# Meta-Learning Approach for Noise Filter Algorithm Recommendation

P. B. Pio[1], L. P. F. Garcia[1], A. Rivolli[2]

[1] Universidade Federal de Brasília, Brazil
pedro.pio@aluno.unb.br, luis.garcia@unb.br
[2] Universidade Tecnológica Federal do Paraná, Brazil
rivolli@utfpr.edu.br

**Abstract.** Preprocessing techniques can increase the quality or even enable Machine Learning algorithms. However, it is not simple to identify the preprocessing algorithms we should apply. This work proposes a methodology to recommend a noise filtering algorithm based on Meta-Learning, predicting which algorithm should be chosen based on a set of features calculated from a dataset. From synthetics datasets, we created the meta-data from an extracted set of meta-features and the f1-score performance metric calculated from the DT, $K$NN, and RF classifiers. To perform the suggestion, we used a meta-ranker that returns the rank of the best algorithms. We selected three noise filtering algorithms, HARF, GE, and ORBoost. To predict the f1-score, we used the PCT, RF, and $K$NN algorithms as meta-rankers. Our results indicate that the proposed solution acquired over 60% and 80% accuracy when considering a top-1 and top-2 approach. It also shows that the meta-rankers, when compared with a random choice and single algorithms as a baseline, provided an overall performance gain for the Machine Learning algorithm.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**.

Keywords: meta-learning, noise detection, preprocessing, machine learning, ranking

## 1. INTRODUCTION

Machine Learning (ML) can be 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, not to mention that this process 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 others 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 a recommendation system that provides a rank of the best noise detection algorithms with MtL techniques. 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 (MFe) 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 preprocessing 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.

The main contributions of this article 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 MFe; (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.

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 in Section 5, we conclude with some discussions and 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: $(i)$ 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; $(ii)$ The feature space $F$ is the set of features extracted from the instances $P$; $(iii)$ The algorithm space $A$ is the set of algorithms considered in the selection; $(iv)$ 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, the 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 MFe, 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 votes to remove or keep the instance; (*iii*) distance-based methods, where the algorithm utilizes $K$ nearest neighbors ($K$NN) 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: (*i*) **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. According to this value, an instance is considered noise and removed; (*ii*) **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; (*iii*) **Outlier Removal Boosting (ORBoost)** [Karmaker and Kwek 2006]: it uses the propriety of AdaBoost [Freund 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. Note that each one approaches noise detection differently: HARF is a voting approach based on an ensemble algorithm, GE is a distance-based algorithm, and ORBoost is an ensemble boosting algorithm.

### 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.

Bilalli [Bilalli et al. 2019] presents 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 MFe 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] presents 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 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

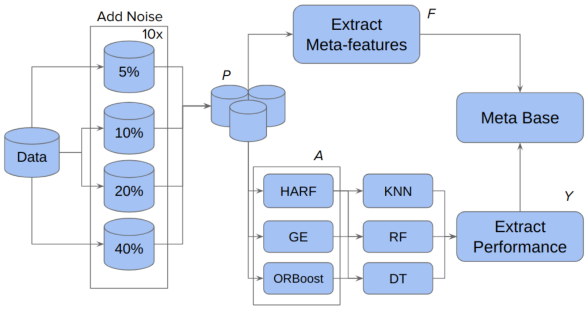


Fig. 1: Base level diagram.

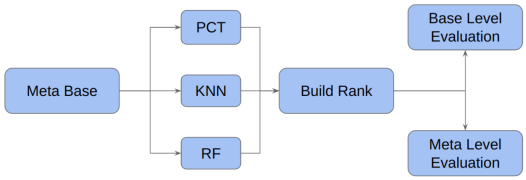


Fig. 2: Meta level diagram.

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 MFe were used to create all nine meta-data. The proposed solution acquired up to 90% accuracy.

Garcia [Garcia et al. 2016] implements an algorithm selection system for noise filter algorithms. 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 MFe and with the performance metric f1-score as a prediction target. They compared three different algorithms to perform the recommendation: $K$NN, RF, and Support Vector Machine (SVM). After the analysis of the mean square error of each regressor, they concluded that RF was a better choice.

In this work we present a noise filtering recommendation methodology. Although a similar problem was approached by [Garcia et al. 2016], 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. We are also generating the results as a rank, increasing the versatility of the results. Considering the MFe choice, Bilalli [Bilalli et al. 2019], due to the computation cost, decided to use only simple, statistical, and information-theoretic features. In this work, we also used landmark and model-based features to increase the information available to the meta-ranker. Furthermore, in addition to the methodology, we provide an analysis of the effects of the noise filtering algorithms in the performance metric and what were the most critical MFe for the recommendation.

## 3. METHODOLOGY

This section describes the proposed methodology for the noise filter algorithm recommendation. We divided it into two levels, the base level, where we build the meta-data set, and the meta level, where we implement the recommendation algorithms and analyze its results.

Figure 1 shows a diagram of the base level, the methodology begins with the data collection. We used the OpenML[1] platform to collect the 323 datasets[2] that were used. The platform allows us to filter the datasets according to our needs. We used sets with up to 10000 instances, 50 attributes, without missing values, and with only two classes, thus, reducing the preprocessing step needed to extract the ML metrics. To guarantee and control the class noise level, we randomly changed the instance's class, generating artificially random noise in 5%, 10%, 20%, and 40% 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 40 synthetics sets for each dataset, forming the $P$ space.

[1] `https://www.openml.org/`
[2] The list of datasets and MFe used can be found at `https://bit.ly/KDMiLeDatasets` and `https://bit.ly/KDMiLeMFes` respectively

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 run the three different filters that compose the algorithm space $A$: **HARF**, **GE**, and **ORBoost**. All filters were implemented with the `NoiseFiltersR` package and their default configurations. After executing the filter, we used three different ML algorithms to extract the f1-score: DT CART [Breiman et al. 2017] algorithm, RF [Breiman 2001] with 100 trees, and $K$NN [Mitchell 1997] with $k = 5$. To compute the MFe we used the `pymfe`[3] [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 73 MFe producing the $F$ space. Combining both the $F$ space and the $Y$ space resulted in our meta base used in the meta level to apply the regressions and form the ranking containing 12915 instances, 73 MFe, and 3 performance metrics.

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 MFe, 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[4] ML algorithms: the Predictive clustering trees (PCT) [Todorovski et al. 2002] that is based on a DT and performs all the regression of each filter at once; the $K$NN [Mitchell 1997] with $k = 5$, which is a distance-based algorithm; and the RF [Breiman 2001] composed with 100 trees, an ensemble algorithm. Note that both $K$NN and RF can only predict the performance of one filter, meaning they will run once for each filter. Since each dataset generates 40 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 40 datasets derivated from the same original OpenML dataset and validate them together.

To compare the results we built three baselines, which are the results when we always select the same filter algorithm and a random algorithm selection, which selects the algorithm randomly. At the base level, we selected the best algorithm predicted by each meta-ranker and compared it with the baselines and the random approach, 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 $k = 2$, meaning we consider a rank as correct if the best algorithm is in the top-1 and top-2 position of the rank. Again, we compare the top-$k$ accuracy with the random approach and 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.

## 4. RESULTS

We first analyze the effects of the filters, verifying if they had a positive or negative impact on the f1-score. Table I presents the percentage of times each filter had a positive, negative, or neutral influence. Both HARF and GE filters usually have a positive effect. Otherwise, compared with the other filters, ORBoost generated negative and neutral results more often.

The percentage of times each algorithm appears in each position in an optimal rank is presented in Table II. Note that the ORBoost is labeled the best more often, however it is also the worst more often, thus, it would probably benefit from an algorithm recommendation technique. Also, HARF is more often found as the second position while GE is more evenly distributed between all placements.

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

[4] In this work we used three classifiers to calculate the f1-score and three regressions algorithm to generate the ranking.

| | DT | | | *K*NN | | | RF | | |
|---|---|---|---|---|---|---|---|---|---|
| | + | - | 0 | + | - | 0 | + | - | 0 |
| **GE** | 89.1 | 9.5 | 1.2 | 87.2 | 11.4 | 1.2 | 89.5 | 9.2 | **1.2** |
| **ORBoost** | 71.0 | **27.5** | **1.4** | 56.0 | **13.7** | **30.1** | 71.3 | **27.4** | **1.2** |
| **HARF** | **94.4** | 4.9 | 0.6 | **88.1** | 11.0 | 0.8 | **95.5** | 3.8 | 0.6 |

Table I: Percentage of times the filters had a positive (+), negative (-), or neutral (0) result.

| | DT | | | *K*NN | | | RF | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1st | 2nd | 3rd | 1st | 2nd | 3rd | 1st | 2nd | 3rd |
| **GE** | 28.9 | 38.0 | 33.0 | 33.1 | **50.7** | 16.1 | 29.9 | 41.9 | 28.1 |
| **ORBoost** | **51.1** | 6.3 | **42.5** | **51.1** | 13.4 | 35.3 | **52.0** | 5.1 | **42.8** |
| **HARF** | 17.8 | **54.8** | 27.3 | 16.1 | 41.8 | **42.0** | 15.9 | **52.0** | 32.0 |

Table II: Percentage of times each algorithm should be ranked first, second or third best.

| | GE | ORBoost | HARF | Random |
|---|---|---|---|---|
| **MtL-RF** | 42.9 | **38.3** | **61.1** | **48.1** |
| **MtL-PCT** | **44.9** | 32.7 | 61.0 | 46.6 |
| **MtL-KNN** | 37.4 | 28.8 | 54.7 | 40.5 |

Table III: The percentage of times the MtL was better compared to each filter and random selection.

| | GE | ORBoost | HARF | Random |
|---|---|---|---|---|
| **MtL-RF** | **12.6** | 24.1 | **17.2** | **18.0** |
| **MtL-PCT** | 17.8 | **19.3** | 20.5 | 19.1 |
| **MtL-KNN** | 23.4 | 25.0 | 27.0 | 25.0 |

Table IV: The percentage of times the MtL was worse compared to each filter and random selection.

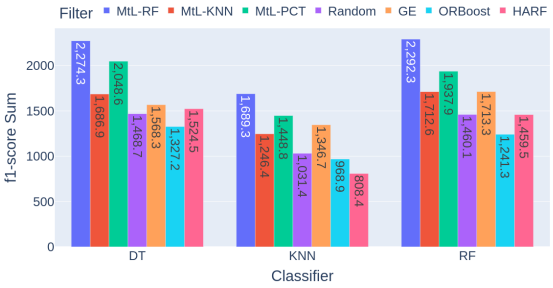


Fig. 3: Graph containing the sum of the gain in all data sets after the application of filters.

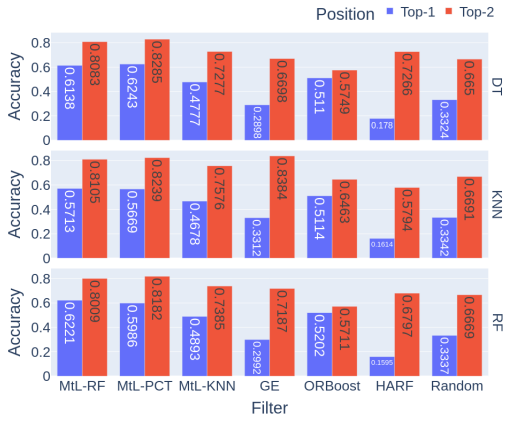


Fig. 4: Meta-rankers accuracy considering the best filter in the first or second position, *top* 1 or *top* 2, respectively.

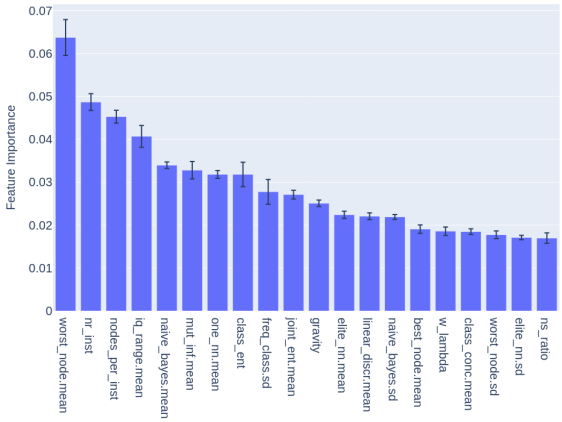


Fig. 5: 20 most important features for the regression of the RF meta-ranker.

We then start evaluating the meta-rankers. We used three ML regressors to create our ranks: PCT (MtL-PCT), RF (MtL-RF), and *K*NN (MtL-KNN). Figure 3 shows the cumulative gain of the f1-score given the first recommendation of each predictor, the baselines, and the random algorithm choice. The results show that MtL-PCT and MtL-RF produced a better gain compared to the baselines, while MtL-KNN is better than ORBoost, HARF baselines, and the random selection, but sometimes worst than the GE. Furthermore, all approaches using the MTL-RF led to the greatest gains in performance.

When examining the number of times each meta-ranker performed better and worse than each approach over all classifiers, Table III and Table IV respectively, once again, show the performance of MtL-RF was the best, however, MtL-PCT sometimes performed better. Also, none of the approaches obtained worse results more often than good results, indicating that they are more effective than the baselines and the random selection.

Another way of evaluating the prediction is by verifying the rank accuracy, allowing us to compute the performance metrics of the meta-rankers and compare the recommendations. Since we are return-

ing the results as a rank, to consider a classification correct, we used a top-$k$ approach meaning that if the best algorithm is in the $k$ first positions in the rank, the recommendation is classified as accurate. Figure 4 presents the accuracy considering the top-1 and top-2 positions. We can notice that MtL-RF and MtL-PCT are, once again, better than the baselines and the random selection, with an exception being the GE filter when using the $K$NN classifier considering the top-2 positions. Also, MtL-KNN top-1 results are always worse than the ORBoost accuracy, enforcing the poor performance of the meta-ranker. We also acquire accuracy of around 60% and 80% for the top-1 and top-2, respectively, on both RF and PCT meta-rankers.

To evaluate the ranks produced by each meta-ranker, we calculated Spearman's rank correlation between the rank acquired by them and the optimal rank. The best ranker was MtL-RF acquiring 0.44, 0.40, and 0.44 correlation when using the DT, KNN, and RF classifiers respectively, MtL-KNN got 0.2, 0.23, and 0.22, while MtL-PCT got 0.43, 0.38, and 0.41. It is important to notice that the poor performance of the MtL-KNN may be due to the high number of MFe suffering with the *curse of dimensionality*, nevertheless, it acquired better results than the random selection, which, as expected, got a 0 correlation.

Finally, we decided to select the best meta-ranker and examine the MFe that were most important for the algorithm. Since we are using leave-one-out cross-validation we had to calculate the mean of all RF we used during the result validation process. Figure 5 presents the mean importance of the top-20 MFe that were used in the algorithm, we can notice that the landmarks features were more present appearing 9 times, followed by information-theoretic (5), statistical (3), simple (2) and model-based (1) [5]. Also, the most important feature, *worst_ node.mean*, is a DT-based landmark, meaning its high importance may be due to the RF DT dependence.

## 5. CONCLUSIONS

In this work, we proposed a methodology to recommend noise filtering algorithms with MtL. From the datasets, we extracted MFe and the f1-score of $K$NN, DT, and RF classifiers. With those values, we built a meta-data that was used as input to a meta-ranker that returned a rank of the suggested filtering algorithms. We selected the PCT, $K$NN, and RF algorithms as meta-ranker. Our experiments indicated that RF was the best choice for a meta-ranker, providing performance gain compared to a random approach and the three baselines, in addition, it acquired around 60% and 80% accuracy in the recommendation when considering the top-1 and top-2 algorithms, respectively. We also studied the MFe importance noticing that landmark features are more present in the top-20 Mfe.

For future works, we intend to expand the preprocessing techniques supported by the system, showing that the same methodology could be applied for problems such as missing values or imbalanced data. We also aim to enhance the number of algorithms and include a hyperparameter optimization, which would increase the solution search space. Finally, it would be interesting to study if the used MFe could be simplified, making it possible to acquire similar results with a smaller set of features or if we could improve the system performance with a more complex set of MFe.

## REFERENCES

Alcobaça, E., Siqueira, F., Rivolli, A., Garcia, L. P. F., Oliva, J. T., and de Carvalho, A. C. P. L. F. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research* 21 (111): 1–5, 2020.

Bilalli, B., Abelló, A., Aluja-Banet, T., and Wrembel, R. Presistant: Learning based assistant for data preprocessing. *Data & Knowledge Engineering* vol. 123, pp. 1–22, 2019.

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

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

Breiman, L. Random forests. *Machine learning* 45 (1): 5–32, 2001.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and regression trees*. Routledge, New York, NY, 2017.

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

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

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

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

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

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

García, S., Luengo, J., and Herrera, F. *Data preprocessing in data mining*. Vol. 72. Springer, Cham, Switzerland, 2015.

Gupta, S. and Gupta, A. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science* vol. 161, pp. 466–474, 2019.

Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated machine learning: methods, systems, challenges*. Springer Nature, Cham, Switzerland, 2019.

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

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

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

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

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

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

Parmezan, A. R. S., Lee, H. D., Spolaôr, N., and Wu, F. C. Automatic recommendation of feature selection algorithms based on dataset characteristics. *Expert Systems with Applications* vol. 185, pp. 115589, 2021.

Rice, J. R. The algorithm selection problem. *Advances in Computers* vol. 15, pp. 65–118, 1976.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J., and de Carvalho, A. C. Meta-features for meta-learning. *Knowledge-Based Systems* vol. 240, pp. 108101, 2022.

Russell, S. J. and Norvig, P. *Artificial Intelligence: a modern approach*. Pearson, Prentice Hall Upper Saddle River, NJ, USA, 2009.

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

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

Todorovski, L., Blockeel, H., and Dzeroski, S. Ranking with predictive clustering trees. In *European Conference on Machine Learning (ECML)*. Springer, Berlin, Heidelberg, pp. 444–455, 2002.

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

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

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

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

Zar, J. H. Spearman rank correlation: overview. *Wiley StatsRef: Statistics Reference Online*, 2014.

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