Data pre-processing pipeline generation for AutoETL

Joseph Giovanelli1,\*, Besim Bilallib , Alberto Abellb

*a University of Bologna, Bologna, Italy*

*bUniversitat Politcnica de Catalunya, BarcelonaTech, Barcelona, Spain*

**Abstract**

Data pre-processing plays a key role in a data analytics process (e.g., applying a classiﬁcation algorithm on a predictive task). It encompasses a broad range of activities that span from correcting errors to se- lecting the most relevant features for the analysis phase. There is no clear evidence, or rules deﬁned, on how pre-processing transformations impact the ﬁnal results of the analysis. The problem is exacerbated when transformations are combined into pre-processing pipeline prototypes. Data scientists cannot eas- ily foresee the impact of pipeline prototypes and hence require a method to discriminate between them and ﬁnd the most relevant ones (e.g., with highest positive impact) for their study at hand. Once found, these prototypes can be instantiated and optimized e.g., using Bayesian Optimization. In this work, we study the impact of transformations when chained together into prototypes, and the impact of transfor- mations when instantiated via various operators. We develop and scrutinize a generic method that allows to generate pre-processing pipelines, as a step towards AutoETL. We make use of rules that enable the construction of prototypes (i.e., deﬁne the order of transformations), and rules that guide the instantia- tion of the transformations inside the prototypes (i.e., deﬁne the operator for each transformation). The optimization of our effective pipeline prototypes provide results that compared to an exhaustive search, get 90% of the predictive accuracy in the median, but with a time cost that is 24 times smaller.

*Keywords:* data pre-processing pipelines, data analytics

1 **1. Introduction**

2 The decision making process has historically been key for the success of any organization or business

3 activity. Lately, with the abundant presence of data, this process has become data-driven, where data

4 are continuously analyzed to be transformed into knowledge. Along the way however, data undergo

5 several (sometimes necessary) processing steps, shown in Figure [1.](#bookmark1) Firstly, data are extracted in a raw

6 format from different sources and then are sifted out such that only a relevant subset is selected. Next,

7 this subset is pre-processed and is fed to a machine learning (ML) algorithm for it to be analyzed. The

8 output of the analysis is then interpreted and the whole process iterates until the results obtained are

9 satisfactory and signiﬁcant for the decisions to be made.

10 Unfortunately, this well known process does not have universal well-deﬁned practices for the differ-

11 ent steps, which translates to the data scientist manually conﬁguring and parameterising the operators

12 for each step until an optimal solution is found — an optimal *data analytics pipeline*. To this end, most

13 [of the time is spent on the heavily laborious work of pre-processing (i.e., 50-80% of the time [1]), where](#bookmark2)

14 the generated output is a *pre-processing pipeline*. Next, once the data is transformed into the proper

15 form, different ML algorithms with different hyperparameters are evaluated over the dataset until an

\* Corresponding author

*Email addresses:* j .giovanelli@unibo .it (Joseph Giovanelli),bbilalli@essi .upc .edu (Besim Bilalli), aabello@essi .upc .edu (Alberto Abell)

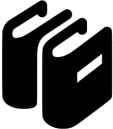
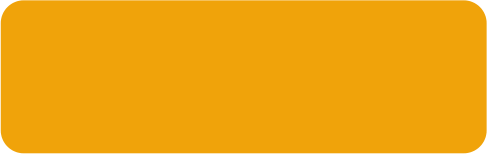
*Preprint submitted to Elsevier February 14, 2022*

© 2021 Elsevier. This manuscript version is made available under the CC-BY-NC-ND 4.0 license <http://creativecommons.org/licenses/by-nc-nd/4.0/>

AutoETL AutoML



DPSO CASH

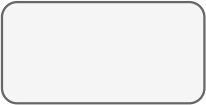


Algorithm selection

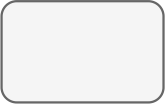
Hyper-

parameter optimization

|  |  |
| --- | --- |
|  |  |

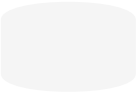


Interpretation/ Evaluation



Data

selection



|  |
| --- |
|  |

Data source

Knowledge

|  |  |
| --- | --- |
| Data pipeline prototype  selection | Data pipeline  prototype  optimization |

Data pre-processing Modeling

Figure 1: Data analytics pipeline generation in a knowledge discovery process.

16 acceptable result is obtained — *ML model*. This whole process requires expertise and is particularly

17 challenging for novice, inexperienced data scientists for whom hand-tuning is no longer an option.

18 Recent developments in algorithm conﬁguration have raised the efﬁciency and effectiveness of auto-

19 matic search, and therefore, for instance, AutoML is now considered a prominent technique for ﬁnding

20 optimal models. [Some AutoML frameworks [2,](#bookmark3) [3], mix-in the pre-processing during the optimization,](#bookmark4)

21 but they are typically limited to very few transformations or do not consider all the data processing

22 phases (e.g., extraction, selection, loading), thus in a way overlooking it. [Inspired from [4], we con](#bookmark5)-

23 tend that there is need for more generic AutoETL techniques, encompassing all the phases of the ETL

24 [process [5]; from its inception via *data extraction*, to the intermediate phase of *data transformation*](#bookmark6), up

25 to the ﬁnal phase, when data reaches its destination, via *data loading*. Assistance is required in every

26 [phase [6]](#bookmark7). Yet, in a data analytics context, as in the case of this work, the critical automation challenge

27 lies more on the *data transformation* phase (see AutoETL and AutoML in Figure [1), which is related to](#bookmark1)

28 the pre-processing of the data. In the literature, this particular problem has been often referred to as the

29 [Data Pipeline Selection and Optimization problem (DPSO) [7], where a *pipeline prototype*](#bookmark8)(sequence

30 of transformations, e.g., missing value imputation followed by normalization) is fed to an optimizer and

31 an optimal instance of the prototype, in the form of a *pipeline* (sequence of operators, e.g., imputation

32 by mean followed by min-max normalization) is found. By considering pre-processing as an integral

33 component of data analytics, and carefully conﬁguring the pre-processing pipelines, it is easy to obtain

34 results that go beyond the ones obtained by only optimizing the learning algorithm.

35 [To brieﬂy illustrate this, we perform an experiment on the well known bank-marketing1](#bookmark9)dataset,

36 [using HyperOpt [8] as an AutoML approach to optimize the parameters of three different ML algorithms,](#bookmark10)

37 namely Naive Bayes (NB), K-Nearest Neighbor (KNN), and Random Forest (RF). We provide an ini-

38 tial budget of 50 iterations for optimizing the hyper-parameters of the algorithms, and after the 50th

39 iteration, we ﬁx the algorithm conﬁguration to the best one achieved so far and start optimizing the pre-

40 [processing pipeline2 .](#bookmark11) In Figure [2, the ratio of the change in terms of predictive accuracy (i.e., ratio of the](#bookmark12)

41 accuracy obtained after the i-th iteration to the baseline/default accuracy) is plotted against the number

42 of different conﬁgurations visited by HyperOpt (i.e., iterations). Observe that after the 11th iteration

43 for NB and KNN, and after the 26th iteration for RF, the lines remain ﬂat. That is, from there on, no

44 improvement is achieved by optimizing the hyper-parameters of the algorithms until the 50th iteration

45 is reached. At this point, a sudden jump is observed and the results start to improve again, going clearly

46 beyond the ones obtained before, thanks to the optimizations performed now over the pre-processing

47 pipeline. Yet, including the pre-processing in a free form in the optimization, heavily increases the

48 search space, making the problem much harder. This is mitigated by creating a pre-processing pipeline

49 prototype that ﬁxes the order of transformations, leaving the freedom to only instantiate and parametrise

50 them. Therefore, the challenge for data scientists is to ﬁnd the right pre-processing pipelines, that is, (i)

51 how to order the transformations (i.e., prototype construction), and (ii) which pre-processing operators

52 to consider in the prototype (i.e., prototype instantiation) such that when optimizing the parameters,

53 better results are obtained. The aim of this work is to study these two questions in order to propose a

[1https://archive.ics.uci.edu/ml/datasets/Bank+Marketing](https://archive.ics.uci.edu/ml/datasets/Bank+Marketing) 2This order is used only for the sake of illustration.

1.4

1.3

1.2

1.1

1.0

0.9

0.8

|  |  |
| --- | --- |
| NB  KNN RF |  |
|  |
|  |

0 10 20 30 40 50 60 70 80 90 100

Configurations visited

Figure 2: Evolution of predictive accuracy during the optimization process. The ﬁrst 50 conﬁgurations optimize only the hyper-parameters and after the 50th conﬁguration, the pre-processing pipeline is optimized instead.

Ratio of pred ict ive accuracy change

54 method for generating effective pre-processing pipeline prototypes that, once instantiated through some

55 optimization technique (e.g., Bayesian optimization),improve the ﬁnal result of the analysis. To keep

56 discussions and experiments simpler, we stick to supervised learning tasks, which encompass algorithms

57 generating a map function based on pairs of input-output exemplars. In particular, this work focuses on

58 classiﬁcation problems (i.e., binary and multiclass), where the output to be predicted is of Categorical

59 type. [Furthermore, this work extends [9], among others with, (i) a new meta-learning step to guide the](#bookmark13)

60 instantiation of transformations where a model is learned to predict the operators for a given transforma-

61 tion (see Section [3.6), (ii) a set of rules extracted through the meta learning process that mitigate the cold](#bookmark14)

62 start problem (see Example 8), (iii) a new cross validation to conﬁrm the initial results (see Section [3.4),](#bookmark15) 63 (iv) a new background section for AutoML and AutoETL (see Section [2), and (v) experiments over new](#bookmark16)

64 datasets (see Section [4)](#bookmark17).

65 **Contributions.** The main contributions of this paper can be summarized as follows:

66 • We empirically evaluate the impact of optimizing the exhaustive set of potential pipeline proto-

67 types and ﬁnd out that at least one different pipeline works best for each dataset and algorithm

68 considered, hence showing that there is no universal pipeline that works best for all of them.

69 • We deﬁne a method that given a classiﬁcation algorithm and a set of pre-processing transforma-

70 tions, is capable of generating the right order between transformations, obtaining effective pre-

71 processing pipeline prototypes, which are then instantiated and further optimized via Bayesian

72 optimization.

73 • We suggest a meta-learning step, where the relationship between pre-processing operators and

74 dataset characteristics is learned in order to create rules that help with the initial instantiation of

75 the pipeline prototypes.

76 **–** We exemplify our meta-learning study generating simple but not obvious and effective rules

77 for two kinds of transformations, namely, Feature Engineering and Rebalancing.

78 • We perform a comprehensive evaluation by comparing the performance of optimizing the pipelines

79 generated following our method, and ﬁnd out that:

80 1. with 24 times less time budget, our proposed pipelines obtain results whose median is above

81 90% of the ones generated via exhaustive search.

82 2. on average, in 73% of the cases, splitting evenly the time budget between pre-processing

83 and hyper-parametrisation outperforms the results of only optimizing the hyper-parameters

84 of the ML algorithm.

85 The remaining of this paper is organized as follows. Section [2](#bookmark16) provides a brief background on

86 AutoETL and AutoML. Section [3](#bookmark18) presents our method of generating effective pipelines. Section [4](#bookmark17)

87 provides an extensive evaluation of the pipelines created using our proposed method. Section [5](#bookmark19)discusses

88 the related work. Finally, Section [6](#bookmark20)provides the conclusions and future work.

89 **2. Background: AutoETL and AutoML**

90 The abundance of data has led to data analytics being prevalent in many disciplines and domains, but

91 since the number of its applications exceeds the number of qualiﬁed experts, more and more non-experts

92 approach the task of data analytics. This has consequently led to the rise of off-the-shelf automated

93 techniques that facilitate its application. AutoML is an umbrella term for automations mainly related

94 to the ML algorithm, and it typically aims to tackle the challenge of Combined Algorithm Selection

95 and Hyperparameter Optimization (CASH). Yet, there is also need for automation in the more generic

96 [aspects of ETL [4], which we coin as AutoETL](#bookmark5). AutoETL encompasses various steps, however, in this

97 work we focus on the phase that is related to the transformation (pre-processing) of the data, typically

98 formalized as DPSO.

99 [CASH and DPSO can be treated as a single optimization problem [10,](#bookmark21) [2]](#bookmark3). However, we consider

100 them separately because this allows to, (i) reduce the search space and, (ii) to explicitly assign different

101 optimization budgets and/or optimization techniques, depending on theirrespective impact to the ﬁnal

102 result of the analysis. Since these problems are similar, the methods initially employed in CASH have

103 been recently considered to solve the DPSO problem too. Therefore, in the following we ﬁrst delve into

104 more details about CASH and then DPSO. In particular, we formalize them and discuss the methods

105 they employ.

106 *2.1. Combined Algorithm Selection and Hyper-parameter optimization (CASH)*

107 The algorithm selection problem is known to exist for a long time and many approaches have been

108 [proposed to solve it [11]](#bookmark22). Recently, in the context of ML algorithms, the problem has been extended to

109 [include the optimization of the hyperparameters too, and thus has been formalized as follows [10]](#bookmark21).

110 Given:

111 • A data-set D divided into D train; D test;

112 • A set of algorithms A = {A1; . . . ; Ak } with associated hyperparameter spaces Λ1 ; . . . ; Λk;

113 • And a loss function L(A; D train; D test);

114 we are searching for:

A\* ∈ argminAi ∈A,λ∈Λi L(A; D train; D test) (CASH)

115 The dataset D is divided into D train and D test, to build and to evaluate the overall performance,

116 calculated through the loss function L. The problem is set up as an optimization problem and, as such,

117 the conﬁguration space is assumed to be known in advance (the set of algorithms {A1; . . . ; Ak } and the

118 related hyper-parameter spaces {Λ1 ; . . . ; Λk }). The goal is then to ﬁnd the best algorithm A\* in the set

119 of algorithms and its best hyperparameters λ\* in the related hyperparameter space.

120 Many optimization techniques have been employed to solve the CASH problem and some of them

121 [are: Grid search [12], Random search [13], Simulating annealing [14], Genetic algorithms](#bookmark23) [[15], Bayesian](#bookmark24)

122 techniques [[16], Bandit-based algorithms](#bookmark25) [[17]](#bookmark26). However, due to their promising results, Bayesian tech-

123 [niques are perhaps the most popular ones [18,](#bookmark27)[19]](#bookmark28). We explore their details and speciﬁcally focus on one

124 [of their incarnation, the Sequential Model-Based Optimization (SMBO) algorithm [20]](#bookmark29).

125 *2.1.1. Sequential Model-Based Optimization (SMBO)*

In an optimization problem, we are searching for the best solution among a set of feasible solutions. The latter can be formalized as follows:

max f (x) x∈B

126 , where B contains all the feasible solutions, or candidates, and it is typically d-dimensional (B ≤ Rd),

127 where d ∈ Z and d > 1. A speciﬁcsolution x ∈ B is evaluated through the function f : Rd —→ R,

128 also called the objective function. In general, in these kinds of problems,f has no special structure like

129 concavity or linearity that would make the optimization easier. In fact, it is considered as a “black-box”

130 function, without any knowledge about its behaviour; being that it maps certain inputs, x ∈ B, to certain

131 outputs, f (x) ∈ R. The goal is then ﬁnding an x that maximizes f (x). The naive solution to the problem

132 would be to systematically evaluate all possible candidates x and choosing the one leading to the highest

133 value off (x), aka *exhaustive search*. Since this evaluates all the potential solutions, it guarantees that

134 it always ﬁnds the best one. Nevertheless, generally, it cannot be applied to real problems due to the

135 large number of candidate solutions to be explored, which dwell in a high-dimensional space, and too

136 expensive objective functions. The result is that not all candidates can be evaluated and we have to ﬁnd a

137 way to wisely choose the most promising ones. Bayesian techniques are part of the family of “surrogate

138 methods”, which create a surrogate model to approximate the objective function and thus, choose a point

139 [in the search space where to evaluate the objective function [21]](#bookmark30). In contrast to the other methods, they

140 build such surrogate models through Bayesian statistics.

141 In short, Bayesian techniques start by evaluating the objective function on an initial observation point

142 of the search space, then the process becomes iterative: the surrogate model is constructed on the basis of

143 the visited points and through an acquisition function — the Bayesian interpretation of the surrogate, the

144 candidate for the next observation is decided. The process ends when a termination condition is reached,

145 generally expressed through a *budget* represented in terms of the *number of iterations* or *execution time*.

146 Given its iterative nature and the fundamental role of the model, this algorithm is called Sequential

147 [Model-Based Optimization [20,](#bookmark29)[22]](#bookmark31). Variations of SMBO exist, depending on the method used to build

148 [the surrogate model (e.g., Gaussian Processes, Random Forest Regressions) [23,](#bookmark32)[24]](#bookmark33).

149 *2.2. Data Pipeline Selection and Optimization (DPSO)*

150 [DPSO was formalized for the ﬁrst time in [25], where some new concepts were introduced](#bookmark34). For

151 instance, a pre-processing *pipeline prototype* or *logical pipeline* is deﬁned as a sequence of kinds of

152 transformations, where each represents a logical concept that can be implemented/instantiated by one or

153 more operators. The prototype thus, deﬁnes only the order between kinds of transformations, without

154 specifying the concrete operators nor their parameters. Yet, the potential operators of each kind and their

155 corresponding parameter search spaces need to be known in advance. Solving the DPSO problem trans-

156 lates to ﬁnding the right instantiation and conﬁguration for each kind of transformation in the pipeline

157 prototype (i.e., optimal operator and optimal parameter values), which is called pre-processing *pipeline*

158 or *physical pipeline*.

159 Formally, given:

160 • A data-set D divided into D train, D test;

161 • A data pipeline prototype P with a conﬁguration space P;

162 • The algorithm A, for which the given pipeline P transforms the data;

163 • And a loss function L(P, A,D train, D test);

164 we are searching for:

P\* ∈ argminP ∈PL(P, A,D train, D test) (DPSO)

 Prototype construction



|  |
| --- |
| Heuristic  rules |

|  |
| --- |
| Framework - related rules |

|  |
| --- |
| Effective  pipeline prototypes |

|  |
| --- |
| Empirical  rules |



3.2.Discover

framework

imposed order

|  |
| --- |
| Machine  learning framework |





|  |
| --- |
| 3.4.Assess  empiric. the  order of  remain. pairs |



|  |
| --- |
| 3.3.Discover  meaningful  order |

|  |
| --- |
| Pre-  processing  pipeline |

V

3.7.Prototype

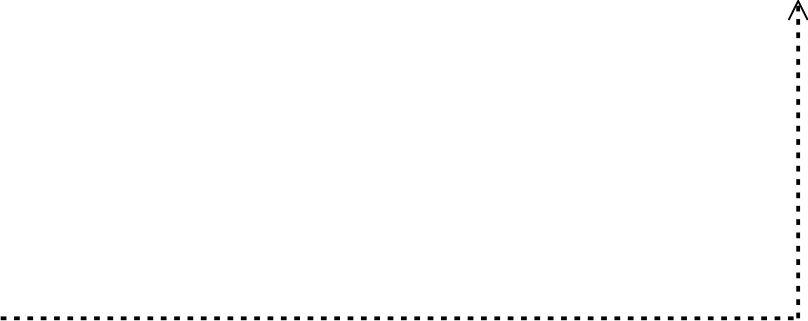
instantiation

3.5.Combine

ordered pairs

of

transformati ons



|  |
| --- |
| Machine  learning experiments |

3.1.Select the

transformati

ons and

operators

V



|  |
| --- |
| 3.6.Discover  rules via  meta-learning |



|  |
| --- |
| Meta-learning rules |

Figure 3: A method for generating pre-processing pipelines.

165 Notice that a prototype imposes an order between the kinds of transformations, but this is an addi- 166 tional problem that is not dealt within DPSO, since it is assumed to be given as part of the input. This is 167 in fact a limitation of the current approaches in DPSO, in that the order of kinds of transformations are 168 ﬁxed a priori without sufﬁciently studying the potential effectiveness of different alternatives.

169 *2.2.1. SMBO as solver forDPSO*

170 [Since DPSO is formalized as an optimization problem, SMBO has been proposed as a valid solver [20]](#bookmark29).

171 In the previous section, we saw the application of SMBO to CASH, but in fact the process of selecting

172 the best algorithm, and its hyperparameters conﬁguration, is identical to selecting the best operator for a

173 transformation, and its parameter conﬁguration. Yet, DPSO requires one more layer, in that transforma-

174 tions need to be chained together into a pipeline. To this end, given a pipeline prototype as input and a

175 budget either in terms of time or number of iterations, SMBO can be conﬁgured to iterate over different

176 conﬁgurations until a near to optimal physical pipeline is found. The objective function of the pipeline

177 is however measured in the context of a given parametrised ML algorithm by applying the pipeline on

178 a dataset, and measuring the performance of the ML algorithm (e.g., predictive accuracy) on the trans-

179 formed output. In this context, by ﬁxing the hyper-parameters of the ML algorithm to the default ones,

180 the performance of the learner is set to measure the effectiveness of the considered data pipeline.

181 **3. Data pre-processing pipeline generation**

182 [Following the notation from [25], we also distinguish between a ﬁxed, ordered sequence of kinds of](#bookmark34)

183 pre-processing transformations, known as *pipeline prototypes*, and a ﬁxed, ordered sequence of operators 184 (i.e., instantiations of transformations) known as (executable) *pipelines*. Typically, pipeline prototype

185 construction is a manual and tedious task, where a data scientist exhaustively iterates over a staggeringly

186 large number of possible pipeline orderings, until he/she ﬁnds one that works best for the problem at

187 hand. This is a challenging task due to the fact that there are no clear rules and guidelines in terms of

188 which permutation of kinds of transformations would work best (i.e., the ﬁnal impact of a pipeline is

189 difﬁcult to foresee). To facilitate it, we propose a method, sketched in Figure [3, that in short breaks](#bookmark35)

190 the combinatorial problem of ﬁnding the best pipeline into studying kinds of transformations in pairs,

191 ultimately, generating effective pipeline prototypes, which are then fed to an optimizer (e.g., we use the

192 [SMBO [8] variant), to be instantiated and further optimized](#bookmark10). Some of the steps of the method are generic

193 and thus can be applied regardless of the context, and yet others are speciﬁc, and depend on the context 194 (i.e., ML framework used or dataset characteristics).

195 The method consists of two ﬂows running in parallel. The ﬁrst ﬂow is responsible for the pipeline

196 prototype construction and the second ﬂow allows to generate rules that guide the instantiation of the

197 transformations inside the pipeline prototype. The output from the two ﬂows is fed to the ﬁnal step where

198 the instantiation and optimization happens. The result is an executable pipeline. Notice that what we

199 [propose is a generic method, however for the sake of an example, we use the OpenML repository [26],](#bookmark36)

200 the Scikit-learn library, and the HyperOpt tool, that internally uses SMBO, to provide a use case.

201 The proposed method starts with the selection of the ML library and optimization framework to be

202 used. On the one hand, this allows to choose the potential kinds of transformations and their available

203 instantiations, and on the other it allows to generate *framework-related rules*, reﬂecting the limitations

204 in the concrete implementation of operators. These rules enable the generation of precedence relation-

205 ships between the kinds of transformations for which they apply. Next, the ﬂow on top continues with a

206 study over all the possible pairs of kinds of transformations, aiming to ﬁnd the correct/meaningful order

207 between them using *generic knowledge* about their behaviour. As a result, a set of *heuristic rules* that de-

208 termines precedences between transformations is generated. Afterwards, for the pairs for which an order

209 cannot be clearly devised, an additional empirical study is proposed. This study may rely on a testbed

210 of dataset representatives, and thus it may implicitly correspond to *domain knowledge*. The output of

211 this step is a set of *empirically learned rules* that determines promising precedences of transformations 212 (i.e. an order that would potentially positively impact the ﬁnal result of the analysis). However, even

213 after this phase, for some pairs of transformations a precedence order may not be found. These are

214 pairs for whom the order is relevant but cannot be decided in advance, thus all their permutations need

215 to be present. Finally, a step of composition follows, where given the overall set of devised rules (i.e.,

216 *framework-related*, *heuristic* and *empirically learned*), transformations are composed into a set of valid

217 and potentially effective pipeline prototypes.

218 Once the prototype is constructed, the ﬂow running in parallel is proposed to help with its instan-

219 tiation. It consists of a meta-learning step, where a set of ML experiments (e.g., pre-processing and

220 classiﬁcation algorithm runs) are used as training data, to predict the initial operator for the transfor-

221 mations inside the pipeline prototype. These rules extract knowledge from past experiments and are

222 complementary to the rules obtained in the ﬁrst ﬂow. They would be used, for example, to ease the cold

223 start problem in the prototype instantiation phase.

224 *3.1. Transformations and Operators*

225 The ﬁrst task in the process consists of selecting the kinds of transformations and their available

226 operators.

227 When combining two different kinds of transformations, it is important to check if, (i) the input

228 and output types of transformations are compatible, (ii) the combination makes sense, and (iii) the

229 combination provides good results for the analysis. As a result, when chaining a pair of transformations,

230 the following precedence relationships arise:

231 1. Compatible/Incompatible pairs. Depending on whether the representation output of the ﬁrst trans-

232 formation is accepted as the representation input of the second one (compatible), or not (incom-

233 patible) (see Section [3.2)](#bookmark37).

234 2. Meaningful/Meaningless pairs. Depending on whether the precedence between them makes sense

235 based on generic knowledge (i.e., based on the literature) over the behaviour of transformations 236 (meaningful), or not (meaningless) (see Section [3.3)](#bookmark38).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transf. Kind | Input | Output | Operator | Parameters |
| Encoding (E) | CA | CO | Ordinal | - |
| One Hot | - |
| Normalization (N) | CO | CO | Standard Scaler | with mean:[True,False] with std:[True,False] |
| Power Transform | - |
| MinMax Scaler | - |
| Robust Scaler | quantile range:[(25,75),(10,90),(5,95)] with centering:[True,False]  with scaling:[True,False] |
| Discretization (D) | CO | CA | KBins | n bins:[3,5,7]  encode:[‘onehot’,‘onehot-dense’,‘ord . ’] strategy:[‘uniform’,‘quant . ’,‘kmeans’] |
| Binarization | threshold: [0, 0 .5, 2, 5] |
| Imputation (I) | CA/CO | CA/CO | Univariate | strategy:[‘most freq . ’,’constant’] |
| Multivariate | initial strategy:[‘most freq’,‘const . ’] order:[‘asc’,‘dsc’,‘rom’,‘arab’,‘rand’] |
| Rebalancing (R)\* | CA/CO | CA/CO | Near Miss | n neighbors:[1,2,3] |
| SMOTE | k neighbors:[5,6,7] |
| Feat. Eng. (F) | CA/CO | CA/CO | PCA | n components:[1,2,3,4] |
| Select K Best | k:[1,2,3,4] |
| PCA + Select K Best | n components:[1,2,3,4] k:[1,2,3,4] |

CA - Categorical, CO - Continuous.

\*All transformations except Rebalancing are taken from scikit-learn.

Table 1: List of transformations applicable to Categorical or Continuous data types.

237 3. Promising/Unpromising pairs. Depending on whether the precedence between them is expected

238 to provide positive impact on the ﬁnal result of the analysis (promising), or not (unpromising) (see

239 Section [3.4)](#bookmark15).

240 Attending to the relationships between its transformations, a prototype can be described as either

241 *compatible*, *well-formed*, or *effective*. A prototype is deﬁned to be *compatible*, if all its precedence

242 relationships are compatible. It is deﬁned as *well-formed*, if all its precedence relationships are both

243 compatible and meaningful. Finally, it is deﬁned as *effective*, if all its precedence relationships are

244 compatible, meaningful, and promising at the same time. In fact, the ultimate goal of our method is to 245 ﬁnd *effective pipelines*.

246 **EXAMPLE 1.** The kinds of transformations selected for the sake of our use case are the following:

247 • Encoding (E) . The process of transforming Categorical attributes into Continuous ones .

248 • Normalization (N) . The process of normalizing Continuous attributes such that their values

249 fall in the same range.

250 • Discretization (D) . The process of transforming Continuous attributes into Categorical ones .

251 • Imputation (I) . The process of imputing missing values.

252 • Rebalancing (R) . The process of adjusting the class distribution of a dataset (i.e . the ratio

253 between the diﬀerent classes/categories represented) .

254 • Feature Engineering (F) . The process of deﬁning the set of relevant attributes (variables,

255 predictors) to be used in model construction.

256 An operator is an actual instantiation/implementation of a kind of transformation. Thus, several

257 operators may implement the same kind of transformation, each having its own set of parameters. For

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **E** | **N** | **D** | **I** | **R** | **F** |
| **E** |  | 0 | 0 | 0 | 0 | 0 |
| **N** | 0 |  | X | 0 | 1 | 0 |
| **D** | 0 | X |  | 0 | 0 | 0 |
| **I** | 1 | 1 | 1 |  | 1 | 1 |
| **R** | 0 | 0 | 0 | 0 |  | 0 |
| **F** | 0 | 0 | 0 | 0 | 0 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **E** | **N** | **D** | **I** | **R** | **F** |
| **E** |  | 1 | 1 | 0 | 1 | 1 |
| **N** | 0 |  | 0 | 0 | 0 | 0 |
| **D** | 0 | 0 |  | 0 | 0 | 0 |
| **I** | 1 | 0 | 1 |  | 1 | 1 |
| **R** | 0 | 0 | 0 | 0 |  | 0 |
| **F** | 0 | 0 | 0 | 0 | 0 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **E** | **N** | **D** | **I** | **R** | **F** |
| **E** |  | 0 | 0 | 0 | 0 | 0 |
| **N** | 0 |  | 0 | 0 | 0 | 1 |
| **D** | 0 | 0 |  | 0 | 0 | 1 |
| **I** | 0 | 0 | 0 |  | 0 | 0 |
| **R** | 0 | 0 | 0 | 0 |  | 0 |
| **F** | 0 | 0 | 0 | 0 | 0 |  |

(a) Compatible precedence. (b) Meaningful precedence. (c) Promising precedence.

**E** - Encoding; **N** - Normalization; **D** - Discretization; **I** - Imputation; **R** - Rebalancing; **F** - Feature Engineering.

1 - a precedence edge exists between the row and the column, 0 - a precedence edge does not exist between the row

and the column, X - the combination is meaningless.

Table 2: Precedence order between pairs of transformations, represented independently for each phase.

258 [our experiments, we selected the operators and parameters from those available in the Scikit-learn3](#bookmark41)

259 library, and they are listed in Table [1.](#bookmark39) *Input* denotes the compatible feature type for a given kind of

260 transformation and can be Continuous (CO) — when it represents measurements on some continuous

261 scale, or Categorical (CA) — when it represents information about some categorical or discrete

262 characteristics. Similarly, *Output* denotes the type of the features after a kind of transformation is

263 applied. Finally, *Operator* denotes the physical instantiation for the kind of transformation, and it

264 can be parametrised using its *Parameters*.

265 *3.2. Framework-related rules*

266 Once the implementation framework is selected, one needs to study it and see if there exist con-

267 straints that limit the interaction between transformations. For instance, applying a transformation may

268 actually invalidate the application of another transformation, because the compatibility of transforma-

269 tions is dependent on the selected ML framework.

270 **EXAMPLE 2.** We studied the transformations implemented in Scikit-learn and detected a set of

271 implicit rules that are shown through an adjacency matrix, corresponding to a precedence graph,

272 in Table [2a](#bookmark40) . Each cell aij denotes a precedence relationship between the row i and column j.

273 Hence, 1 means that an edge exists between the transformation in the row and the transformation

274 in the column, whereas 0 means that such an edge does not exist, hence a precedence order is not

275 established for that pair. For example, most Scikit-learn transformations cannot be applied in the

276 presence of missing values. This is why in every pair of transformations where Imputation is involved,

277 [except the one with Normalization4](#bookmark42), Imputation goes ﬁrst. Furthermore, Scikit-learn transformations

278 are applied only to all compatible attributes of a given dataset. Generally, Categorical attributes are

279 physically represented as strings and Continuous attributes as numbers . However, a transformation

280 that is meant to be applied, say to Continuous attributes, cannot be applied over a dataset that

281 contains both Continuous and Categorical attributes (i.e., a dataset containing both numbers and

282 strings); Scikit-learn cannot deal with arrays of mixed types . In that case, all the Categorical

283 attributes need to be encoded into numeric representations, even if they represent a categorical

284 value. That is, a value can be a number but represent a category. This is what happens when

285 Normalization and Discretization are meant to be applied to a dataset containing mixed types of

286 attributes. In order for them to be applied to datasets of mixed types, an Encoding transformation

287 needs to be applied ﬁrst. A similar constraint is imposed when considering Rebalancing and Feature

288 Engineering, since these transformations do not accept inputs containing strings (i.e., representing

[3https://scikit-learn.org](https://scikit-learn.org)

4Normalization transformations are the only ones that Scikit-learn can apply on datasets with missing values.

289 a Categorical type) . For the rest of the pairs of transformations there are no constraints imposed

290 by the framework, thus any order of such transformations is permitted, reﬂected by a 0 in Table [2a](#bookmark40) .

291 The graph obtained in this case exclusively corresponds to the limitations of Scikit-learn (as a matter

292 of fact, if another framework were to be chosen, it may have looked diﬀerently) .

293 *3.3. Heuristic rules*

294 In the previous section, we proposed to derive a precedence based on the constraints of the frame-

295 work. Now, we want to study the precedence independently of the framework, and ﬁnd *meaningful pairs*.

296 That is, for every given pair, we want to ﬁnd the relative order, based on generic, domain-independent

297 knowledge (i.e., literature) about transformations and their applicability. To this end, some of the con-

298 straints imposed by the framework may be contradicted here, but this is resolved in the last step of the

299 proposed method, when we take the union of the rules and hence construct the ﬁnal pipeline prototypes 300 (see Section [3.5)](#bookmark43). Brieﬂy, in a combination where Imputation is involved, it is advised to apply Impu-

301 tation ﬁrst. Next, an Encoding transformation makes sense to be combined in any order with the rest of

302 transformations, except Imputation. Combining Discretization with Normalization does not make sense,

303 due to the fact that after the Discretization step, Continuous attributes are transformed into Categorical

304 ones, and hence Normalization cannot be applied. Similarly, applying Normalization ﬁrst, changes the

305 scale of the values and hence impacts the Discretization step. Finally, a meaningful precedence can be

306 derived when combining Normalization with Rebalancing. In this case, Normalization should be applied 307 ﬁrst, since otherwise Rebalancing would impact the scale of the values to be normalized.

308 **EXAMPLE 3.** For our use case, Table [2b shows the heuristic rules obtained considering domain](#bookmark40)-

309 [independent knowledge about transformations [27]](#bookmark44) . In comparison with the results from Table [2a,](#bookmark40)

310 observe that the constraints on the Imputation transformation still hold, that is, it is correct to apply

311 Imputation ﬁrst when combining it with another transformation. This time even when combining it

312 with Normalization — note the diﬀerence with Table [2a](#bookmark40) . The constraints of Encoding are however

313 not present in Table [2a, hence not considering the framework, Discretization combined with Encoding](#bookmark40)

314 is a meaningful combination — when a mixed type dataset is considered, but incompatible from the

315 point of view of Scikit-learn .

316 *3.4. Empirically learned rules*

317 The two previously proposed steps (i.e., *framework-related* and *heuristic rules*), do not guarantee

318 that for each pair of transformations we will obtain a precedence order. Therefore, for the cases where

319 they are not sufﬁcient to determine the precedence, a third viewpoint can be considered. That of learn-

320 ing a promising order by empirically studying the impact of the combinations on the ﬁnal result of the

|  |  |  |
| --- | --- | --- |
| **Algorithm 1** Find a promising pipeline prototype for transformations T1 and T2 | | |
| **Require:** d, a  **Require:** T1 → T2 , T2 → T1 1: acc baseline = Acc(d, a); | | # dataset, classiﬁcation algorithm  # precedence orders of a pair of transformations # get baseline performance of algorithm on d |
| 2: | [pipelineT1→T2, accT1→T2] = SMBO(T1 → T2, d, a)  # get pipeline and accuracy for T1 → T2 | |
| 3: | [pipelineT2→T1, accT2→T1] = SMBO(T2 → T1, d, a)  # get pipeline and accuracy for T2 → T1 | |
| 4: | **if** *IsValid*(accT1→T2 , accT2→T1 , acc baseline) **then** # see Table [3](#bookmark46) for the rules applied | |
| 5: | **return** *Winner*([pipelineT1→T2 , accT1→T2], [pipelineT2→T1 , accT2→T1])  # see column *Winner prototype* in Table [3](#bookmark46) | |
| 6: | **else** | |
| 7: | **return** 。 | |
| 8: | **end if** | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Pipeline 1 | Pipeline 2 | Valid result | Valid score | Winner prototype |
| 1. | 。 → 。 | 。 → 。 | Draw | accbaseline | Baseline |
| 2. | 。 → 。 | confT2 → 。 | Draw | accbaseline | Baseline |
| 3. | 。 → 。 | 。 → confT1 | Draw | accbaseline | Baseline |
| 4. | 。 → 。 | confT2 → confT1 | Draw  confT2 → confT1 | accbaseline accT2 →T1 | Baseline T2 → T1 |
| 5. | 。 → confT2 | 。 → 。 | Draw | accbaseline | Baseline |
| 6. | 。 → confT2 | confT2 → 。 | Draw  。 → confT2 confT2 → 。 | accT2 | T2 T2 T2 |
| 7. | 。 → confT2 | 。 → confT1 | Draw | accT2 or accT1 | T1 orT2 |
| 8. | 。 → confT2 | confT2 → confT1 | Draw  confT2 → confT1 | accT2  accT2 →T1 | T2  T2 → T1 |
| 9. | confT1 → 。 | 。 → 。 | Draw | accbaseline | Baseline |
| 10. | confT1 → 。 | confT2 → 。 | Draw | accT1 or accT2 | T1 orT2 |
| 11. | confT1 → 。 | 。 → confT1 | Draw  confT1 → 。 。 → confT1 | accT1 | T1 T1 T1 |
| 12. | confT1 → 。 | confT2 → confT1 | Draw  confT2 → confT1 | accT1  accT2 →T1 | T1  T2 → T1 |
| 13. | confT1 → confT2 | 。 → 。 | Draw  confT1 → confT2 | accbaseline accT1 →T2 | Baseline T1 → T2 |
| 14. | confT1 → confT2 | confT2 → 。 | Draw  confT1 → confT2 | accT2  accT1 →T2 | T2  T1 → T2 |
| 15. | confT1 → confT2 | 。 → confT1 | Draw  confT1 → confT1 | accT1  accT1 →T2 | T1  T1 → T2 |
| 16. | confT1 → confT2 | confT2 → confT1 | Draw  confT1 → confT2 confT2 → confT1 | accT1 or accT2 accT1 →T2  accT2 →T1 | T1 orT2 T1 → T2 T2 → T1 |

。- SMBO ﬁnds a better result without instantiating a transformation (or both) in the pair. confT - The conﬁguration (i.e., operator and its parameters) found for T by SMBO.

accT - The accuracy of the ML algorithm over the data transformed with a pipeline T.

Table 3: Validation rules.

321 analysis, using different classiﬁcation problems in the training. For every selected pair of transforma-

322 tions,for a given classiﬁcation algorithm, we propose to check which order of the pair improves most

323 the performance (e.g., predictive accuracy) of the algorithm over a set of datasets (preferably from dif-

324 ferent domains). Like this, for each dataset we can get a precedence order that gives better results (i.e.,

325 promising precedence) in terms of predictive accuracy (other metrics can be used as well).

326 *3.4.1. Algorithm*

327 To ﬁnd a promising precedence order between a given pair of transformations, we propose Algo-

328 rithm [1.](#bookmark45) To compute the impact of transformations, we ﬁrst get the accuracy of the ML algorithm over

329 the original non-transformed dataset (see line 1). Afterwards, for each precedence order between the

330 pairs of transformations, we ﬁnd both their optimized executable pipelines (i.e., using SMBO), and the

331 accuracies of the ML algorithm (with default parametrisation) over the datasets transformed using the re-

332 spective pipelines (see lines 2-3). Based on the comparison between the respective optimized pipelines,

333 we get the winner inline 5. However, beforehand, inline 4, we perform a validity check. This is because

334 when optimizing a pre-processing pipeline, SMBO may not instantiate a transformation with an operator

335 at all (i.e., represented with a 。symbol). Hence, given a pair of transformations, where one or both of

336 them may not be instantiated, SMBO may generate 16 possible scenarios. They are listed in Table [3,](#bookmark46)

337 and make up the validation rules for Algorithm [1](#bookmark45) (see line 4).

338 Brieﬂy, if among the optimized pairs of transformations (same transformations but in reverse order)

339 obtained from SMBO, one or both of them contain a 。operator, their results are considered valid, only

340 if they have equal scores (i.e., a draw). This is because, if one has a higher score, it means that during

341 the optimization phase it was more advantageous than the other, since it could ﬁnd a conﬁguration that

342 should have been found by both of the pairs, given enough budget. In our SMBO runs, such invalid

343 results account for less than 10% and in those cases datasets are discarded from the study (see line 7).

344 In particular, in Table [3, the ﬁrst two columns denote the pipeline instantiations for the respective](#bookmark46)

345 pairs of transformations (i.e., T1 → T2 and T2 → T1). Next, *Valid result* denotes the expected result

346 when comparing the results of the pipelines in the same row. For instance, in the ﬁrst row, if both

347 transformations in the pipelines are not instantiated during the optimization, a valid result is a draw, and

348 a *Valid score* for the respective result is the baseline accuracy, and the *Winner prototype* is the prototype

349 that is in accordance with the expected result, which in this case is the Baseline (i.e., prototype consisting

350 of only the ML algorithm, where no transformations are applied).

351 For the sake of another example, let us check row 2 in Table [3.](#bookmark46) Running SMBO, the best result for

352 the ﬁrst pair is the pipeline 。 → 。, and for the second pair, the pipeline T2 → 。. In this case, the

353 comparison between the results of these pipelines should be equal (i.e., draw), and the score should be

354 that of the baseline. Otherwise, if say, the score of the second pipeline was higher, it would mean that

355 for the ﬁrst pair, SMBO was not given enough time to ﬁnd the pipeline with higher score (i.e., T2 → 。).

356 The same logic applies also for the other rows where a 。operator is involved.

357 **EXAMPLE 4.** For the sake of this work, we considered three classiﬁcation algorithms (i.e . , *NB*,

358 *RF*, *KNN*) and 80 datasets from the OpenML repository. The datasets, were compiled from three

359 [OpenML benchmarks, namely, the OpenMLCC18 benchmark5](#bookmark48)[, the AutoML benchmark6](#bookmark49), and the

360 [Classiﬁcation algorithms benchmark7](#bookmark50). For the ﬁnal set, we ﬁltered out datasets with more than 361 10% of missing values — not to include bias due to the heavy pre-processing we need to perform

362 on top of them, and we ﬁltered out the datasets with more than 5 million instances — because of

363 the computation time required to process them. As a result we obtained 60 datasets from the ﬁrst

364 benchmark, 17 from the second, and 3 more from the third to reach a total of 80 datasets.

[5https://www.openml.org/s/99/data](https://www.openml.org/s/99/data) [6https://www.openml.org/s/271/data](https://www.openml.org/s/271/data) [7https://www.openml.org/s/1/data](https://www.openml.org/s/1/data)

*T*1 = Feat. Eng., *T*2 = Normalize

 *T*1  *T*2  *T*1→*T*2  *T*2→*T*1 Zz Baseline

40

Number of wins Number of wins

30

20

10

0

40

30

20

10

0

Number of wins Number of wins

40

30

20

10

0

|  |  |  |
| --- | --- | --- |
|  |  |  |

NB KNN RF

Algorithms

*T*1 = Feat. Eng., *T*2 = Rebalance

40

30

20

10

0

|  |  |  |
| --- | --- | --- |
|  |  |  |



NB KNN RF

Algorithms

*T*1 = Discretize, *T*2 = Feat. Eng.

|  |  |  |
| --- | --- | --- |
|  |  |  |

NB KNN RF

Algorithms

*T*1 = Discretize, *T*2 = Rebalance

|  |  |  |
| --- | --- | --- |
|  |  |  |



NB KNN RF

Algorithms

Figure 4: Number of datasets for which a given pipeline prototype is declared the winner.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T1 | T2 | T1 → T2 | T2 → T1 | alpha | p-value |
| F D F D | N F R R | 3  70  49  66 | 88  7  61  86 | 0.05 0.05 0.05 0.05 | **0**  **0**  8.53e-01 9.38e-01 |

N - Normalization; D - Discretization; R - Rebalancing; F

- Feature Engineering.

Table 4: Binomial test for determining the order between pairs of transformations.

365 Given the proposed algorithm (i.e . Algorithm [1), we could try to learn the precedence of every](#bookmark45)

366 pair of transformations, but would just be a waste of resources, because we can see in Table [2a](#bookmark40)

367 and [2b, that some precedences are already decided for one reason or another](#bookmark40) . Hence, only pairs of

368 transformations with a 0 for both directions (in both Table [2a and](#bookmark40) [2b) need to be studied further](#bookmark40) .

369 That is, they make sense to be combined together, but a precedence order could not be determined

370 through *framework-related* or *heuristic rules*. Thus, for instance, pairs involving Encoding are not

371 considered in this phase, since for them an order is already imposed by the framework (see Table [2a)](#bookmark40) .

372 To this end, the pairs of transformations we consider for the third precedence graph include only 373 {F, N}, {F, D}, {F, R}, and {R, D}.

374 Applying Algorithm [1, we obtain a promising order for each pair of transformations considered](#bookmark45) .

375 Since SMBO is a randomized algorithm we experimented with (i) running it several times splitting

376 the budget, and (ii) running it only once with the entire budget. For the experiments considered,

377 no signiﬁcant diﬀerences where observed, therefore we opted for running it once with the entire

378 budget (i.e., 200 seconds per run), which allows for more conﬁgurations to be visited in a single run .

379 Aggregating all the results, Figure [4](#bookmark51) shows the number of datasets, for which a given prototype (see

380 Table [3](#bookmark46), column *Winner prototype* for the list of labels) is selected as the winner. For instance, for

381 the pair {F, N} (i.e., Feature Engineering, Normalization), the prototype winning in more datasets

382 for *KNN* and *NB* is N → F. This means that in general, better results are obtained if Normalization

383 is applied before Feature Engineering.

384 Next, only N appears as ﬁrst for *RF* and second best for *KNN* and *NB*, which means that

385 for many datasets, considering diﬀerent algorithms, it results better to apply only Normalization

386 without combining it with Feature Engineering. The third position is for 。 → 。, which means

387 that for some datasets it is better not to apply any of the transformations (in any combination) .

388 The remaining prototypes winning in some datasets are F (only Feature Engineering), and F → N 389 (Feature Engineering preceding Normalization) . Finally, for three datasets, that are omitted from

390 the ﬁgure, there were no winning pipelines (i.e., pipelines resulted in a draw) .

391 Since our goal is to ﬁnd the best order for a pair of transformations, we focus on the performances

392 of the pipelines where both of the transformations are instantiated (i.e . , T1 → T2 versus T2 → T1) .

393 To do this, we check whether the diﬀerence between the number of datasets where they each appear

394 to win are statistically signiﬁcant by running a binomial test assuming a theoretical probability of 395 0.5. The results are shown in Table [4.](#bookmark52) In summary, the results from Table [4](#bookmark52) indicate that, with 396 95% conﬁdence we can assume that for the pair {F, N}, N → F performs better than F → N ,

397 hence Normalization should precede Feature Engineering. On the other hand, for {D, F}, D → F

398 performs better than F → D, hence Discretization should precede Feature Engineering. Finally, for

399 the remaining transformations, {F, R} and {R, D}, a precedence order can not be pre-assumed since

400 the results obtained are not signiﬁcant. Using these results, we create the *Promising precedence*

401 adjacency matrix shown in Table [2c, where as one can observe, precedence edges are introduced for](#bookmark40) 402 {N, F} and {D, F}, but no edges exist neither for {F, R}, nor for {R, D}.

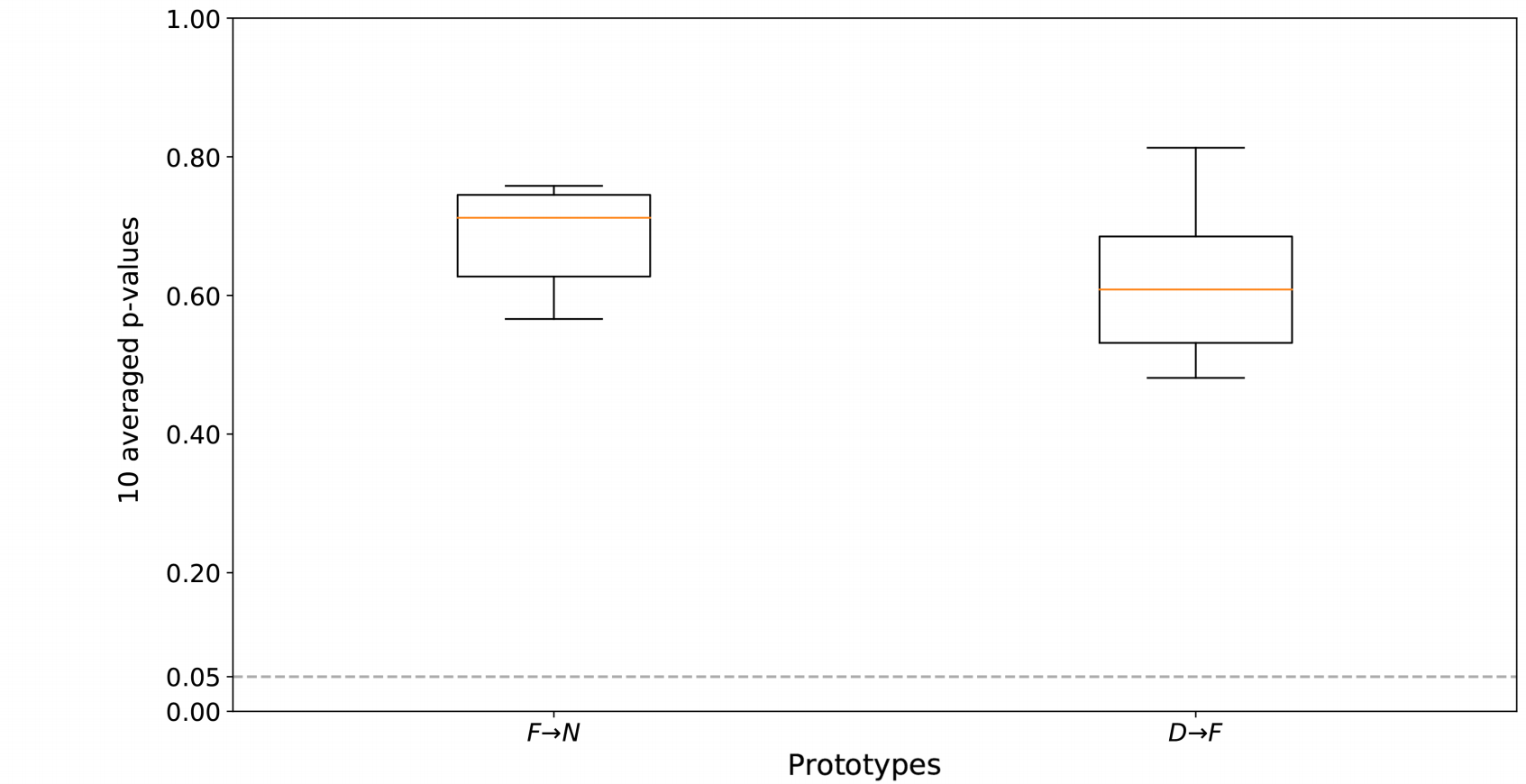


Figure 5: The distribution of the p-values obtained after repeating the chi-square test for 10 times, for the 10 times 4-fold

cross-validation.

403 *3.4.2. Cross-validation*

404 After running Algorithm [1](#bookmark45) to empirically ﬁnd a winner between two pairs of transformations, we

405 may obtain a different distribution of the number of wins for the pairs, depending on the datasets con-

406 sidered. To show that the results obtained with the initial set of datasets are generalizable, we propose

407 to perform an additional cross-validated experiment, where the set of datasets considered can be ran-

408 domly split into many folds. Then, for each fold, the results can be compared to the rest, with the aim

409 of checking whether the distributions are similar. This check can be done via a signiﬁcance test (e.g.,

410 chi-square). To this end, if the distributions between the folds are similar, it means that the obtained

411 results are independent of the datasets considered, since no matter the combination of the datasets, the

412 results are the same and thus generalizable.

413 **EXAMPLE 5.** In our use case, to show that the results do not depend on the datasests selected,

414 we re-run the experiments (i.e., 10-times each), but this time splitting the datasets into 4-folds.

415 The goal was to check if the results of the precedence orders from the diﬀerent folds (i.e., for

416 each experiment considering a randomly diﬀerent set of datasets) are similar between them (i.e . ,

417 follow the same distributions) . To conﬁrm this hypothesis, we perform a chi-square test between

418 the results (precedence orders) obtained in a single fold in comparison to the three remaining folds,

419 hence comparing 25% of the datasets to the rest. In particular, to conﬁrm the hypothesis, we need

420 to ﬁnd results that accept the null hypothesis of the chi-square test which states that ”there is no

421 signiﬁcant diﬀerence between the distributions”. To do that, sticking to the 95% conﬁdence interval,

422 we need to look for p-values greater than 0.05 . That is, the higher the p-values, the more we accept

423 the null hypothesis, the more similar the distributions. Looking at the p-values we found out that

424 they were all much higher than 0.05 . Speciﬁcally, the scores of the chi-square tests of the folds (one

425 fold compared to the rest) are averaged and, after having repeated this procedure 10 times, instead

426 of using a table we depict the 10 averaged p-values using box-plots in Figure [5.](#bookmark53) We conclude that,

427 for both of the rules (i.e . , F → N and F → D), the signiﬁcance test indicates a compliance between

428 the new results (Figure [5) and those illustrated above (Table](#bookmark53) [4)](#bookmark52) .

429 *3.5. Effective pipeline prototypes*

430 In this task we foresee the composition of the previously deﬁned rules (i.e., for the pairs of transfor-

431 mations), to generate the ﬁnal set of rules that would allow to compose longer chains — consisting of

N

R



F

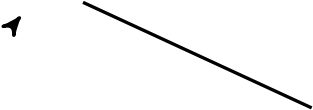


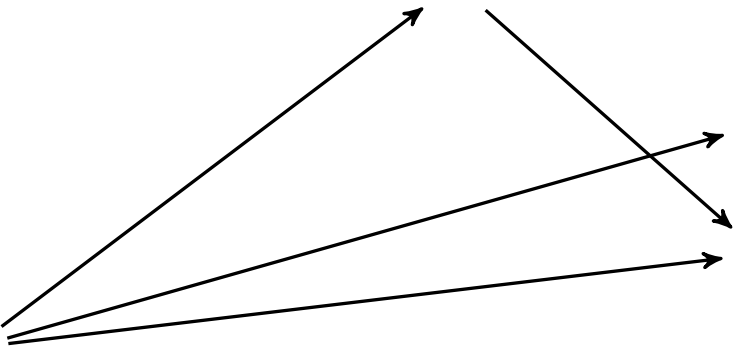
D

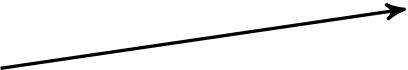
Figure 6: Precedence graph generated from Table [5.](#bookmark54) E - Encoding; N - Normalization; D - Discretization; I - Im- putation; R - Rebalancing; F - Feature Engineering.

|  |
| --- |
| ID Pipeline prototype |
| 1 I → E → N → R → F |
| 2 I → E → N → F → R |
| 3 I → E → R → D → F |
| 4 I → E → D → R → F |
| 5 I → E → D → F → R |

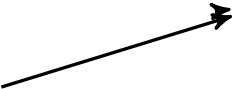
Table 6: Effective pipeline prototypes generated from Figure 6. E - Encod- ing; N - Normalization; D - Discretiza- tion; I - Imputation; R - Rebalancing; F - Feature Engineering.











A



I

E

432 more than two transformations. This is when we resolve the inconsistencies and also deﬁne precedences

433 for the pairs of transformations that may not have any precedence deﬁned already — in that case, we ba-

434 sically take into account all the permutations. This step allows to ﬁnally generate the possible effective

435 pipeline prototypes.

436 **EXAMPLE 6.** To generate the ﬁnal pipeline prototypes, in this step we combine all the matrices

437 generated by the previous steps. That is, we take the union of the edges (represented by 1’s) from

438 the matrices in Table [2](#bookmark40) (a,b,c), and create a new ﬁnal adjacency matrix, shown in Table [5.](#bookmark54) This is

439 the matrix that will allow us to generate the ﬁnal eﬀective pipeline prototypes .

440 Observing the table, one can realize that for pairs {F, R} and {R, D}, no precedence edges

441 exist. This means that these pairs are somewhat equally relevant from either direction (any order),

442 and thus when generating the ﬁnal prototypes, both options should appear .

443 For a better reading, in Figure 6, we visualize Table [5](#bookmark54) in form of a graph, where nodes represent

444 the kinds of transformations and the directed edges represent a precedence order between them. Out

445 of the graph, we generate the ﬁnal pipeline prototypes by taking all the maximum length variations 446 (ordered arrangements without repetition) of the nodes, respecting the precedence rules (i.e., not

447 contradicting the direction of existing edges) . The result is the set of ﬁve pipeline prototypes shown

448 in Table 6 . This set consisting of *compatible*, *meaningful* and *promising* pairs of transformations is

449 the set of recommended *eﬀective pipeline prototypes*.

450 *3.6. Meta-learning rules*

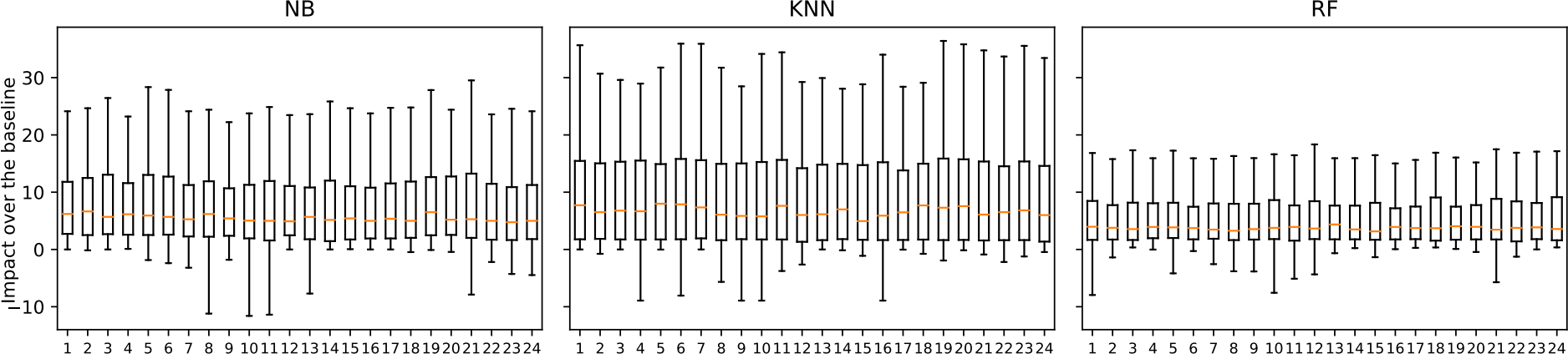
451 Once the pipeline prototype is constructed, that is, the order between the kinds of transformations

452 is deﬁned, what follows is the instantiation of transformations with the physical operators. For that,

453 one can rely completely on the optimization algorithm, and let the algorithm choose the right operators.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **E** | **N** | **D** | **I** | **R** | **F** |
| **E** |  | 1 | 1 | 0 | 1 | 1 |
| **N** | 0 |  | X | 0 | 1 | 1 |
| **D** | 0 | X |  | 0 | 0 | 1 |
| **I** | 1 | 1 | 1 |  | 1 | 1 |
| **R** | 0 | 0 | 0 | 0 |  | 0 |
| **F** | 0 | 0 | 0 | 0 | 0 |  |

Table 5: Union of rules from Table [2.](#bookmark40) **E** - Encoding; **N** - Normalization; **D** - Discretization; **I** - Imputation; **R** - Rebalancing; **F** - Feature Engineering 1 - an edge exists, 0 - edge does not exist, X - the combination is meaningless.



Prototypes ID Prototypes ID Prototypes ID

Figure 7: The impact of the different pipeline prototypes over the baseline (i.e., when no transformation is applied).

454 However, given the way optimization algorithms work (e.g., SMBO) — successively ﬁnding better and

455 better instantiations, there is a cold-start problem, where in the beginning, the algorithm does not have

456 enough information in order to come up with the most promising initial instantiations, and a wrong

457 choice may affect the optimization process.

458 *3.6.1. Exploratory analysis*

459 Given the availability of the experimental SMBO executions (executed in an exhaustive manner,

460 considering all the pipeline prototypes), one can perform an exploratory analysis with the aim of remov-

461 ing useless prototypes, pipelines or operators. Hence, further tweaking the search space. In particular,

462 starting from the highest level, that of prototypes, then going to the physical pipelines, and ﬁnally to the

463 actual operators inside the pipeline, one can analyze if:

464 • there exist some combination of transformations in the form of prototypes (see Table [7](#bookmark56) for the

465 exhaustive list of prototypes), that are generally useless (i.e., in terms of their impact to the ﬁnal

466 accuracy), and thus can be discarded a priori in order to reduce the search space,

467 • there are some physical pipelines that are consistently chosen more often than others by the opti-

468 mization algorithm, meaning that they are more useful than others,

469 • within the physical pipelines, some transformations are chosen more often than others, meaning

470 that they provide more positive impact.

471 **EXAMPLE 7.** We performed the above-mentioned analysis to our use case, but it did not lead

472 to any conclusive or signiﬁcant results. In particular, as shown in Figure [7, we could not ﬁnd any](#bookmark55)

473 useless prototypes — not positively impacting the ﬁnal accuracy, that could be discarded a priori

474 from the potential list of prototypes . Actually, as we will show in Section [4.1, all of them lead to](#bookmark57)

475 the best in one case or another, which does not mean the epsilon improvement some provide is

476 worth the search cost you incur in considering them (but this more in-depth analysis is done later) .

477 Next, as shown in Figure [8, there were no physical pipelines shown to be more useful](#bookmark58) — hence

478 more often selected, than others. Even if N → R is clearly above, it barely reaches 30% in KNN.

479 Finally, observing Figure [9, it is clear that some kinds of transformations are chosen more often,](#bookmark59)

480 but looking closely (i.e., the shaded bars), it is not clear which operator brings more beneﬁt. For

481 instance, Normalization is present in 90% of the pipelines, but it is not easy to distinguish which

482 kind of Normalization (i.e., actual operator) is more beneﬁcial. For this, we need more complex rules

483 or guidelines that may help in ﬁnding the right operator to use.

484 *3.6.2. Meta-learning*

485 To mitigate the red cold-start problem, we propose to perform meta-learning (shown in Figure [3),](#bookmark35)

486 where we intend to use the knowledge extracted from historical data in order to devise rules that may

487 help the optimization algorithm in its initial phase. Meta-learning is the process of ‘learning on top of 16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *D*  *D*→*F*  *D*→*F R* | *D*→*R*  *D*→*R*→*F*  *F*→*R* | *N*  *N*→*F*  *N*→*F R* | *N*→*R*  *N*→*R*→*F*  *R* | *R*→*D*  *R*→*D*→*F*  *R*→*F* |



30%

25%

20%

Usage

15%

10%

5%

0%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

NB KNN RF

Algorithms

Figure 8: Percentage of use of the different physical pipelines.

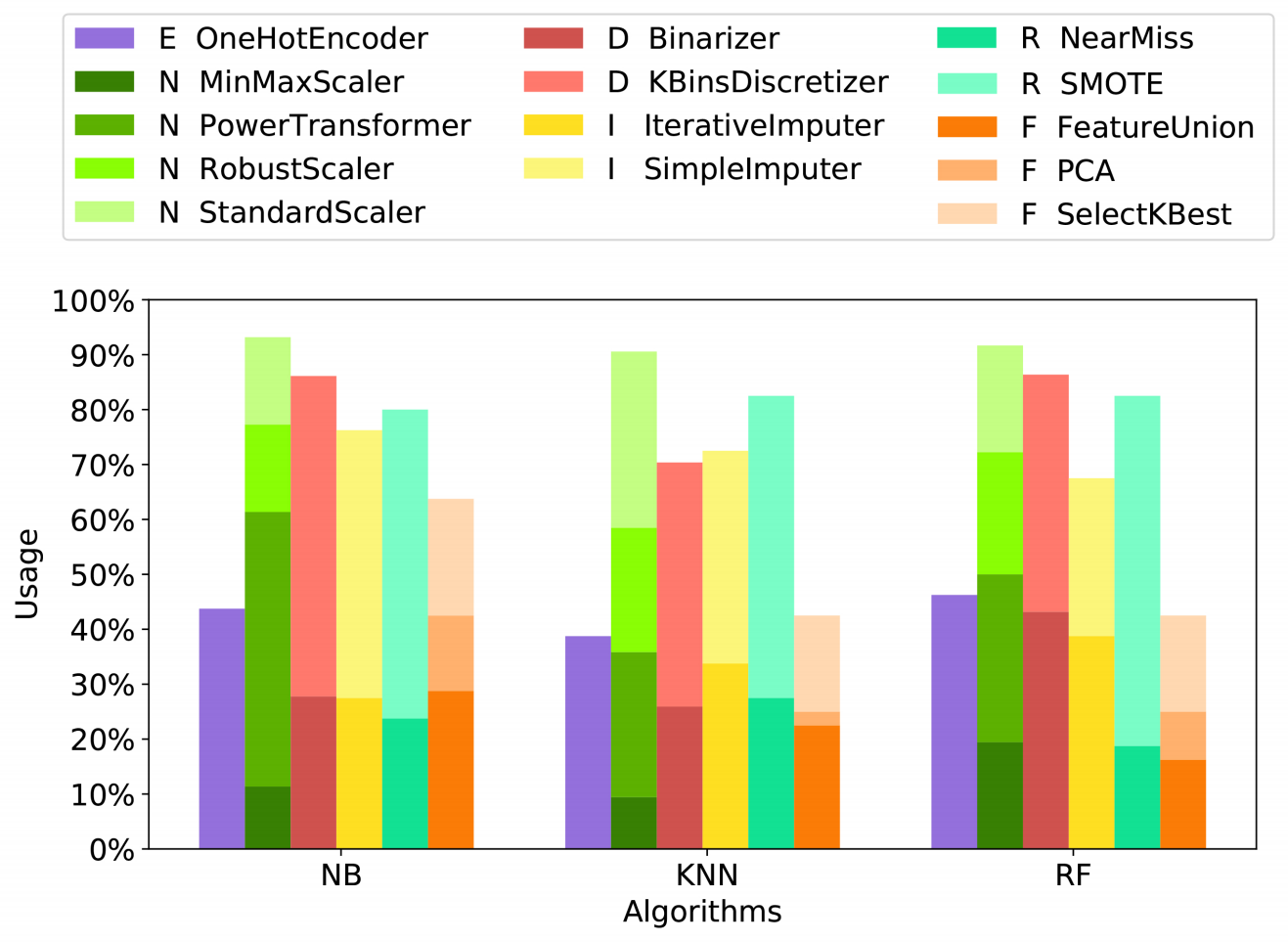


Figure 9: Percentage of use of a transformation in a physical pipeline.

488 learning’, or learning a model using historical data from ML experiments. Traditionally, it has been

489 used for predicting the performance (e.g., predictive accuracy) of an algorithm on a given dataset. That

490 is, given some historical runs of the performance of classiﬁcation algorithms over various datasets (i.e.,

491 meta-database: consisting of datasets characteristics as predictive variables and the performance of the

492 classiﬁcation algorithm as the response variable), one can learn a model (i.e., meta-model), that is able to

493 [predict the performance of a given classiﬁcation algorithm on a new dataset [28]](#bookmark60). Lately, this technique

494 has been extended in order to predict the impact of transformations over the performance of classiﬁcation

495 [algorithms and thus rank transformations based on their impact [29,](#bookmark61) [30,](#bookmark62) [31]](#bookmark63). The same idea can be

496 applied for learning the best operator for a given transformation. That is, through meta-learning one can

497 learn the intrinsic relationship between dataset characteristics and the operator performance, and thus

498 come up with rules that are not obvious and are effective at the time of instantiating a transformation.

499 The main idea is to build a model, that is able to predict the operator for a certain kind of transformation,

500 given the meta-features extracted from the dataset considered for the optimization. This translates to

501 answering the following question: “given that we know the dataset characteristics and having selected a

502 certain kind of transformation (e.g., missing value imputation), what is the optimal physical algorithm 503 (see Table [1) we need to select, to obtain the highest improvement possible in terms of classiﬁcation](#bookmark39)

504 accuracy (i.e., when the classiﬁcation algorithm is applied over the transformed dataset)?” . In particular,

505 the model can generate a set of complementary rules that help in the optimization, providing a good

506 starting instantiation for some of the transformations in the prototype.

507 To train the model we need a meta-dataset that can be (i) generated through optimization algorithms 508 (e.g., SMBO executions), (ii) generated manually through simple evaluations of classiﬁcation algorithms

509 over transformed datasets, or (iii) assumed already given (e.g., OpenML).

510 Given a meta-dataset, we propose to learn to predict the best instantiation (operator) for a given

511 transformation, where among the classes we can include the class None too. This means that one of the

512 possible predictions is to not instantiate a transformation at all, hence remove it from the pipeline.

513 **EXAMPLE 8.** Our training dataset (sometimes referred to as ‘meta-database’ or ‘meta-dataset’)

514 for the meta-learning is compiled through SMBO runs on the OpenML datasets (see Section [3.4.1)](#bookmark47) .

515 That is, we ﬁrst extract the dataset characteristics/proﬁles (i.e., number of features, number of

516 instances, number of missing values, etc), and then by applying SMBO optimization, on classiﬁcation

517 algorithms and pre-processing pipelines (as explained in Section [3.5), for each dataset, we retrieve the](#bookmark43)

518 evaluations (i.e., predictive accuracy) of the algorithms over the optimized pipelines. This gives us

519 the presumably optimal physical pipelines and their impact on the accuracy of the learning algorithms

520 for each dataset at hand. Given such information, our aim is to now save time and improve the

521 instantiation of the operators for each transformation considered in the prototype .

522 [We trained several diﬀerent Conditional Inference Trees [32] because they produce models that](#bookmark64)

523 can be easily read and interpreted. Speciﬁcally, the independence of each variable (meta-features in

524 our case) with the class (operator of a speciﬁc transformation) is tested through a statistical test.

525 The split is made on the variable with the lowest p-value. We report the p-value too, so that it can

526 be seen how strong the association is (i.e., why that variable was chosen) . We stick with the p-value

527 threshold of 0.05, and devise a rule from any branch of the tree that is within the threshold. In the

528 following, we describe the rules obtained within the selected signiﬁcance threshold.

529 **Rules for Feature Engineering**. The available operators in Scikit-learn for Feature Engineering 530 (see Table [1) are:](#bookmark39) PCA (Principal Component Analysis), Feature Selection (Select K Best),

531 Both (PCA + Select K Best), and None. The tree generated for the Feature Engineering trans-

532 formation is shown in Figure [10.](#bookmark65) The leaves show the selected operator frequency. For the sake of

533 simplicity, we do not consider the union of PCA and Select K Best as an operator per se, instead

534 we distribute that contribution to the two operators that compose it. Observe that there is a strong

535 correlation between the Feature Engineering operator and the entropy of the class attribute. Indeed,

536 such a meta-feature achieved a p-value smaller than 0.001 . We can clearly read that if the Class

537 Entropy is low, then Feature Selection is way more chosen than the other options (see Node 538 2) . Recall that the entropy of an attribute is a measure of how much disorder there is among its

539 instances. The less is that value, the easier is the classiﬁcation problem. As a consequence, it is rea-

540 sonable to think that the easier the classiﬁcation problem is, the more likely is the fact that the class

541 can be described by a low number of features. Hence, the Feature Selection technique can

542 be successfully applied. Conversely, Node 5 shows that, when the Class Entropy is high, it is better

543 to not apply any Feature Engineering operator . As a matter of fact, a high value of Class Entropy

544 involves a high number of classes and/or few instances per class, hence a really diﬃcult problem. In

545 such cases, reducing the dimensionality of the dataset does not lead to any improvement. Finally,

546 when the Class Entropy is in between, there is no clear winner, and thus other non-obvious factors

547 may aﬀect the choice of the operator .

548 **Rules for Rebalancing**. [As for Rebalancing, the operators considered from the imblearn8](#bookmark66) library

549 are: Near Miss, SMOTE, and None. The ﬁrst is an undersampling algorithm which randomly

550 eliminates the samples from the larger class. Instead, the second is an oversampling technique that

551 creates samples of the minority class, as a linear combination of them. As shown in Figure [11, the](#bookmark67)

552 meta-feature Majority Class Percentage has a p-value of 0.014 . This can be read as, in case of an

553 unbalanced class problem (i.e., Node 3: Majority Class Percentage greater than 56), an oversampling

554 of the minority class(es) is preferred to a downsampling of the majority one(s) . However, when

555 the Majority Class Percentage is smaller than 56%, the situation is not that clear, and there is

556 no technique that is applied signiﬁcantly more often than the rest; they are close to each other.

557 Therefore, it is diﬃcult to understand which problems (which dataset characteristics do they have)

558 belong to Node 2 . In summary, when the majority class has no more than 56% , it implies that it is

559 an unbalanced class, and as mentioned above, SMBO tends to choose the same operator . However,

560 when the majority class has less than 56%, it may imply that: (i) there are just two classes and the

561 problem counts as a balanced problem, so no operator needs to be applied, or (ii) it is a multi-class

562 problem, and thus there is no clear winner in terms of operators .

563 *3.7. Prototype instantiation*

564 The prototypes from the top ﬂow and the meta-lerning rules from the bottom ﬂow (if the optimization

565 framework permits), are ﬁnally fed to the ﬁnal step which deals with the instantiation and optimization

566 of the prototypes. In this task we run an optimization algorithm that is executed until an optimal pipeline

567 is found.

568 **EXAMPLE 9.** In our ﬁnal execution, we run SMBO to ﬁnd a suitable instantiation for the suggested

569 prototypes . The simple but not obvious meta-learning rules, even though not included in our ﬁnal

570 execution, because of the implementation considered (i.e., HyperOpt), can potentially be used to

571 ease the cold-start problem.

572 **4. Evaluation**

573 The aim of our experimental study is three-fold:

574 1. Check whether there exists a universal pipeline prototype that works best for any classiﬁcation

575 problem considered (i.e., dataset and ML algorithm) (Section [4.1)](#bookmark57).

576 2. Assess and compare the performance of the effective pipelines constructed using our method

577 against the set of exhaustively generated pipeline prototypes (Section [4.2)](#bookmark68).

578 3. Assess and compare the impact of dedicating a portion of the optimization time to the effective

579 pipelines constructed using our method, with the impact of using the whole optimization time for

580 the hyper-parameters of the ML algorithm (Section [4.3)](#bookmark69).

581 The experiments were performed on an Intel Core i7 machine with 12 cores, running at 3.20 GHz

582 with 64 GB of main memory. As a platform for running the SMBO optimization algorithm we use

583 HyperOpt. Furthermore, the datasets used in the experiments are the ones from the OpenML repository 584 (see Section [3.4.1)](#bookmark47). Finally, the classiﬁcation algorithms considered are *NB*, *KNN*, and *RF*. All the

585 [experiments for a single algorithm, on average took approximately two weeks9 .](#bookmark70)

[8https://pypi.org/project/imbalanced-learn](https://pypi.org/project/imbalanced-learn)

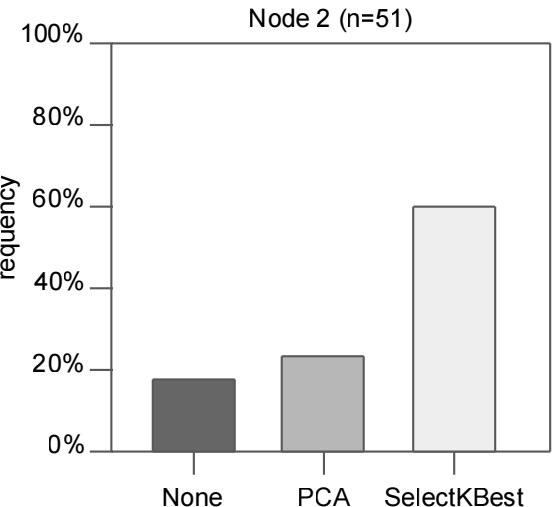
9The source code and the datasets for reproducing the experiments can be found in [https://github.com/josephgiovanelli/effective preprocessing pipeline evaluation](https://github.com/josephgiovanelli/effective_preprocessing_pipeline_evaluation)

Class Entropy p<0.001

> 0.863

0.863

Class Entropy p<0.001

1.787 > 1.787

F

Frequency

Frequency

Node 5 (n=78)

100%

100%

Node 4 (n=111)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | | |
|  |  |
|  |  |
|  |  |  |

80%

80%

60%

60%

40%

40%

20%

20%

0%

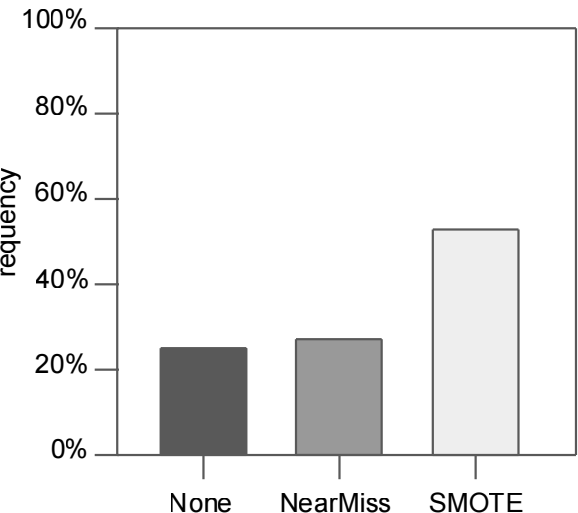
0%

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | |  | |
|  |  |
|  |  |  |  |  |

None PCA SelectKBest

None PCA SelectKBest

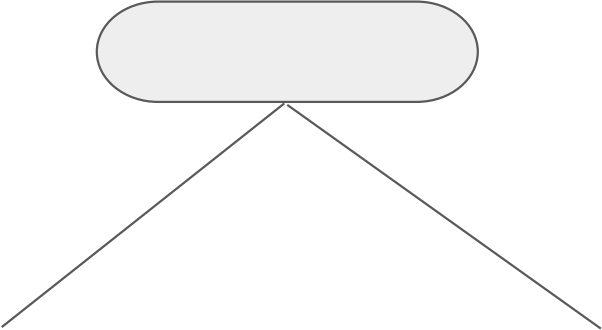
Figure 10: Conditional Inference Tree built for the *Features Engineering* transformation.



Node 2 (n=150)

Node 3 (n=90)

|  |  |
| --- | --- |
|  |  |



MajorityClassPercentage p=0.014

> 0.56

100%

0.56

80%

Frequency

60%

F

40%

20%

0%

None NearMiss SMOTE

Figure 11: Conditional Inference Tree built for the *Rebalancing* transformation.

586 *4.1. Universal pipeline prototype*

587 The goal of this experiment is to demonstrate the difﬁculty of blindly ﬁnding the right pipeline

588 prototype (i.e., without considering any meaningful or promising precedence). In Table [7, we list the](#bookmark56)

589 exhaustive set of pipeline prototypes generated considering the compatible precedence graph in Table [2a](#bookmark40) 590 (i.e., 24 compatible permutations). In a real scenario, this number would be too high for splitting the

591 time budget in order to optimize them. Yet, for the sake of this experiment, we exhaustively optimize

592 all the prototypes, for each dataset. Thus, for each pipeline prototype and for each dataset, the SMBO

593 algorithm is conﬁgured to assign a 200 seconds time budget to the phase of instantiating and optimizing

594 the pipeline prototype, and another 200 seconds to the phase of optimizing the hyper-parameters of the

595 ML algorithm.

596 The results obtained are shown in Figure [12.](#bookmark71) The enumerated prototypes are listed in the ordinate 20

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Pipeline prototype | ID | Pipeline prototype |
| 1 | I  ~~)~~ E  ~~)~~ N  ~~)~~ D  ~~)~~ F  ~~)~~ R | 13 | I  ~~)~~ E  ~~)~~ F  ~~)~~ N  ~~)~~ D  ~~)~~ R |
| 2 | I  ~~)~~ E  ~~)~~ N  ~~)~~ D  ~~)~~ R  ~~)~~ F | 14 | I  ~~)~~ E  ~~)~~ F  ~~)~~ N  ~~)~~ R  ~~)~~ D |
| 3 | I  ~~)~~ E  ~~)~~ N  ~~)~~ F  ~~)~~ D  ~~)~~ R | 15 | I  ~~)~~ E  ~~)~~ F  ~~)~~ D  ~~)~~ N  ~~)~~ R |
| 4 | I  ~~)~~ E  ~~)~~ N  ~~)~~ F  ~~)~~ R  ~~)~~ D | 16 | I  ~~)~~ E  ~~)~~ F  ~~)~~ D  ~~)~~ R  ~~)~~ N |
| 5 | I  ~~)~~ E  ~~)~~ N  ~~)~~ R  ~~)~~ D  ~~)~~ F | 17 | I  ~~)~~ E  ~~)~~ F  ~~)~~ R  ~~)~~ N  ~~)~~ D |
| 6 | I  ~~)~~ E  ~~)~~ N  ~~)~~ R  ~~)~~ F  ~~)~~ D | 18 | I  ~~)~~ E  ~~)~~ F  ~~)~~ R  ~~)~~ D  ~~)~~ N |
| 7 | I  ~~)~~ E  ~~)~~ D  ~~)~~ N  ~~)~~ F  ~~)~~ R | 19 | I  ~~)~~ E  ~~)~~ R  ~~)~~ N  ~~)~~ D  ~~)~~ F |
| 8 | I  ~~)~~ E  ~~)~~ D  ~~)~~ N  ~~)~~ R  ~~)~~ F | 20 | I  ~~)~~ E  ~~)~~ R  ~~)~~ N  ~~)~~ F  ~~)~~ D |
| 9 | I  ~~)~~ E  ~~)~~ D  ~~)~~ F  ~~)~~ N  ~~)~~ R | 21 | I  ~~)~~ E  ~~)~~ R  ~~)~~ D  ~~)~~ N  ~~)~~ F |
| 10 | I  ~~)~~ E  ~~)~~ D  ~~)~~ F  ~~)~~ R  ~~)~~ N | 23 | I  ~~)~~ E  ~~)~~ R  ~~)~~ D  ~~)~~ F  ~~)~~ N |
| 11 | I  ~~)~~ E  ~~)~~ D  ~~)~~ R  ~~)~~ N  ~~)~~ F | 23 | I  ~~)~~ E  ~~)~~ R  ~~)~~ F  ~~)~~ N  ~~)~~ D |
| 12 | I  ~~)~~ E  ~~)~~ D  ~~)~~ R  ~~)~~ F  ~~)~~ N | 24 | I  ~~)~~ E  ~~)~~ R  ~~)~~ F  ~~)~~ D  ~~)~~ N |

Table 7: Exhaustive set of pipeline prototypes generated using the compatible precedence graph of Table [2a](#bookmark40). E - Encoding; N

- Normalization; D - Discretization; I - Imputation; R - Rebalancing; F - Feature Engineering.

20% 18% 16% 14% 12% 10% 8% 6% 4% 2% 0%

Percentage of cases for wh ich a prototype

achieved the best performance



|  |  |
| --- | --- |
|  | NB  KNN  RF |

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Prototype ID

Figure 12: Comparison of the goodness of the exhaustive set of prototypes.

597 axis and each stacked bar represents the percentage of cases for which that prototype achieved the best

598 performance across different ML algorithms (the contribution of each algorithm is represented with a

599 different color). In an ideal scenario, for a pipeline to be considered *universal*, it should perform best in

600 all or at least most of the cases, which is clearly not happening. Observe that, even the best performing

601 pipeline is only the best in 19% of the cases, which is obviously far from being *universal*. Hence all (or

602 at least several) pipelines need to be evaluated together, in order to obtain better solutions.

603 *4.2. Exhaustive versus effective prototypes*

604 Given that there is no single universal pipeline, one can opt for feeding all the possible prototypes 605 (see Table [7) to the optimization algorithm in order to get the best solutions out of them](#bookmark56). As before, we

606 assign a budget of 200 seconds for the optimization of each prototype, hence 80 minutes in total for all

607 the set of 24 *exhaustive prototypes* in order to ﬁnd the optimal pipeline for every dataset. On the other

608 hand, we take only the ﬁve *effective prototypes* resulting from the application of our method and assign

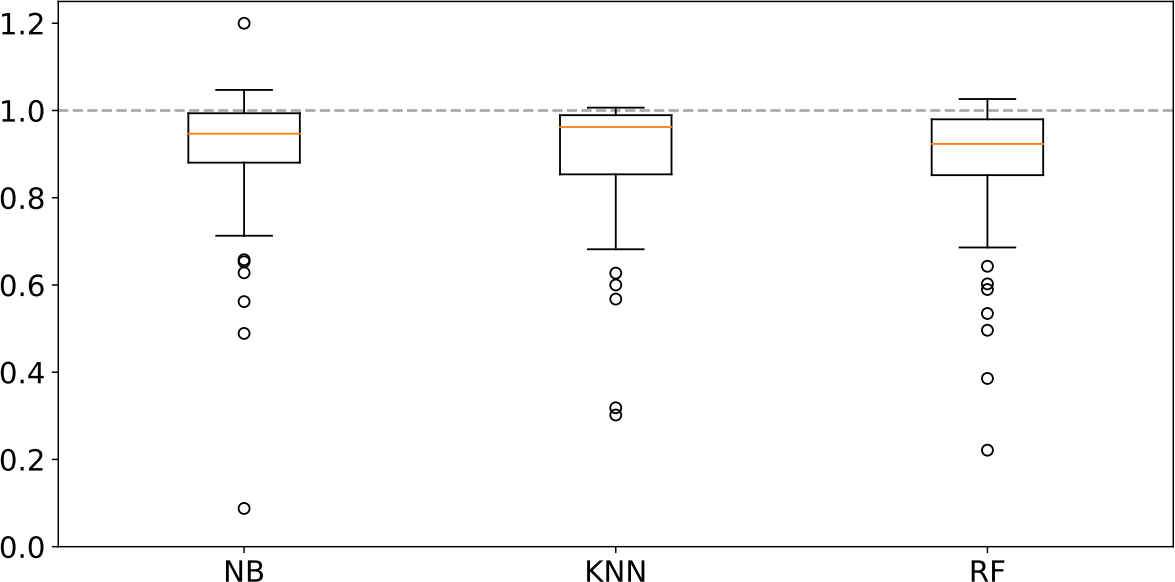
609 just 40 seconds time budget for the optimization of each one of them, hence 200 seconds in total. With

610 the aim of comparing the two, and thus roughly understanding how close we are to the optimal case,

611 in both cases, we dedicated the same time budget (i.e., 200 seconds) for the phase of optimizing the

612 hyper-parameters of the ML algorithm. In order to evaluate how close the *effective prototypes* are to the

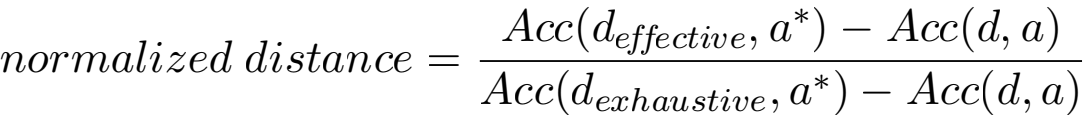
613 *exhaustive ones*, we calculate the *normalized distance* from the result to the optimum:



Norma l ized distance

Algorithms

Figure 13: Normalized distances between the scores obtained by optimizing our effective prototypes and the ones obtained optimizing the exhaustive set.



614 where, Acc(d, a) is the baseline performance (i.e., predictive accuracy of the algorithm a with de- 615 fault hyper-parameters over the original dataset d). Acc(d effective, a\* ) is the accuracy of the optimized 616 algorithm a\* over the dataset d effective transformed using the optimized instantiation of the effective 617 set of prototypes (i.e., our approach). Finally, Acc(d exhaustive, a\* ) is the accuracy of the optimized 618 algorithm a\* over the dataset d exhaustive transformed using the optimized pipeline instantiation of the 619 exhaustive set of prototypes. The subtraction by Acc(d, a) is done with the aim of weighting the dif- 620 ﬁculty of a dataset, hence allowing for comparisons in terms of the gain in accuracy. To this end, the

621 bigger the potential gain (denominator) is, the bigger the obtained gain (numerator) must be, for the

622 latter to be relevant.

623 The results obtained for every dataset and algorithm are shown as boxplots in Figure [13.](#bookmark72) Observe

624 that, most of the cases are very close to the results obtained using the exhaustive set, the median distances

625 being 91.51%, 93.13%, 88.97%, for NB, KNN, and RF, respectively. In general, in 75% of the cases

626 the chosen pipelines are above 80%, and only few outliers are below 60%. Curiously, in some cases,

627 we outperform the results over the exhaustive set of pipelines, but this is due to the randomness of the

628 optimization algorithm, which unless it is given an unrealistically high budget of time, is not capable of 629 ﬁnding the true optimal solution. We discarded the option of assigning a larger budget since this was not

630 practical considering the huge search space and the lack of any guarantee of improvement.

631 To summarize, the experiment shows that with roughly 24 times less time budget, we can obtain

632 results that are as good as 90% in the median compared to the exhaustive ones. The raw results (i.e.,

633 without the normalized distances) can be found on the aforementioned github page.

634 *4.3. Complementing hyper-parameter optimization with pre-processing*

635 We have just shown that our effective pipeline prototypes have similar impact as the exhaustive

636 prototypes. Now we want to compare the impact of effective prototypes against optimizing only the

637 hyper-parameters of the ML algorithm. That is, we want to examine whether dedicating a part of the

638 optimization budget to the pre-processing pipeline impacts more (positively) the results of the analysis,

639 [than using the whole budget for the hyper-parameter optimization10 .](#bookmark73)

640 To this end, for the latter we now dedicate the total optimization budget (i.e., 400 seconds), and for

641 [the former, inspired by [25], we split the budget 50-50 between the pre-processing pipeline optimization](#bookmark34)

10To enable the application of the ML algorithms on all the datasets, whenever required, we apply the necessary transforma- tion (e.g, imputation or encoding).

22

 Pre-processing + Hyper-parameter optimization  Hyper-parameter optimization

NB KNN

RF

1

1

Normal ized impact

0.5

0.5

0

0

Datasets Datasets Datasets

Figure 14: The impact of dedicating a portion of the optimization budget to pre-processing compared to using the whole optimization budget for the hyper-parameter optimization.

642 and the hyper-parameter optimization (i.e., 200 seconds for the pre-processing, and 200 seconds for the

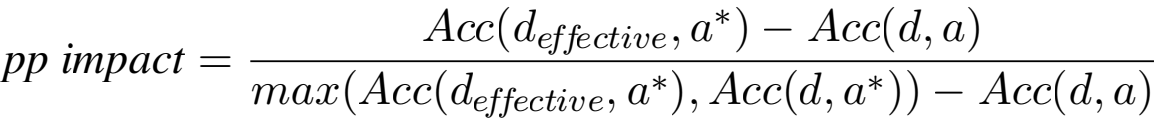
643 hyper-parameter optimization). The time for the pre-processing is further split among the ﬁve different

644 pipeline prototypes (i.e., 40 seconds each).

645 To compare the results, we calculate the impact using the formulas below, that correspond to the

646 normalized distance from either pre-processing or hyper-parameter optimization to the maximum im-

647 provement that can be achieved, regardless of whether pre-processing is applied or not.

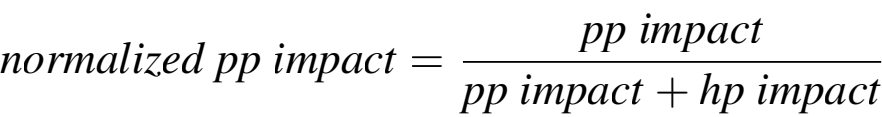


*hp impact* = Acc(d, a\* ) — Acc(d, a)

max(Acc(d effective, a\* ), Acc(d, a\* )) — Acc(d, a)

648 where, Acc(d, a) is the baseline accuracy (i.e., predictive accuracy of the algorithm a with default 649 hyper-parameters over the original dataset d). Acc(d effective, a\* ) is the accuracy of the optimized algo- 650 rithm a\* over the dataset d effective transformed using the optimized instantiation of the effective set of 651 prototypes obtained using our method. Finally, Acc(d, a\* ) is the accuracy of the optimized algorithm a\* 652 (i.e, using the entire budget) over the original dataset d.

To obtain relative values that sum to 1, we normalize the impacts dividing them by their sum. For instance, for the pre-processing score we calculate the following:



653 We perform the same for the hyper-parameter impact and plot the results obtained for all the algo-

654 rithms and datasets in Figure [14, where each bar represents the results obtained for a single dataset](#bookmark74). The

655 different colors represent the impact values of pre-processing and hyper-parameter optimization.

656 Observing the bar-charts one can see that (i) dedicating a portion of the budget to pre-processing,

657 brings beneﬁt to the analysis in most of the cases (i.e., 73% of the cases), and (ii) the impact of hyper-

658 parameter optimization, increases with the increase of the number of hyper-parameters of the ML al-

659 gorithm (e.g., hyper-parameter optimization impacts more RF than NB). Overall, we can conclude that

660 pre-processing is a critical step that once effectively applied may have a high positive impact on the ﬁnal

661 result of the analysis.

662 **5. Related work**

663 A lot of ongoing research aims at addressing the problem of providing user assistance for the data

664 analytics process. [Speciﬁcally, they can be classiﬁed into three main categories [33]: distributed, cloud](#bookmark75)-

665 based, and centralized. The ﬁrst two try to address the problem of Big Data. Thus, clusters of several

666 machines are employed to distribute the workload. On the contrary, this is not a fundamental requirement

667 for centralized solutions. Indeed, the overhead of using a cluster is not worth for relatively small datasets.

668 Since our work belongs to the category of centralized solutions, in the following, we provide examples

669 of them.

670 As already mentioned before, the data analytics process consists of different steps. In general, there

671 is a trend to develop (semi) automatic systems that assist the user in one or many steps altogether. At the

672 beginning, the focus was to provide support exclusively for the learning step (i.e., the CASH problem).

673 Recently however, the direction has shifted towards designing systems that additionally or speciﬁcally

674 provide user assistance in the data pre-processing step (i.e., the DPSO problem).

675 When it comes to data pre-processing, different works have tackled this problem from different per-

676 spectives. For instance, there are works that aim to apply pre-processing for the sake of guaranteeing data

677 quality, or enabling data exchange, or even data integration. That is, they consider data pre-processing in

678 [isolation or apart from data analysis [34,](#bookmark76)[35,](#bookmark77)[36,](#bookmark78)[37]](#bookmark79). In this, and our related work however, we consider

679 only the works that see pre-processing as an integral part of data analytics and hence apply it for the

680 sake of improving the ﬁnal result of the analysis.

681 Finally, there are works that aim at fully automating the data analytics process (i.e., automatically

682 generate data analytics ﬂows), which roughly translates to combining DPSO with CASH, where the

683 borderline between the latter two becomes blurry. Nevertheless, we tentatively group the works based

684 on the type of the problem they aim to solve.

685 *5.1. DPSO*

686 [In DPD [25], the DPSO problem, as we use it in this work, is formally deﬁned](#bookmark34). Authors demonstrate

687 the impact of optimizing the pre-processing pipeline, but considering only a single ﬁxed pipeline proto-

688 type. However, as we have already seen (Section [4.1), a single ﬁxed prototype cannot perform best for](#bookmark57)

689 every dataset. [Therefore, we build on top of [25], and instead of relying on a ﬁxed prototype, we deﬁne](#bookmark34)

690 a method to generate the right pipeline prototypes to be optimized.

691 [In PRESISTANT [30,](#bookmark62) [38,](#bookmark80) [31], we tackled the problem of recommending pre-processing operators](#bookmark63)

692 to the non-expert data analyst. The goal, and at the same time the challenge was to identify the pre-

693 processing operators, and rank them in advance, based on their potential impact to the ﬁnal analysis.

694 However, we did not consider pre-processing pipelines, but only single transformations, expecting that

695 the analyst applies the process iteratively. In this work, we consider sets of transformations and thus

696 study the impact of combining transformations into a pipeline.

697 [In ActiveClean [39], authors deﬁne an algorithm that aims at prioritizing the cleaning of records that](#bookmark81)

698 are more likely to affect the results of the statistical modeling problems, assuming that the latter belong

699 to the class of convex loss models (i.e., linear regression and SVMs). Hence, instead of recommending

700 the transformations to be applied, the system recommends the subset of data which needs to be cleaned

701 at a given point. The type of pre-processing to be applied is left to the user, assuming that the user is an

702 expert.

703 [In Learn2Clean [40], based on a reinforcement learning technique, for a given dataset, and an ML](#bookmark82)

704 model, an optimal sequence of operators for pre-processing the data is generated, such that the quality

705 of the ML model is maximized. [Here, similarly to [25], the pipeline prototype is ﬁxed in advance](#bookmark34).

706 Our work is a step further in that we help to choose the right pipeline prototype, instead of ﬁxing it in

707 advance.

708 [In Alpine Meadow [41], authors follow a similar approach to ours in that they deﬁne two steps for](#bookmark83)

709 the pre-processing phase. One, the so called *logical pipeline plan*, which is roughly equivalent to the

710 *pipeline prototypes* deﬁned in this work, and the second the *physical pipeline plan* which translates to

711 *pipelines* used in this work. The physical plan is generated through a combination of Bayesian optimiza-

712 tion, meta-learning, and multi-armed bandits. For the logical plans, they rely on rules but without clear

713 evidence on how they are generated. [Moreover, it is not clear whether the logical plan is ﬁxed as in [25]](#bookmark34)

714 and if some further adjustment from the user is required.

715 *5.2. CASH*

716 The task in solving the CASH problem is to automatically ﬁnd an optimized instantiation for the

717 hyper-parameters of the ML algorithm. Most of the works use Bayesian optimization methods to tune

718 [and optimize them [42,](#bookmark84)[43,](#bookmark85)[44]](#bookmark86). Since Bayesian optimization is randomized, meta-learning has been used

719 [to ﬁnd a good seed for the search [45]](#bookmark87). Most of these works however, only minimally consider the data

720 pre-processing step. [Auto-WEKA [43], based on the Java machine learning library Weka, is the pioneer](#bookmark85)

721 of the ﬁeld. The authors formalized the problem of algorithm selection and their associated hyper-

722 parameter optimization, and solved it in a combined search space. Sequential Model-based Algorithm

723 Conﬁguration (SMAC) is used to explore the large search space.

724 [Autostacker [46] combines a hierarchical stacking architecture and an evolutionary algorithm (EA)](#bookmark88).

725 Stacking is an ensemble method that involves the concatenation of several classiﬁers, so that the later

726 layers can learn the mistakes that classiﬁers in the previous layers make. Even if it brings some beneﬁts,

727 this approach affects the search space: way larger than that of a single classiﬁer. In a nutshell, such

728 concatenations are randomly generated and then optimized. The one that achieves the higher predictive

729 accuracy is chosen. Rather than Bayesian Optimization, to ﬁnd suitable hyper-parameters, the authors

730 utilize a basic Evolutionary Algorithm.

731 [OBoe [47] exploits collaborative ﬁltering for AutoML, choosing models that have performed well on](#bookmark89)

732 similar datasets. It collects a large number of datasets and applies different ML algorithms (with different

733 hyper-parameters conﬁgurations). In this way, a matrix of cross-validated errors is built. Common

734 approaches typically compute dataset meta-features and use them to predict the error of a particular

735 machine learning model, but OBoe works exactly the other way around. PCA is applied on such a

736 matrix in order to ﬁnd latent meta-features. Given a new dataset, some basic algorithms are applied to

737 infer a feature vector (i.e., the value of the latent meta-features). Finally, the feature vector is leveraged

738 to estimate the cross-validated error of more complex algorithms.

739 *5.3. DPSO + CASH*

740 [Auto-sklearn [42] is based on the popular Python library scikit-learn](#bookmark84). The authors, inspired by

741 Auto-Weka, address the problem with the Sequential Model-based Algorithm Conﬁguration (SMAC).

742 Furthermore, they improve the approach by adding a meta-learning phase at the beginning (to warm-start

743 the Bayesian Optimization) and an ensemble technique at the end (to suggests multi-classiﬁers). Such

744 a system considers pre-processing transformations to generate end-to-end analytic pipelines. Yet, they

745 consider a small set of transformations and also consider a single ﬁxed pipeline prototype. Our work in a

746 way is complementary to this, since instead of a priori ﬁxing the prototype, we can construct a potentially

747 optimal one (or a set), and then provide it to the tool for it to be instantiated and further optimized.

748 [TPOT [44] is a tree-based pipeline optimization tool using genetic programming while requiring little](#bookmark86)

749 to no expertise from the user. In TPOT however, they only consider one transformation inside the

750 optimization process (i.e., Feature Engineering).

751 [ML-Plan [3] uses hierarchical planning, a particular form of AI planning, to propose a solution to](#bookmark4)

752 both the pre-processing and the modeling phases. As in context-free grammars, there are complex tasks 753 (non-terminal symbols) that are derived as long as primitive tasks (terminal symbols) are not obtained.

754 Typically, standard graph search algorithms (e.g., depth-ﬁrst search, best-ﬁrst search, etc.) are employed

755 to solve such problems. ML-Plan successively creates solutions in a global search instead of changing

756 given solutions in a local search. However, due to the problem constraints, they adopt a randomized

757 best-ﬁrst search, randomly choosing the solution path.

758 [AutoBazaar [48] is a Python open-source tool](#bookmark90). [Like in ML-Plan [3], both pre-processing and mod](#bookmark4)-

759 eling phases are covered. Here the last step of a prototype is the machine learning algorithm. The

760 approach involves two different steps. Firstly, a *catalog* proposes a collection of prototypes (with an

761 ML algorithm as last step) based on the task and the dataset itself. Secondly, the optimization process

762 starts tuning the prototypes until either the time budget is expired or the prototypes are all optimized.

763 In particular, a *selector* and a *tuner* work in synergy. The former decides which prototype should be

764 optimized next. Such a task is treated as a multi-armed bandit problem. As to the tuner, Bayesian Op-

765 timizationis chosen. At the end, the prototype that achieved the higher predictive accuracy is elected.

766 However, AutoBazaar strictly depends on the catalog. Such a component memorizes all the possible

767 primitives and supported tasks. The prototypes are hard-coded for each task. Thus, it is neither ﬂexible

768 nor maintainable. If a task is not implemented, the approach cannot suggest a solution.

769 To summarize, full automation of data analytics has been the ultimate goal of many research works.

770 Yet, such an automation has shown to be computationally expensive, mainly due to the search space

771 involved (i.e., pre-processing and mining operators). Therefore, the usability of these approaches in

772 realistic scenarios is sometimes limited. Our approach of ﬁnding a set of effective pipeline prototypes

773 can be seen as complementary to these solutions, since it helps in pruning the large space and guiding

774 the search, hence reducing their cost.

775 **6. Conclusions and future work**

776 In this work, we ﬁrst studied the overall impact of transformations when chained together inside

777 pre-processing prototypes and then delved into examining the impact of instantiating transformations

778 via various operators. As a result, we deﬁned a method that allows to generate effective pre-processing

779 pipelines. That is, pipelines that consist of, (i) compatible pairs of transformations with respect to the

780 framework used, (ii) meaningful pairs of transformations in terms of general knowledge (best practices),

781 and (iii) promising pairs of transformations that once applied are expected to provide higher overall

782 impact (domain knowledge). In addition, via the meta-learning step proposed, we aim to guide the

783 instantiation of transformations in order to facilitate ﬁnding better instantiations.

784 An extensive evaluation on 80 datasets with heterogeneous characteristics, from sample size to fea-

785 ture types, and a set of classiﬁcation algorithms (i.e., Naive Bayes, Random Forest, K-Nearest Neigh-

786 bours), showed that our devised pipeline prototypes give promising results. More speciﬁcally, we were

787 able to observe that:

788 – The overall impact of optimizing pre-processing is not negligible and it may boost the performance

789 of the overall analytics (e.g., predictive accuracy).

790 – There is no universal pre-processing pipeline prototype that works best for every dataset and

791 algorithm.

792 – With 24 times less time budget, our proposed pipeline prototypes were able to obtain results that

793 were as good as 90% in the median of the optimal ones found through an exhaustive search.

794 – Dedicating a portion of the time to the pre-processing optimization, instead of dedicating it entirely

795 to hyper-parameter optimization may boost the ﬁnal result of the analysis. On average, in 73% of

796 the cases including pre-processing in the optimization, outperformed the results of only optimizing

797 hyper-parameters.

798 The results indicate that pre-processing can boost the performance of the ML algorithm. Hence, it

799 must be considered as an integral part of the data analytics optimization process.

800 Finally, previous works have shown the effectiveness of meta-learning for solving the cold start

801 [problem [45], hence as immediate future work, we intend to extend an optimization framework (i.e](#bookmark87).,

802 HyperOpt) with a complementary meta-learning module that can ease the cold-start problem, facilitating

803 the search for optimal instantiations.

804 **Acknowledgments.** This work was supported by the DOGO4ML project, funded by the Spanish

805 Ministerio de Ciencia e Innovacin under project/funding scheme PID2020-117191RB-I00 / AEI / 806 10.13039/501100011033. We thank University of Bologna for issuing a grant for author’s research

807 stay at Universitat Politcnica de Catalunya. Finally, we thank Matteo Golfarelli for his comments and

808 feedback on this work.

809 **References**

810 [1] M. A. Munson, A study on the importance of and time spent on different modeling steps, SIGKDDExplor. Newsl. 13 (2) 811 (2012) 65–71.

812 [2] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efﬁcient and robust automated machine

813 learning (2015) 2962–2970.

814 [3] F. Mohr, M. Wever, E. Hllermeier, Ml-plan: Automated machine learning via hierarchical planning, Machine Learning

815 107 (8) (2018) 1495–1515.

816 [4] L. Muoz, J.-N. Mazn, J. Trujillo, Automatic generation of etl processes from conceptual models, in: Proceedings of

817 the ACM Twelfth International Workshop on Data Warehousing and OLAP, DOLAP ’09, 2009, p. 33–40.

818 [5] A. A. Vaisman, E. Zimnyi, Data Warehouse Systems - Design and Implementation, Data-Centric Systems and Applica-

819 tions, Springer, 2014.

820 [6] B. Bilalli, A. Abell, T. Aluja-Banet, R. Wrembel, Towards intelligent data analysis: The metadata challenge, in: M. Ra- 821 machandran, G. B. Wills, R. J. Walters, V. M. Muoz, V. Chang (Eds.), Proceedings of the International Conference on

822 Internet of Things and Big Data, IoTBD 2016, Rome, Italy, April 23-25, 2016, pp. 331–338.

823 [7] A. Quemy, Data pipeline selection and optimization, in: Proceedings of the 21st International Workshop on De-

824 sign, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT Joint Conference,

825 DOLAP@EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019, 2019.

826 [8] J. Bergstra, D. Yamins, D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of

827 dimensions for vision architectures, ICML’13, 2013, pp. 115–23.

828 [9] J. Giovanelli, B. Bilalli, A. Abell, Effective data pre-processing for automl, in: K. Stefanidis, P. Marcel (Eds.), Pro- 829 ceedings of the 23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data 830 (DOLAP), Vol. 2840 of CEUR Workshop Proceedings, CEUR-WS.org, 2021, pp. 1–10.

831 [10] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, K. Leyton-Brown, Auto-weka 2.0: Automatic model selection and hyper-

832 parameter optimization in weka, Journal of Machine Learning Research 18 (2017) 1–5.

833 [11] F. Serban, J. Vanschoren, J. Kietz, A. Bernstein, A survey of intelligent assistants for data analysis, ACM Computing

834 Surveys 45 (3) (2013) 1–35.

835 [12] D. C. Montgomery, Design and analysis of experiments, John wiley & sons, 2017.

836 [13] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization., Journal of machine learning research 13 (2) 837 (2012).

838 [14] P. J. Van Laarhoven, E. H. Aarts, Simulated annealing, in: Simulated annealing: Theory and applications, Springer, 1987,

839 pp. 7–15.

840 [15] O. Kramer, Genetic algorithms, in: Genetic algorithm essentials, Springer, 2017, pp. 11–19. 841 [16] P. I. Frazier, A tutorial on bayesian optimization (2018). [arXiv:1807.02811.](http://arxiv.org/abs/1807.02811)

842 [17] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyper-

843 parameter optimization, The Journal of Machine Learning Research 18 (1) (2017) 6765–6816.

844 [18] M.-A. Zller, M. F. Huber, Survey on automated machine learning, arXiv preprint arXiv:1904.12054 9 (2019).

845 [19] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, Y. Yu, Taking human out of learning applications: A

846 survey on automated machine learning (2018). [arXiv:1810.13306.](http://arxiv.org/abs/1810.13306)

847 [20] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm conﬁguration, in:

848 International conference on learning and intelligent optimization, Springer, 2011, pp. 507–523.

849 [21] F. Archetti, A. Candelieri, Bayesian Optimization and Data Science, 1st Edition, Springer International Publishing, 2019. 850 [22] F. Hutter, H. H. Hoos, K. Leyton-Brown, K. P. Murphy, An experimental investigation of model-based parameter optimi- 851 sation: Spo and beyond, in: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, 2009,

852 pp. 271–278.

853 [23] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, NeurIPS ’12, 854 2012, pp. 2960–2968.

855 [24] M. Wistuba, N. Schilling, L. Schmidt-Thieme, Scalable gaussian process-based transfer surrogates for hyperparameter

856 optimization, Mach. Learn. 107 (1) (2018) 43–78.

857 [25] A. Quemy, Two-stage optimization for machine learning workﬂow, Information Systems 92 (2020) 101483.

858 [26] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, Openml: Networked science in machine learning, SIGKDD Explo-

859 rations 15 (2) (2013) 49–60.

860 [27] T. Dasu, T. Johnson, Exploratory Data Mining and Data Cleaning, 1st Edition, John Wiley & Sons, Inc., New York, NY,

861 USA, 2003.

862 [28] P. Brazdil, C. G. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning - Applications to Data Mining, Cognitive Technolo-

863 gies, Springer, 2009.

864 [29] B. Bilalli, A. Abell, T. Aluja-Banet, On the predictive power of meta-features in openml, Int. J. Appl. Math. Comput.

865 Sci. 27 (4) (2017) 697–712.

866 [30] B. Bilalli, A. Abell, T. Aluja-Banet, R. Wrembel, Intelligent assistance for data pre-processing, Comput. Stand. Inter-

867 faces 57 (2018) 101–109.

868 [31] B. Bilalli, A. Abell, T. Aluja-Banet, R. Wrembel, PRESISTANT: learning based assistant for data pre-processing, Data

869 Knowl. Eng. 123 (2019).

870 [32] T. Hothorn, K. Hornik, A. Zeileis, Unbiased recursive partitioning: A conditional inference framework, Journal of Com-

871 putational and Graphical Statistics 15 (3) (2006) 651–674.

872 [33] R. Elshawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges, arXiv preprint

873 arXiv:1906.02287 (2019).

874 [34] F. Geerts, G. Mecca, P. Papotti, D. Santoro, The llunatic data-cleaning framework, PVLDB End. 6 (9) (2013) 625–636. 875 [35] Z. Khayyat,I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quian-Ruiz, N. Tang, S. Yin, Bigdansing: A 876 system for big data cleansing, SIGMOD ’15, 2015, pp. 1215–1230.

877 [36] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, Y. Ye, Katara: A data cleaning system powered by

878 knowledge bases and crowdsourcing, SIGMOD ’15, 2015, pp. 1247–1261.

879 [37] Z. Jin, M. R. Anderson, M. Cafarella, H. V. Jagadish, Foofah: A programming-by-example system for synthesizing data

880 transformation programs, SIGMOD ’17, 2017, pp. 1607–1610.

881 [38] B. Bilalli, A. Abell, T. Aluja-Banet, R. F. Munir, R. Wrembel, PRESISTANT: data pre-processing assistant, CAiSE

882 Forum ’18, 2018, pp. 57–65.

883 [39] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, K. Goldberg, Activeclean: Interactive data cleaning for statistical modeling,

884 PVLDB 9 (12) (2016) 948–959.

885 [40] L. Berti-quille, Learn2clean: Optimizing the sequence of tasks for web data preparation, WWW ’19, 2019, pp. 2580–

886 2586.

887 [41] Z. Shang, E. Zgraggen, B. Buratti, F. Kossmann, P. Eichmann, Y. Chung, C. Binnig, E. Upfal, T. Kraska, Democratizing

888 data science through interactive curation of ML pipelines, SIGMOD ’19, 2019, pp. 1171–1188.

889 [42] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Efﬁcient and robust automated machine

890 learning, NeurIPS ’15, 2015, pp. 2962–2970.

891 [43] C. Thornton, F. Hutter, H. H. Hoos, et al., Auto-weka: Combined selection and hyperparameter optimization of classiﬁ-

892 cation algorithms, in: KDD, 2013, pp. 847–855.

893 [44] R. S. Olson, N. Bartley, R. J. Urbanowicz,J. H. Moore, Evaluation of a tree-based pipeline optimization tool for automat-

894 ing data science, GECCO ’16, 2016, pp. 485–492.

895 [45] M. Feurer, J. T. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, AAAI ’15, 896 2015, pp. 1128–1135.

897 [46] B. Chen, H. Wu, W. Mo, I. Chattopadhyay, H. Lipson, Autostacker: A compositional evolutionary learning system, in:

898 Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 402–409.

899 [47] C. Yang, Y. Akimoto, D. W. Kim, M. Udell, Oboe: Collaborative ﬁltering for automl model selection, in: Proceedings of 900 the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1173–1183.

901 [48] M. J. Smith, C. Sala, J. M. Kanter, K. Veeramachaneni, The machine learning bazaar: Harnessing the ML ecosystem for 902 effective system development, CoRR abs/1905.08942 (2019).