Noise filter with hyperparameter recommendation: a meta-learning approach

Pedro B Pio[1] **, Adriano Rivolli**[2] **, André C. P. L. F. de Carvalho** [3] **, Luís P. F. Garcia** [1]

[1]Department of Computer Science
University of Brasília (UnB) – Brasília, DF – Brazil

[2]Universidade Tecnológica Federal do Paraná (UTFPR)
Cornélio Procópio, PR – Brazil

[3]Institute of Mathematical and Computer Sciences
University of São Paulo (USP) – São Carlos, SP – Brazil

pedrobpio@gmail.com, rivolli@utfpr.edu.br, andre@icmc.usp.br,

luis.garcia@unb.br

Abstract. *Applying Machine Learning (ML) algorithms to a dataset can be time-consuming. It usually involves, not only selecting and fine-tuning the algorithm, but also other steps, such as data preprocessing. To reduce this time, the whole or a subset of this process has been automated by Automated ML (AutoML) techniques, which can include Bayesian Optimization, Genetic Programming, and Meta-Learning techniques. However, despite it often being a necessary stage, preprocessing is commonly not well handled in AutoML tools. In this work, we propose and experimentally investigate the use of meta-learning to recommend noise detection algorithms and the values for their hyperparameters. The proposed approach produces a ranking of the best noise filters for a given dataset, reducing the development cost of ML-based solutions and improving their predictive performance. To validate the process, we generated* 10740 *noisy datasets, which we describe using* 97 *meta-features. For each dataset, we applied* 8 *noise filters, which increased to* 27 *when we added variations of hyperparameter values. Next, we applied* 4 *ML algorithms to this data and created a performance ranking, which we used as a meta-target to induce* 3 *meta-regressors. We compared these* 3 *meta-regressors and the results with and without hyperparameters for the noise filters. According to the experimental results, the introduction of hyperparameter recommendation resulted in a higher gain in the F1-Score performance metric. However, it came at the cost of lower accuracy in the Top-K ranking evaluation.*

1. Introduction

The design of a good solution for a data analysis problem using a Machine Learning (ML) algorithm is not a simple task. There are several methodologies developed to guide users to extract information from datasets based on those algorithms, such as the Cross-Industry Standard Process for Data Mining (CRISP-DM) [Martínez-Plumed et al. 2019] or the Knowledge Discovery in Databases (KDD) [Fayyad et al. 1996]. Despite their differences, these methodologies share fundamental steps, including preprocessing, algorithm selection, and algorithm hyperparameters optimization.

In recent years, significant efforts have been made to enhance the accessibility of ML [Nguyen et al. 2019]. This progress is evident in the development of ML frameworks such as Weka, Scikit-learn, H2O, and Tensorflow, which provide users with powerful tools and libraries to implement ML algorithms. Another notable advancement is the emergence of Automated ML (AutoML), a field dedicated to automating the entire ML process, including selecting suitable preprocessing techniques, ML algorithms, and their hyperparameters [Truong et al. 2019]. AutoML systems may rely on several methods to build the ML workflow, where, among the most common are: Meta-Learning (MtL), Bayesian Optimization, and Genetic Programming [Nagarajah and Poravi 2019]. In this work, we will be focusing on MtL approaches.

Often referred to as learning to learn, MtL encompasses the process of acquiring knowledge from previous tasks or experiences [Vanschoren 2019]. Within the context of AutoML, MtL is employed for algorithm selection. It utilizes a set of features known as Meta-Features (MFs). These MFs capture essential characteristics of the data, allowing the system to extract valuable knowledge and determine the most appropriate algorithm for a given task [Brazdil et al. 2009].

Despite the crucial role of preprocessing in the ML pipeline [García et al. 2015], AutoML systems often struggle to handle this step effectively [Truong et al. 2019]. In this work, we are using MtL to suggest the most suitable Noise Filter (NF) algorithms, a preprocessing technique that aims to find noisy instances on the data, which commonly appear in the data, and may reduce the performance of the classifiers [Frénay and Verleysen 2013].

In this work, we present a MtL approach to suggest the most suitable NF and their hyperparameters, evaluating the effects of the NFs and their hyperparameters over several datasets and comparing the MtL results with the same approach when the hyperparameter recommendations are not applied. Our solution produces a rank ordering of the best filters, which provides more flexibility for the user and is easily adaptable to different types of preprocessing algorithms. As far as we know, this is the first study including hyperparameters recommendation and noise detection algorithms with MtL.

The rest of this paper is divided as follows: Section 2 presents the background and some related works on the topic; Section 3 introduces the proposed methodology; Section 4 shows the results of the experiments; and Section 5 concludes the work and presets the future works.

2. Background and Related works

In this section, we present the theoretical background needed for the problem of algorithm recommendation with MtL and NF. Afterward, we show a few approaches covering it in the literature with some related works.

2.1. Meta-learning and algorithm selection

Commonly known as learn how to learn, MtL, refers to any type of learning based on previous experiences with other tasks, which include: learning from model evaluations, where the learning process is based on previous task evaluations; learning from task proprieties, where proprieties called MFs, containing the task knowledge, are extracted from

the task to enable the learning process; and learning from prior models, which uses previous models structures and parameters to learn [Vanschoren 2019].

Considering the different types of tasks, Brazdil et al. [Brazdil et al. 2022] provide a list of some MtL tasks, such as algorithm selection, hyperparameter optimization, workflow or pipeline synthesis, architecture search, and few-shot learning. In this work, we are focusing on MtL from Tasks proprieties for algorithm selection and hyperparameter optimization. The origin of this approach can be traced to Rice's algorithm selection problem (1976) [Rice 1976]. Rice, being one of the first to formalize this problem, divided it into four spaces:

- Problem space (P): space containing the problem $x \in P$ to be solved;
- Feature space (F): features extracted from P by a function $f(x)$;
- Algorithm Space (A): available algorithms $a \in A$ that can solve problem P;
- Performance Space (Y): metrics $y \in Y$ used to evaluate the solution.

With the four spaces, Rice proposed that, from the set of features F, it was possible to map the algorithms $a \in A$ according to a performance metric $y \in Y$ based on a function $S(f(x))$. Ideally, the function S would return the best algorithm a to solve the problem x.

Smith-Miles, K (2008) [Smith-Miles 2008] expanded Rice‘s algorithm selection problem with a framework that introduced the concept of MtL to it. Smith-Miles divided the framework into three phases: in the first phase, the meta-data, also called meta-base, is created. From P we extract F, execute all algorithms $a \in A$ to extract the metric $y \in Y$, and create a meta-base; in the second phase, we apply empirical rules to implement the algorithm selection; finally, in the last step, we perform theoretical support to the empirical rules and refine the algorithms set.

Another crucial aspect of MtL is the MFs, which contain the information used to solve the task [Vanschoren 2019]. Usually, the MFs are divided into classes. However, sometimes they are not clearly delimited. Thus, in this work, we follow the class division proposed by Rivolli et al. [Rivolli et al. 2022] where the MFs are divided into the following groups: Simple, including simple features such as the number of classes; Statistical, which are calculated from statistics; Information-theoretic, belonging to the information theory field; Model-based, extracted from ML models structures and parameters; Landmarking, calculated from ML performances; and Others which include features that do not fit in the other groups.

Moreover, MtL can be divided into two levels: the meta level and the base level. At the base level, the goal is to study what is used to create the meta-base, including the A, F, and Y spaces. At the meta level, the meta-base is evaluated and the MtL task is implemented [Brazdil et al. 2009].

2.2. Noise Detection

Noise detection is a type of preprocessing. Therefore, following Familli's definition [Famili et al. 1997], it is a transformation on a dataset that solves a problem found in the data, consequently making the dataset more useful for its application. When considering noise detection, the intention is to correct the mistakes that may appear on datasets, which are called noise [Frénay and Verleysen 2013].

There are mainly two kinds of noise[1], the attribute and class noise. However, class noise usually produces a larger impact on ML algorithms performance [Zhu and Wu 2004]. For that reason, this work focuses on class noise. This type of noise can appear in three forms [Frénay and Verleysen 2013]: *completely at random*, where it occurs evenly over the classes and does not depend on any attribute; *at random*, where it is also independent, however, it may not occur evenly over the classes; and *not at random* where the frequency of the noise may be influenced by other factors such as a specific attribute on the data.

Finding noise in datasets is not trivial and can often involve applying one or more ML algorithms [Gupta and Gupta 2019]. After the noise is found, we should correct or remove it from the dataset. This process can be performed by several NF, which may be classified as [Frénay and Verleysen 2013]: (i) Classification filters; (ii) distance-based filters; (iii) voting filters; (iv) ensemble or boosting filters.

In this work, we selected eight NFs to compose the algorithms set, of which three belong to the distance-based class, three were voting algorithms, and two were ensemble filters:

- High Agreement Random Forest (HARF) [Sluban et al. 2014]: a voting filter is based on the Random Forest (RF) algorithm, where each tree that forms the RF is granted a vote regarding the class of an instance. If the total of votes is not higher than an agreement level, the instance is considered noisy and is removed;
- Dynamic Classification Filter (DCF) [Garcia et al. 2012]: a voting algorithm executes nine classifiers, k-Nearest Neighbors (KNN) with k = 3, 5, and 7, Support Vector Machine (SVM), two Decision Trees (DT) implementations CART and C4.5, RF, Naive Bayes (NB), and MultiLayer Perceptron (MLP). A number between $1 \leq m \leq 9$ of algorithms is selected based on their result similarities, and each one votes if it is a noisy instance. If the majority votes yes, the instance is removed;
- Hybrid Repair-Remove Filter (HRF) [Miranda et al. 2009]: a voting algorithm that uses four ML algorithms, KNN, MLP, SVM, and CART. Each one vote if the instance should be classified as noise, if the majority votes as yes, the instance is removed or altered depending on the KNN vote;
- Outlier Removal Boosting Filter (ORB) [Karmaker and Kwek 2006]: an ensemble algorithm that uses the tendency of overfitting of the Adaboost algorithm to determine if an instance is noisy. After each iteration of the algorithm, it tracks the weight of the instances. If the weight is higher than a threshold d, the instance is considered as noisy and is removed;
- Edge Boosting Filter (EDB) [Wheway 2001]: an ensemble filter that also uses the Adaboost algorithm, where, after m iterations of the algorithm, it computes the edge value of each instance. If the value is above a threshold t, and the quantity of removed instances is not higher than a predefined maximum percentage of instances, the instance is removed;
- Generalized Edition (GE) [Koplowitz and Brown 1981]: a distance-based algorithm that is a variation of the Edited Nearest Neighbour (ENN) algorithm. Considering the $k-1$ instance nearest neighbors and a threshold k', if the instance has

[1]Considering tabular data.

at least k' neighbors of the same class, it is relabeled as that class. Otherwise, the instance is removed;

- All-k Edited Nearest Neighbors (AENN) [Tomek 1976]: this distance-based algorithm consists of running the ENN algorithm k times, with k varying from 1 to k. If the instance is considered as noise for any of these executions, the instance is considered as noisy and is removed;
- Preprocessing Instances that Should be Misclassified (PRISM) [Smith and Martinez 2011]: a distance-based filter that computes five heuristics: one distance based, two likelihood-based, and two extracted based on the C4.5 leaves. Those heuristics are evaluated based on a function[2]. If it satisfies the proposed function, the instance is removed.

2.3. Related Works

Leyva et al. (2013) [Leyva et al. 2013] presents a framework for recommending Instance Selection (IS) algorithms by predicting both the accuracy obtained after applying the KNN algorithm and the number of instances that will be removed according to the user's preference. The study uses a total of 40 datasets obtained from the KEEL platform, eight MFs, six IS algorithms to be recommended, and six algorithms serving as meta-regressors to evaluate which one performs the best. The approach showed good results considering the computational cost of executing the recommendation.

Garcia et al. (2016) [Garcia et al. 2016a] conducted the selection of algorithms to recommend noise detection algorithms. They used 53 datasets from UCI and KEEL to create the meta-base, in which noise was generated in a quantity between 5% to 20% of the total instances. 70 MFs were extracted from each dataset to build the meta-base and apply the meta-regressors. Six filters were selected as the algorithm set, and three regression algorithms, KNN, RF, and SVM, were used to predict how well each filter could identify the noise on the datasets. When analyzing the Mean Square Error of the regressors and comparing them with the best general filter algorithm and random algorithm choices, they concluded that RF achieved the best performance.

Garcia et al. (2016) [Garcia et al. 2016b] presents an MtL system that seeks to recommend the noise detection algorithm that achieves the best accuracy in removal. The study uses five different noise reduction algorithms to form 26 ensemble combinations that are evaluated and used in the recommendation. They used 90 datasets to create the meta-data, which were modified randomly generating 5%, 10%, 20%, and 40% of noise on each one. A total of 70 MFs are used to classify the best algorithm ensemble, and five ML techniques are compared to perform meta-classification. The best classifier was DT, which achieved a classification accuracy close to 75%.

Pio et al. (2022) [Pio et al. 2022] recommend noise detection algorithms by predicting and identifying the algorithm that generates the most gain in the F1-Score metric. The system uses 73 MFs extracted from 323 different datasets that were modified to contain 5%, 10%, 20%, and 40% of noisy instances. Three NFs were used: HARF, ORB, and GE. Three base-learners were selected to calculate performance metrics: DT, RF, and KNN. And three regressors were used to build the ranking recommended algorithms: RF, PCT, and KNN. The results showed that it was possible to improve performance by

[2]More details can be found at [Smith and Martinez 2011].

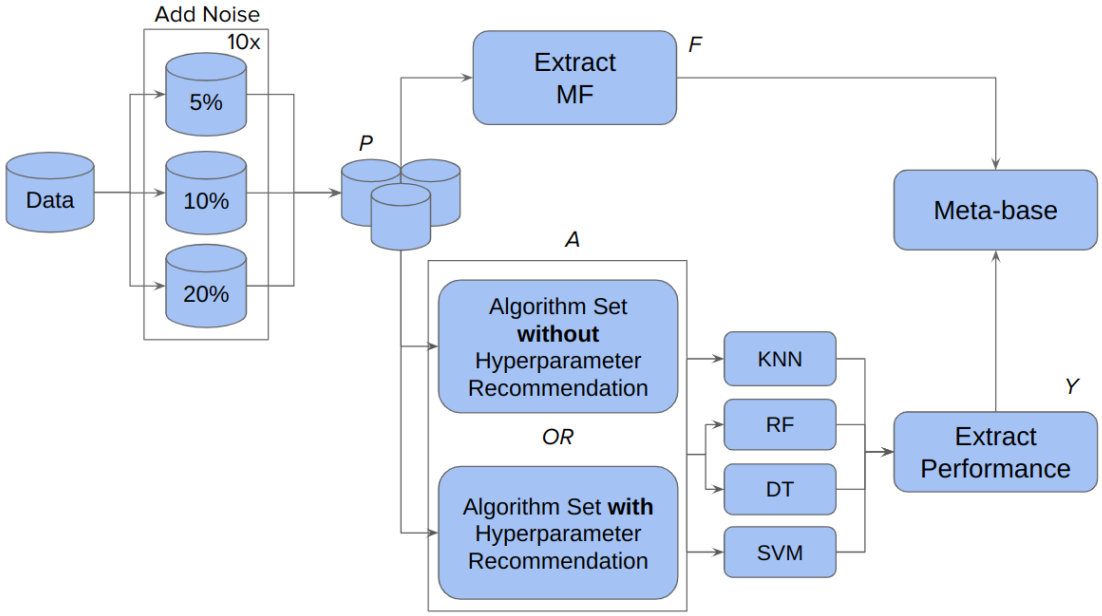

Figure 1. Diagram presenting the process used to create each meta-base.

always choosing the algorithm present in the first position of the rank, with RF as the meta-regressor that obtained the best results.

This work extends the solution proposed by Pio et al. (2022) [Pio et al. 2022], incorporating hyperparameter suggestion into the process. While previous studies have explored MtL-based hyperparameter recommendation [Parmezan et al. 2021, de Morais et al. 2016], we believe we are the first to implement it for noise detection and compare its performance against an approach without hyperparameter optimization. Also, only our and [Pio et al. 2022] solutions present the recommendation as a ranking, providing more flexibility in selecting which algorithm better suits a specific problem, where the users can either choose the first position in the rank or try others based on their individual needs and preferences.

3. Methodology

In this work, the methodology is divided into two levels: the base level and the meta level. At the base level, our goal is to build the meta-base, which means constructing the P spaces and extracting F and Y. At the meta level, we utilize the meta-base to learn, perform recommendations, and evaluate the results.

Figure 1 shows a diagram of the base level starting from the data extraction to the creation of the meta-base. The process begins with the datasets that we extracted from OpenML[3] platform, which was chosen due to the number of datasets available and the possibility to filter the dataset based on features such as the number of classes. To reduce the complexity of the experiment, we decided to limit the number of instances and the number of attributes of the datasets, only extracting the sets with 100 to 20000 instances, 3 to 100 attributes, and with two classes. Although we are limiting the size and the number

[3] https://www.openml.org/

of classes of the datasets, the entire methodology can be applied without it. Where larger datasets would increase the computational time to build the meta-base and work with more than two classes would require a set of NFs and ML algorithms that support more than two classes. We also removed the datasets that had missing values, which reduced the amount of preprocessing needed to compute the MFs and the performance metrics. After extracting the data from the platform, since some of our NFs and ML algorithms only support numerical attributes, we implemented a simple preprocessing step to replace the categorical attributes for dummy variables. Finally, we removed the datasets with more than 200 attributes. This data extraction and preprocessing step resulted in 358 datasets[4].

To ensure that a percentage of noise is always present in our datasets, we generate it artificially and completely at random. Since we are working only with binary datasets, this process consists of randomly selecting instances and changing their class. We selected 5%, 10%, and 20% of the instances. To avoid bias, we repeated this sampling ten times for each percentage, creating a total of 30 noisy datasets per original dataset. The result is the P space, formed by the 10740 noisy datasets and used to compute the F1-Score performance metric Y and the MFs F.

To compute F1-Score, we first executed the NFs, which compose the A space. Afterward, we run ML classification algorithms called base-learner. With the results of the base-learner classification, we compute the F1-Score performance metric. In this process, we selected four base-learners[5]: KNN, RF, DT, and SVM algorithms, each resulting in a unique meta-base. Furthermore, to compare the effects of hyperparameter variations on the algorithms and allow its recommendations, we created two A spaces, one with and the other without hyperparameter recommendation, resulting in a total of eight meta-bases.

Each A space is composed by $(a_i, \lambda_{ij}) \in A$, where a_i is the algorithm and λ_{ij} is a set of hyperparameters that is acceptable by a_i. Meaning we considered each combination of NFs and its hyperparameter as a unique algorithm. We selected eight NFs [6]: HARF; HRF; DCF; EDB; ORB; PRISM; AENN; and GE. The hyperparameters used for each filter are presented in Table 1, where the parameters in bold are the default values, and the ones with more than one value listed are the ones we varied for optimization. Therefore, the A space that only used the default parameters is composed of eight algorithms, while the one with hyperparameters variations is formed by 27. It is important to emphasize that PRISM, HRF, and DCF were not optimized, where HRF and DCF were due to the computational cost of the filter while PRISM implementation did not offer any hyperparameter to optimize.

To create the set of features F, we used the `pymfe` package which allowed us to extract simple, statistical, information-theoretic, model-based, and landmarking MFs. We extracted a total of 97 MFs[7]. With both F and Y, we can build the meta-bases which use the F as attributes and the Y as the target.

The meta-bases are then used at the meta level to build a ranking of the best algorithms. The rankings are created based on the result obtained by the regressors, which

[4]The list of datasets can be found at `https://bit.ly/Datasets_ENIAC_2023`

[5]The base-learners were implemented with `scikit-learn` default values.

[6]All filters were implemented with the `NoisefiltersR` package.

[7]The list of MFs used can be found at `https://bit.ly/MF_ENIAC_2023`

Table 1. Table containing the NFs and their hyperparameters configurations, the default values are presented in bold, and the parameters with more than a value were the ones that were being optimized.

NF	Hyperparameters
HARF	AgreementLevel = **70**; 75; 80; 85; 90 percent Number of trees= 500 nfolds = 10
HRF	Consensual voting Hybrid approach (remove or repair instances)
DCF	nfolds = 10 m = 3 Consensual voting
EDB	Quantity of iterations = 15 Removal percent = 0.05 threshold = 5; 10; **15**; 20
ORB	Quantity of iterations = 20 threshold d = 3; 7; **11**; 15; 19
PRISM	None
AENN	$k = 3, \mathbf{5}, 7, 9, 11$
GE	$k = 3, \mathbf{5}, 7, 9, 11$ k' = k/2

aims to predict the gain on the F1-Score generated by the execution of each NF based on the set of MFs. Since we generated 30 noisy datasets for each extracted dataset, during the training and validation, we used a variation of the leave-one-out cross-validation where, instead of validating one dataset separately, we grouped all sets of 30 datasets derived from the same dataset and validated them together. The predictions are then ordered according to the higher performance gain forming the ranking. We compared three regressors, called meta-regressors: RF, Gradient Boosting (GB), and KNN[8]. Each meta-regressor generated a ranking that was evaluated at the base and meta level.

At the base level evaluation, we compared the total gain in the F1-Score generated by the first position filter of each rank with a baseline filter, which is the filter that acquired a higher sum of F1-Score over all datasets. At the meta level evaluation, we compute the accuracy obtained by each rank with two baselines which were the filters that appeared most frequently at first position on an optimal ranking. However, since we are generating rankings of algorithms, to compute the accuracy, we have to define when the ranking was correct. We decided to follow the Top-K approach, where we would consider the ranking correct if the best filter was in the first K positions of it. Finally, we also decided to use Spearman's rank correlation to compare the rankings generated by the meta-regressors with the optimal ranking.

4. Results

Following the proposed methodology we generated all the noisy datasets, executed all NFs, and extracted the F1-Score performance metric for each base-learner. This allowed

[8]The meta-regressors were implemented with `scikit-learn` default values.

us to study the effects of each NF in the base-learner F1-Score. The meta-base analysis sought to identify the filters' frequency that led to the highest overall gain in the performance metric. Figure 2 presents the sum of the F1-Score gain over all noisy datasets after execution of each filter, where the filter name is followed by, if applicable, its hyperparameter variation value (for instance, AENN11 or GE11 indicates that $k = 11$). The results show that DCF was the filter that causes higher performance gain, followed by the variations of AENN. We also noted that the ensemble filters ORB and EDB usually induce lower gains, and PRISM was always the worst filter.

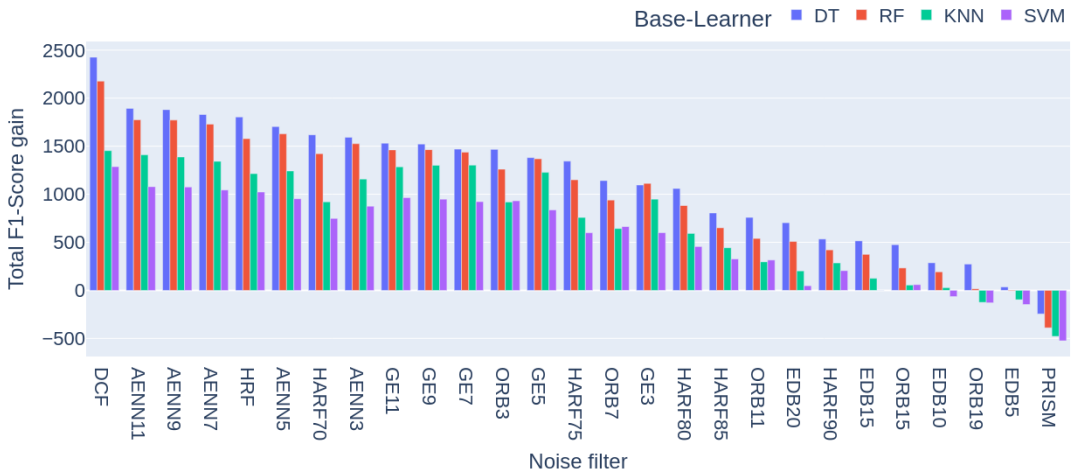

Figure 2. Total gain obtained on all datasets after applying each NF.

To review how often each NF appeared on each position, giving us more information on how well each filter performed, we ordered all the results creating an optimal ranking that shows the optimal order of algorithms for each dataset. The results revealed that the ORB filters appeared most often in the first position, and ORB3 was most frequently the best. However, those filters were also contradictorily ordered as one of the worst filters frequently. Also, when considering the set of algorithms without hyperparameter variations, the DCF is the filter found most times at the first position for both RF and DT base-learners.

To verify the frequency that each NFs resulted in a positive and negative result, we executed each NF, comparing the F1-Score performance metric in all base-learners with the result we acquired when no NF was applied. Figure 3 presents the number of times each filter generated a positive or negative result in the F1-Score on all base-learners. We note that HARF70 was the one with the most positive results, while ORB19 was the one with the most negative effects. In fact, the ensemble filters had a higher number of negative results, which explains that even being often ranked as the best filters, the ORB filters do not produce a higher overall performance gain, as shown in Figure 2.

After examining the impact of the filter on the performance metric, we started to evaluate the proposed MtL approach. First, we study the base level, comparing the F1-Score gain obtained by the MtL approach with the baseline. Figure 4 shows the total F1-Score gain of the MtL when we used the RF (MtL-RF), GB (MtL-GB), and KNN (MtL-KNN) regressors compared with the obtained performance gain when always selecting

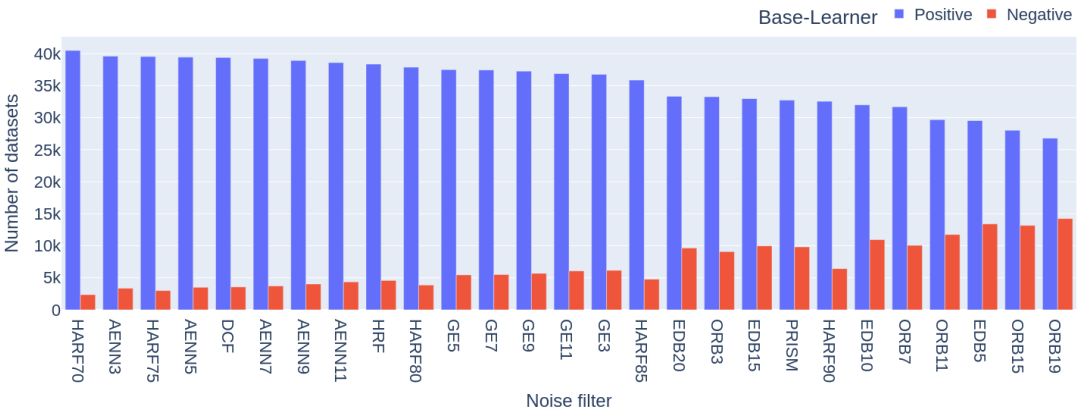

Figure 3. Times each filter resulted in positive or negative results in the performance metrics considering all base-learners.

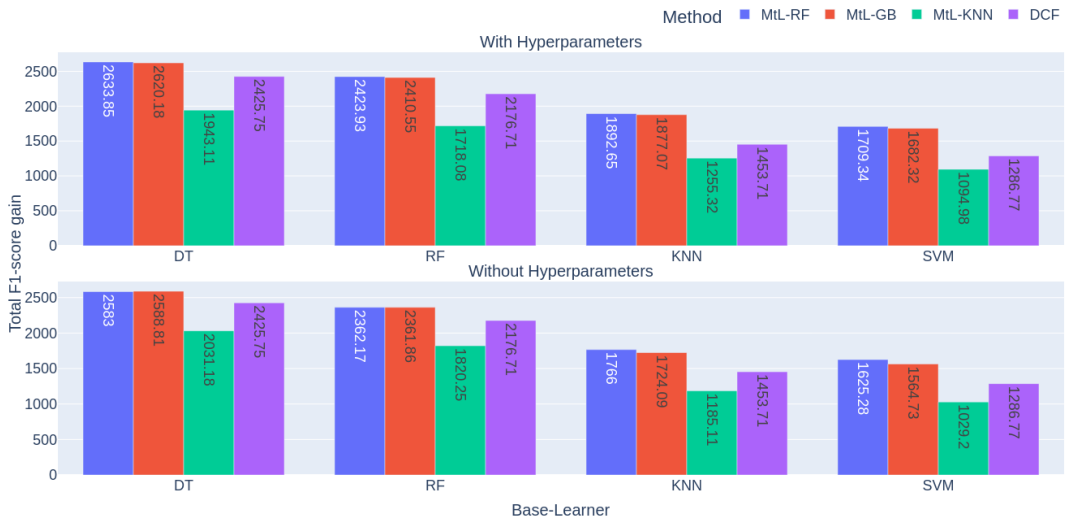

Figure 4. MtL approaches performance gain compared with the DCF baseline.

DCF filter as the baseline. The first row presents the outcomes using the hyperparameter recommendation, while the second shows the result when we only suggest the algorithms. We note that introducing the hyperparameter recommendation usually resulted in a performance gain when compared with the implementation without hyperparameter variations, the only exceptions being the MtL-KNN with the DT and RF base-learners. Nevertheless, both implementations obtained better results than the baseline when considering the MtL-RF and MtL-GB.

To verify if the base level results we obtained were statistically significant, we applied the Friedman-Nemenyi test [Demšar 2006]. Figure 5 illustrates the Nemenyi test obtained after discarding the null hypothesis. Each dot represents the result of the test, and the lines show its critical distance. If the lines overlap any dots, it means that the results were not significantly different from each other. The test showed that only the MtL-RF and DCF with hyperparameters and the MtL-GB and DCF without hyperparam-

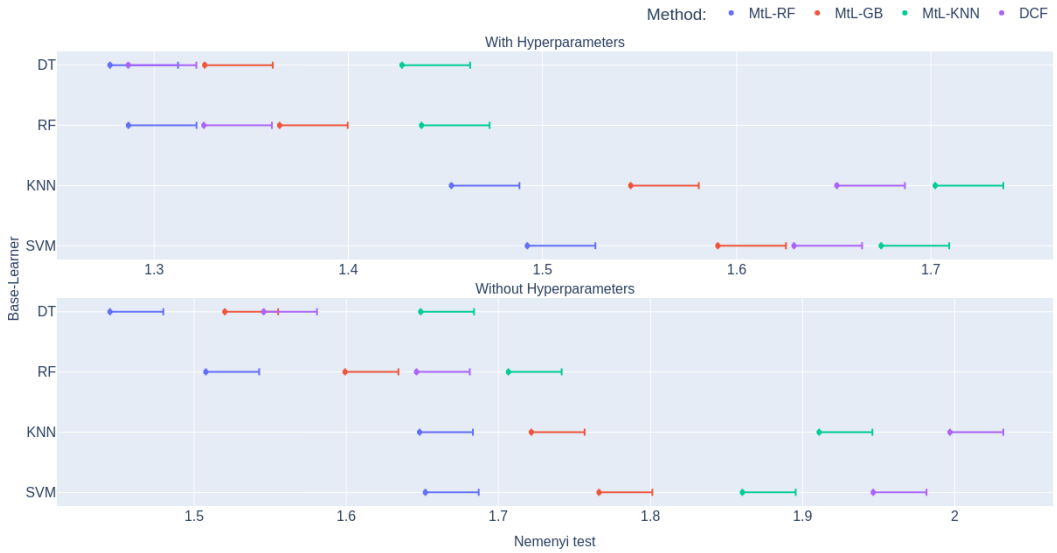

Figure 5. Nemenyi test results on the MtL approaches and the DCF filter.

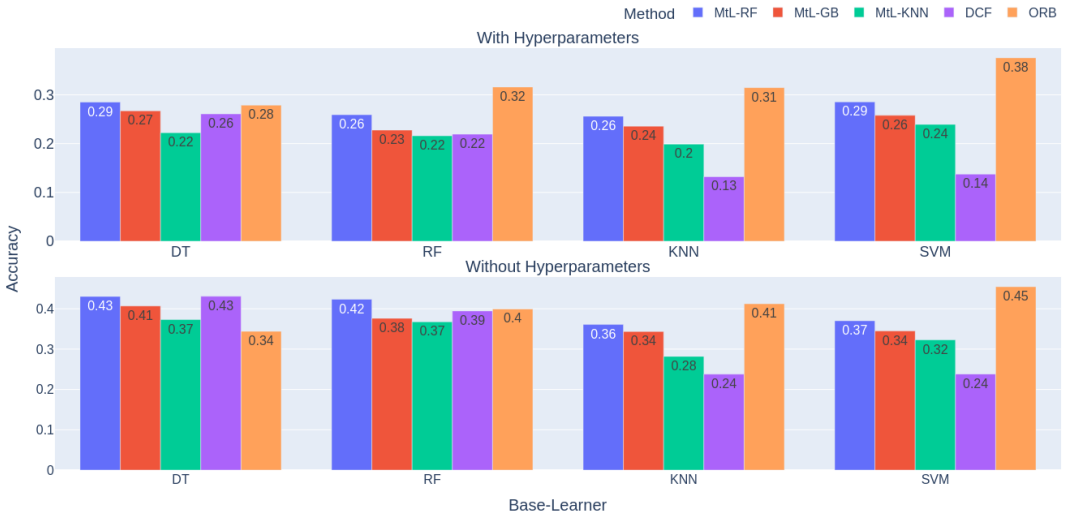

Figure 6. MtL approaches accuracy compared with DCF and ORB baselines when considering the Top-1 position of the ranking as correct.

eters failed, both when using DT as a base-learner. However, even in these cases, we managed to obtain at least one MtL approach with higher performance gain that was also significantly different from the baseline.

At the meta level, we evaluated the recommendation based on its accuracy. Figure 6 and 7 illustrate the accuracy obtained by our approaches considering the Top-1 and Top-3 results, respectively. Both results followed the same pattern. However, due to the higher number of available algorithms, when considering the hyperparameters recommendation, we obtained lower accuracy, achieving up to 29% on the Top-1 and up to 55% on the Top-3. While the approach without hyperparameters achieved up to 43% and 87%, obtaining

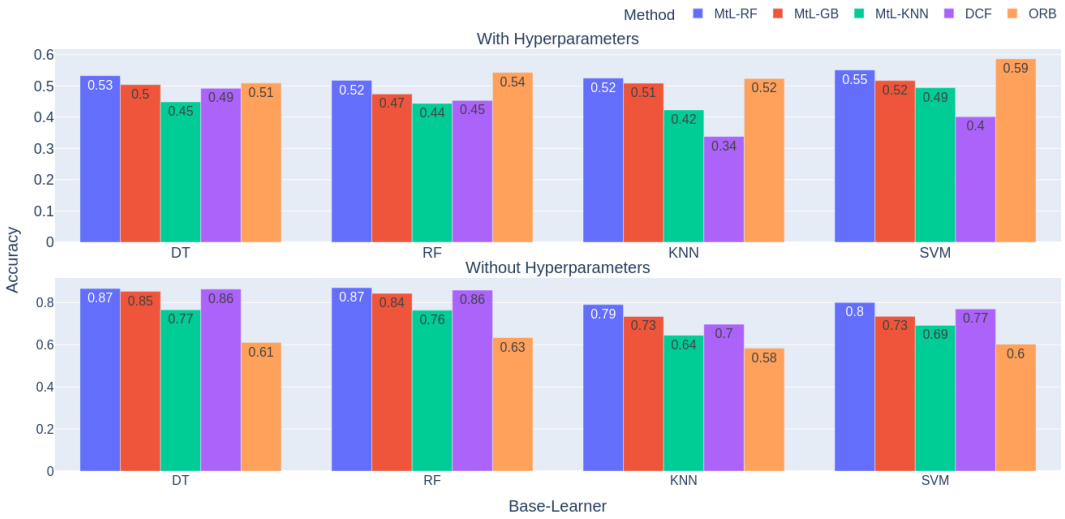

Figure 7. MtL approaches accuracy compared with DCF and ORB baselines when considering the Top-3 positions of the ranking as correct.

better results than both baselines on the Top-3.

The last step in the evaluation is to analyze the correlation rankings generated by the MtL approaches and the optimal ranking, obtained by ordering all base-learners F1-Score after each NF. Table 2 presents the mean Spearman's rank correlation of each dataset between the MtL ranks and the optimal rank considering the approaches with and without hyperparameters recommendations. In both cases, the MtL-RF obtained the best correlation, reaching up to 58% of correlation. We also noted that MtL-KNN was always the worst approach.

Table 2. Spearman correlation between the rankings obtained by the MtL and the optimal ranking with and without hyperparameters optimization.

	DT		**KNN**		**RF**		**SVM**	
Hyperparameter	With	Without	With	Without	With	Without	With	Without
MtL-RF	0.57 ± 0.33	0.56 ± 0.31	0.55 ± 0.35	0.54 ± 0.33	0.58 ± 0.33	0.57 ± 0.31	0.52 ± 0.37	0.51 ± 0.35
MtL-GB	0.56 ± 0.32	0.55 ± 0.29	0.51 ± 0.36	0.52 ± 0.33	0.56 ± 0.33	0.55 ± 0.30	0.47 ± 0.37	0.47 ± 0.35
MtL-KNN	0.43 ± 0.37	0.39 ± 0.33	0.40 ± 0.40	0.39 ± 0.36	0.44 ± 0.37	0.40 ± 0.33	0.40 ± 0.40	0.39 ± 0.36

Comparing the meta-regressors, MtL-RF is usually the best choice, acquiring higher accuracy, correlation, and performance gain. We also note that the hyperparameter recommendation provides a higher performance gain on the F1-Score. Since the MtL-RF with hyperparameters was the approach with higher performance gain, we decided to verify its most important MFs for the meta-regressor. Figure 8 presents the 20 MFs that were most important during the RF regression. In total, we listed 29 MFs, of which 10 were statistical, 6 belong to the information theory group, 5 to the model-based group, 5 was Landmarking, and 3 were part of the simple class. We also noted that the most important feature in the regression was mutual information.

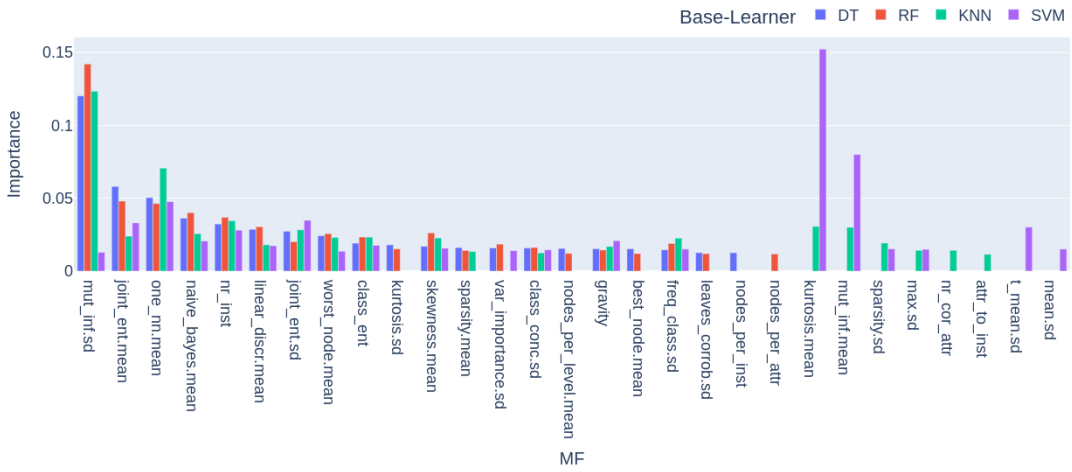

Figure 8. Top 20 MFs importance for each base-learner on the MtL-RF when considering the hyperparameters.

5. Conclusion

In this work, we presented a MtL approach to suggest NFs and their hyperparameters and to compare their benefits in contrast with when we do not consider hyperparameter variations. Our test confirmed that adding the complexity of hyperparameter recommendation can generate a higher gain in the F1-Score performance metric. However, it also reduces the Top-K accuracy of the ranking generated by the system. The results also showed that the RF meta-regressor obtained the best rankings, with a higher correlation between the optimal ranking, F1-Score performance gain, and accuracy.

For future works, we expect to enhance the hyperparameter optimization to more than one parameter per algorithm and compare it with other recommendation techniques, such as Bayesian optimization or genetic programming. We also intend to study the influences of different sets of MFs, investigating if we could improve the result of the MtL and finding a reduced group of MFs that better fits this problem. Finally, we would like to expand the recommendation to include other preprocessing techniques, such as missing values imputation, or data balancing techniques, which could result in a preprocessing pipeline recommendation system.

Acknowledgment

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

References

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. (2009). *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer, Berlin, Heidelberg, 1 edition.

Brazdil, P., van Rijn, J. N., Soares, C., and Vanschoren, J. (2022). *Metalearning: Applications to Automated Machine Learning and Data Mining*. Springer Nature.

de Morais, R. F., Miranda, P. B., and Silva, R. M. (2016). A meta-learning method to select under-sampling algorithms for imbalanced data sets. In *5th Brazilian Conference on Intelligent Systems*, pages 385–390. IEEE.

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

Famili, A., Shen, W.-M., Weber, R., and Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent Data Analysis*, 1(1):3–23.

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*, pages 50–56, Portland, OR. AAAI Press.

Frénay, B. and Verleysen, M. (2013). Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869.

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

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.

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.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*, volume 72. Springer, Cham, Switzerland, 1 edition.

Gupta, S. and Gupta, A. (2019). Dealing with noise problem in machine learning datasets: A systematic review. *Procedia Computer Science*, 161:466–474.

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

Koplowitz, J. and Brown, T. A. (1981). On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3):251–255.

Leyva, E., González, A., and Pérez, R. (2013). Knowledge-based instance selection: A compromise between efficiency and versatility. *Knowledge-Based Systems*, 47:65–76.

Martínez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Kull, M., Lachiche, N., Ramirez-Quintana, M. J., and Flach, P. (2019). Crisp-dm twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3048–3061.

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.

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

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., López García, Á., Heredia, I., Malík, P., and Hluchỳ, L. (2019). Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52(1):77–124.

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.

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.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

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.

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.

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.

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

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

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*, pages 1471–1479, Portland, OR. IEEE.

Vanschoren, J. (2019). Meta-learning. In *Automated Machine Learning*, pages 35–61. Springer Nature, Cham, Switzerland.

Wheway, V. (2001). Using boosting to detect noisy data. In *Pacific Rim International Conference on Artificial Intelligence*, pages 123–130.

Zhu, X. and Wu, X. (2004). Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210.