Journal of Information and Data Management, 2024, 15:1, doi: 10.5753/jidm.2024.3365

Two Meta-learning approaches for noise filter algorithm recommendation

Pedro B. Pio [**University of Brasilia** | ***pedrobpio@gmail.com***]
Adriano Rivolli [**Federal University of Technology – Paraná, Brazil (UTFPR)** | ***rivolli@utfpr.edu.br***]
André C. P. L. F. de Carvalho [**University of São Paulo** | ***andre@icmc.usp.br***]
Luís P. F. Garcia [**University of Brasilia** | ***luis.garcia@unb.br***]

Department of Computer Science, University of Brasília, Asa Norte, Brasília, DF, 70910-900, Brazil.

Received: 30 April 2023 • **Published:** 23 February 2024

Abstract Preprocessing techniques can increase the predictive performance, or even allow the use, of Machine Learning (ML) algorithms. This occurs because many of these techniques can improve the quality of a dataset, such as noise removal or filtering. However, it is not simple to identify which preprocessing techniques to apply to a given dataset. This work presents two approaches to recommend a noise filtering technique using meta-learning. Meta-learning is an automated machine learning (AutoML) method that can, based on a set of features extracted from a dataset, induce a meta-model able to predict the most suitable technique to be applied to a new dataset. The first approach returns a ranking of the noise filter techniques using regression models. The second sequentially applies multiple meta-models, to decide the most suitable noise filter technique for a particular dataset. For both approaches we extract the meta-features from use synthetics datasets and use as meta-label the f1-score value obtained by different ML algorithms when applied to these datasets. For the experiments, eight noise filtering techniques were used. The experimental results indicated that the rank approach acquired higher performance gain than the baseline, while the second obtained higher predictive performance. The ranking based approach also ranked the best algorithm in the top-3 positions with high predictive accuracy.

Keywords: Meta-Learning, Noise Detection, Preprocessing, Machine Learning, Algorithm Recommendation, Ranking

1 Introduction

Machine Learning (ML) has been defined as the ability to adapt to new circumstances and to detect and extrapolate patterns [Russell and Norvig, 2009]. Nowadays, to facilitate the implementations of ML algorithms, we have several frameworks, such as Weka, Scikit-learn, H2O, Tensorflow, and Pytorch. However, applying ML algorithms to datasets and acquiring information from them is very time-consuming. Usually, the user follows a Data Mining (DM) methodology such as CRISP-DM [Wirth and Hipp, 2000] and KDD [Fayyad *et al*., 1996]. Those methodologies indicate that a few steps will be necessary to successfully extract the information, such as preprocessing, algorithm selection, hyperparameters tunning, and presentation. Unfortunately, choosing the preprocessing algorithm is not trivial and can influence the entire ML process. In fact, the choice of the preprocessing techniques that should be applied may vary according to the ML algorithm that is selected [García *et al*., 2015].

Since the demand for ML systems has grown in the past years, forming market pressure for ML specialists, and the process to build ML systems is commonly tedious and repetitive, the idea of automating the ML process became promising [Truong *et al*., 2019]. This field, called Automated ML (AutoML), aims to automatically find the best approach for a particular problem when provided with a dataset [Hutter *et al*., 2019]. For this, the AutoML systems may search for the best data preprocessing, features engineering, ML models, hyperparameters of the algorithm, and architecture [Truong *et al*., 2019]. At each step, the AutoML systems will have to search for algorithms, a problem usually solved with Bayesian optimization, Genetic Programming, or Meta-Learning (MtL) [Nagarajah and Poravi, 2019]. It is also important to notice that the preprocessing step is not extensively covered by most of the AutoML solutions [Truong *et al*., 2019].

The preprocessing step covers all actions applied before the data analysis starts [Famili *et al*., 1997]. It is essentially a transformation applied to the raw data returning a new dataset ready for data analysis. There are several different reasons why it is necessary to implement a preprocessing technique: data may have missing values, too many or not enough attributes, noisy instances, and other problems [Famili *et al*., 1997]. However, since noise data may provide lower accuracy for classifiers, it is hard to find a generalized algorithm to remove it, and they commonly appear on real data [Zhu and Wu, 2004]. In this work, we focus on noise detection algorithms.

Our goal is to present two different noise detection algorithm recommendation systems using MtL, compare their results, and identify the advantages of each approach. MtL, commonly known as learning how to learn, is a form of using previous experiences to solve similar tasks [Vanschoren, 2019]. When applied to algorithm selection problems, it performs the recommendation based on a set of meta-features (MF) extracted from the dataset that contains relevant information that influences the algorithm choice [Brazdil *et al*., 2009].

Since 50% to 80% of the DM time is dedicated to prepro-

cessing the data [Munson, 2012], most of the AutoML lack extensive preprocessing support [Truong *et al*., 2019], and MtL is frequently used for algorithm recommendation [Vanschoren, 2019]. We believe that a noise detection algorithm recommendation could be useful in an AutoML system, improving the quality and flexibility of current solutions. Also, MtL is a feasible approach to predict the best algorithm reducing the suggestion cost based on previous experience.

This work is an extension of Pio *et al*. [2022]. And its main contributions can be summarized as follows: (i) Applies MtL to produce a rank of the most suitable noise detection algorithm for a given dataset from a set of MF; (ii) Proposes a flexible methodology that could be expanded to other preprocessing techniques and integrated into current AutoML systems; (iii) Evaluates the effects of the noise filter algorithms on datasets and both meta and base levels of the MtL approach; (iv) Compares two different MtL methodologies for noise detection evaluating its meta and base levels and identifying the advantages and disadvantages of each one.

The rest of this article is divided as: In Section 2, we present a theoretical background on MtL and noise detection methods and introduce some previous works that use MtL to suggest preprocessing techniques; Section 3 explains the proposed methodology; In Section 4, we present the results and some discussions; and in Section 5, we present the conclusions and propose some future works.

2 Background and Related Works

2.1 Algorithm Selection and Meta-learning

One of many utilities for MtL applications is to use previous knowledge to select an algorithm [Vanschoren, 2019]. Rice [Rice, 1976] was one of the first to propose a solution for the algorithm selection problem. Rice divided it into four different spaces:

- The problem space P is the set of problems involved that usually has high dimensions and some independent characteristics that are important for the algorithm selection;
- The feature space F is the set of features extracted from the instances P;
- The algorithm space A is the set of algorithms considered in the selection;
- The performance space Y contains the metrics used to evaluate the performance of the algorithms.

Rice proposed that, from P we could extract $f(x) \in F$ reducing the problem complexity, from F we use the function $S(f(x))$ to map the algorithm $a \in A$. Ideally, $S(f(x))$ will result in the best algorithm according to the performance metric $y \in Y$. Smith-Miles [Smith-Miles, 2008] expanded the solution proposed by Rice to support MtL dividing it into three phases. The first one aims to build a meta-data set formed by the features F, extracted from P, and the algorithm performance results Y, acquired from the algorithms A. In the second phase, we use empirical rules in the meta-data to perform the algorithm selection. In phase three, theoretical support is applied to adjust the empirical rules and refines the algorithms.

We separate the MtL process into the base level, used to build the meta-data from the P, F, A, and Y space, and the meta level, where we interpret the meta-data [Brazdil *et al*., 2009]. At the base level occurs the extraction of the MF, forming the F space, which can be divided into six groups [Rivolli *et al*., 2022]: simple; statistical; information-theoretic; model-based; landmarking; and others. The recommendation process happens at the meta level, where we can apply classifications, regressions, or rank algorithms to predict which algorithm to choose based on the F and Y [Brazdil *et al*., 2009].

2.2 Noise detection

In real-world datasets, it is common to occur mistakes in its values called noise, which can appear both in the attributes and the data class. When performing data analysis, the class noise will probably cause more interference in the results than the attributes noise [Zhu and Wu, 2004].

If the noise is detected, we may choose between ignoring, removing (filtering), or altering the instance [Gupta and Gupta, 2019]. Usually, when filtering or modifying the instance, we can apply methods such as [Frénay and Verleysen, 2014] (i) classification filter, where a classification algorithm is used to identify the noisy instance and then remove it; (ii) voting filter, where multiple algorithms are executed and each one vote to remove or keep the instance; (iii) distance-based methods, where the algorithm utilizes K nearest neighbors (KNN) sensibility to noise to identify it; (iv) ensemble or boosting methods, where proprieties of the algorithm, such as the tendency to overfitting, are used to identify the noise.

A list of noise filters is presented by Morales [Morales *et al*., 2017], of which we used the following algorithms:

- **High Agreement Random Forest (HARF)** [Sluban *et al*., 2014]: Uses the Random Forest (RF) classifier as a noise filter. Instead of classifying each instance, it defines a percentage threshold of agreement trees. If the quantity of agreement trees is lower than the threshold, the instance is considered as noisy and is removed;
- **Dynamic Classification Filter (DCF)** [Garcia *et al*., 2012]: it is a voting filter that uses a set of classifiers algorithms to define if an instance is noisy, where each algorithm vote once. Considering the algorithms K-Nearest Neighbor (KNN) (with k = {3, 5, 7}), Support Vector Machine (SVM), CART, C4.5, RF, Naive Bayes e Multilayer Perceptron (MLP)) as the set of classifiers, it selects m algorithms based on its predictions similarities. The instances labeled as noisy by the majority of classifiers are removed;
- **Hybrid Repair-Remove Filter (HRF)** [Miranda *et al*., 2009]: It uses four classifiers to vote if the instance is noisy (SVM, MLP, CART, and KNN). If the instance is considered noisy by the majority of the classifiers, the algorithm may decide to remove or modify the instance label according to the result obtained by the KNN classifier.
- **Outlier Removal Boosting (ORB)** [Karmaker and Kwek, 2006]: it uses the propriety of AdaBoost [Fre-

und and Schapire, 1995] to enhance the weights of the outliers instances to implement the filter. If the weights are higher than a defined threshold d it is considered noise and removed.
- **Edge Boosting Filter (EDB)** [Wheway, 2001]: uses AdaBoost to classify if an instance is noisy. It applies the AdaBoost for m iterations, computing the *edge value* of each instance, if it is higher than a threshold t, the instance is considered as noisy and is removed;
- **Generalized Edition (GE)** [Koplowitz and Brown, 1981]: Is a variation of the ENN [Wilson, 1972] algorithm that allows the possibility to correct the noisy instance. The instance is corrected if the number of agreement neighborhoods is higher than k'. Otherwise, it is removed;
- **All-k Edited Nearest Neighbors (AENN)** [Tomek, 1976]: it applies the ENN algorithm multiple times with K varying from 1 to k. If any instance is classified as noisy by any of the ENN executions, it is considered as noisy and is removed;
- **Preprocessing Instances that Should be Misclassified (PRISM)** [Smith and Martinez, 2011]: It computes five heuristics, one based on distance, two based on likelihood and two extracted from leaves of the algorithm C4.5. the five heuristics are passed to a function[1], and, depending on the returned result, the instance is classified as noisy and is removed.

Note that this algorithm set A includes voting, distance-based, and ensemble noise filters. Where HARF, DCF, and HRF are voting filters, GE, AENN, and PRISM are distance-based, and the ensemble filters are ORB and EDB.

2.3 Related Works

To validate the possibility of using an MtL approach to recommend preprocessing algorithms, we performed a systematic review on the subject of MtL and preprocessing techniques. Although it is not the focus of this work to present the results of this review, in this section, we present some studies that use MtL techniques to recommend preprocessing algorithms.

Garcia [Garcia *et al*., 2016a] implements an algorithm selection system for noise filter algorithms aiming to select the algorithm which identifies the noisy instance with higher precision. It uses 53 datasets extracted from UCI and KEEL. They artificially generate noise into these datasets with a percentage varying between 5% to 20% of the total instances. The meta-data was built with 70 MF and with the performance metric f1-score as a prediction target. They compared three different algorithms to perform the recommendation: KNN, RF, and SVM. After the analysis of the mean square error of each regressor, they concluded that RF was a better choice.

Garcia [Garcia *et al*., 2016b] presents a MtL system that aims to recommend the noise detection algorithm that achieves the best precision in removal. The study uses five different noise reduction algorithms forming 26 combinations of ensembles that are evaluated and used as the algorithm set. the study selected 90 datasets that were subjected to an artificial noise generation process, creating noise in 5% 10, 20, and 40% of the instances. The MtL used a total of 70 MF to perform the classification of the best ensemble of algorithms. TO define the best meta-classifier, the authors compared five ML techniques and concluded that the DT obtained the best performance, with approximately 75% of accuracy.

Bilalli [Bilalli *et al*., 2019] introduces a preprocessing recommendations system that ranks the most suitable preprocessing techniques. The recommendation includes discretization, normalization, missing data imputation, and dimensionality reduction techniques. To create the meta-data, they used over 500 real datasets extracted from OpenML, of which they extracted over 60 MF and applied five different ML algorithms to calculate the performance metrics. They compared the results with baseline algorithms and real users. In both cases, the recommendation was efficient, acquiring better accuracy than random algorithms and non-specialists.

Parmezan [Parmezan *et al*., 2021] proposes a methodology to suggest a feature selection algorithm applying MtL systems in sequence. First, they used multiple MtL systems to select the type of algorithm that is more adequate, then, depending on the selected type, it suggests an algorithm, and finally, another MtL system is applied to suggest the algorithm parameters. A total of nine meta-data were created, two to select the type of algorithm, two to select the algorithm, and five to suggest the algorithm with its parameters. Five algorithms are supported: CBF, CFS InfoGain, Relief, and wrapper subset evaluation. 213 datasets were selected and 161 MF were used to create all nine meta-data. The proposed solution acquired up to 90% accuracy.

Pio [Pio *et al*., 2022] perform the recommendation of noise detection algorithms by trying to identify the algorithm that generates the most gain in the *f1-score* performance metric. A total of 323 datasets were used and modified to contain 5, 10, 20, and 40 percent of noisy instances. A set of three noise filters, HARF, ORB, and GE, and 73 MF were utilized to perform the recommendation which was presented to the user as a ranking of the best algorithms. The work compared three regressors that were adapted from the rank, concluding that the solution could generate performance gain in the f1-score metric and that the RF algorithm acquired the best results.

In this work, we present two noise filtering recommendation methodologies and compare their results. Although a similar problem was approached by [Garcia *et al*., 2016a] and [Garcia *et al*., 2016b], here we are trying to predict the performance of a specific classification algorithm, verifying if the combination of filtering and classifier is adequate, while in the aforementioned work, the authors predict how well the filter would find the noise. More precisely, this work compares two MtL approaches, the first is an extension of the methodology proposed by Pio [Pio *et al*., 2022]. The second approach is adapted from Parmezan's [Parmezan *et al*., 2021] solutions to feature selection recommendation, thus, we used a set of MtL in sequences to suggest a noise filter algorithm. Both approaches are explained in Section 3 and their results are shown and compared in Section 4.

[1] defined by the author

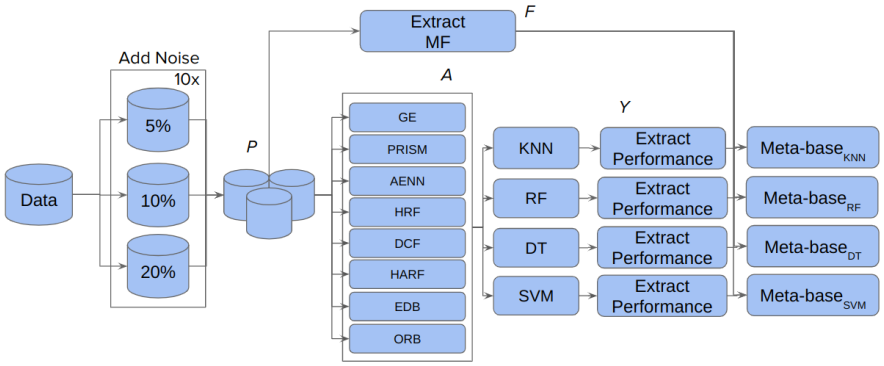

Figure 1. Base level diagram.

3 Methodology

This section describes the proposed methodologies for each MtL approach. We divided it into two subsections: Firstly we show the ranking approach (MtL-Rank), which was expanded from [Pio *et al.*, 2022]; secondly, we present the approach based on Parmezan's solution [Parmezan *et al.*, 2021], where multiple MtL models are used in sequence to provide the suggestion (MtL-Multi).

3.1 MtL-Rank

To explain each approach, we divided it into the base and meta levels. The base level is where we build the meta-base, and the meta level is where we build the recommendation and analyze the results. Figure 1 shows a diagram of the MtL-Rank base level, the process begins with the data collection. We used the OpenML[2] platform to collect the datasets that were used. The platform allows us to filter the datasets according to our needs. We used sets with 100 to 20000 instances, 3 to 100 attributes, without missing values, and with only two classes, thus, reducing the preprocessing step needed to extract the ML metrics. To ensure that all ML and noise filter algorithms would run, we converted all categorical attributes into dummy variables and excluded datasets with more than 200 attributes due to high dimensionality, resulting in a total of 358 datasets.[3]. Note that, those filters were used to standardize the datasets, facilitate the tests, and reduce the execution time of the noise filters and the ML algorithms. In fact, this methodology could work without any restrictions regarding the number of classes, attributes, or instances in the datasets. Afterwards, to control the class noise level on each dataset, we randomly changed the instance's class, generating artificially random noise in 5%, 10%, and 20% of the instances. To avoid bias in cases where the noise is applied in instances outliers, we repeated the process 10 times for each percentage of noise, resulting in 30 synthetics sets for each dataset, forming the P space.

With the class noise introduced in the datasets, we calculate the performance metric Y after applying the noise filter algorithms. To calculate Y, first, we apply the filter and then run a classification algorithm that will allow us to compute the f1-score. We executed the eight different filters that compose the algorithm space A: **GE**, **PRISM**, **AENN**, **HRF**, **DCF**, **HARF**, **EDB**, and **ORB**. All filters were implemented with the `NoiseFiltersR` package and their default configurations. After executing the filter, we used four different ML algorithms as base-learners to extract the f1-score: DT CART [Breiman *et al.*, 2017] algorithm, RF [Breiman, 2001], KNN [Mitchell, 1997] and SVM [Cortes and Vapnik, 1995], all with the default hyperparameters of the `Scikit-Learn` implementation. To compute the MF we used the `pymfe`[4] [Alcobaça *et al.*, 2020] library, which allows us to calculate a large variety of features, including: simple, statistical, information-theoretic, model-based, and landmarking. We extracted a total of 97 MF producing the F space. Combining both the F space and the Y space resulted in four meta-bases, one for each base-learner, used in the meta level to apply the regressions and form the ranking containing 10740 instances, 97 MF, and 8 performance metrics, which are later used as the regressors targets.

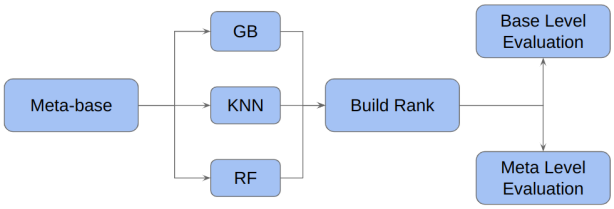

Figure 2. Meta level diagram.

Figure 2 shows the meta level, where we implement the meta-ranker and evaluate its performance. The induced meta-model is obtained by an ML algorithm that performs a regression. The goal is, based on the MF, to predict the ML classifier f1-score on the dataset after the filter execution. Later, the set of regressions is transformed into a rank, being ordered and then labeled according to the higher f1-score, in case of a draw between different approaches both are labeled equally as the best.

To perform the regression, we used three different[5] ML algorithms: the Gradient Boosting (GB) [Friedman, 2001] that is boosting ensemble algorithm; the KNN [Mitchell, 1997], which is a distance-based algorithm; and the RF [Breiman, 2001] a bagging ensemble algorithm, once again,

[2] `https://www.openml.org/`

[3] The list of datasets and MF used can be found at `https://bit.ly/JIDM_Datasets` and `https://bit.ly/JIDM_MF` respectively

[4] `https://pymfe.readthedocs.io/en/latest/`

[5] In this work we used four classifiers to calculate the f1-score, we call them base-learners, and three regressions algorithm to generate the ranking.

all algorithms were executed with `Scikit-Learn` default parameters. Since each dataset generates 30 synthetics noisy datasets, during the training, to avoid bias, we implemented a variation of the leave-one-out cross-validation [Cawley and Talbot, 2003], in which we separated each set of the 30 datasets derivated from the same original OpenML dataset and validate them together.

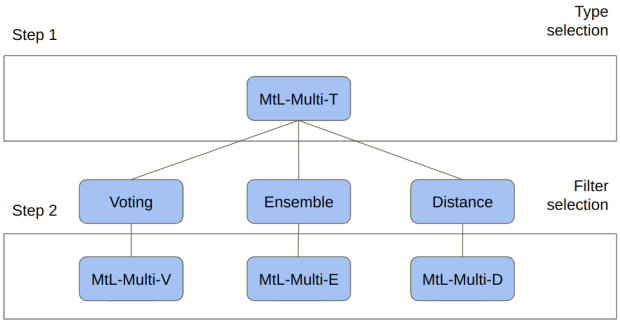

Figure 3. Steps of MtL-Multi.

To compare the results we selected two baselines, which represent the results acquired if we always used the filters DCF and ORB. Those algorithms were chosen because they provided the best overall performance gain and accuracy, respectively. At the base level, we selected the best algorithm predicted by each meta-ranker and compared it with the baselines, allowing us to quantify the gain in the f1-score metric we acquired with each ML algorithm. At the meta level, we evaluated the accuracy of the top-k best position in the generated rank, we used $k = 1$ and $k = 3$, meaning we consider a rank as correct if the best algorithm is in the top-1 and top-3 position of the rank. Again, we compare the top-k accuracy with the baselines, allowing us to identify when the recommendations were efficient. Finally, to evaluate the rank, we apply Spearman's rank correlation [Zar, 2014] between the optimal rank and the one returned by the meta-ranker.

3.2 MtL-Multi

The second approach used in this study was based on Parmezan's solution [Parmezan *et al.*, 2021], however, it was adapted to perform the noise algorithm recommendation. This recommendation is performed in two steps. Firstly, a MtL model is used to decide the most appropriate type of filter. Subsequently, according to the chosen filter type, another MtL model is selected to decide the most suitable algorithm. Figure 3 illustrates the two steps of the MtL-Multi, considering that we used three different filter types, it is required a total of four MtL systems to perform the recommendation. It is important to note that, unlike MtL-Rank, this approach does not provide the result as rank, instead, it returns only the selected algorithm.

To enable a fair comparison between the two approaches, we decided to maintain as much as possible the four spaces. Thus, we used the same set of datasets (P), MF (F), algorithms (A), and performance metrics (Y). However, the algorithm space was divided into three categories: voting algorithms (HARF, DCF, and HRF), distance-based algorithms (AENN, GE, and PRISM), and ensemble algorithms (ORB and EDB).

Figure 4 illustrates the construction of each meta-base, denoted by meta-base$_{T,D,V,E}$, which are used to predict the algorithm type, the distance algorithm, the voting algorithm, and the ensemble algorithm, respectively. The meta-bases differ mainly in the algorithm set and labeling. To build the meta-base$_T$ we executed all noise filters and labeled the dataset with the type of filter that achieved the highest performance gain. The other three meta-bases focused only on a specific type of noise filter and were labeled with the algorithm name that generated the highest performance gain considering its reduced algorithm set. We extracted the performance metric Y using the same four base-learners as in MtL-Rank.

With the meta-bases, we can perform the recommendation and evaluate the results. We employed three classifiers, namely GB, KNN, and RF, to generate the recommendations, and evaluated their performance at both the meta and base levels. The results were compared with the same baselines selected in the MtL-Rank approach.

4 Results

Before evaluating each MtL approach, we analyzed the meta-base. This included examining the impact of the noise filter on performance metrics, identifying which filters produced the highest performance gain for each base-learner, determining the optimal rank for each filter in terms of performance gain, and verifying the distribution of classes within the MtL-Multi meta-bases.

First, we analyze the filters' impacts, checking if the effects in the f1-score are positive or negative. Figure 5 displays the number of datasets that exhibited positive and negative results in the performance metric after applying each filter for each base-learner. HARF was the filter that produced the highest number of positive results, followed by either AENN or DCF. In contrast, ORB was always the filter with more negative results, followed by either PRISM or EDB.

However, the results were different when we considered the sum of the f1-score gain. Figure 6 shows the total performance gain obtained by each filter across all datasets. HARF, instead of being the first, is either the fourth or fifth filter with higher performance gain, meaning that, even though it may not decrease performance, it may not provide a significant performance boost compared to other filters that yield more positive results. Overall, DCF was the filter with the best performance, while PRISM was the only one with negative performance gain.

To determine how frequently each filter was ranked as the best and the worst, we assigned a rank to each algorithm based on its performance gain across all datasets. Figure 7 presents the frequency with which each algorithm appeared on each rank position considering all noisy datasets. The results showed that ORB was ranked as the best algorithm most frequently, however, it was also commonly ranked as one of the worst. DCF was most often ranked as the best or second-best filter, while HRF was always the third-best. Those results help to explain why ORB did not have the worst perfor-

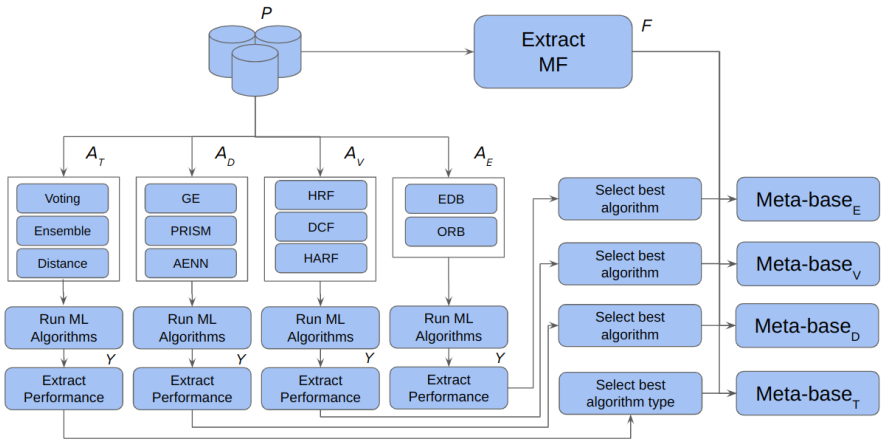

Figure 4. Base level diagram of MtL-Multi.

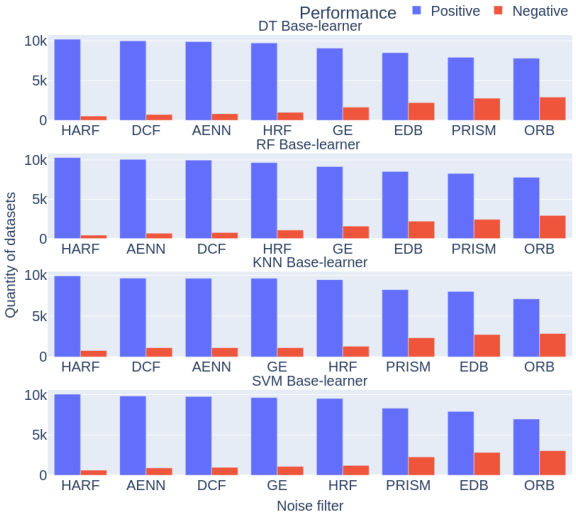

Figure 5. Times the filters had a positive or negative result in the performance metric.

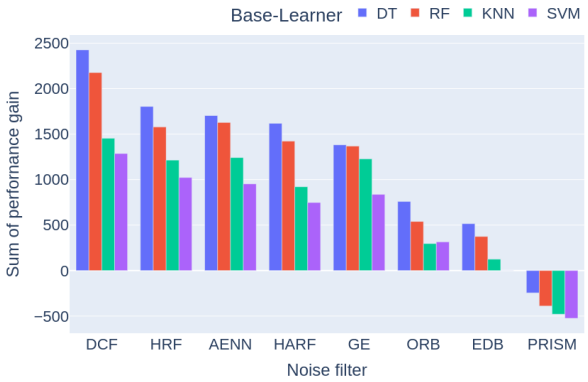

Figure 6. Plot containing the sum of the gain in all data sets after the application of filters.

mance gain despite being the one with the highest number of negative results. This suggests that ORB would probably benefit from an algorithm recommendation technique.

Finally, we also verified the class distributions in the four meta-bases used in the MtL-Multi approach. Figure 8 displays the frequency with which each class was ranked as the best across all four MtL-Multi meta-bases considering all noisy datasets. We observed that none of the meta-bases had a balanced distribution of classes, where DCF, ORB, and AENN were the dominant classes in their respective meta-base. Moreover, we found that the distance-based filters had worse overall performance when compared with both voting and ensemble filters.

4.1 MtL-Rank Results

After analyzing the effects of the filters on the performance metric, we started to examine the MtL approaches. We first evaluated the MtL-Rank approach, verifying its overall performance gain and accuracy. In this evaluation, we used three ML regressors to generate the ranks: RF (MtL-Rank-RF), GB (MtL-Rank-GB), and KNN (MtL-Rank-KNN).

Figure 9 presents the total performance gain obtained by the MtL-Rank-RF, MtL-Rank-GB, MtL-Rank-KNN compared to DCF as the baseline. Both MtL-Rank-RF and MtL-Rank-GB outperformed the baseline, showing that the approach can provide actual performance gain. To verify if the results were statistically significant, we conducted the Friedman-Nemenyi test[Demšar, 2006]. Figure 10 shows the results of the Nemenyi test, where each dot represents the value obtained by the test, and each line indicates its respective critical distance. The results revealed that only two methods were not significantly different: MtL-Rank-GB and DCF, while using DT as the base-learner.

When evaluating the MtL-Rank accuracy, since we are returning the results as a rank, to consider a classification correct, we used a top-k with $K = 1$ and $k = 3$ method to evaluate it. Figures 11 and 12 present the accuracy considering the top-1 and top-3 method, respectively, comparing it with two baselines, the DCF and ORB filters. The results showed that the MtL-Rank approach is always better than at least one baseline when considering the top-1, while the top-3 analysis managed to achieve up to 87% accuracy, better than all baselines, with the MtL-Rank-RF.

Finally, to evaluate the ranks produced by the MtL-Rank, we calculated Spearman's rank correlation between the rank acquired by them and the optimal rank. The best ranker was MtL-Rank-RF acquiring 0.57, 0.55, 0.58, and 0.52 correlation when using the DT, KNN, RF, and SVM base-learners respectively, MtL-Rank-GB got 0.56, 0.51, 0.56, and 0.47,

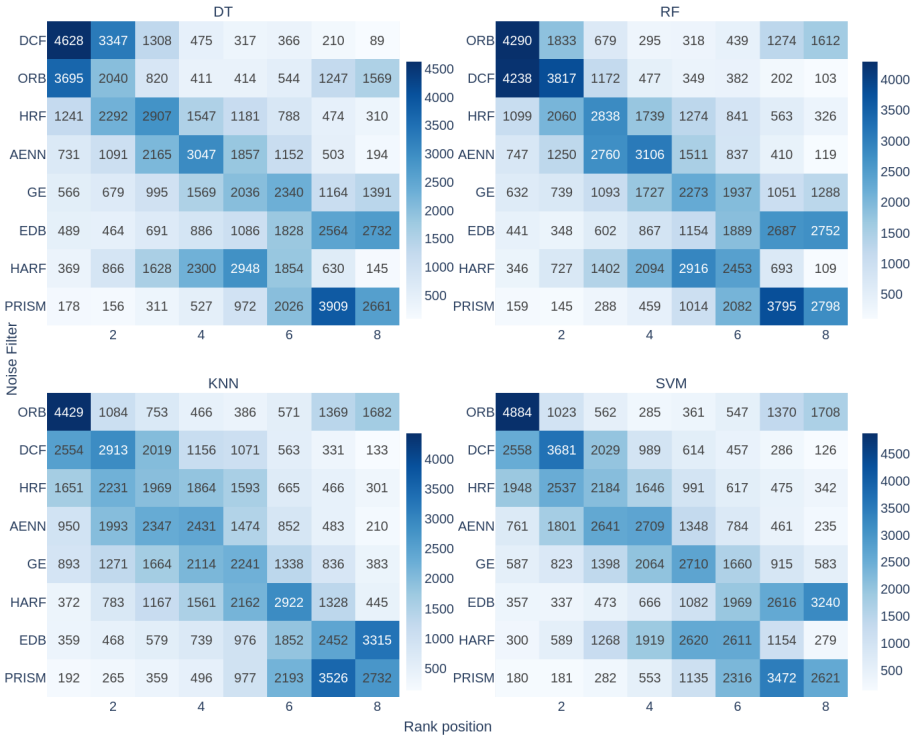

Figure 7. Times the filters had a positive or negative result in the performance metric.

Figure 8. MtL-Multi meta-bases classes distributions.

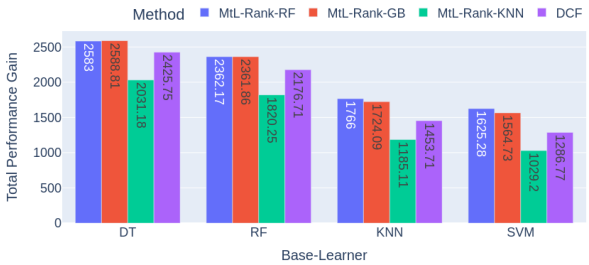

Figure 9. MtL-Rank performance gain compared with DCF baseline.

while MtL-Rank-KNN got 0.43, 0.40, 0.44, and 0.40. It is important to notice that the poor performance of the MtL-Rank-KNN may be due to the high number of MF suffering with the *curse of dimensionality*.

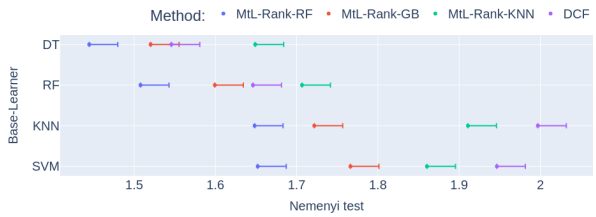

Figure 10. Results of the Friedman-Nemenyi for the MtL-Rank approaches and DCF baseline performance gain.

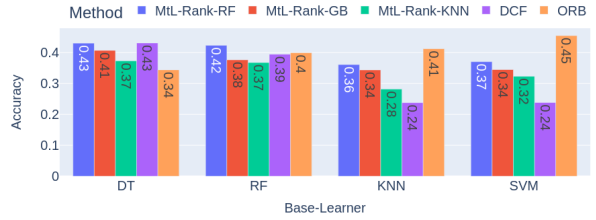

Figure 11. MtL-Rank accuracy compared with DCF and ORB baselines when considering the top-1 positions of the rank as correct.

4.2 MtL-Multi Results

Similarly to the MtL-Rank, to evaluate the MtL-Multi approach, we used three meta-classifiers: RF (MtL-Multi-RF), GB (MtL-Multi-GB), and KNN (MtL-Multi-KNN). Once again, each is analyzed based on its performance gain and accuracy.

Figure 13 displays the total performance gain obtained by the MtL-Multi-RF, MtL-Multi-GB, MtL-Multi-KNN compared to the DCF filter as the baseline. The results indicate that the MtL-Multi approaches did not manage to obtain

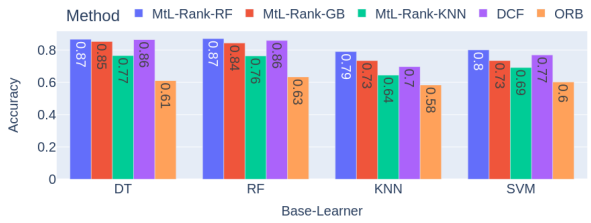

Figure 12. MtL-Rank accuracy compared with DCF and ORB baselines when considering the top-3 positions of the rank as correct.

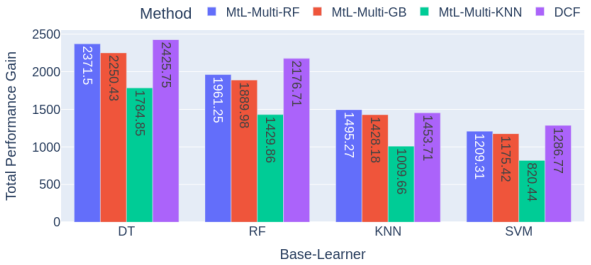

Figure 13. MtL-Rank performance gain compared with DCF baseline.

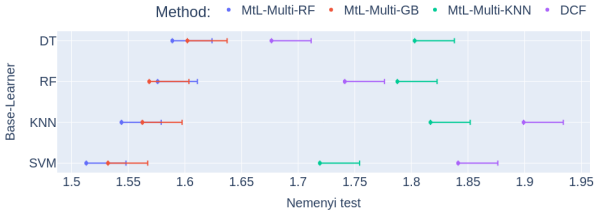

Figure 14. Results of the Friedman-Nemenyi for the MtL-Multi approaches and DCF baseline performance gain.

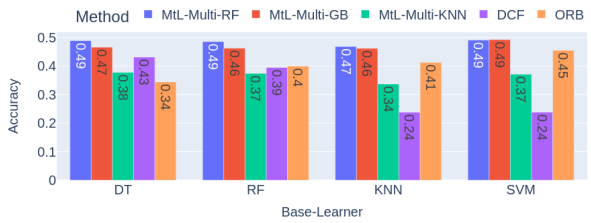

Figure 15. MtL-Rank accuracy compared with DCF and ORB baselines.

higher performance gains than the baselines. Furthermore, after applying the Friedman-Nemenyi test, Figure 14, both MtL-Multi-RF and MtL-Multi-GB were not significantly different from each other.

However, even though the MtL-Multi did not provide a performance gain when compared with the DCF, both MtL-Mult-RF, and MtL-Mult-GB obtained a higher accuracy than all baselines. Figure 15 presents the accuracy obtained by each MtL-multi compared with the two baselines. The results showed that the solution acquired an accuracy of up to 49%, higher than that obtained by the top-1 MtL-Rank and any baseline.

4.3 Discussions

The comparison between the two approaches reveals that each one has its advantages. The MtL-Rank approach obtained a higher performance gain, while the MtL-Multi achieved higher accuracy in its predictions. This difference can be attributed to the training goal of each methodology. Since MtL-Rank trains and forms its ranks with meta-regressors, it is optimized to always look for higher performance gain. Conversely, the MtL-Multi is trained with classifiers, which are optimized to find the correct label, resulting in an emphasis on accuracy during training.

Another factor that may have influenced the results is the combinations of both P and A spaces. When we analyzed the filters' performances, we observed that ORB was the algorithm that was most frequently ranked as the best but also had a higher number of negative results in terms of performance. This means that when the recommendation method mistakenly suggests this filter, it has a higher chance of reducing the total performance gain, which is one of the metrics that we utilized to analyze the results. Needless to say, since MtL-Multi was optimized for higher accuracy, it suggested the ORB more frequently than the MtL-Rank approach.

Finally, we decided to verify the importance of each MF for the MtL. Thus, we selected the approach with the highest overall performance gain, which was MtL-Rank-RF, and examined the feature importance for each base learner. The results are presented in Figure 16, which shows the 20 most important features in the RF regression for each base learner. In total, 30 MFs[6] with 11 belonging to the statistical group, 7 to the information-theoretic group, 5 being landmarking features, 4 being model-based, and only 3 being simple. Usually, the most important feature is mutual information, with the exception of when the SVM was the base-learner, in this case, kurtosis had more influence in the regression.

5 Conclusions

In this work, we presented and compared two methodologies to recommend noise filtering algorithms with MtL. The first one, MtL-Rank, presents the suggestion in the form of a rank of the best algorithms, while the other one, MtL-Multi, performs the recommendation in two steps, first defining the filters type and then deciding what algorithm of the selected type should be selected. Our goal was to identify the algorithm that generated the most gain over the f1-score after the execution of a base-learner. The results showed that MtL-Rank approaches managed to obtain higher performance gain than the MtL-Rank and the baseline. However, when considering the recommendation accuracy, MtL-Multi was the best choice, achieving up to 49% of accuracy, better than any baselines or the top-1 results of MtL-Rank approaches. It is worth noticing that, taking into account the top-3 results, the MtL-Rank, we acquired up to 87% accuracy.

For future works, we intend to expand the process to support hyperparameters recommendations, which could increase the solution search space and provide better results. It

[6]The description of each MF can be found at `https://pymfe.readthedocs.io/en/latest/auto_pages/meta_features_description.html`

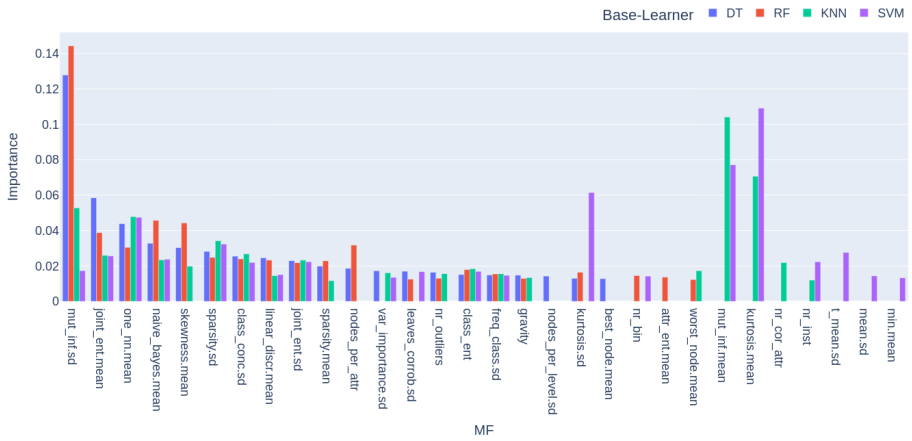

Figure 16. 20 most important features for each base-learner for MtL-Rank-RF meta-regressor.

would also be interesting to study different sets of MF. Even though we presented a collection of MF who was more influential in the regression, this process could be further analyzed to find an optimal MF set. Finally, those methodologies could be expanded to other types of preprocessing techniques, such as missing values imputation, feature selection, or data balancing, resulting in a more general preprocessing recommendation system.

Authors' Contributions

Pedro B. Pio contributed by performing the experiments, writing and reviewing the study. Adriano Rivolli and André C. P. L. F. de Carvalho contributed to writing and reviewing the study. Luís P. F. Garcia contributed by supervising the experiments, writing, and reviewing the study. All authors read and approved the final manuscript.

Availability of data and materials

The list of datasets and MF used can be found at `https://bit.ly/JIDM_Datasets` and `https://bit.ly/JIDM_MF` respectively.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

Alcobaça, E., Siqueira, F., Rivolli, A., Garcia, L. P. F., Oliva, J. T., and de Carvalho, A. C. P. L. F. (2020). Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(1):4503–4507. DOI: 10.5555/3455716.3455827.

Bilalli, B., Abelló, A., Aluja-Banet, T., and Wrembel, R. (2019). Presistant: Learning based assistant for data preprocessing. *Data & Knowledge Engineering*, 123:1–22. DOI: 10.1016/j.datak.2019.101727.

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. (2009). *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer, Berlin, Heidelberg, 1 edition. DOI: 10.1007/978-3-540-73263-1.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32. DOI: 10.1023/a:1010933404324.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017). *Classification and regression trees*. Routledge, New York, NY. DOI: 10.1201/9781315139470.

Cawley, G. C. and Talbot, N. L. (2003). Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers. *Pattern Recognition*, 36(11):2585–2592. DOI: 10.1016/S0031-3203(03)00136-5.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20:273–297. DOI: 10.1007/bf00994018.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30. DOI: 10.5555/1248547.1248548.

Famili, A., Shen, W.-M., Weber, R., and Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent data analysis*, 1(1):3–23. DOI: 10.1016/S1088-467X(98)00007-9.

Fayyad, U. M., Haussler, D., and Stolorz, P. E. (1996). Kdd for science data analysis: Issues and examples. In *Second International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 50–56, Portland, OR. AAAI Press. DOI: 10.5555/3001460.3001471.

Frénay, B. and Verleysen, M. (2014). Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869. DOI: 10.1109/TNNLS.2013.2292894.

Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139. DOI: 10.1006/jcss.1997.1504.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Symposium on Knowledge Discovery, Mining and Learning*, 29(2):1189–1232. DOI: 10.1214/aos/1013203451.

Garcia, L. P., de Carvalho, A. C., and Lorena, A. C. (2016a). Noise detection in the meta-learning level. *Neurocomputing*, 176:14–25. DOI: 10.1016/j.neucom.2014.12.100.

Garcia, L. P., Lorena, A. C., Matwin, S., and de Carvalho, A. C. (2016b). Ensembles of label noise filters: a ranking approach. *Data Mining and Knowledge Discovery*, 30(5):1192–1216. DOI: 10.1007/s10618-016-0475-9.

Garcia, L. P. F., Lorena, A. C., and Carvalho, A. C. (2012). A study on class noise detection and elimination. In *Brazilian Symposium on Neural Networks (BRACIS)*, pages 13–18. DOI: 10.1109/SBRN.2012.49.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*, volume 72. Springer, Cham, Switzerland, 1 edition. DOI: 10.1007/978-3-319-10247-4.

Gupta, S. and Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474. DOI: 10.1016/j.procs.2019.11.146.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature, Cham, Switzerland. DOI: 10.1007/978-3-030-05318-5.

Karmaker, A. and Kwek, S. (2006). A boosting approach to remove class label noise. *International Journal of Hybrid Intelligent Systems*, 3(3):169–177. DOI: 10.1109/ICHIS.2005.1.

Koplowitz, J. and Brown, T. A. (1981). On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3):251–255. DOI: 10.1016/0031-3203(81)90102-3.

Miranda, A. L., Garcia, L. P. F., Carvalho, A. C., and Lorena, A. C. (2009). Use of classification algorithms in noise detection and elimination. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 417–424. DOI: 10.1007/978-3-642-02319-4_50.

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill series in computer science. McGraw Hill, New York, NY.

Morales, P., Luengo, J., Garcia, L. P., Lorena, A. C., de Carvalho, A. C., and Herrera, F. (2017). The noisefiltersr package: Label noise preprocessing in r. *The R Journal*, 9(1):219–228. DOI: 10.32614/RJ-2017-027.

Munson, M. A. (2012). A study on the importance of and time spent on different modeling steps. *ACM SIGKDD Explorations Newsletter*, 13(2):65–71. DOI: 10.1145/2207243.2207253.

Nagarajah, T. and Poravi, G. (2019). A review on automated machine learning (automl) systems. In *5th International Conference for Convergence in Technology (I2CT)*, pages 1–6, Bombay, India. IEEE. DOI: 10.1109/I2CT45611.2019.9033810.

Parmezan, A. R. S., Lee, H. D., Spolaôr, N., and Wu, F. C. (2021). Automatic recommendation of feature selection algorithms based on dataset characteristics. *Expert Systems with Applications*, 185:115589. DOI: 10.1016/j.eswa.2021.115589.

Pio, P. B., Garcia, L. P., and Rivolli, A. (2022). Meta-learning approach for noise filter algorithm recommendation. In *X Symposium on Knowledge Discovery, Mining and Learning*, pages 186–193. SBC. DOI: 10.5753/kdmile.2022.227958.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118. DOI: 10.1016/S0065-2458(08)60520-3.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J., and de Carvalho, A. C. (2022). Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101. DOI: 10.1016/j.knosys.2021.108101.

Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: a modern approach*. Pearson, Prentice Hall Upper Saddle River, NJ, USA, 3 edition. DOI: 10.5555/1671238.

Sluban, B., Gamberger, D., and Lavrač, N. (2014). Ensemble-based noise detection: noise ranking and visual performance evaluation. *Data Mining and Knowledge Discovery*, 28(2):265–303. DOI: 10.1007/s10618-012-0299-1.

Smith, M. R. and Martinez, T. (2011). Improving classification accuracy by identifying and removing instances that should be misclassified. In *International Joint Conference on Neural Networks*, pages 2690–2697. DOI: 10.1109/IJCNN.2011.6033571.

Smith-Miles, K. A. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25. DOI: 10.1145/1456650.1456656.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(6):448–452. DOI: 10.1109/TSMC.1976.4309523.

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., and Farivar, R. (2019). Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In *31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1471–1479, Portland, OR. IEEE. DOI: 10.1109/ICTAI.2019.00209.

Vanschoren, J. (2019). Meta-learning. In *Automated Machine Learning*, pages 35–61. Springer Nature, Cham, Switzerland. DOI: 10.1007/978-3-030-05318-5_2.

Wheway, V. (2001). Using boosting to detect noisy data. In *Pacific Rim International Conference on Artificial Intelligence*, pages 123–130. DOI: 10.1007/3-540-45408-X_13.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):408–421. DOI: 10.1109/TSMC.1972.4309137.

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *4th International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, New York, NY. AAAI Press.

Zar, J. H. (2014). Spearman rank correlation: overview. *Wiley StatsRef: Statistics Reference Online*. DOI: 10.1002/9781118445112.stat05964.

Zhu, X. and Wu, X. (2004). Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210. DOI: 10.1007/s10462-004-0751-8.