Gerling, Alexander et al.

Article — Published Version

Comparison of algorithms for error prediction in manufacturing with automl and a cost-based metric

Journal of Intelligent Manufacturing

Provided in Cooperation with:
Springer Nature

Suggested Citation: Gerling, Alexander et al. (2022) : Comparison of algorithms for error prediction in manufacturing with automl and a cost-based metric, Journal of Intelligent Manufacturing, ISSN 1572-8145, Springer US, New York, NY, Vol. 33, Iss. 2, pp. 555-573, https://doi.org/10.1007/s10845-021-01890-0

This Version is available at:
https://hdl.handle.net/10419/307585

Journal of Intelligent Manufacturing (2022) 33:555–573
https://doi.org/10.1007/s10845-021-01890-0

Comparison of algorithms for error prediction in manufacturing with automl and a cost-based metric

Alexander Gerling[1,2,3] · **Holger Ziekow**[1] · **Andreas Hess**[1] · **Ulf Schreier**[1] · **Christian Seiffer**[1] · **Djaffar Ould Abdeslam**[2,3]

Received: 28 May 2021 / Accepted: 29 November 2021 / Published online: 3 January 2022

Abstract

In order to manufacture products at low cost, machine learning (ML) is increasingly used in production, especially in high wage countries. Therefore, we introduce our PREFERML AutoML system, which is adapted to the production environment. The system is designed to predict production errors and to help identifying the root cause. It is particularly important to produce results for further investigations that can also be used by quality engineers. Quality engineers are not data science experts and are usually overwhelmed with the settings of an algorithm. Because of this, our system takes over this task and delivers a fully optimized ML model as a result. In this paper, we give a brief overview of what results can be achieved with a state-of-the-art classifier. Moreover, we present the results with optimized tree-based algorithms based on RandomSearchCV and HyperOpt hyperparameter tuning. The algorithms are optimized based on multiple metrics, which we will introduce in the following sections. Based on a cost-oriented metric we can show an improvement for companies to predict the outcome of later product tests. Further, we compare the results from the mentioned optimization approaches and evaluate the needed time for them.

Keywords Hyperparameter optimization · Manufacturing · Metrics · Machine Learning · Production Line

Introduction

Machine Learning (ML) has been increasingly used in the area of manufacturing in recent years to predict errors (Caggiano et al., 2019; Hirsch et al., 2019; Li et al., 2019). An approach to use ML to optimize makespan in job shop scheduling problems can be found in (Dao et al., 2018). Deep learning methods for example are used to predict product quality with data from parallel (Zhenyu et al., 2020) or dynamic non-linear processes (Wang & Jiao, 2017; Yuan et al., 2020). Also, there are data-driven approach for complex production systems (Ren et al., 2020). With the further development of artificial intelligence, research for data-driven quality prediction methods are expanding in various fields (Kirchen et al., 2017; Liu et al., 2019; Tangjitsitcharoen et al., 2017). In manufacturing, especially quality engineers and data scientists analyse production errors using quality data. A review of problems and challenges of data science approaches for quality control in manufacturing is given in (Wilhelm et al., 2020). The paper explains manufacturing domain-specific challenges such as concept drift, diverse error types and cost-sensitive modelling. However, as ML

✉ Alexander Gerling
alexander.gerling@hs-furtwangen.de;
alexander.gerling@uha.fr

Holger Ziekow
holger.ziekow@hs-furtwangen.de

Andreas Hess
andreas.hess@hs-furtwangen.de

Ulf Schreier
ulf.schreier@hs-furtwangen.de

Christian Seiffer
christian.seiffer@hs-furtwangen.de

Djaffar Ould Abdeslam
djaffar.ould-abdeslam@uha.fr

[1] Business Information Systems, Furtwangen University of Applied Science, 78120 Furtwangen, Germany

[2] IRIMAS Laboratory, Université de Haute-Alsace, 68100 Mulhouse, France

[3] Université de Strasbourg, Strasbourg, France

techniques get more popular and tools mature, the applications become practicable even for non-experts. Specifically, Automated Machine Learning (AutoML) promises to make the application of ML more feasible and reduce the required level of expertise from the users. Various AutoML solutions emerged over the past few years (Candel et al., 2016; Feurer et al., 2019; Golovin et al., 2017; Kotthoff et al., 2019). These tools cover many steps of the data science pipeline, such as feature engineering or hyperparameter tuning for an algorithm. However, the heuristics of AutoML tools are generic and independent of a given domain. Hence, they may not be best tailored to find the solution in a particular use case, such as manufacturing quality management. Especially, if it comes to specific domain problems, like unbalanced class distribution, the state-of-the-art metrics can mislead the user to a wrongly chosen algorithm or parameter setting. Therefore, it is useful to test different metrics for a specific use case like in (Zhou et al., 2016). The performance of an algorithm is directly related to expenses and savings in the production process. PREFERML (Proactive Error Avoidance in Production through Machine Learning) (Ziekow et al., 2019) is a project that investigates challenges for the production and provides a holistic ML system solution in the context of error detection and error prevention in the production. Therefore, we deal with different and specific problems in the production in relation to data and adapt our system accordingly to this environment. At one point our system selects an algorithm that is optimized for the given production data. Since the data varies from one product to another, we assume that there is not one single optimal algorithm (Ho & Pepyne, 2002); however, we aim to provide a flexible tool that always supplies optimised results according to the provided data. The main objective is to reduce costs and save money based on our solution. In our case, we save costs by predicting a corrupted product part at an early stage. In addition, the system can provide hints for further investigations to the user. In this paper, we analyse the application of AutoML techniques using real manufacturing data from our project partner. We analyse the impact of using different mechanisms with twelve different classifiers. Further, we compare several metrics to improve the performance of our AutoML tool based on decision tree algorithms. Here the question arises, which metric shows a justification for the use of ML and saves costs in production. Therefore, we provide a metric which improves optimization results for the use in a manufacturing domain and demonstrate the effects in experiments with real world data. Moreover, we analyse to which extend hyperparameter tuning yields improvements to the results.

The following research questions (Q) emerge from the above description for our use case:

- (Q1) How good are results of established ML algorithms based on real production data?
- (Q2) How suitable are existing metrics for real production data?
- (Q3) How effective is hyperparameter tuning for improving cost–benefit?

The present paper is organized as follows: Sect. 2 a description of the domain and the specific challenges of our use case are given. In Sect. 3 describes the literature review. Fundamental knowledge about the used metrics is provided in Sect. 4. In Sect. 5 we explain a simplified illustration of our AutoML tool. Furthermore, we describe the used datasets as well as the hyperparameter for the AutoML tool. An algorithm comparison is given in Sect. 6. In Sect. 7 we visualize and evaluate the optimized results. Section 8 summarizes the content of this paper and shows the added value based on the answered research questions.

Domain description

In this section, we explain the production line and the specific challenges in this domain. A detailed explanation about the domain description and the production line structure are given in (Gerling et al., 2020). The goal of a manufacturing company is to produce as many flawless products as possible. These products are often built in a production line and get tested at test stations *T1* to *Tn*. There is a particular order between tests stations, where $Ta < Tb$ if *Ta* precedes *Tb* in the production process. With the data from test stations *Ta,* we aim to predict errors detected in test stations *Tb* with $Ta < Tb$. The objective for the company is to minimize the rate of errors in the production line—this becomes apparent by low error rates in the data. Usually, we have highly unbalanced datasets to train the ML model, because of the few errors in the production line. The number of errors in the data is a critical problem for the resulting performance of a ML model. If we check a product, it is not advisable to only check the final test station and to merge the data from all previous test stations. This merge procedure reduces the number of errors in the data. As described in (Hirsch et al., 2019) the number of instances *i1* to *in*, especially error instances, could be very low in the final test station. The corrupted tested devices are split into multiple error groups. Analysing a specific error group further reduces the number of recorded errors and makes the process of training a ML model even more difficult. By checking the results of the test station *Ta* towards test station *Tb* in the production process, we have the advantage to prevent errors at an early stage during the production. This will save costs across the entire production process and not only at the final test station. Through this type of classification, we can benefit from more usable data.

In Fig. 1 we illustrate an example of a production line with four test stations. Along the production process some parts

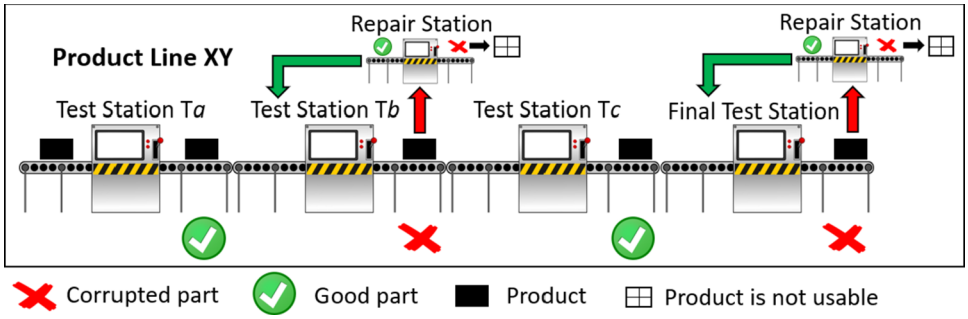

Fig. 1 Product in production line

turn out to be corrupted (illustrated as a corrupted product in Fig. 1). These corrupted product parts can be repaired in a separate process and returned to the production process. Corrupted product parts that cannot be repaired are not useable for production anymore. Data combinations of separate test stations prior to the final test station could be used, to identify productions errors. Every test station has multiple and generally different test features $f_1 to f_n$, which vary between different products and test stations. We use these test features to train our ML model. Pre-processing the data is required to remove unusable test features like text descriptions. By preference, only numerical features are left after pre-processing to train the ML model, because not all algorithms can handle for example text for the model training. For our experiments, we applied pre-processing to the quality data to make usable datasets for the ML training. To train a ML model with data, we have to choose a ML algorithm. At this stage we face the next challenge because a production quality engineer wants to understand why an algorithm has made a specific decision. Therefore, our system should be capable to explain the results. The rationale is to increase trust in the ML system as well as interpretability. The second requirement concerns the metrics to evaluate the results. A metric must be adjustable to a specific use case to provide a cost benefit to the user. Further, it should be possible to simply recognize a usable model for the prediction. When an ML system is applied, this does not automatically lead to a reduction of the costs. A metric that fulfils this requirement is presented in the next section.

With the mentioned preconditions, a result identified as a production error can be analyzed. A crucial point to use ML in the manufacture is the benefit of prediction explainability and visualizations. Therefore, we create visualisations where rules can be derived to correct or interpret the error message based on a single feature or of combinations features e.g., histogram or heatmap. Without any advanced tools, a quality engineer has just simple analytic tools e.g., mathematical tools with classical statistical functions. However, these are not sufficient to understand complex causes of an error and are limited in their ability to analyse them, especially regarding the number of features in the data.

Literature review

In this section, we describe related work which we have divided into the following groups: AutoML, Quality Prediction, imbalanced classification, and cost-based metric.

Literature for AutoML (Olson & Moore, 2019) explains an open-source genetic programming based AutoML system named TPOT. This tool automatically optimizes a series of ML models and feature pre-processors. The objective is to optimize classification accuracy on a supervised classification task. For a given problem domain (Olson et al., 2016) TPOT designs and optimizes the necessary ML pipeline without any involvement of a human being. To do so, TPOT uses a version of genetic programming—an evolutionary computation technique. With genetic programming, it is possible to automatically create computer programs (Banzhaf et al., 1998). TPOT uses similar algorithms for supervised classification as we do. The difference here is that we set our focus to the tree-based algorithms and do not use for example logistic regression. A major difference to TPOT is that we do not generate a code for further use. However, our holistic system has also a pre-processing and feature engineering pipeline.

In (Candel et al., 2016) an open-source ML tool named H20 gets described. Their objective is it to optimize ML for Big Data (Kochura et al., 2017). Because the H2O tool is fast and scalable, it is well suited for deep learning with specific algorithms. Additionally, this tool provides boosting and bagging ensembles and further supports algorithms to use. The H2O tool uses in-memory compression to handle a huge amount of data. A wide variety of program languages are supported. The H2O tool can be used as a standalone solution or together in a cluster solution. The developers of this tool collaborate with industrial partners and other research institutions. This tool has already been deployed in different domains and can be used by a wide variety of users with the Flow web-based GUI. Our AutoML tool solution differs here, because we provide a specific production domain solution with adjusted functionality. A key point to mention is the adjusted metric for the production and the strong support for the tasks of the quality engineer.

(Krauß et al., 2020) shows possibilities and limits of applying AutoML in production. Further, it includes an evaluation of available systems. Moreover, a comparison of AutoML and a manual implementation from data scientists in a predictive quality use case was held. At the moment, AutoML still requires programming knowledge and was outperformed by the implementation of the data scientists. One specific point was the preparation of the needed data. Without predefined domain knowledge, an AutoML system cannot merge the data correctly. Additionally, the integration of the data or the extraction from a database is problematic. A solution to this could be in form of an expert system. A further point was the deployment of the results or the models for the end-user. In conclusion, it can be said that AutoML systems provide the chance to increase the efficiency in a ML projects. This could be done by automating the necessary procedure within Data Integration, Data Preparation, Modelling and Deployment. The expertise of a data scientist and domain knowledge should be included to obtain satisfying results. Nevertheless, the latest developments provide indicators for future improvements towards the automation of specific steps within the ML pipeline.

Several approaches to automate machine learning appeared in recent years. (Maher et al., 2019) describe a meta learning-based framework for automated selection and hyperparameter tuning for ML algorithms (SmartML). SmartML has a meta learning feature that emulates the role of a domain expert in the field of ML. They use a meta learning mechanism to select an algorithm to reduce the parameter-tuning search space. The SmartML provides a model interpretability package to explain their results. The SmartML tool differs significantly from our approach. With our AutoML tool we pre-process the data with the provided background information of a product. This could be done with the aid of a quality engineer or a ML expert. Furthermore, we only use decision tree-based algorithms to provide human recognizable and acceptable decisions. This allows us to gain confidence in the given results. We only use fast and simple algorithms for our experiments. A critical difference is that SmartML is not specialized on manufacturing data. Crucial points, like highly unbalanced data or the selection of a specific metric are not supported. This is where our AutoML tool differs from existing solutions and provides a specialized solution for manufacturing data.

In (Golovin et al., 2017) the black-box optimization tool Google Vizier is described. This tool has become the de facto parameter tuning engine for Google and is used as an internal service for performing black-box optimization. This tool supports various algorithms and uses them for training. The results are saved in a persistent database for the purpose of transfer learning. Vizier uses also RandomSearchCV (Bergstra & Bengio, 2012) and GridSearchCV (Worcester, 2019) to optimize the hyperparameter for algorithms. In (Golovin et al., 2017) the authors do not present in detail which algorithms could be used to train a ML model and it seems to be a general solution to optimize algorithms. In comparison to our approach, we already provide pre-selected decision tree-based algorithms and offer a use case specific metric. Thus, we provide a more specific solution for the manufacturing domain.

AUTO WEKA (Thornton et al., 2013) is an open-source automation framework for algorithm selection and hyperparameter optimization based on Bayesian optimization using sequential model-based algorithm configuration (SMAC) and Tree-structured Parzen Estimator (TPE) (Kotthoff et al., 2019). The target user group of AUTO WEKA are not only experts, but also novice users in the field of ML. Some parts of AUTO WEKA overlap with our approach. We also support the use of novice users with our tool. One of our target groups are quality engineers. This target group has often no experience with ML at all. For the optimization task, we also use Distributed Hyperparameter Optimization (HyperOpt, 2020) with TPE as an estimator. Additionally, we provide the possibility to optimize the algorithm with the RandomSearchCV approach. With AUTO WEKA it is possible to create individual metrics and use them for the evaluation. Here AUTO WEKA could take advantage of our metric and implement it, to provide a manufacturing adjusted metric. A point where we differ from AUTO WEKA is the number of classifiers, we provide with our AutoML tool. For our tool, we only use tree-based algorithms to support the information value of the results.

Literature for quality prediction In (Sankhye & Hu, 2020) the objective was to design machine learning based classification methods for quality compliance. Afterwards, a validation of the models via case study of a multi-model appliance production line was shown. In this case study, the proposed model for the classification could achieve a Cohen's Kappa (Cohen, 1960) of 0.91 and an accuracy of 0.99 for the compliance quality of unit batches. The main objective is the implementation of a predictive model for compliance quality, which could be enabled with the proposed method. Another aspect of this paper was to emphasize the importance of dataset knowledge and feature construction within the training of the classification models. In this work two algorithms, namely RandomForest and XGBoost, was used but the second algorithm achieved better results. Further the Cohen's Kappa metric was used to tackle the imbalanced dataset problem. In this work, a classical machine learning approach without AutoML is used. The disadvantage here is that only two algorithms are used for classification. What is more important, however, is the selection of the metric. In (Delgado & Tibau, 2019), the author compared the Matthews correlation coefficient (MCC) (Matthews, 1975) and Cohen's Kappa and concluded that Cohen's Kappa should be avoided as a performance measure for classification.

In (Zonnenshain & Kenett, 2020) they discuss the challenges for quality engineering in the future. Moreover, they consider the future directions for quality and reliability engineering. This is done in the context of how opportunities from Industry 4.0 could be used. The paper shows how important data has become for quality engineering and the evolution of quality models. Moreover, they describe quality as a data driven discipline.

Literature for imbalanced classification In (Kim et al., 2018) investigates the case of imbalanced classification of manufacturing quality conditions by using several cost-sensitive decision tree ensembles. A real-life die-casting data set was used to compare the various classifiers. In this work, the authors had to deal with strong unbalanced data, which demonstrates the need to allocate costs to different classes. To do so, three cost-sensitive ensembles based on a decision tree algorithm were selected, namely AdaC1, AdaC2 and AdaC3. Those had to compete with 19 different algorithms. As results, the AdaC2 algorithm could provide the best results. To compare the algorithms the accuracy, balanced accuracy, precision, recall, F-measure, G-Mean and AUC were chosen as performance metrics. This work is similar to ours in terms of the highly unbalanced data and the approach to assigning costs to the different prediction classes. Our AutoML tool also uses decision tree-based algorithms for the fast and understandable prediction. We distinguish ourselves by the use of high-performance decision tree-based algorithms. Further, we optimize the prediction by using a cost-sensitive metric.

In (Moldovan et al., 2017) the authors investigated state-of-the-art approaches for using ML techniques on the SECOM dataset. This dataset contains data from a semiconductor manufacturing process and therefore represents an unbalanced real-world dataset. Based on this dataset, three different feature selection methods were used. Further, the performance of three sample classification algorithms was compared. To measure the performance of the classifier, the F-measure, recall, False-Positive-Rate, precision and accuracy metrics was used. This paper only focuses on the SECOM dataset and therefore, provides only a process to achieve the best results specifically for this dataset. In our work, we focus on the ML part and the optimization of the algorithm. Further, we use various manufacturing datasets for our experiments. We also consider the unbalanced data problem and tackle this by adjusting the hyperparameter within the optimization.

Literature for cost-based metric One of the first and best-known paper regarding MetaCost and cost-sensitive classifier is presented in (Domingos, 1999). With this approach, the classifier will be adjusted for the different costs of errors. This procedure is well suited for imbalanced datasets to assign a cost value to the different classes. With this approach a large cost reduction compared to cost-blind classifiers can be achieved. The cost matrix for a two-class classification is set to C(0,0) = C(1,1) = 0; C(0,1) = 1000; C(1,0) = 1000r, where r was set alternately to 2, 5, and 10. In this case C(0,1) and C(1,0) are only relevant because of the ratio r. Number 0 represent the minority class and 1 shows the majority class. In our approach, we set our focus on the True Positive (TP) and False Positive (FP) respectively C(0,0) and C(0,1) and use a factor alpha to adjust the real cost savings (see the equation for Expected Benefit Rate Positives Only (EBRP) in the subsequent section).

In (Loyola-González et al. 2019) a proposal of an algorithm for discovering cost-sensitive patterns in class imbalance problems was given. Further, this pattern is used for classification with a pattern-based classifier. This proposal can obtain cost-sensitive patterns, which leads to lower misclassification costs in comparison to patterns mined by well-known state-of-the-art pattern miners. In the approach of (Loyola-González et al., 2019), the cost matrices are adjusted to C(0,0) = C(1,1) = 0, which means that the resulting costs of True Negative (TN) and TP are 0. The costs of FP are set consecutively to C(0,1) = 2, 5, 10, 20 and further propose to use the imbalance ratio of training database as a cost. The costs of False Negative (FN) is set to C(1,0) = 1. To control the results from the classifier, they use a normalized expected cost (NEC) (normalized misclassification cost (Drummond & Holte, 2006)), here given in the original full form:

$$\mathrm{NEC} = \frac{\mathrm{TP} * \mathrm{C}(0,0) + \mathrm{FP} * \mathrm{C}(0,1) + \mathrm{FN} * \mathrm{C}(1,0) + \mathrm{TN} * \mathrm{C}(1,1)}{\mathrm{D}p * \mathrm{C}(0,0) + \mathrm{D}p * \mathrm{C}(0,1) + \mathrm{D}n * \mathrm{C}(1,0) + \mathrm{D}n * \mathrm{C}(1,1)} \quad (1)$$

Dp and Dn are the numbers of instances from the minority (Dp) and majority class (Dn). All possible costs were taken into account in this approach and got represented by a normalized value. We can use the NEC formula as a basis for our special use case. In our work, we care about the True Positives and False Positives because this shows the difference to the as-is situation for companies without ML in their production. Further, we use the place holder alpha for the real costs, which can be adjusted by the cost value of a product. Additionally, in our approach we consider the specific use case of the production domain.

Metrics

In this section, we explain the metrics used for our experiments in Sects. 6 and 7. We built an AutoML tool that independently chooses one of five different algorithms. These five algorithms are the DecisionTree, RandomForest, XGBoost, CatBoost and LightGBM Classifiers. Additionally, we use five metrics to further improve the results with hyperparameter tuning. As first metric to mention is the MCC

(Boughorbel et al., 2020; Shmueli, 2019). The MCC value gets calculated by:

$$\mathrm{MCC} = \frac{\mathrm{TP} * \mathrm{TN} - \mathrm{FP} * \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(TN+FN)}} \quad (2)$$

Based on the MCC we measure the quality of a binary classification. The MCC metric is a balanced measure method that takes TP, FP, TN and FN into account. In our case a TN is a good part predicted as good part and a FN is a corrupted part predicted as a good part. A TP is a corrupted part predicted as corrupted part and a FP is a good part predicted as a corrupted part. The MCC is used for different class sizes which is necessary in our case (Boughorbel et al., 2017).

For the second and third metric, we use the Area Under the Curve (AUC) calculation for the Receiver Operating Characteristic (ROC) and Precision Recall Curve (PRC). The ROC AUC metric is suitable for balanced classes. At first sight, the ROC AUC metric is not helpful at all because the classes are extremely unevenly distributed. Nevertheless, to handle the unbalanced distribution we use a hyperparameter named'class_weight' respectively'scale_pos_weight'. Depending on the setting of this hyperparameter it considers the classes as equal and changes the training of the model. The PRC AUC metric is also suitable for a binary classification. Moreover, this metric looks at the positive class; in our case this is the prediction of an error. Therefore, the PRC AUC metric provides us with a value of how good it predicts the minority class. The AUC represents the performance of a binary classification (Sarath, 2018). The fourth metric is a calculation to predict the cost savings for each positive predicted forecast named EBRP:

$$\mathrm{EBRP} = \frac{\mathrm{TP} * \alpha - \mathrm{FP}}{\mathrm{TP}+\mathrm{FP}} \quad (3)$$

Symbol α (alpha) represents the cost saving factor or ratio of how much a correctly identified error in relation to an incorrectly identified error will save us and is therefore a placeholder for the real costs of a product or product part. Moreover, the user of this metric can adjust the α parameter with the ratio of the cost saving factor.

The fifth metric is similar to the fourth metric, a calculation to predict the cost savings for all predictions. EBR (Expected Benefit Rate):

$$\mathrm{EBR} = \frac{\mathrm{TP} * \alpha - \mathrm{FP}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}} \quad (4)$$

The EBRP and ERB are our own cost-based metrics, which are based on the cost formula of (Domingos, 1999). Within both metrics, the counter represents the absolute savings or our expected benefit. We obtain the total costs by TP*C(0,0) + FP*C(0,1) + FN* C(1,0) + TN*C(1,1). Here the individual parts represent C(0,0) = α, C(0,1) = -1, C(1,0) = 0, C(1,1) = 0, which leads us to the counter TP* α—FP. The denominator is used to normalise the savings by dividing the counter by all instances for the EBR metric and only the positive predictions for the EBRP metric. A key point to mention is that we want to quantify the benefit to use ML for the production. For the as-is situation without a ML system, we do not have the chance to use the potential to save costs in the production. To express a positive result, we use the terms of cost savings or benefit. Costs are negative cost savings or a negative benefit. By using a ML system, a corrupt product could be detected, which reduces the costs for the company. This would change a FN to a TP. To explain it more simply, we would correctly predict a corrupted part that would otherwise proceed further in the production line. A FP produce costs but usually less in comparison to a TP, which we express with α. However, dependent on the production process, these costs may be less than the savings of a TP, which we express with α as well. An important aspect is that a TP still produces costs, but we cannot influence these costs. A further point is that the EBRP and EBR metric are intuitive with respect to the usability of the model. Values above zero indicate that a reduction of costs could be realized with those metrics. This is a crucial benefit in comparison to other metrics. Moreover, no extended tests must be executed to show a benefit from a model in comparison to a baseline.

In this paper we analyse which of the above introduced metrics is most suitable for a manufacturing use case. A crucial factor for a manufacturing case is the cost saving for correctly predicted fault devices. Even if the quality of a ML model is not great, it could save money by these criteria. We focus on hyperparameter that assign weights to classes ('class_weight' and'scale_pos_weight' in the used libraries), as they are very important for our experiments. Because we used unbalanced data to train the ML model, it simplifies the handling of the data. Therefore, we do not have to handle this issue explicitly when preparing the data.

Automl tool and setup

In this section, we describe the workflow of our AutoML tool. Followed by an explanation of the datasets and their class distribution. Afterwards we describe the setup used for our experiments and the associated hyperparameters.

Workflow

In Fig. 2 we illustrate the workflow in our tool. As first step, we read and clean the data e.g., from missing values. The next step is to prepare the data by removing unnecessary features e.g., features with the same value and check the data format. Followed by creating new derived features. This should be carried out with the help of domain

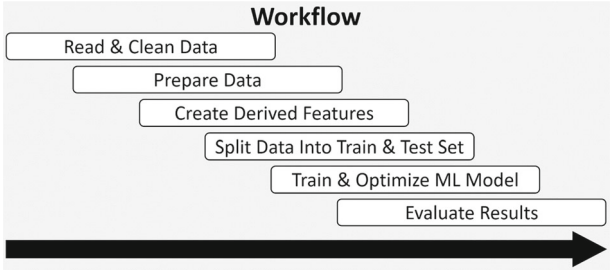

Fig. 2 Simplified Workflow

knowledge to derive product-specific features. For example, product features which represents electronic components can be grouped. This should lead to a better performance. However, an empirical evaluation of the benefits of domain knowledge is out of scope of this paper and is deferred to future work. The domain knowledge can be stored in a PTMD as described in (Gerling et al., 2020). In the following step, we split the data in a predefined percentage split into a train and test set. The following step is the training and optimization of the ML model by using hyperparameter tuning. At the last step, we evaluate the results and check created visualizations.

Dataset and class distribution

Table 1 shows the used datasets and the class distribution. Glass1 and Yeast-0–2–5–6 (Alcalá-Fdez et al., 2011) are unbalanced open-source binary datasets and we use them to check the quality of a classifier with an unbalanced dataset. UCI SECOM (Dua & Graff, 2020) is a real-world dataset and is more relevant for us than Glass1 or Yeast-0–2-5–6 because it contains data from a semi-conductor manufacturing process. The features from the UCI SECOM dataset are e.g., signals or variables collected from sensors and process measurement points in the manufacturing process (Dua & Graff, 2020). For our purpose's dataset Line A, Line B, Line C, Line X, Line Y and Line Z are especially interesting. These datasets represent a classification of station T*a* to station T*b* in a production chain from our partner SICK AG (SICK AG, 2020). Line X is particularly interesting here because it shows a data chain over several test stations with 1071 features. Further, it shows how highly unbalanced the datasets in the real production can be. Dataset Glass1 has here the lowest imbalance ratio with 0.550725 and Line Z the highest with 0.002103. Due to the strong imbalance in Line Z, which represent just a few errors in comparison to good products, the results are particularly interesting. All the used datasets contain only numerical features. A failure instance is represented as class 1 or 1 in the dataset and a flawless instance is be represented by class 0 or 0. A row corresponds to an instance and columns represent the features in the dataset. The column'IR (Class 1/Class 0)' in Table 1, shows how strongly the classes in the dataset are unbalanced. Moreover, these datasets were chosen randomly from a dataset pool to have an overview of possible results for this kind of data.

Test setup (machine hardware, software versions)

For our experiments we used a machine with Windows 10 64Bit. The test system has an i9-9999KS (16×4 GHz) processor and 64 GB RAM. We used the Anaconda Distribution with Numpy Version: 1.18.1, Pandas Version: 1.0.1, Scikit-learn Version: 0.22.1, Catboost Version: 0.21, Lightbgm Version:2.3.0, Py-xgboost Version 0.90, HyperOpt Version 0.2.2 and Python 3.7.6. All experiments shown were executed on the CPU.

Data preparation for training

The data was cleaned of nan values and the format of the data was adjusted. For the classification, we set the split for the training set to 67% of the total amount of errors in the data. Therefore, we have constantly 33% of the total amount of errors to validate the quality of the ML model. Additionally, we use a cross validator with shuffle mode and five splits for the training set.

Table 1 Datasets

Dataset	Class 0	Class 1	Instances	Features	IR (class 1/class 0)
Glass1	138	76	214	9	0.550725
Yeast-0–2–5–6	905	99	1004	8	0.109392
Line A	57,499	530	58,029	63	0.009218
Line B	7127	73	7200	16	0.010243
Line C	88,928	553	89,481	22	0.006219
Line X	19,692	102	19,794	1071	0.00518
Line Y	89,204	687	89,891	16	0.007701
Line Z	190,183	400	190,583	19	0.002103
UCI SECOM	1463	104	1567	592	0.071087

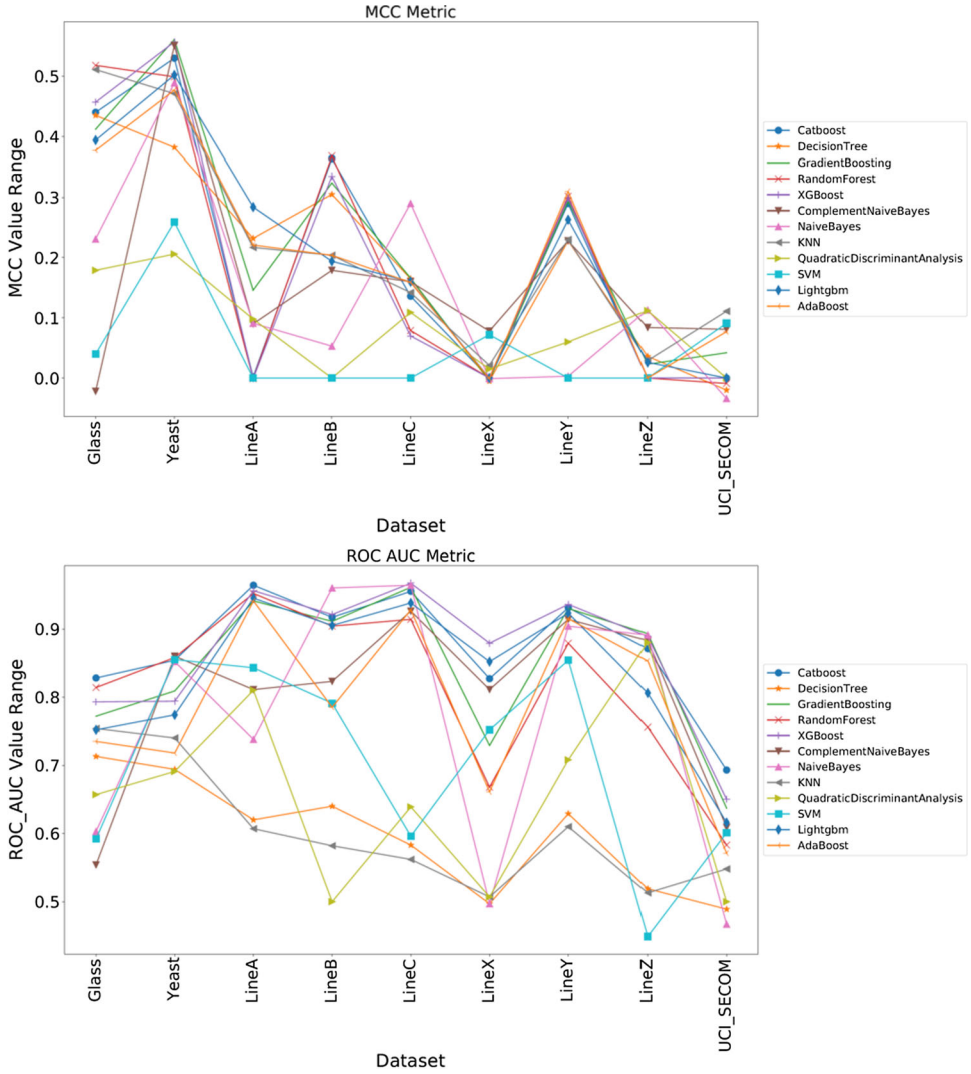

Fig. 3 Algorithm comparison part 1

Hyperparameters

In the appendix (Table 8) we summarise up all hyperparameters with the explanation that we used for the different decision tree-based algorithms. Not all classifiers use the same parameters and hence we show an accumulated table with the used hyperparameters in Table 8. In Table 9 we show the used hyperparameters with the value range for the associated algorithms. All test runs were performed with the maximum amount of CPU threads via the adjustment of the n_jobs, thread_count or thread parameter depending on the algorithm. Only the Decision Tree algorithm could not be adjusted, because there is no such parameter. We obtained the explanations from the official websites (Decision Tree Classifier, 2020; Random Forest Classifier, 2020; XGBoost, 2020; CatBoost, 2020; LightGBM, 2020).

Parameter optimization

We used RandomSearchCV and HyperOpt as search approaches for the experiments. RandomSearchCV is a state-of-the-art approach for hyperparameter optimization. To optimize the parameter of a model, RandomSearchCV uses a random set of parameters from the given parameter range. HyperOpt use the Bayesian optimization method and is created for large-scale optimization for models with several parameters. The Bayesian optimization method should find better model settings than the random search method in less iterations, by rating the hyperparameter that appear more promising from past results. For our experiments we used the TPE estimator for HyperOpt, which is one of three implemented algorithms.

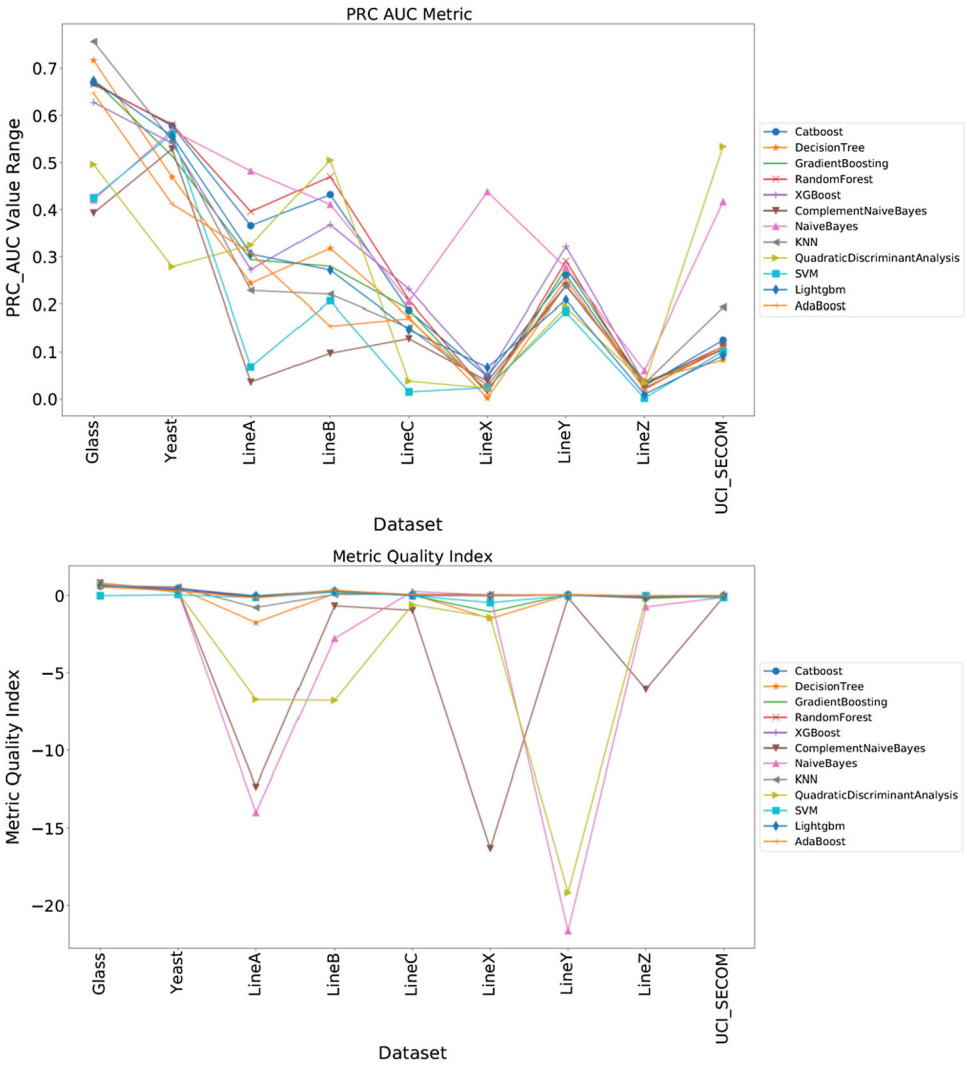

Fig. 4 Algorithm comparison part 2

Algorithm comparison

In this section we visualize the results based on the implementation of twelve algorithms in Figs. 3 and 4. The objective of this comparison is to get an overview, how good the classifiers perform with hardly any adjustments. Those results are used as a baseline for the metric comparison. To get comparable results from the different algorithm, we had to adjust the parameters for k-nearest neighbors (KNN) hyperparameter'n_neighbors' 1, Support Vector Machine (SVM) hyperparameter'kernel' linear.

Table 2 shows the best and worst achievable classification results regarding the used dataset. The best case is calculated by P *α and the worst case is represented by -N. The value P represents the number of corrupted parts (TP + FN) and the value N the number of good parts (TN + FP) in the test data. These values are corresponding to the used dataset. To calculate the Metric Quality Index (MQI) we use:

Table 2 Best- and worst-case classification results

Dataset	Best case classification result	Worst case classification result
Glass1	260	− 41
Yeast-0–2–5–6	340	− 218
Line A	1760	− 54,328
Line B	250	− 1977
Line C	1830	− 31,764
Line X	350	− 9792
Line Y	2880	− 52,086
Line Z	1330	− 90,533
UCI SECOM	350	− 706

$$\mathrm{MQI} = \frac{\mathrm{TP} * \alpha - \mathrm{FP}}{\mathrm{P} * \alpha} \tag{5}$$

The MQI represents a normalized value to compare the results of optimizations. With the MQI, we can compare the

results of different metrics in a single visualization or table. The best possible result for this column is 1.0 and a result above 0.0 will save costs. We assume that a correctly predicted corrupted product part (TP) is worth 10 times more than a good product that must be tested in a separate test station again costs (FP) (i.e., the savings enable by a true positive are assumed to be 10 times higher than the cost resulting from a false positive). We set α to 10 but this value must be adjusted for every product. The value 10 is a common recommendation for the cost matrices in other works (Domingos, 1999; Du et al., 2007; Krętowski & Grześ, 2006; Zhang et al., 2007).

The comparison of Figs. 3 and 4 answer the question (Q1) how good the results of established ML algorithms are based on real production data. Table 3 summarizes the results from Figs. 3 and 4. In these two figures, we show which results can be achieved with the different algorithms and get an overview of their performance. As indicated in the introduction, the results have shown that there is no single optimal algorithm. Therefore, it is necessary to compare different algorithms with each other for every dataset. An abnormality in the results can be seen using the MCC and ROC AUC metric. The result from the ROC AUC metric with the Catboost classifier in Line X implies a good result with the value of 0.814. This would lead us to the assumption that this specific model could be used to predict product errors. When considering the outcome of the MCC metric, we see that this result is misleading. The value of the MCC metric (− 0.001) seems like a random outcome. We would not use a model that has an MCC value below 0. This effect results from the strong unbalanced classes in the dataset. ROC AUC and PRC AUC are particularly affected by this effect. Therefore, it is necessary to find an adjusted metric for our use case. The Naive Bayes and the Complement Naive Bayes algorithms produce strong negative outliers within the MQI Metric. Further, the SVM algorithm performs poorly over all datasets with most of the metrics. Nevertheless, we show a benefit to use ML in this use case with the results from the MQI metric. Therefore, ML could already be used to predict production errors. Another aspect that can be observed is that the decision tree-based algorithms are often close in terms of results and provide reasonable results. For new data we suggest to use a simple decision tree algorithm to get a first impression of the possible results and error cause(s).

Metric comparison for hyperparameter tuning

In this section, we first show insights into results from a test with RandomSearchCV. Further, we show the average optimization time for RandomSearchCV and HyperOpt by all executed runs. Moreover, we take a look at the used hyperparameters of a run execution to compare them. Afterwards, we show a comparison of all optimized results.

We optimized with RandomSearchCV and HyperOpt for maximal EBR, EBRP, MCC, ROC AUC and PRC AUC. For each approach we checked all metrics, MQI and computation time. Table 4 represents the results of the AutoML tool with RandomSearchCV approach and an initial seed. The same seed was used for all results in Table 4. The name of the algorithm for the best result is abbreviated as follows in the Name column: RFC = Random Forest Classifier, CBC = Cat Boost Classifier, DTC = Decision Tree Classifier, XGBC = XGBoost Classifier, LGBMC = LGBM Classifier. The computation time for a program execution is illustrated in the column'ComTime with format h:mm:ss.

In all our experiments, the models that were obtained through optimizing for EBR perform well compared to results with other metrics. That is, the EBR metrics yields scores that are either better or similar to the scores of the best alternatives. Our provided metrics EBR and EBRP leads to most of the best MQI results. The results also illustrate the effect of negatives monetary effects that could results from using the wrong metric for optimization. In Line X we show a MCC value of 0.198 (MCC metric). However, more important are the increasing costs in this specific case, leading to a MQI of − 0.354. To compare, our metric EBR shows a MCC value of 0.118 but reduces the costs by a MQI of 0.025. Even more crucial is the comparison in Line Z. With the EBR metric we get a neutral result. On the contrary the MCC metric would increase costs dramatically by a MQI of -2.888. This example shows the necessity of our EBR and EBRP metrics. Moreover, we show how suitable existing metrics for real production data are (Q2) and present a much better metric for our specific use case.

The next point is the needed time to improve the algorithm with hyperparameter tuning. Table 5 shows the average time for a program execution with the associated dataset. RandomSearchCV is the fastest competitor in this comparison, but the needed time for an execution with HyperOpt is still acceptable regarding the cost savings that result from the optimization. The maximum amount of time for an execution took average 50 min in Table 5. This is due to the high number of features in dataset X. Nevertheless, the measured times are close to each other.

The visualized results in Table 6 also hold further information with regards to the reusability of hyperparameters. We take a closer look at the results of Line B, Line C, Line X and UCI Secom from the EBRP Metric in Table 4. These datasets created the best results using the Random Forest Classifier. The question rises if the same classifier has used the same hyperparameters to produce the result. Table 6 shows the used hyperparameters for the mentioned datasets:

The used hyperparameters shown in Table 6 indicate that it is necessary to optimize the hyperparameters individually

Table 3 Algorithm comparison

Dataset	Catboost	Decision Tree	Gradient boosting classifier	Random Fast	XGBoost	Complement naive bayes	Naive bayes	KNN	Quadratic discriminant analysis	SVM	Lightgbm	AdaBoost
MCC metric (MCC value in table)												
Glass 1	0.584	0.560	0.617	**0.618**	0.497	0.302	0.382	0.497	0.404	0.000	0.443	0.518
Yeast-0–2–5–6	**0.615**	0.500	0.547	0.567	0.591	0.538	0.433	0.614	0.392	0.226	0.486	0.421
Line A	0.035	0.024	0.004	− 0.004	− 0.001	0.049	0.042	0.039	**0.070**	− 0.004	0.062	0.039
Line B	0.492	0.215	0.441	0.451	**0.502**	0.157	0.109	0.159	0.015	0.488	0.439	0.476
Line C	0.131	0.167	0.068	0.077	0.043	0.162	**0.310**	0.180	0.191	0.000	0.111	0.161
Line X	− 0.001	0.001	0.005	0.000	0.000	− 0.001	**0.251**	0.146	− 0.014	0.050	0.000	0.097
Line y	0.190	0.127	**0.214**	0.201	0.165	0.199	0.000	0.132	− 0.008	0.000	0.146	0.169
Line Z	0.000	0.033	− 0.001	0.000	0.000	0.084	0.123	0.015	**0.139**	0.000	0.005	0.000
UCI SECOM	0.000	0.037	− 0.030	− 0.008	0.000	− 0.041	− 0.019	− 0.033	− 0.008	− 0.042	0.000	**0.051**
ROC_AUC metric (ROC_AUC value in table)												
Glass 1	**0.899**	0.780	0.864	0.880	0.880	0.681	0.797	0.749	0.791	0.221	0.880	0.823
Yeast-0–2–5–6	0.847	0.737	0.819	0.837	0.830	**0.862**	0.839	0.778	0.834	0.830	0.800	0.764
Line A	0.816	0.551	0.817	0.765	0.804	0.767	0.691	0.559	0.767	0.714	**0.818**	0.807
Line B	0.928	0.596	0.891	0.917	0.935	0.832	0.946	0.557	0.824	**0.962**	0.936	0.902
Line C	0.955	0.588	0.962	0.919	**0.968**	0.918	0.968	0.578	0.924	0.474	0.953	0.959
Line X	0.814	0.501	0.715	0.582	0.744	0.558	**0.831**	0.570	0.475	0.754	0.772	0.815
Line y	0.870	0.570	**0.914**	0.842	0.910	0.905	0.809	0.554	0.462	0.849	0.895	0.867
Line Z	0.860	0.525	0.892	0.806	0.910	0.902	**0.910**	0.507	0.874	0.315	0.835	0.855
UCI SECOM	0.559	0.525	0.469	0.481	0.527	0.446	0.471	0.489	0.499	**0.560**	0.445	0.434
PRC_AUC metric (PRCC_AUC value in table)												
Glass 1	**0.864**	0.783	0.789	0.853	0.837	0.536	0.628	0.752	0.624	0.258	0.853	0.762
Yeast-0–2–5–6	0.674	0.596	0.619	0.647	0.661	0.665	0.622	**0.692**	0.630	0.602	0.539	0.544

Table 3 continued

Dataset	Catboost	Decision Tree	Gradient boosting classifier	Random Fast	XGBoost	Complement naive bayes	Naive bayes	KNN	Quadratic discriminant analysis	SVM	Lightgbm	AdaBoost
Line A	0.043	0.088	0.031	0.012	0.041	0.009	**0.424**	0.083	0.392	0.006	0.047	0.033
Line B	0.491	0.230	0.273	0.502	0.492	0.121	0.499	0.181	0.425	**0.647**	0.520	0.500
Line C	0.187	0.175	0.191	0.205	0.211	0.107	**0.222**	0.189	0.132	0.005	0.134	0.162
Line X	0.014	0.032	0.007	0.005	0.018	0.004	**0.391**	0.151	0.002	0.014	0.014	0.077
Line y	0.175	0.133	0.206	0.218	0.209	0.113	0.205	0.140	**0.351**	0.111	0.146	0.154
Line Z	0.011	0.039	0.026	0.026	0.039	0.046	**0.067**	0.017	0.039	0.001	0.009	0.023
UCI SECOM	0.053	**0.127**	0.043	0.043	0.072	0.039	0.065	0.024	0.024	0.052	0.073	0.046
MQI; formula = ((TP*α) − FP)/best case result [α = 10]; (MQI in table)												
Glass 1	0.638	0.704	0.677	0.642	0.662	**0.819**	0.804	0.662	0.777	0.000	0.654	0.562
Yeast-0–2–5–6	0.491	0.494	0.429	0.432	0.462	0.350	0.285	**0.568**	0.203	0.059	0.394	0.309
Line A	− 0.026	− 1.751	− 0.068	− 0.132	− **0.004**	− 12.346	− 13.996	− 0.770	− 6.713	− 0.127	− 0.007	− 0.136
Line B	0.276	0.140	**0.360**	0.236	0.312	− 0.660	− 2.768	0.080	− 6.772	0.240	0.300	0.340
Line C	0.037	0.89	0.014	0.014	0.006	− 0.946	**0.280**	0.101	− 0.579	0.000	0.039	0.076
Line X	0.006	− 1.486	− 1.046	0.000	0.000	− 16.311	0.040	**0.066**	− 1.406	− 0.454	0.000	0.023
Line y	0.061	0.037	**0.080**	0.058	0.041	− 0.129	− 21.647	0.055	− 19.175	0.000	0.045	0.057
Line Z	**0.000**	− 0.168	− 0.020	**0.000**	**0.000**	− 6.056	− 0.727	− 0.071	− 0.197	**0.000**	− 0.130	− 0.001
UCI SECOM	0.000	− 0.043	− 0.097	0.003	0.000	− 0.069	− 0.109	− 0.046	− 0.003	− 0.137	0.000	**0.014**

Bold values indicate the best result for a dataset

Table 4 Random search CV with initial seed results

Dataset/algorithm and Metric	Random search CV (expected benefit rate)						Random search CV (expected benefit rate Pos.)					
	MCC	ROC_AUC	PRC_AUS	ComTime	MOI	Algorithm	MCC	ROC_AUC	PRC_AUS	ComTime	MOI	Algorithm
Glass 1	0.390	0.880	0.823	0:00:21	**0.892**	RFC	0.652	0.890	0.855	0:00:21	0.750	XGBC
Yeast-0–2–5–6	0.495	0.851	0.630	0:00:24	**0.561**	RFC	0.539	0.795	0.592	0:00:22	0.502	XGBC
Line A	0.185	0.812	0.046	0:02:23	**0.62**	LGBMC	0.129	0.788	0.040	0:2:09	0.030	CBC
Line B	0.425	0.938	0.472	0:01:42	0.488	CBC	0.499	0.953	0.508	0:01:39	**0.520**	RFC
Line C	0.334	0.966	0.237	0:10:08	**0.338**	RFC	0.334	0.966	0.237	0:10:30	**0.338**	RFC
Line X	0.118	0.763	0.073	0:21:29	0.025	CBC	0.259	0.905	0.151	0:20:42	**0.202**	RFC
Line y	0.289	0.907	0.199	0:07:11	**0.253**	RFC	0.222	0.841	0.129	0:07:12	0.115	CBC
Line Z	0.000	0.917	0.044	0:14:04	**0.000**	RFC	0.011	0.962	0.012	0:12:54	− 0.027	CBC
UCI SECOM	0.000	0.538	0.087	0:05:07	0.000	XGBC	0.072	0.613	0.080	0:05:18	**0.034**	RFC
	Randon search CV (ROC AUC)						Randon search CV (PRC AUC)					
Glass 1	0.652	0.890	0.855	0:00:22	0.750	XGBC	0.576	0.889	0.863	0:00:20	0.773	LGBMC
Yeast-0–2–5–6	0.495	0.851	0.630	0:00:22	**0.561**	CBC	0.623	0.832	0.663	0:00:23	0.544	RFC
Line A	0.125	0.814	0.038	0:02:16	− 0.024	XGBC	0.039	0.751	0.247	0:02:08	− 7.995	DTC
Line B	0.518	0.945	0.513	0:01:37	0.444	XGBC	0.405	0.960	0.523	0:01:37	0.196	RFC
Line C	0.208	0.963	0.155	0:09:34	− 0.780	RFC	0.194	0.948	0.348	0:09:30	− 0.755	DTC
Line X	− 0.000	0.826	0.041	0:18:18	− 0.005	LGBMC	0.005	0.523	0.444	0:21:16	− 22.914	DTC
Line y	0.150	0.894	0.128	0:07:06	− 0.755	RFC	0.150	0.845	0.210	0:06:48	− 1.109	DTC
Line Z	0.083	0.906	0.023	0:13:06	− 7.085	RFC	0.067	0.848	0.199	0.13:54	− 10.148	DTC
UCI SECOM	0.025	0.581	0.057	0:05:09	− 0.320	RFC	0.0	0.542	0.088	0:05:29	0.000	LGBMC
	Randon search CV (MCC)											
Glass 1	0652	0.890	0.855	0:00:21	0.750	XGBC						
Yeast-0–2–5–6	0.623	0.832	0.663	0:00:22	0.544	RFC						
Line A	0.185	0.812	0.046	0:02:15	**0.062**	LGBMC						
Line B	0.518	0.945	0.513	0:01:42	0.444	XGBC						
Line C	0.334	0.966	0.237	0:09:53	**0.338**	RFC						
Line X	0.198	0.888	0.119	0:20:19	− 0.354	RFC						
Line y	0.289	0.907	0.199	0:07:17	**0.253**	RFC						
Line Z	0.099	0.911	0.052	0:14:21	− 2.888	RFC						
UCI SECOM	0.025	0.581	0.057	0:05:05	− 0.320	RFC						

Bold values indicate the best result for a dataset

Table 5 Comparison of the average optimization time of all run executions

Dataset	HyperOpt	RandomSearchCV
Glass1	0:00:47	0:00:31
Yeast-0–2–5–6	0:00:50	0:00:32
Line A	0:04:40	0:04:29
Line B	0:02:42	0:02:37
Line C	0:12:44	0:11:20
Line X	0:48:32	0:49:56
Line Y	0:09:20	0:08:57
Line Z	0:20:27	0:16:36
UCI SECOM	0:16:28	0:21:15

for each ML model, as our experiments do not show a set of parameters that is optimal across different cases. A comparison of different winner parameter settings for other datasets could be research in the future.

In Figs. 5 and 6 we illustrate all executions with all five metrics compared by the MQI, which are distinguished by different seeds for the run execution. The illustrations are divided by the two optimization approaches. With these illustrations we show the variance of the results of all eight run executions with the different optimization approaches. Further, we show in each visualization as baseline the results of the algorithm comparison from Fig. 4 (MQI metric) with an orange boxplot as sixth result.

Based on the characteristic of the datasets Line Z and Line X we can derive some further information. Line Z has the most unbalanced dataset and this is reflected in the results. Nevertheless, we achieved an EBR Result which is slightly better than 0. Line X has the highest number of features and for this dataset, we can also achieve an EBR result above 0. This proves to us that real world data from manufacturing with a high number of features can achieve usable results.

In 12 out of 18 experiments based on the cost-oriented metrics (EBR & EBRP) RandomSearchCV shows a better mean MQI value of all datasets compared to HyperOpt, whereas it is the other way round for 6 out of 18 experiments. In Fig. 7 we show that, RandomSearchCV indicate a slight advantage, leading to a preferable use for manufacturing data.

In all three illustrations we use a vertical line to show where a new dataset has begun and to highlight our EBR Metric. For these visualizations we had to adjust the y-axes to the maximal reachable result 1.0 and the threshold 0.0. Some metrics performed below the threshold of 0.0. For the

Table 6 Used hyperparameters for random forest classifier

Parameter/line	Line B	Line C	Line X	UCI Secom
n_estimators	1000	1000	100	800
min_weight_fraction_leaf	0	0	0	0.1
min_samples_split	7	7	7	5
min_samples_leaf	7	7	2	5
max_depth	5	5	9	7
class_weight	0: 1, 1: 20	0: 1, 1: 20	0: 1, 1: 147	0: 1, 1: 20

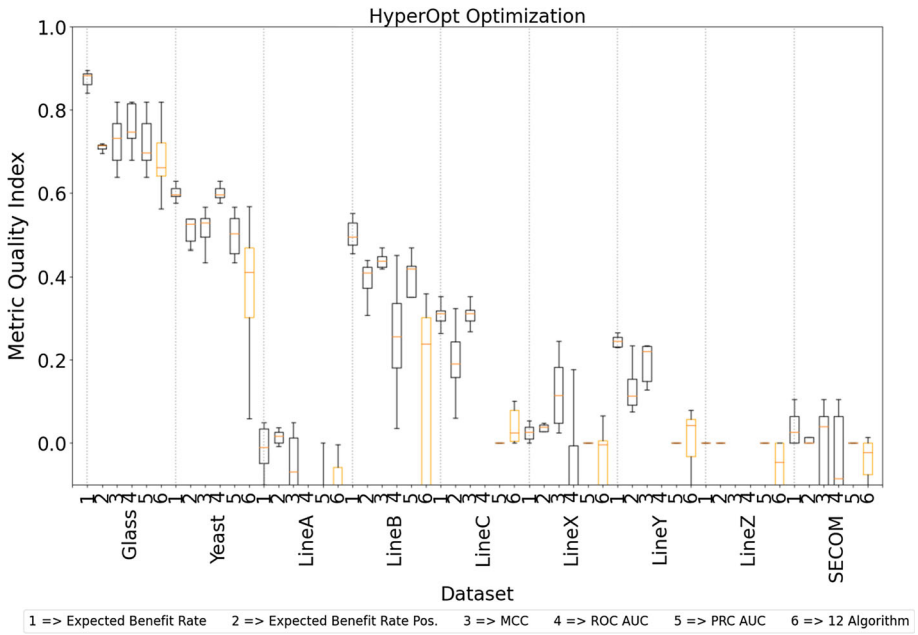

Fig. 5 Comparison of used metrics part 1

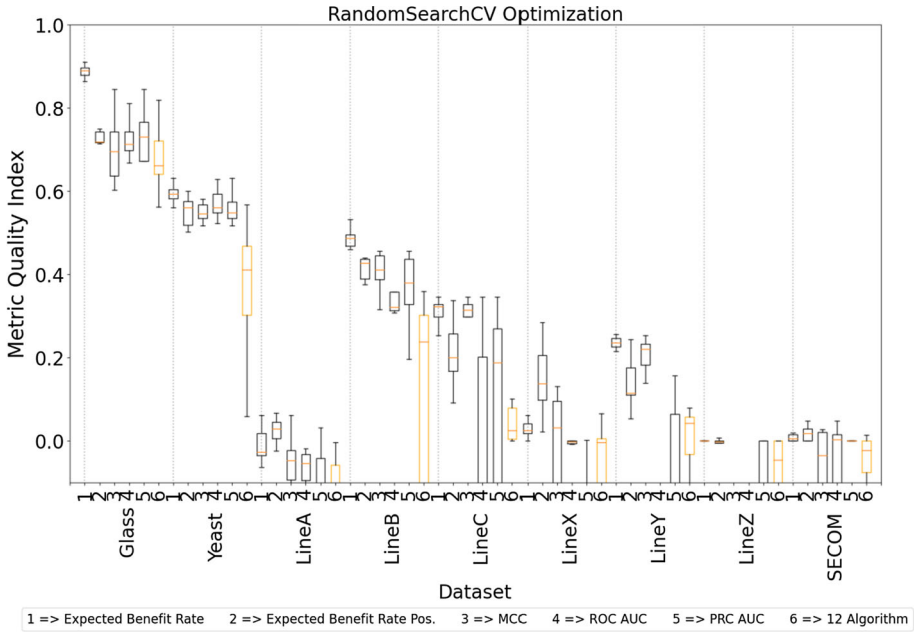

Fig. 6 Comparison of used metrics part 2

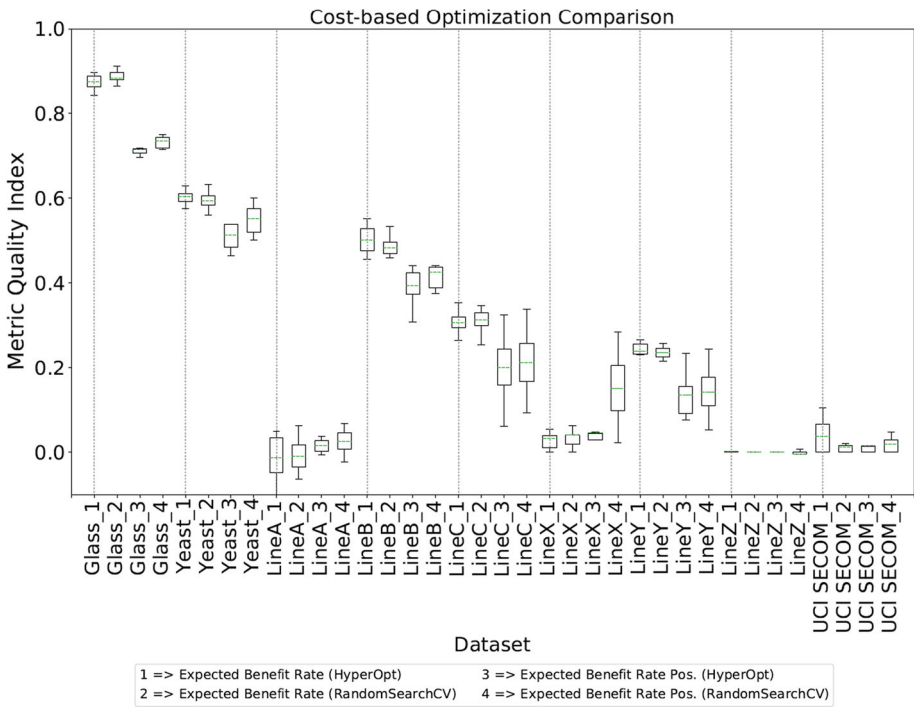

Fig. 7 Cost-based Metric Optimization Comparison

sake of simplicity, we do not illustrate results below -0.1 in the visualization. The first point to mention is that our EBR metric outperforms the other metrics in the majority of tested cases. For example, the results of the MCC metric of the dataset Line Z are even outside the chosen range of the figure. An interesting point is that the PRC metric performed worse than expected even though this metric should be favourable for imbalanced datasets.

Finally, in Table 7 we compare the cost savings from the optimized results in Figs. 5 and 6 with the worst, best and mean results from Fig. 4 with the MQI metric. We took the worst, best and mean results based on the MQI metric of all run executions in Figs. 5 and 6. In Table 7 we find some interesting information about the results. First, the best results outperform the algorithm results from Fig. 4. Secondly, we can get a worse result through the optimization, due to overfitting of the model. However, nearly all results are still in a positive range or at least neutral EBR value, except Line A. Further, all of these results are from algorithms, which are not easy to interpret for a human being. Therefore, we pro-

Table 7 Optimization comparison

Dataset/metric	Worst results from Fig. 4 (MQI)	Best results from Fig. 4 (MQI)	Median result from Fig. 4 (MQI)	Worst result from hyperparameter tuning based on hyperopt (Fig. 5)	Best result from hyperparameter tuning based on hyperopt (Fig. 5)	Median result from hyperparameter tuning based on hyperopt (Fig. 5)	Worst result from hyperparameter tuning based on randomsearchCV (Fig. 6)	Best result from hyperparameter tuning based on randomsearchCV (Fig. 6)	Median result from hyperparameter tuning based on randomsearchCV (Fig. 6)
	MQI	MQI	MQI	MQI	MQI	MQI	MQI	MQI	MQI
Glass1	0.000	0.819	0.662	**0.823**	**0.911**	**0.884**	**0.842**	**0.896**	**0.89**
Yeast-0–2–5–6	0.059	0.568	0.412	0.55	**0.632**	**0.597**	**0.576**	**0.644**	**0.594**
Line A	− 13.996	− 0.004	− 0.134	− 0.063	**0.062**	− 0.011	− 0.1	**0.049**	− 0.026
Line B	− 6.772	0.360	0.238	**0.428**	**0.533**	**0.494**	**0.456**	**0.552**	**0.488**
Line C	− 0.946	0.280	0.026	0.254	**0.346**	**0.312**	0.265	**0.353**	**0.3235**
Line X	− 16.311	0.066	− 0.003	0	**0.148**	0.0265	0	**0.105**	0.025
Line Y	− 21.647	0.080	0.043	**0.216**	**0.257**	**0.2455**	**0.178**	**0.266**	**0.2365**
Line Z	− 6.056	0.000	− 0.045	0	**0.006**	0	0	**0.012**	0
UCI SECOM	− 0.137	0.014	− 0.023	0	**0.048**	**0.027**	0	**0.105**	0.0055

Bold values indicate the best result for a dataset

pose utilizing our metric for production data with low error rates as it creates a useful contribution to save costs in the production process. In this comparison, both optimization approaches perform similarly well.

Conclusion

In summary, this paper makes four core contributions. First, we compare state-of-the-art algorithms for production data and showed that decision trees are well suited, based on the achieved results. Especially the advantage of the explainability and interpretability of a Decision Tree algorithm could be utilized to search for the root cause of an error. Second, we provide an adjusted metric (EBR) that fits to the needs of a production environment. The economic benefits of a model can be quantified trough the value of the EBR metric. With this point, we show a clear advantage over state-of-the-art metrics. Therefore, we fulfil the mentioned requirements for our metric from Sect. 3. Third, we compare the RandomSearchCV and HyperOpt approach for hyperparameter tuning in this context. As last contribution, we answer our research questions for our manufacturing use case. (Q1) The results from the established ML algorithms are acceptable and can already provide a benefit. (Q2) The existing metrics can mislead a user and increase the costs dramatically for a product. (Q3) By using hyperparameter tuning, we can benefit from a substantial better result.

We are going to improve our data pre-processing and extend the feature engineering pipeline for our AutoML tool. Additionally, we want to exploit information from a knowledge base about production in order to use it for the data preparation and feature engineering.

Acknowledgements We want to thank the company SICK AG for the cooperation in this project. Special thanks go to Nadine Gericke for the helpful comments.

Funding Open Access funding enabled and organized by Projekt DEAL. This project was funded by the German Federal Ministry of Education and Research, funding line'Forschung an Fachhochschulen mit Unternehmen (FHProfUnt)', contract number 13FH249PX6. The project is also co-financed by SICK AG. The responsibility for the content of this publication lies with the authors.

Appendix

See Tables 8 and 9.

Table 8 Hyperparameter explanation

Hyperparameter	Explanation
n_estimators	The number of trees in the forest
max_depth	The maximum depth of a tree
min_child_weight	The minimum sum of instance weight (hessian) needed in a child
reg_alpha	L1 regularization term on weights
reg_lambda	L2 regularization term on weights
gamma	The minimum loss reduction required to make a further partition on a leaf node of the tree
colsample_bytree	The subsample ratio of columns when constructing each tree
learning_rate	Step size shrinkage used in update to prevents overfitting
min_samples_split	The minimum number of samples required to split an internal node
min_samples_leaf	The minimum number of samples required to be at a leaf node
min_weight_fraction_leaf	The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node
class_weight	The weights associated with classes. The'balanced' mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data
scale_pos_weight	Control the balance of positive and negative weights, useful for unbalanced classes
l2_leaf_reg	The Coefficient at the L2 regularization term of the cost function
num_leaves	The maximum number of leaves in one tree
max_features	The number of features to consider when looking for the best split
iterations	The maximum number of trees that can be built when solving machine learning problems

Table 9 Algorithms and hyperparameter

Algorithm	Hyperparameter
Xgboost	n_estimators: [100, 300, 500, 800, 1000] max_depth: [3, 5, 7, 9] min_child_weight: [1, 2, 3] reg_alpha: [0, 0.005, 0.01, 0.05] reg_lambda: [0, 0.2, 0.4, 0.6, 0.8, 1] gamma: [0.0, 0.1, 0.2, 0.3] colsample_bytree: [0.3, 0.4, 0.5, 0.7, 1.0] learning_rate: [0.1, 0.15, 0.20, 0.25, 0.3, 0.35, 0.4] scale_pos_weight: [B*0.5, B, B*2, M*0.5, M, M*2, 1] (B = 10) (M = (sum(negative instances) / sum(positive instances)))
Random Forest	n_estimators: [100, 300, 500, 800, 1000] min_samples_split: [2, 5, 7, 9] min_samples_leaf: [1, 2, 5, 7] min_weight_fraction_leaf: [0, 0.05, 0.1, 0.15] max_depth: [3, 5, 7, 9] class_weight: [dict({0:1, 1:1}, dict({0:1, 1:5}), dict({0:1, 1:10}), dict({0:1, 1:20}), dict({0:1, 1:int(M)}), dict({0:1, 1:int(M)*2}), dict({0:1, 1:int(M)*0.5})] (M = (sum(negative instances) / sum(positive instances)))
Catboost	iterations: [100, 300, 500, 800, 1000] learning_rate: [0.15, 0.20, 0.25, 0.3, 0.35, 0.4] l2_leaf_reg: [2, 3, 5, 7] depth: [3, 5, 7, 9] scale_pos_weight: [B*0.5, B, B*2, M*0.5, M, M*2, 1] (B = 10) (M = (sum(negative instances) / sum(positive instances)))

Table 9 continued

Algorithm	Hyperparameter
LightBGM	learning_rate: [0.1, 0.15, 0.20, 0.25, 0.3, 0.35, 0.4] num_leaves: [8, 12, 16, 31] colsample_bytree: [0.3, 0.4, 0.5, 0.7, 1.0] max_depth: [3, 5, 7, 9, − 1] reg_alpha: [0, 0.005, 0.01, 0.05] reg_lambda: [0, 0.005, 0.01, 0.05] min_child_weight: [0.001, 1, 2, 3] class_weight: [dict({0:1, 1:1}, dict({0:1, 1:5}), dict({0:1, 1:10}),dict({0:1, 1:20}), dict({0:1, 1:int(M)}), dict({0:1, 1:int(M)*2}), dict({0:1, 1:int(M)*0.5})] (M = (sum(negative instances) / sum(positive instances)))
Decision Tree	max_depth: [None, 3, 5, 7, 9] max_features: ['auto'] min_samples_split: [2, 5, 7, 9] min_samples_leaf: [1, 2, 5, 7] class_weight: [dict({0:1, 1:1}, dict({0:1, 1:5}), dict({0:1, 1:10}), dict({0:1, 1:20}), dict({0:1, 1:int(M)}), dict({0:1, 1:int(M)*2}), dict({0:1, 1:int(M)*0.5})] (M = (sum(negative instances) / sum(positive instances)))

References

Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., & Herrera, F. (2011). Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing 17*.

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998) Genetic programming: An introduction on the automatic evolution of computer programs and its applications.

Bergstra, J. & Bengio, Y. (2012) Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 81–305.

Boughorbel, S., Fethi, J., Mohammed, E.-A. (2020). Optimal classifier for imbalanced data using matthews correlation coefficient metric. PLOS ONE. Public Library of Science. Retrieved February 12, 2020 from https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0177678

Caggiano, A., Zhang, J., Alfieri, V., Caiazzo, F., Gao, R., & Teti, R. (2019). Machine learning-based image processing for on-line defect recognition in additive manufacturing. *CIRP Annals, 68*(1), 451–454. https://doi.org/10.1016/j.cirp.2019.03.021

Candel, A., Parmar, V., LeDell, E., & Arora, A. (2016). Deep learning with H_2O. H_2O. AI Inc.

"CatBoost" CatBoost. Documentation. Retrieved February 10, 2020 from https://catboost.ai/docs/concepts/python-reference_parameters-list.html

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*(1), 37–46.

Dao, T. K., Pan, T. S., & Pan, J. S. (2018). Parallel bat algorithm for optimizing makespan in job shop scheduling problems. *Journal of Intelligent Manufacturing, 29*(2), 451–462.

"DecisionTreeClassifier" scikit. Retrieved February 10, 2020. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Domingos, P. (1999). "MetaCost." In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining—KDD 99, 155–64. https://doi.org/10.1145/312129.312220

Drummond, C., & Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance. *Machine Learning, 65*(1), 95–130. https://doi.org/10.1007/s10994-006-8199-5

Du, J. X., Cai, Z. X., & Ling, C. X. (2007). Cost-sensitive decision trees with prepruning. *Advances in Artificial Intelligence (Lecture Notes in Computer Science)* 171–79

Dua, D. & Graff, C. (2020). UCI machine learning repository. Irvine, CA, University of California, School of Information and Computer Science. Retrieved January 25, 2020 from http://archive.ics.uci.edu/ml

Delgado, R., & Xavier-Andoni, T. (2019). Why Cohen's Kappa should be avoided as performance measure in classification. *PLoS ONE*. https://doi.org/10.1371/journal.pone.0222916

Feurer, M., Aaron, K., Katharina, E., Jost, T. S., Manuel, B., & Frank, H. (2019). Auto-Sklearn: Efficient and robust automated machine learning. *Automated Machine Learning the Springer Series on Challenges in Machine Learning*. https://doi.org/10.1007/978-3-030-05318-5_6

Gerling, A., Ulf, S., Andreas, H., Alaa, S., Holger, Z., & Djaffar, A. (2020). A reference process model for machine learning aided production quality management. In: Proceedings of the 22nd international conference on enterprise information systems 1, 515–23. https://doi.org/10.5220/0009379705150523

Golovin, D., Benjamin S., Subhodeep, M., Greg, K., John, K., & Sculley, D. (2017). Google Vizier. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining—KDD 17. https://doi.org/10.1145/3097983.3098043

Hirsch, V., Peter, R., & Bernhard, M. (2019) Data-driven fault diagnosis in end-of-line testing of complex products. In: 2019 IEEE international conference on data science and advanced analytics (DSAA), https://doi.org/10.1109/dsaa.2019.00064

Ho, Y., & Pepyne, D. (2002). Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications, 115*, 549–570. https://doi.org/10.1023/A:1021251113462

"HyperOpt" Hyperopt Documentation. Retrieved February 10, 2020 from http://hyperopt.github.io/hyperopt/#hyperopt-distributed-asynchronous-hyper-parameter-optimization

Kim, A., Oh, K., Jung, J. Y., & Kim, B. (2018). Imbalanced classification of manufacturing quality conditions using cost-sensitive decision tree ensembles. *International Journal of Computer Integrated Manufacturing, 31*(8), 701–717.

Kirchen, I., Vogel-Heuser, B., Hildenbrand, P., Schulte, R., Vogel, M., Lechner, M., et al. (2017). Data-driven model development for quality prediction in forming technology. In 2017 IEEE

15th international conference on industrial informatics (INDIN) (pp.775–780). IEEE. https://doi.org/10.1109/indin.2017.8104871

Krauß, J., Pacheco, B. M., Zang, H. M., & Schmitt, R. H. (2020). Automated machine learning for predictive quality in production. *Procedia CIRP, 93*, 443–448. https://doi.org/10.1016/j.procir.2020.04.039

Kochura, Y., Sergii, S., Oleg, A., Michail, N., & Yuri, G. (2017). Performance analysis of open source machine learning frameworks for various parameters in single-threaded and multi-threaded modes. *Advances in Intelligent Systems and Computing II Advances in Intelligent Systems and Computing*. https://doi.org/10.1007/978-3-319-70581-1_17

Kotthoff, L., Chris, T., Hoos, H. H., Hutter, F., & Kevin, L.-B. (2019). Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. *Automated Machine Learning the Springer Series on Challenges in Machine Learning*. https://doi.org/10.1007/978-3-030-05318-5_4

Krętowski, M., & Marek, G. (2006). Evolutionary induction of cost-sensitive decision trees. *Lecture Notes in Computer Science Foundations of Intelligent Systems*. https://doi.org/10.1007/11875604_15

Li, Z., Zhang, Z., Shi, J., & Dazhong, Wu. (2019). Prediction of surface roughness in extrusion-based additive manufacturing with machine learning. *Robotics and Computer-Integrated Manufacturing, 57*, 488–495. https://doi.org/10.1016/j.rcim.2019.01.004

Liu, G., Gao, X., You, D., & Zhang, N. (2019). Prediction of high power laser welding status based on PCA and SVM classifcation of multiple sensors. *Journal of Intelligent Manufacturing, 30*(2), 821–832. https://doi.org/10.1007/s10845-016-1286-y

"LightGBM" Parameters—LightGBM 2.3.2 documentation. Retrieved February 10, 2020 from https://lightgbm.readthedocs.io/en/latest/Parameters.html

Loyola-Gonzalez, O., Jose, F. C. O., Martinez-Trinidad, J.A.C.-O., & Milton, G.-B. (2019). Cost-sensitive pattern-based classification for class imbalance problems. *IEEE Access, 7*, 60411–60427. https://doi.org/10.1109/access.2019.2913982

Maher, M. M., Maher, Z. A., & Sherif, S. (2019) SmartML: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. EDBT: 22nd international conference on extending database technology. https://doi.org/10.5441/002/edbt.2019.54

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)—Protein Structure 405*(2), 442–51. https://doi.org/10.1016/0005-2795(75)90109-9.

Moldovan, D., Cioara, T., Anghel, I., & Salomie, I. (2017). Machine learning for sensor-based manufacturing processes. In: 13th IEEE international conference on intelligent computer communication and processing (ICCP), 147–54. https://doi.org/10.1109/ICCP.2017.8116997

Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., La, C. K., & Moore, J. H. (2016). Automating biomedical data science through tree-based pipeline optimization. *Applications of Evolutionary Computation Lecture Notes in Computer Science*. https://doi.org/10.1007/978-3-319-31204-0_9

Olson, R. S., & Moore, J. H. (2019). TPOT: A tree-based pipeline optimization tool for automating machine learning. *Automated Machine Learning the Springer Series on Challenges in Machine Learning*. https://doi.org/10.1007/978-3-030-05318-5_8

"RandomForestClassifier" scikit. Retrieved February 10, 2020 from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Ren, L., Meng, Z., Wang, X., Zhang, L., & Yang, L. T. (2020). A data-driven approach of product quality prediction for complex production systems. *IEEE Transactions on Industrial Informatics*. https://doi.org/10.1109/TII.2020.3001054

Sankhye, S., & Hu, G. (2020). Machine learning methods for quality prediction in production. *Logistics, 4*(4), 35.

Sarath, S. (2018). Area under the ROC curve—explained. *Medium. Medium*. https://medium.com/@sarath13/area-under-the-roc-curve-explained-d056854d3815

Shmueli, B. (2019). Matthews correlation coefficient is the best classification metric you've never heard of. *Medium. Towards Data Science*. https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3bf50a2f3e9a

SICK AG. Retrieved April 23, 2020 from https://www.sick.com/de/de/

Tangjitsitcharoen, S., Thesniyom, P., & Ratanakuakangwan, S. (2017). Prediction of surface roughness in ball-end milling process by utilizing dynamic cutting force ratio. *Journal of Intelligent Manufacturing, 28*(1), 13–21. https://doi.org/10.1007/s10845-014-0958-8

Thornton, C., Frank H., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining—KDD. https://doi.org/10.1145/2487575.2487629

Wang, G., & Jiao, J. (2017). A kernel least squares based approach for nonlinear quality-related fault detection. *IEEE Transactions on Industrial Electronics, 64*(4), 3195–3204. https://doi.org/10.1109/TIE.2016.2637886

Wilhelm, Y., Schreier, U., Reimann, P., Mitschang, B., & Ziekow, H. (2020). Data Science approaches to quality control in manufacturing: A review of problems, challenges and architecture. In: Symposium and Summer School on Service-Oriented Computing (pp. 45–65). Springer.

Worcester, P. (2019). A comparison of grid search and randomized search using scikit learn. *Medium. Noteworthy—The Journal Blog*. https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85?gi=2eb9f2afe9a3

"XGBoost" XGBoost Parameters - xgboost 1.1.0-SNAPSHOT documentation. Accessed February 10, 2020. https://xgboost.readthedocs.io/en/latest/parameter.html.

Yuan, X., Lin, L., Yalin, W., Chunhua, Y., & Weihua, G. (2020). Deep learning for quality prediction of nonlinear dynamic processes with variable attention-based long short-term memory network. *The Canadian Journal of Chemical Engineering, 98*(6), 1377–1389. https://doi.org/10.1002/cjce.23665

Zhang, S., Zhu, X., Zhang, J., & Zhang, C. (2007). Cost-time sensitive decision tree with missing values. *Knowledge Science, Engineering and Management (Lecture Notes in Computer Science)* 447–59.

Zhenyu, L., Zhang, D., Jia, W., Lin, X., & Liu, H. (2020). An adversarial bidirectional serial–parallel LSTM-based QTD framework for product quality prediction. *Journal of Intelligent Manufacturing 31*, 1511–1529. https://link.springer.com/article/10.1007/s10845-019-01530-8

Zhou, L., Wang, H., Lu, Z. M., Nie, T., & Zhao, K. (2016). Face recognition based on LDA and improved pairwise-constrained multiple metric learning method. *Journal of Information Hiding and Multimedia Signal Processing, 7*(5), 1092.

Ziekow, H., Schreier, U., Saleh, A., Rudolph, C., Ketterer, K., Grotzinger, D., & Gerling, A. (2019) Proactive error prevention in manufacturing based on an adaptable machine learning environment. *Research to Application 113*.

Zonnenshain, A., & Kenett, R. S. (2020). Quality 4.0—the challenging future of quality engineering. *Quality Engineering*. https://doi.org/10.1080/08982112.2019.1706744