), see Sections 6.2.1.1and 6.2.1.2, respectively. Lastly, the multi-class direct classification method is given in Section 6.2.1.3.

6.2.1.1 One vs. Rest approach

Suppose there are M classes in the multi-class imbalanced problem. In the OvR decomposition, each of the M classes is trained against the remaining $(M-1)$ classes [225]. In other words, an M-class classification problem is decomposed into M binary classification problems. When predicting the final label for a test sample, each binary classifier provides a prediction with confidence, and the prediction with the *highest confidence* is usually determined as the final label for this test sample. While OvR provides the convenience of treating multi-class scenarios as binary scenarios, it also brings further imbalance into the binary subsets. In addition, all the individual classifiers are trained with the complete dataset, ensuring no information is dropped in the training procedure. However, this also preserves the overlapping regions, a factor leading to the degradation of the classification performance [220].

6.2.1.2 One vs. One approach

In the OvO decomposition, each of the M classes is trained against one of the remaining classes [226]. Thus, an M-class classification problem is decomposed into $M(M-1)/2$ binary problems, i.e. $M(M-1)/2$ classifiers will be built. The final predictions are usually determined via the *majority voting* among the $M(M-1)/2$ classifiers. Each binary classifier is only trained with pairs of classes; this makes the decision boundaries much simpler and properly addresses the overlapping issue. However, when pairing the classes, the number of binary classifiers increases quadratically in M [227]. The training time can be long if M is large.

6.2.1.3 Multi-class direct classification

The class decomposition methods are typically time-consuming as they transform the single multi-class problem into multiple binary problems, i.e., decomposing the input data into smaller parts or features. The *multi-class direct classification* (direct method) indicates the approach using a single classification algorithm to map input features to output (multi) classes directly, making it faster compared to the class decomposition approaches [228]. Hence, this approach only applies to the classification algorithms that can be modified, e.g., [229], [230] proposed to adjust the decision function in support vector machines, or are naturally designed to be applicable to multi-class problems. For examples, decision trees [71], [231], support vector machines [21], [232], k-nearest neighbors [233]–[235], logistic regression [236], and random forest [72], are suitable algorithms.

6.2.2 Performance Metrics

Table 6.1: Confusion matrix for a multi-class classification problem

True/Actual Class	Predicted Class: A	B	...	M
A	$\mathbf{TP_A}$	$E_{A,B}$	...	$E_{A,M}$
B	$E_{B,A}$	$\mathbf{TP_B}$	...	$E_{B,M}$
...	...	...	**...**	...
M	$E_{M,A}$	$E_{M,B}$	...	$\mathbf{TP_M}$

The assessment method is key in evaluating a classification performance to choose the right classification model for the given problem. In a classification problem, the confusion matrix is a common method to determine the performance of a classifier, as it can provide classification results. For instance, Table 6.1 shows the confusion matrix for a multi-class classification problem with M classes $(\mathrm{A}, \mathrm{B}, \ldots, \mathrm{M})$. As shown, $\mathrm{TP_A}$ is the number of True Positive (TP) samples in class A, and $\mathrm{E_{A,B}}$ is the number of samples from class A that were incorrectly predicted as class B. Hence, the number of False Negatives in class A ($\mathrm{FN_A}$) is the sum of $\mathrm{E_{A,B}}$ to $\mathrm{E_{A,M}}$, i.e., $\mathrm{FN_A} = \mathrm{E_{A,B}} + \ldots + \mathrm{E_{A,M}}$, which indicates the sum of all class A samples that were misclassified. Whereas the number of False Positives (FP) in class A is the sum of all samples that were misclassified as class A, i.e., $\mathrm{FP_A} = \mathrm{E_{B,A}} + \ldots + \mathrm{E_{M,A}}$.

Performance metrics for multi-class classification are usually decomposed into multiple single-class performance metrics by converting the confusion matrix in Table 6.1 into multiple 2×2 confusion matrices:
$\begin{bmatrix} \mathrm{TP_A} & \mathrm{FP_A} \\ \mathrm{FN_A} & \mathrm{TN_A} \end{bmatrix}, \ldots, \begin{bmatrix} \mathrm{TP_M} & \mathrm{FP_M} \\ \mathrm{FN_M} & \mathrm{TN_M} \end{bmatrix}$
The common per-class measurement metrics are presented in Table 2.1.

To compute an overall performance, the scores per class can be averaged to obtain a single score [227], [237]. There are three ways:

- **Macro** approach averages all per-class scores using the arithmetic mean of those values without considering the sample size difference between classes.
- **Weighted** approach is similar to the macro process but takes the sample size rate of classes, e.g., the sample size rate of class A is the number of samples of class A over the total number of samples.
- **Micro** approach computes the corresponding performance metrics by counting the sums of the True Positives (TP), False Negatives (FN), True Negatives (TN), and False Positives (FP).

In this chapter, we use the penalized geometric mean micro ($\mathrm{GM}^{\mathbf{P}}_{\mathrm{micro}}$) as the objective function to maximize, calculated as:

$$\begin{aligned} \mathrm{GM}^{\mathbf{P}}_{\mathrm{micro}} &= \sqrt{\mathrm{Specificity}^{\mathbf{P}}_{\mathrm{micro}} \times \mathrm{Sensitivity}^{\mathbf{P}}_{\mathrm{micro}}} \\ &= \sqrt{\frac{\sum_{i=1}^{\mathrm{M}} \mathrm{TN}^{\mathbf{P}}_i}{\sum_{i=1}^{\mathrm{M}} \mathrm{TN}^{\mathbf{P}}_i + \sum_{i=1}^{\mathrm{M}} \mathrm{FP}^{\mathbf{P}}_i} \times \frac{\sum_{i=1}^{\mathrm{M}} \mathrm{TP}^{\mathbf{P}}_i}{\sum_{i=1}^{\mathrm{M}} \mathrm{TP}^{\mathbf{P}}_i + \sum_{i=1}^{\mathrm{M}} \mathrm{FN}^{\mathbf{P}}_i}} \end{aligned} \tag{6.3}$$

where $\mathrm{TP}^{\mathbf{P}}_i, \mathrm{TN}^{\mathbf{P}}_i, \mathrm{FP}^{\mathbf{P}}_i, \mathrm{FN}^{\mathbf{P}}_i$ denote the number of penalized true positives, penalized true negatives, penalized false positives and penalized false negatives samples in class $i, i \in \mathrm{M}$ classes, respectively. Those values are based on the proposed penalized confusion matrix.

Based on the input (standard) confusion matrix $C_{\mathrm{M} \times \mathrm{M}}$ and a penalty matrix $\mathbf{P}_{\mathrm{M} \times \mathrm{M}}$ (defined by user). We take the Hadamard product, i.e., the pairwise product, of the two matrices, i.e., $(C'_{\mathrm{M} \times \mathrm{M}})_{ij} = (C_{\mathrm{M} \times \mathrm{M}})_{ij} \cdot (\mathbf{P}_{\mathrm{M} \times \mathrm{M}})_{ij}$, where C' denotes the penalized confusion matrix.

6.3 Experiments

In this section, we briefly introduce the dataset (Section 6.3.1) and the experimental procedure (Section 6.3.2).

6.3.1 Datasets

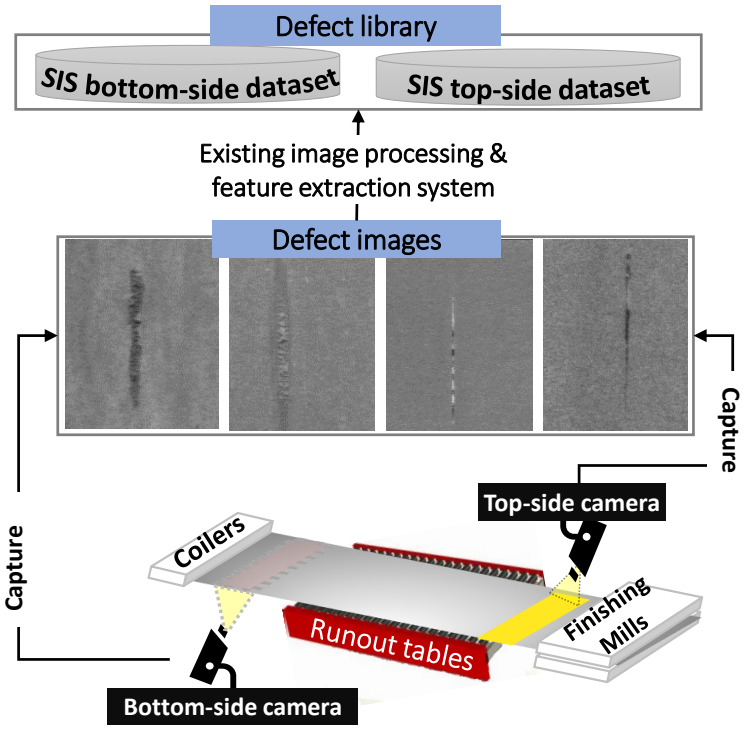


Figure 6.2: Schematic explanation of how the defect images are captured. Defect images are taken from TATA's official website[1], for illustration.

The appearance of the surface of a steel product is one of the major quality aspects. Therefore, surface defects should be avoided or at least known. A camera-based Surface Inspection System (SIS) is used in various process lines to identify those defects in the industry [238]. Grey value images taken from the surface by the SIS contains information on the defects. These images of various defects occurring in production are assessed and gathered in defined classes within a so-called *defect library*. Figure 6.2 shows a diagram, illustrating how the defect images are captured in the production process. The defect library is used to train and test classifiers, and these classifiers are finally used to identify the new surface defects from production. Thus, stable, accurate, and high classification

[1] https://automation.tatasteel.com/products/rolling-mills/squins-surface-quality-inspection-system/

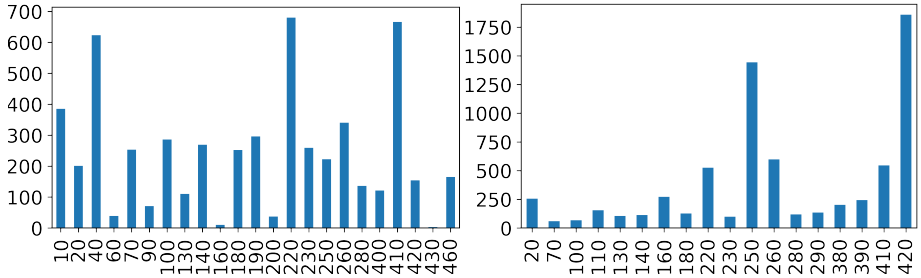


Figure 6.3: The distribution of samples over classes for the top-side (left) and bottom-side (right) camera datasets. The top-side dataset contains 23 classes and 5578 samples, while the bottom-side dataset contains 18 classes and 6908 samples (158 attributes in both cases).

performance is a must in the quality control procedure. However, the imbalance in the number of various defect types makes it challenging to obtain a stable and accurate classification performance.

The images captured by the SIS cameras are processed in the feature extraction module. Then, relevant defect features, e.g., geometrical, textural, and moment features, are extracted for classification. Both the images and information after extraction are stored in the defect library. The surface defects dataset used in this chapter is taken from a defect library after a specific selection was made (for confidentiality reasons). The library is split into two datasets with 158 features: the *top-side* camera and the *bottom-side* camera dataset. The top-side dataset contains 5578 samples distributed in 23 classes. The bottom-side dataset contains 6908 samples distributed in 18 classes. The distribution of the classes on surface defects data used for the experiments is given in Figure 6.3.

6.3.2 Experimental procedure

In this chapter, we experiment with two datasets (top- and bottom-side, see Section 6.3.1) with three multi-class classification strategies, i.e., One vs. Rest (OvR), One vs. One (OvO), and direct method (see Section 6.2.1). TPE as implemented in the `Python` package HyperOpt[2] (version 0.2.5) is used as the mere CASH optimization algorithm with a budget of 500 function evaluations. We reuse the search space identical to Section 4.2, with 5 classification algorithms (Support Vector Machines (SVM), Random Forest (RF), k-Nearest Neighbors

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

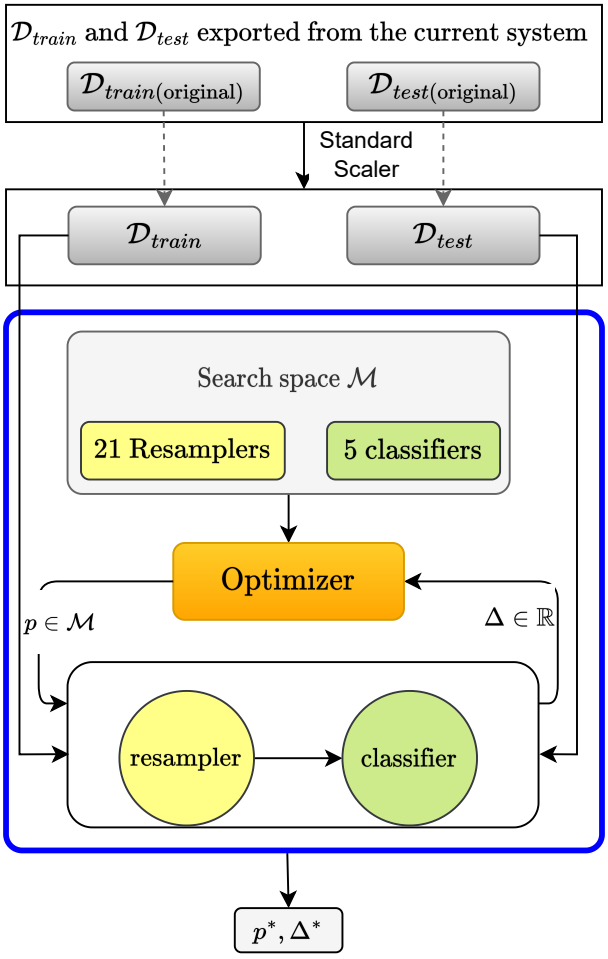


Figure 6.4: Flowchart of the experimental setup.

(KNN), Decision Tree (DT) and Logistic Regression (LR)) and 21 choices of resampling techniques.

In this study, we set up three independent experiments, each representing a different approach mentioned in Section 6.2.1, i.e., One vs. Rest, One vs. One and Direct method. Our experiments aim to compare the current classification system (*current system* in figures and tables below) used by TATA[3]. We use the same training and test datasets as the *current system* for a fair comparison. The

[3]For reasons of confidentiality, since proprietary software of a supplier is used by the industrial partner, no details about the algorithmic approach taken by the currently used system are available.

Table 6.2: Average penalized geometric mean (micro), rounded to 5 decimals over 10 repetitions for the 2 datasets. Boldface highlights the best-performing method per dataset and underline indicates results that are significantly different from the best method in that group according to a Wilcoxon signed-rank test ($p < 0.05$).

Dataset	Direct method	OvO	OvR	Current system
Top side	0.88293	0.88475	**0.88729**	0.81308
Bottom side	0.90990	**0.91275**	0.91245	0.79811

current system executes 10 times on each of the tested datasets. For each execution, the considered dataset is randomly split into training (80%) and test (20%) sets. The prediction performances are reported in Section 6.3.3, and the train/test sets are exported to use in our experiments. i.e., we have $2 \times 10 = 20$ different train/test sets in total. The overall structure of our implementation is summarized in Figure 6.4. The experimental process begins with a data normalization step by applying the so-called Standard Scaler[4] function to the input dataset, i.e., resulting in zero mean and a standard deviation of one. Then, the training and test datasets are fed into the optimization phase.

During the optimization process, the training dataset is used for the ML pipeline proposed by the optimizer. The ML pipeline is then measured by evaluating its prediction performance on the test dataset. We note that the performance is computed based on the penalized confusion matrix that is recomputed by multiplying values in the standard confusion matrix with the corresponding values in the penalty matrix, which is defined by TATA's domain experts (see Figure 6.5). The final evaluation value is calculated by computing the geometric mean (micro) on that penalized confusion matrix. Lastly, the reported result of each method for an individual dataset is averaged over 10 executions.

6.3.3 Results

In this section, we report the results and discuss our insights. The experimental results are summarized in Table 6.2 to illustrate the performance differences between the three classification strategies used: Direct method and two decomposition approaches, i.e., One vs. One (OvO) and One vs. Rest (OvR), and to compare them against the classification approach used in the *current system*. The highest performance for each dataset is highlighted in bold. The methods performing

[4]Standard scaler is implemented in the python library scikit-learn (version 0.23.2).

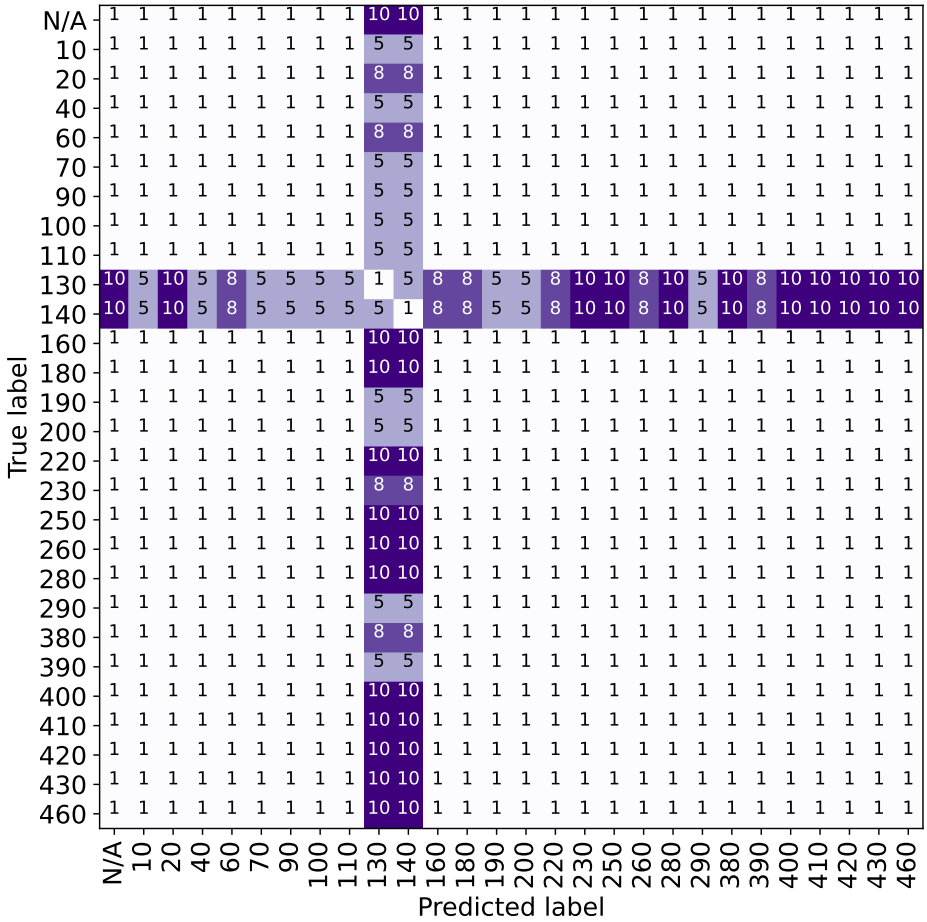


Figure 6.5: Penalty matrix used in our experiments

significantly worse than the best according to the Wilcoxon signed-rank test with $\alpha = 0.05$ are underlined.

Additionally, the distribution of the used performance metric, i.e., the penalized geometric mean (micro), over 10 repetitions for the two tested datasets, is visualized in Figure 6.6. Each box plot represents 10 repetitions. The horizon inner line shows the median. The whiskers show the lowest and the highest observed value[5]. The color dots show the observed values, and the dots outside the whisker represent the outliers. The box covers the first to the third quantiles.

The results allow the following insights:

[5]whisker scale is set as 1.5.

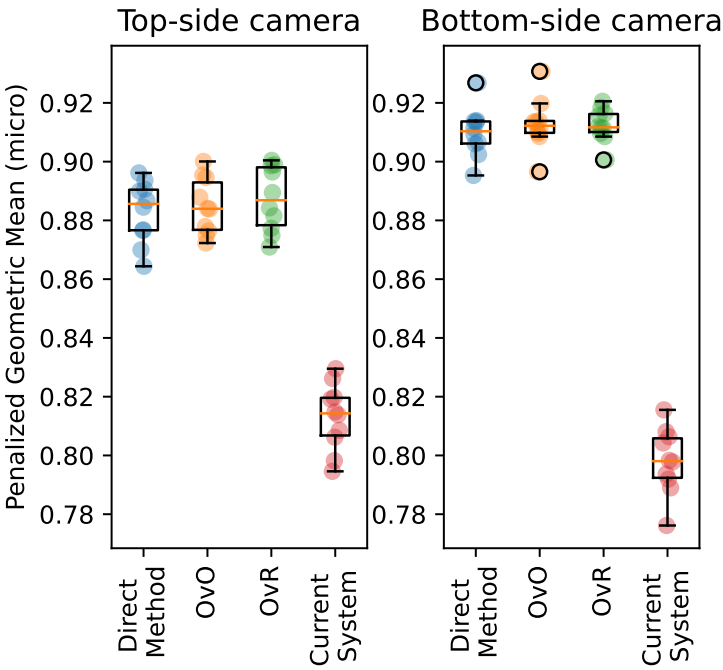


Figure 6.6: Box plots showing the distribution of classification results over 10 repetitions for two examined datasets.

- According to the results of the Wilcoxon signed-rank test, our experimental approaches significantly outperform the current approach used at TATA (*current system*). Additionally, from Figure 6.6, the median and whiskers of our three approaches are higher than those of the *current system*.

- Overall, the decomposition approaches produce the highest performance for both tested cases. More precisely, OvO shows the highest result on the "Bottom side camera" dataset, while OvR achieves the highest result on the "Top side camera" dataset.

- Additionally, according to our experimental results, *Direct method* does not outperform decomposition approaches but is not significantly worse than any decomposition approaches over all tested cases.

As mentioned in Section 6.2, the decomposition approaches are more expensive than the direct classification approach. To investigate this in more detail, we provide Figure 6.7 to show the running time of 10 executions for the 3 experimental approaches. The colour box shows the running time for 1 execution of 500 function

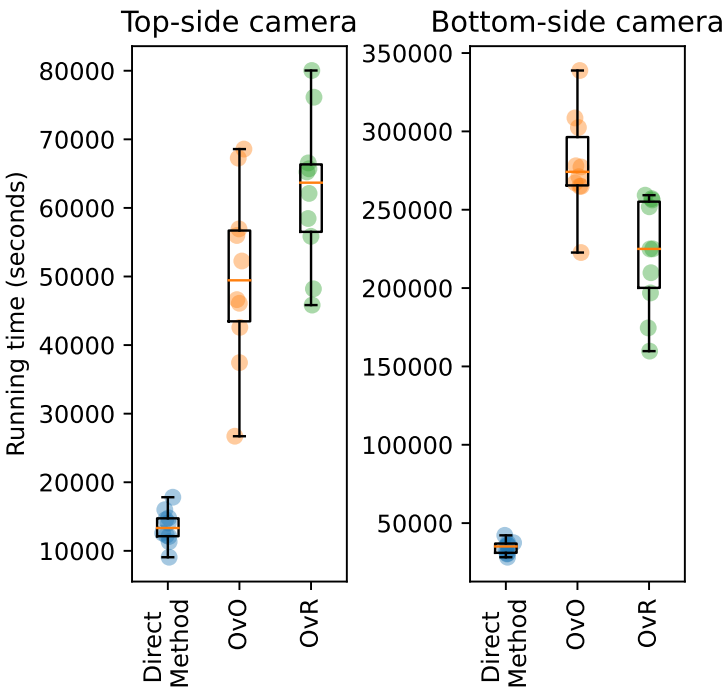
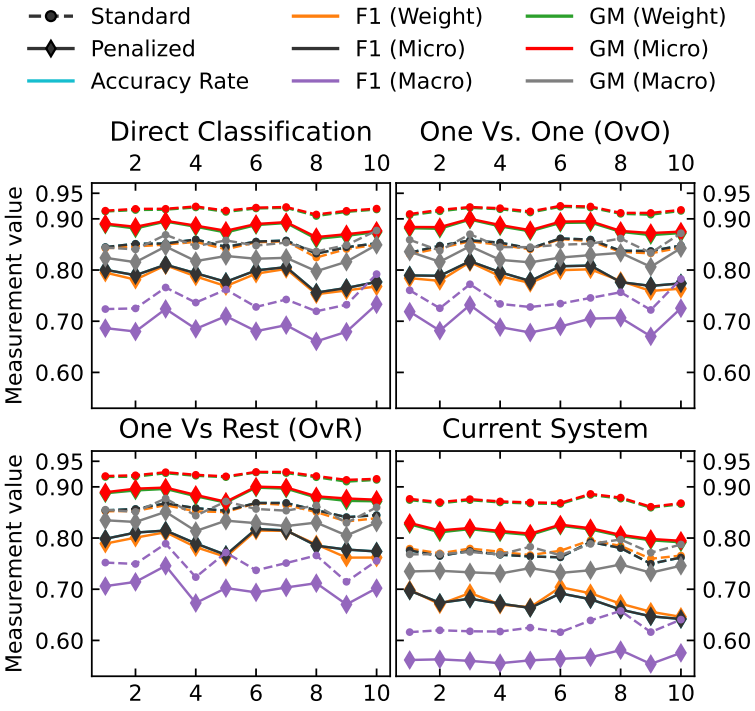


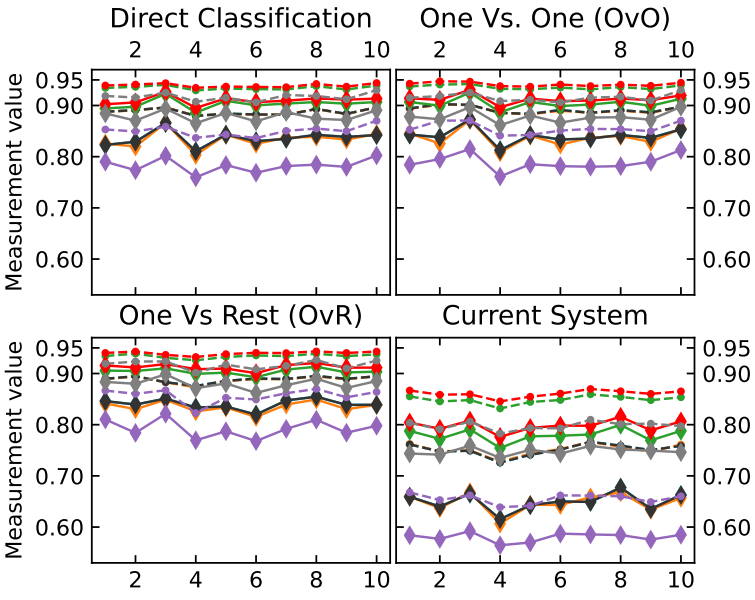
Figure 6.7: Running time over 500 function evaluations over 10 repetitions for two examined datasets.

evaluations. The box covers the first to the third quantiles. The horizon inner line shows the median. The whiskers show the fastest and the slowest execution. We can observe that the *direct method* is the fastest of the three experimental approaches. Notably, the average running time of the OvO and OvR on the "Top side camera" dataset is slower than the *direct classification* approach, approx 372% and 464%, respectively. In the same computation way on the "Bottom side camera", they are 811% and 643%. This is consistent with our presupposition, because the two decomposition approaches, i.e., OvO and OvR, convert the original multi-class dataset into multiple binary-class datasets, resulting in increased resource consumption for each iteration.

Figure 6.8 shows the results of 7 measurement metrics (i.e., F1 (weight), F1 (Micro), F1 (Macro), GM (Weight), GM (Macro), GM (Micro) and Accuracy rate) with and without penalization, by re-evaluating these metrics on the best-found ML pipelines once optimization processes are over. The results are shown for each of the ten runs performed. The dashed line with the dots marker shows the



(a) SIS top-side camera.



(b) SIS bottom-side camera

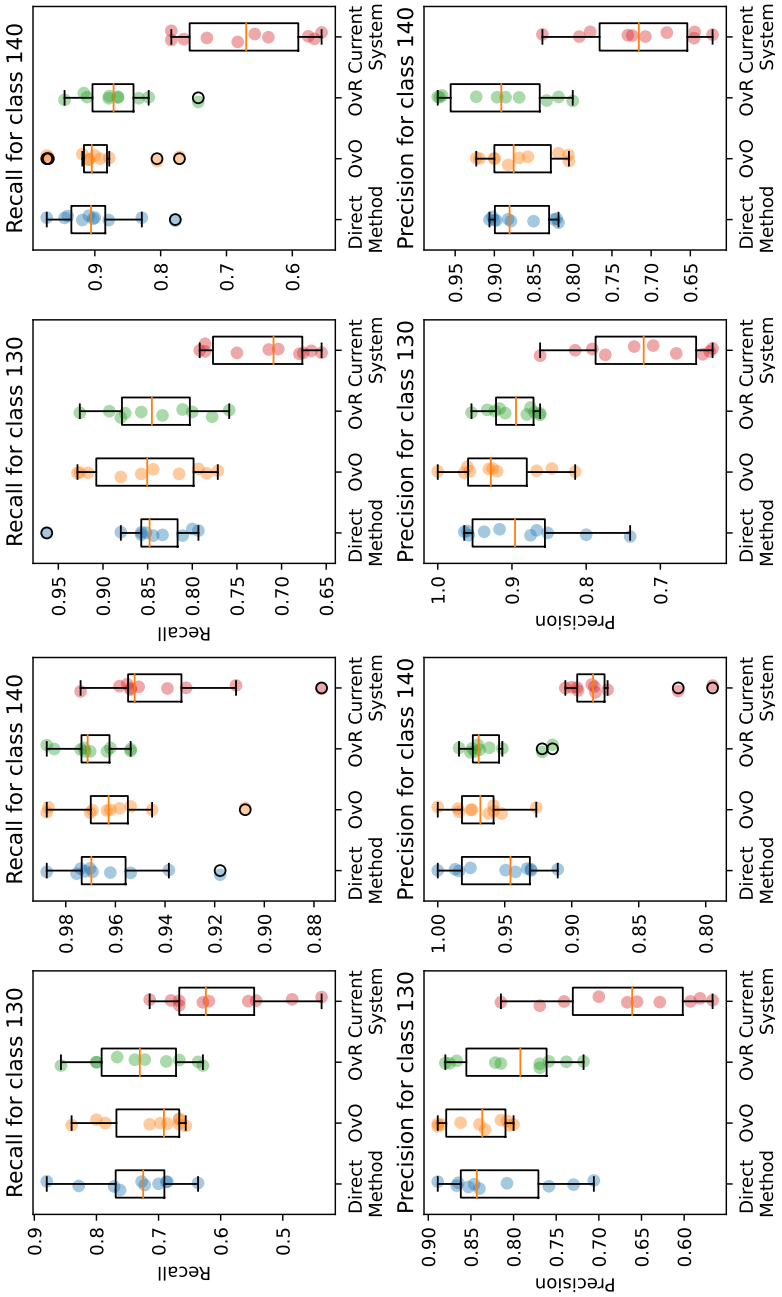
Figure 6.8: Comparisons between standard vs. penalized version of 7 measurement metrics on two datasets for three proposed vs. current approaches.

value of those standard metrics, i.e., without penalization. The solid line with the diamond marker shows those measurement metrics with our penalization approach. Penalized values are always lower than the corresponding standard values. A possible explanation is that the ML model misclassifies samples of the important classes. Lastly, we can observe that the penalized and the corresponding values of the used 7 measurement metrics are strongly correlated.

To investigate the predictive performance for the important classes (i.e., class 130 and 140), we provide Figure 6.9 to show recall and precision for those two important classes of 10 executions for the two tested datasets (i.e., Figure 6.9a for the top-side camera dataset and Figure 6.9b for the bottom-side camera dataset). Each box plot represents 10 repetitions. The box covers the first to the third quantiles. The horizontal inner line shows the median. The whiskers show the highest and the lowest values. The colour dots show the observed values, and the dots outside the whisker denotes the outliers. We can observe that the three CASH approaches produce better precision and recall scores for the two important classes on the two examined datasets.

6.4 Conclusion

In this chapter, we proposed an efficient approach to solving the steel surface defect classification, where the defect classes are (1) imbalanced and have (2) unequal importance. Firstly, we applied Bayesian Optimization (BO) to optimize the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem (i.e., the combination of resampling and classification algorithms, with their hyperparameter setting), to improve the classification performance for this class imbalance problem. Second, we propose a novel penalization approach to compute the classification performance metrics for unequal importance of classes. Based on our experimental results (Figure 6.6) and the running time analysis (Figure 6.7), we observed that the CASH approach clearly improves classification performance compared to the current classification system in use by TATA. Additionally, the direct classification method is much cheaper than the two experimented decomposition approaches (i.e., One vs. One and One vs. Rest) and not significantly worse than any of those in both test cases. Hence, we recommend to use the direct classification method to deal with similar problems. Lastly, the penalized performance metrics are strongly correlated to the standard metrics and



(a) SIS top-side camera.

(b) SIS bottom-side camera.

Figure 6.9: Recall and Precision for important classes over 10 repetitions per examined method.

efficient in measuring the important misclassified cases. Finally, future work will apply the proposed penalty approach to other industries.