Data pre-processing pipeline generation for AutoETL

Joseph Giovanelli1,*∗*, Besim Bilallib, Alberto Abello´b

*aUniversity of Bologna, Bologna, Italy*

*bUniversitat Polite`cnica 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.](#_bookmark0) 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]),](#_bookmark44) where

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](mailto:j.giovanelli@unibo.it) (Joseph Giovanelli), [bbilalli@essi.upc.edu](mailto:bbilalli@essi.upc.edu) (Besim Bilalli),

[aabello@essi.upc.edu](mailto:aabello@essi.upc.edu) (Alberto Abello´)

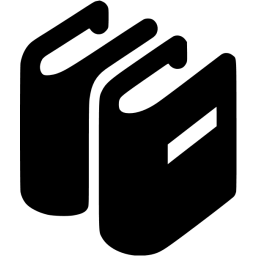
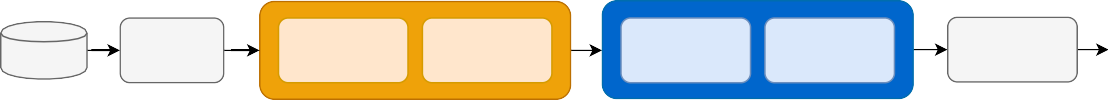
*Preprint submitted to Elsevier February 14, 2022*

© 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

Data source



Data selection

Data pipeline prototype selection

Data pipeline prototype optimization

Algorithm selection

Hyper- parameter optimization

Interpretation/ Evaluation

Knowledge

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,](#_bookmark45) [3],](#_bookmark46) mix-in the pre-processing during the optimization,

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],](#_bookmark47) we con-

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

24 process [[5];](#_bookmark48) from its inception via *data extraction*, to the intermediate phase of *data transformation*, up

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

26 phase [[6].](#_bookmark49) 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),](#_bookmark0) which is related to

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],](#_bookmark50) where a *pipeline prototype* (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-marketing[1](#_bookmark1) dataset,

36 using HyperOpt [[8]](#_bookmark51) as an AutoML approach to optimize the parameters of three different ML algorithms,

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 pipeline[2](#_bookmark2). In Figure [2,](#_bookmark3) the ratio of the change in terms of predictive accuracy (i.e., ratio of the

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

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

1..4

NB

KNN RF

Ratio of predictive accuracy change

1.3

1.2

1.1

1.0

0.9

0.8

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.

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],](#_bookmark52) among others with, (i) a new meta-learning step to guide the

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

61 tion (see Section [3.6),](#_bookmark24) (ii) a set of rules extracted through the meta learning process that mitigate the cold

62 start problem (see Example 8), (iii) a new cross validation to confirm the initial results (see Section [3.4),](#_bookmark13)

63 (iv) a new background section for AutoML and AutoETL (see Section [2),](#_bookmark4) and (v) experiments over new

64 datasets (see Section [4).](#_bookmark29)

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](#_bookmark4) provides a brief background on

86 AutoETL and AutoML. Section [3](#_bookmark6) presents our method of generating effective pipelines. Section [4](#_bookmark29)

87 provides an extensive evaluation of the pipelines created using our proposed method. Section [5](#_bookmark42) discusses

88 the related work. Finally, Section [6](#_bookmark43) 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],](#_bookmark47) which we coin as AutoETL. 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,](#_bookmark53) [2].](#_bookmark45) 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 their respective 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].](#_bookmark54) 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].](#_bookmark53)

110 Given:

111 • A data-set *D* divided into *Dtrain, Dtest*;

112 • A set of algorithms *A* = *{A*1*, . . . , Ak}* with associated hyperparameter spaces Λ1*, . . . ,* Λ*k*;

113 • And a loss function *L*(*Ai , Dtrain, Dtest*);

*λ*

114

we are searching for:

*A∗∗ ∈* argmin*Ai∈A,λ∈*Λ*i L*(*Ai , Dtrain, Dtest*) (CASH)

*λ λ*

115 The dataset *D* is divided into *Dtrain* and *Dtest*, 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 *{A*1*, . . . , Ak}* and the

118 related hyper-parameter spaces *{*Λ1*, . . . ,* Λ*k}*). 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]](#_bookmark55), Random search [[13]](#_bookmark56), Simulating annealing [[14]](#_bookmark57), Genetic algorithms [[15]](#_bookmark58), Bayesian

122 techniques [[16]](#_bookmark59), Bandit-based algorithms [[17]](#_bookmark60). However, due to their promising results, Bayesian tech-

123 niques are perhaps the most popular ones [[18,](#_bookmark61) [19].](#_bookmark62) We explore their details and specifically focus on one

124 of their incarnation, the Sequential Model-Based Optimization (SMBO) algorithm [[20].](#_bookmark63)

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:

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

max *f* (*x*)

*x∈B*

, where *B* contains all the feasible solutions, or candidates, and it is typically d-dimensional (*B ⊆* R*d*), where *d ∈* Z and *d >* 1. A specific solution *x ∈ B* is evaluated through the function *f* : R*d −→* R, also called the objective function. In general, in these kinds of problems, *f* has no special structure like

concavity or linearity that would make the optimization easier. In fact, it is considered as a “black-box” function, without any knowledge about its behaviour; being that it maps certain inputs, *x ∈ B*, to certain outputs, *f* (*x*) *∈* R. The goal is then finding an *x* that maximizes *f* (*x*). The naive solution to the problem would be to systematically evaluate all possible candidates *x* and choosing the one leading to the highest

value of *f* (*x*), aka *exhaustive search*. Since this evaluates all the potential solutions, it guarantees that it always finds the best one. Nevertheless, generally, it cannot be applied to real problems due to the large number of candidate solutions to be explored, which dwell in a high-dimensional space, and too expensive objective functions. The result is that not all candidates can be evaluated and we have to find a way to wisely choose the most promising ones. Bayesian techniques are part of the family of “surrogate methods”, which create a surrogate model to approximate the objective function and thus, choose a point in the search space where to evaluate the objective function [[21].](#_bookmark64) In contrast to the other methods, they build such surrogate models through Bayesian statistics.

In short, Bayesian techniques start by evaluating the objective function on an initial observation point of the search space, then the process becomes iterative: the surrogate model is constructed on the basis of the visited points and through an acquisition function — the Bayesian interpretation of the surrogate, the candidate for the next observation is decided. The process ends when a termination condition is reached, generally expressed through a *budget* represented in terms of the *number of iterations* or *execution time*. Given its iterative nature and the fundamental role of the model, this algorithm is called Sequential Model-Based Optimization [[20,](#_bookmark63) [22].](#_bookmark65) Variations of SMBO exist, depending on the method used to build the surrogate model (e.g., Gaussian Processes, Random Forest Regressions) [[23,](#_bookmark66) [24].](#_bookmark67)

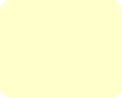
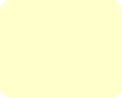
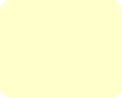
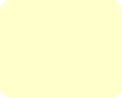
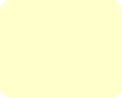
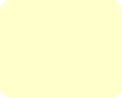
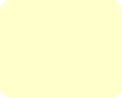
*2.2. Data Pipeline Selection and Optimization (DPSO)*

DPSO was formalized for the first time in [[25],](#_bookmark68) where some new concepts were introduced. For instance, a pre-processing *pipeline prototype* or *logical pipeline* is defined as a sequence of kinds of transformations, where each represents a logical concept that can be implemented/instantiated by one or more operators. The prototype thus, defines only the order between kinds of transformations, without specifying the concrete operators nor their parameters. Yet, the potential operators of each kind and their corresponding parameter search spaces need to be known in advance. Solving the DPSO problem trans- lates to finding the right instantiation and configuration for each kind of transformation in the pipeline prototype (i.e., optimal operator and optimal parameter values), which is called pre-processing *pipeline* or *physical pipeline*.

Formally, given:

* A data-set *D* divided into *Dtrain, Dtest*;
* A data pipeline prototype *P* with a configuration space *P*;
* The algorithm *A*, for which the given pipeline *P* transforms the data;
* And a loss function *L*(*P, A, Dtrain, Dtest*); we are searching for:

*P∗ ∈* argmin*P∈P L*(*P, A, Dtrain, Dtest*) (DPSO)



Prototype construction

Effective pipeline prototypes

Machine learning framework

framework

i r

3.3.Discover meaningful order

3.4.Assess empiric. the order of remain. pairs

3.5.Combine ordered pairs of transformati ons

Pre- processing pipeline

3.1.Select the transformati ons and operators

Machine learning experiments

3.7.Prototype instantiation

3.6.Discover rules via meta-learning

Meta-learning rules

Empirical rules

Heuristic rules

Framework - related rules

3.2.Discover

mposed orde

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 for DPSO*

170 Since DPSO is formalized as an optimization problem, SMBO has been proposed as a valid solver [[20]](#_bookmark63).

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],](#_bookmark68) we also distinguish between a fixed, ordered sequence of kinds of

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,](#_bookmark5) that in short breaks

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]](#_bookmark51) variant), to be instantiated and further optimized. 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],](#_bookmark69)

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).](#_bookmark9)

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).](#_bookmark12)

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

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

1. Promising/Unpromising pairs. Depending on whether the precedence between them is expected to provide positive impact on the final result of the analysis (promising), or not (unpromising) (see Section [3.4).](#_bookmark13)

Attending to the relationships between its transformations, a prototype can be described as either *compatible*, *well-formed*, or *effective*. A prototype is defined to be *compatible*, if all its precedence relationships are compatible. It is defined as *well-formed*, if all its precedence relationships are both compatible and meaningful. Finally, it is defined as *effective*, if all its precedence relationships are compatible, meaningful, and promising at the same time. In fact, the ultimate goal of our method is to find *effective pipelines*.

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

* + Encoding (*E*). The process of transforming Categorical attributes into Continuous ones.
  + Normalization (*N* ). The process of normalizing Continuous attributes such that their values fall in the same range.
  + Discretization (*D*). The process of transforming Continuous attributes into Categorical ones.
  + Imputation (*I*). The process of imputing missing values.
  + Rebalancing (*R*). The process of adjusting the class distribution of a dataset (i.e. the ratio between the different classes/categories represented).
  + Feature Engineering (*F* ). The process of defining the set of relevant attributes (variables, predictors) to be used in model construction.

An operator is an actual instantiation/implementation of a kind of transformation. Thus, several operators may implement the same kind of transformation, each having its own set of parameters. For

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

* + 1. Compatible precedence.
    2. Meaningful precedence.
    3. 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

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

our experiments, we selected the operators and parameters from those available in the Scikit-learn[3](#_bookmark10) library, and they are listed in Table [1.](#_bookmark7) *Input* denotes the compatible feature type for a given kind of transformation and can be Continuous (CO) — when it represents measurements on some continuous scale, or Categorical (CA) — when it represents information about some categorical or discrete characteristics. Similarly, *Output* denotes the type of the features after a kind of transformation is applied. Finally, *Operator* denotes the physical instantiation for the kind of transformation, and it can be parametrised using its *Parameters*.

* 1. *Framework-related rules*

Once the implementation framework is selected, one needs to study it and see if there exist con- straints that limit the interaction between transformations. For instance, applying a transformation may actually invalidate the application of another transformation, because the compatibility of transforma- tions is dependent on the selected ML framework.

EXAMPLE 2. We studied the transformations implemented in Scikit-learn and detected a set of implicit rules that are shown through an adjacency matrix, corresponding to a precedence graph, in Table [2a.](#_bookmark8) Each cell *aij* denotes a precedence relationship between the row *i* and column *j*. Hence, 1 means that an edge exists between the transformation in the row and the transformation in the column, whereas 0 means that such an edge does not exist, hence a precedence order is not established for that pair. For example, most Scikit-learn transformations cannot be applied in the presence of missing values. This is why in every pair of transformations where Imputation is involved, except the one with Normalization[4](#_bookmark11), Imputation goes first. Furthermore, Scikit-learn transformations are applied only to all compatible attributes of a given dataset. Generally, Categorical attributes are physically represented as strings and Continuous attributes as numbers. However, a transformation that is meant to be applied, say to Continuous attributes, cannot be applied over a dataset that contains both Continuous and Categorical attributes (i.e., a dataset containing both numbers and strings); Scikit-learn cannot deal with arrays of mixed types. In that case, all the Categorical attributes need to be encoded into numeric representations, even if they represent a categorical value. That is, a value can be a number but represent a category. This is what happens when Normalization and Discretization are meant to be applied to a dataset containing mixed types of attributes. In order for them to be applied to datasets of mixed types, an Encoding transformation needs to be applied first. A similar constraint is imposed when considering Rebalancing and Feature Engineering, since these transformations do not accept inputs containing strings (i.e., representing

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

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

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

a Categorical type). For the rest of the pairs of transformations there are no constraints imposed by the framework, thus any order of such transformations is permitted, reflected by a 0 in Table [2a.](#_bookmark8) The graph obtained in this case exclusively corresponds to the limitations of Scikit-learn (as a matter of fact, if another framework were to be chosen, it may have looked differently).

* 1. *Heuristic rules*

In the previous section, we proposed to derive a precedence based on the constraints of the frame- work. Now, we want to study the precedence independently of the framework, and find *meaningful pairs*. That is, for every given pair, we want to find the relative order, based on generic, domain-independent knowledge (i.e., literature) about transformations and their applicability. To this end, some of the con- straints imposed by the framework may be contradicted here, but this is resolved in the last step of the proposed method, when we take the union of the rules and hence construct the final pipeline prototypes (see Section [3.5).](#_bookmark23) Briefly, in a combination where Imputation is involved, it is advised to apply Impu- tation first. Next, an Encoding transformation makes sense to be combined in any order with the rest of transformations, except Imputation. Combining Discretization with Normalization does not make sense, due to the fact that after the Discretization step, Continuous attributes are transformed into Categorical ones, and hence Normalization cannot be applied. Similarly, applying Normalization first, changes the scale of the values and hence impacts the Discretization step. Finally, a meaningful precedence can be derived when combining Normalization with Rebalancing. In this case, Normalization should be applied first, since otherwise Rebalancing would impact the scale of the values to be normalized.

EXAMPLE 3. For our use case, Table [2b](#_bookmark8) shows the heuristic rules obtained considering domain- independent knowledge about transformations [[27].](#_bookmark70) In comparison with the results from Table [2a,](#_bookmark8) observe that the constraints on the Imputation transformation still hold, that is, it is correct to apply Imputation first when combining it with another transformation. This time even when combining it with Normalization — note the difference with Table [2a.](#_bookmark8) The constraints of Encoding are however not present in Table [2a,](#_bookmark8) hence not considering the framework, Discretization combined with Encoding is a meaningful combination — when a mixed type dataset is considered, but incompatible from the point of view of Scikit-learn.

* 1. *Empirically learned rules*

The two previously proposed steps (i.e., *framework-related* and *heuristic rules*), do not guarantee that for each pair of transformations we will obtain a precedence order. Therefore, for the cases where they are not sufficient to determine the precedence, a third viewpoint can be considered. That of learn- 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 *T*1 and *T*2

**Require:** *d*, *a* # dataset, classification algorithm

**Require:** *T*1 *→ T*2, *T*2 *→ T*1 # precedence orders of a pair of transformations

1: *accbaseline* = *Acc*(*d, a*); # get baseline performance of algorithm on d

2: [*pipelineT*1*→T*2 *, accT*1*→T*2 ] = *SMBO*(*T*1 *→ T*2*, d, a*)

# get pipeline and accuracy for *T*1 *→ T*2

3: [*pipelineT*2*→T*1 *, accT*2*→T*1 ] = *SMBO*(*T*2 *→ T*1*, d, a*)

# get pipeline and accuracy for *T*2 *→ T*1

4: **if** *IsValid*(*accT*1*→T*2 *, accT*2*→T*1 *, accbaseline*) **then** # see Table [3](#_bookmark15) for the rules applied

5: **return** *Winner*([*pipelineT*1*→T*2 *, accT*1*→T*2 ]*,* [*pipelineT*2*→T*1 *, accT*2*→T*1 ])

# see column *Winner prototype* in Table [3](#_bookmark15)

6: **else**

7: **return** ∅

8: **end if**

2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Pipeline 1 | Pipeline 2 | Valid result | Valid score | Winner prototype |
| 1. | ∅ *→* ∅ | ∅ *→* ∅ | Draw | *accbaseline* | Baseline |
| 2. | ∅ *→* ∅ | *confT*2 *→* ∅ | Draw | *accbaseline* | Baseline |
| 3. | ∅ *→* ∅ | ∅ *→ confT*1 | Draw | *accbaseline* | Baseline |
| 4. | ∅ *→* ∅ | *confT → confT* | Draw  1 *confT → confT* | *accbaseline*  *accT →T* | Baseline  *T*2 *→ T*1 |
| 5. ∅ *→ confT*2 | | ∅ *→* ∅ | Draw | *accbaseline* | Baseline |
| 6. ∅ *→ confT*2 | | *confT*2 *→* ∅ | Draw  ∅ *→ confT*2 | *accT*2 | *T*2  *T*2 |
|  | |  | *confT*2 *→* ∅ |  | *T*2 |
| 7. ∅ *→ confT*2 | | ∅ *→ confT*1 | Draw | *accT*2 or *accT*1 | *T*1 *or T*2 |
| 8. | ∅ *→ confT* | *confT → confT* | Draw  1 *confT → confT* | *accT*2  *accT →T* | *T*2  *T*2 *→ T*1 |
| 9. *confT*1 *→* ∅ | | ∅ *→* ∅ | Draw | *accbaseline* | Baseline |
| 10. *confT*1 *→* ∅ | | *confT*2 *→* ∅ | Draw | *accT*1 or *accT*2 | *T*1 *or T*2 |
| 11. *confT*1 *→* ∅ | | ∅ *→ confT*1 | Draw  *confT*1 *→* ∅ | *accT*1 | *T*1  *T*1 |
|  | |  | ∅ *→ confT*1 |  | *T*1 |

2 1 2 1

2 2

2 1 2 1

1. *confT*

*→* ∅ *confT*

*→ confT*

Draw *accT*1 *T*1

1 2 1

*confT*2

*→ confT*1

*accT*2*→T*1

*T*2 *→ T*1

1. *confT*

*→ confT*

∅ *→* ∅ Draw *accbaseline* Baseline

1 2 *confT*

1

*→ confT*2

*accT*1*→T*2

*T*1 *→ T*2

1. *confT*

*→ confT*

*confT*

*→* ∅ Draw *accT*2 *T*2

1 2 2

*confT*1

*→ confT*2

*accT*1*→T*2

*T*1 *→ T*2

1. *confT*

*→ confT*

∅ *→ confT*

Draw *accT*1 *T*1

1 2 1

*confT*1

*→ confT*1

*accT*1*→T*2

*T*1 *→ T*2

1. *confT*1 *→ confT*2 *confT*2 *→ confT*1

Draw *accT*1 or *accT*2 *T*1 *or T*2 *confT*1 *→ confT*2 *accT*1*→T*2 *T*1 *→ T*2

*confT*2 *→ confT*1 *accT*2*→T*1 *T*2 *→ T*1

∅ - 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.](#_bookmark14) 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 in line 5. However, beforehand, in line 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,](#_bookmark15)

337 and make up the validation rules for Algorithm [1](#_bookmark14) (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

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

if they have equal scores (i.e., a draw). This is because, if one has a higher score, it means that during the optimization phase it was more advantageous than the other, since it could find a configuration that should have been found by both of the pairs, given enough budget. In our SMBO runs, such invalid results account for less than 10% and in those cases datasets are discarded from the study (see line 7).

In particular, in Table [3,](#_bookmark15) the first two columns denote the pipeline instantiations for the respective pairs of transformations (i.e., *T*1 *→ T*2 and *T*2 *→ T*1). Next, *Valid result* denotes the expected result when comparing the results of the pipelines in the same row. For instance, in the first row, if both

transformations in the pipelines are not instantiated during the optimization, a valid result is a draw, and a *Valid score* for the respective result is the baseline accuracy, and the *Winner prototype* is the prototype that is in accordance with the expected result, which in this case is the Baseline (i.e., prototype consisting of only the ML algorithm, where no transformations are applied).

For the sake of another example, let us check row 2 in Table [3.](#_bookmark15) Running SMBO, the best result for the first pair is the pipeline ∅ *→* ∅, and for the second pair, the pipeline *T*2 *→* ∅. In this case, the comparison between the results of these pipelines should be equal (i.e., draw), and the score should be

that of the baseline. Otherwise, if say, the score of the second pipeline was higher, it would mean that for the first pair, SMBO was not given enough time to find the pipeline with higher score (i.e., *T*2 *→* ∅). The same logic applies also for the other rows where a ∅ operator is involved.

EXAMPLE 4. For the sake of this work, we considered three classification algorithms (i.e., *NB*, *RF*, *KNN*) and 80 datasets from the OpenML repository. The datasets, were compiled from three OpenML benchmarks, namely, the OpenMLCC18 benchmark[5](#_bookmark17), the AutoML benchmark[6](#_bookmark18), and the Classification algorithms benchmark[7](#_bookmark19). For the final set, we filtered out datasets with more than 10% of missing values — not to include bias due to the heavy pre-processing we need to perform on top of them, and we filtered out the datasets with more than 5 million instances — because of the computation time required to process them. As a result we obtained 60 datasets from the first benchmark, 17 from the second, and 3 more from the third to reach a total of 80 datasets.

5<https://www.openml.org/s/99/data> 6<https://www.openml.org/s/271/data> 7<https://www.openml.org/s/1/data>



*T*1 *T*2 *T*1 *T*2 *T*2 *T*1 Baseline

40 *T*1 = Feat. Eng., *T*2 = Normalize 40 *T*1 = Discretize, *T*2 = Feat. Eng.

30 30

Number of wins

Number of wins

20 20

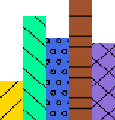
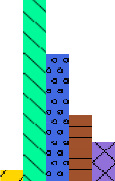
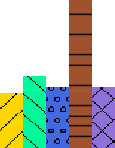
10 10

0

NB KNN RF

Algorithms

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

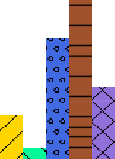
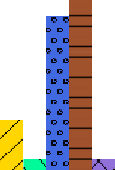
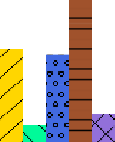


0

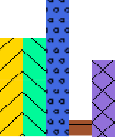
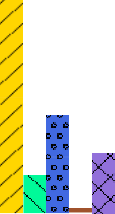
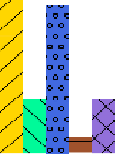
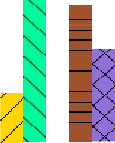
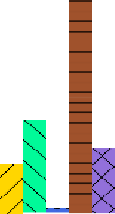
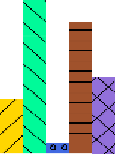
NB KNN RF

Algorithms

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



30 30



Number of wins

Number of wins

20 20

10 10

0

NB KNN RF

Algorithms

0

NB KNN RF

Algorithms

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *T*1 | *T*2 | *T*1 *→ T*2 | *T*2 *→ T*1 | alpha | p-value |
| *F* | *N* | 3 | 88 | 0.05 | **0** |
| *D* | *F* | 70 | 7 | 0.05 | **0** |
| *F* | *R* | 49 | 61 | 0.05 | 8.53e-01 |
| *D* | *R* | 66 | 86 | 0.05 | 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

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

Given the proposed algorithm (i.e. Algorithm [1),](#_bookmark14) we could try to learn the precedence of every pair of transformations, but would just be a waste of resources, because we can see in Table [2a](#_bookmark8) and [2b,](#_bookmark8) that some precedences are already decided for one reason or another. Hence, only pairs of transformations with a 0 for both directions (in both Table [2a](#_bookmark8) and [2b)](#_bookmark8) need to be studied further. That is, they make sense to be combined together, but a precedence order could not be determined through *framework-related* or *heuristic rules*. Thus, for instance, pairs involving Encoding are not considered in this phase, since for them an order is already imposed by the framework (see Table [2a).](#_bookmark8) To this end, the pairs of transformations we consider for the third precedence graph include only

*{F, N}*, *{F, D}*, *{F, R}*, and *{R, D}*.

Applying Algorithm [1,](#_bookmark14) we obtain a promising order for each pair of transformations considered.

Since SMBO is a randomized algorithm we experimented with (i) running it several times splitting the budget, and (ii) running it only once with the entire budget. For the experiments considered, no significant differences where observed, therefore we opted for running it once with the entire budget (i.e., 200 seconds per run), which allows for more configurations to be visited in a single run. Aggregating all the results, Figure [4](#_bookmark20) shows the number of datasets, for which a given prototype (see Table [3,](#_bookmark15) column *Winner prototype* for the list of labels) is selected as the winner. For instance, for

the pair *{F, N}* (i.e., Feature Engineering, Normalization), the prototype winning in more datasets for *KNN* and *NB* is *N → F* . This means that in general, better results are obtained if Normalization

is applied before Feature Engineering.

Next, only *N* appears as first for *RF* and second best for *KNN* and *NB*, which means that for many datasets, considering different algorithms, it results better to apply only Normalization

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

that for some datasets it is better not to apply any of the transformations (in any combination). The remaining prototypes winning in some datasets are *F* (only Feature Engineering), and *F → N*

(Feature Engineering preceding Normalization). Finally, for three datasets, that are omitted from the figure, there were no winning pipelines (i.e., pipelines resulted in a draw).

Since our goal is to find the best order for a pair of transformations, we focus on the performances of the pipelines where both of the transformations are instantiated (i.e., *T*1 *→ T*2 versus *T*2 *→ T*1).

To do this, we check whether the difference between the number of datasets where they each appear to win are statistically significant by running a binomial test assuming a theoretical probability of

0.5. The results are shown in Table [4.](#_bookmark21) In summary, the results from Table [4](#_bookmark21) indicate that, with 95% confidence we can assume that for the pair *{F, N}*, *N → F* performs better than *F → N* , hence Normalization should precede Feature Engineering. On the other hand, for *{D, F}*, *D → F* performs better than *F → D*, hence Discretization should precede Feature Engineering. Finally, for the remaining transformations, *{F, R}* and *{R, D}*, a precedence order can not be pre-assumed since

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

adjacency matrix shown in Table [2c,](#_bookmark8) where as one can observe, precedence edges are introduced for

*{N, F}* and *{D, F}*, but no edges exist neither for *{F, R}*, nor for *{R, D}*.

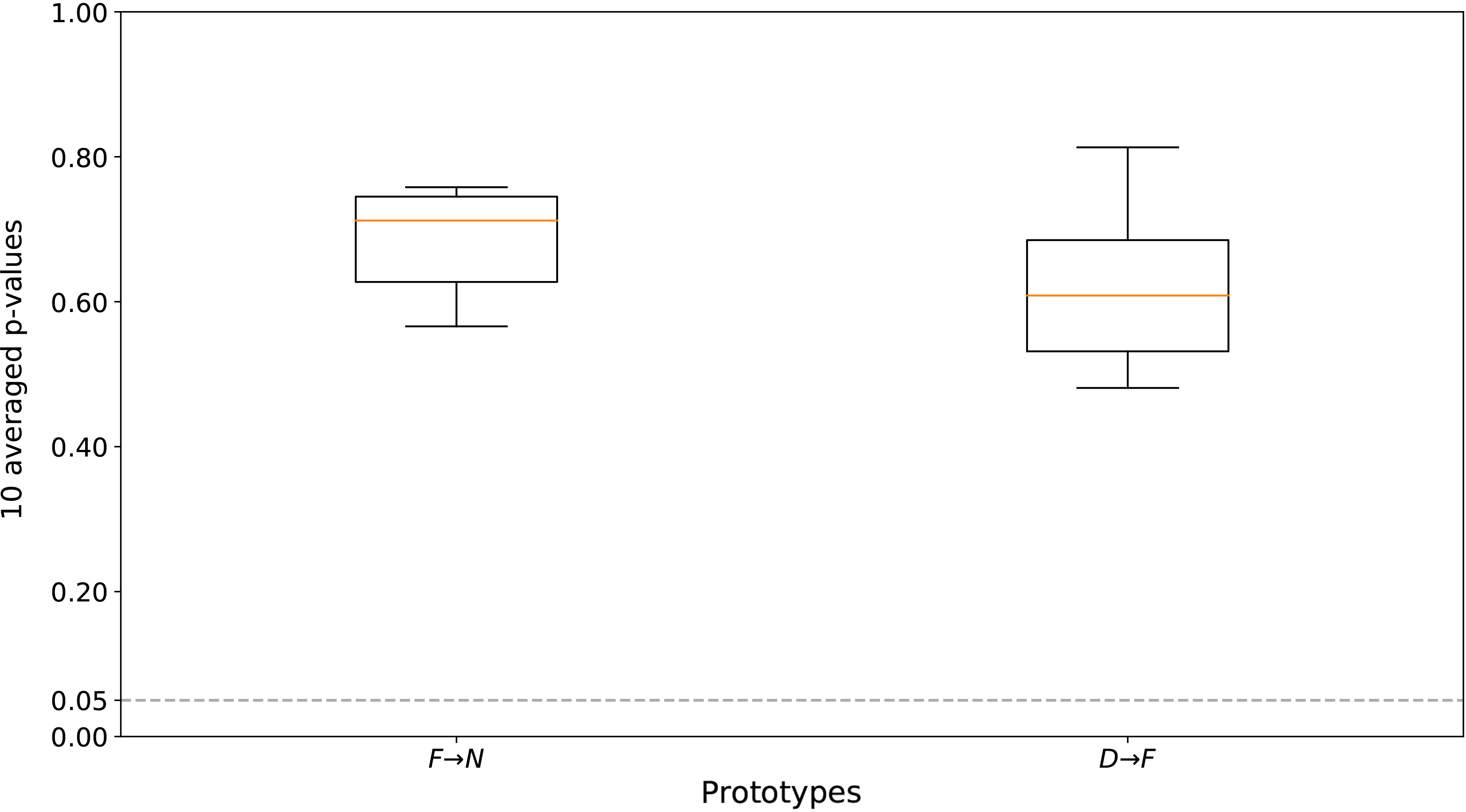


Figure 5: The distribution of the p-values obtained after repeating the chi-square test for 10 times, for the 10 times 4-fold cross-validation.

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

*3.4.2. Cross-validation*

After running Algorithm [1](#_bookmark14) to empirically find a winner between two pairs of transformations, we may obtain a different distribution of the number of wins for the pairs, depending on the datasets con- sidered. To show that the results obtained with the initial set of datasets are generalizable, we propose to perform an additional cross-validated experiment, where the set of datasets considered can be ran- domly split into many folds. Then, for each fold, the results can be compared to the rest, with the aim of checking whether the distributions are similar. This check can be done via a significance test (e.g., chi-square). To this end, if the distributions between the folds are similar, it means that the obtained results are independent of the datasets considered, since no matter the combination of the datasets, the results are the same and thus generalizable.

EXAMPLE 5. In our use case, to show that the results do not depend on the datasests selected, we re-run the experiments (i.e., 10-times each), but this time splitting the datasets into 4-folds. The goal was to check if the results of the precedence orders from the different folds (i.e., for each experiment considering a randomly different set of datasets) are similar between them (i.e., follow the same distributions). To confirm this hypothesis, we perform a chi-square test between the results (precedence orders) obtained in a single fold in comparison to the three remaining folds, hence comparing 25% of the datasets to the rest. In particular, to confirm the hypothesis, we need to find results that accept the null hypothesis of the chi-square test which states that ”there is no significant difference between the distributions”. To do that, sticking to the 95% confidence interval, we need to look for p-values greater than 0.05. That is, the higher the p-values, the more we accept the null hypothesis, the more similar the distributions. Looking at the p-values we found out that they were all much higher than 0.05. Specifically, the scores of the chi-square tests of the folds (one fold compared to the rest) are averaged and, after having repeated this procedure 10 times, instead of using a table we depict the 10 averaged p-values using box-plots in Figure [5.](#_bookmark22) We conclude that,

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

the new results (Figure [5)](#_bookmark22) and those illustrated above (Table [4).](#_bookmark21)

* 1. *Effective pipeline prototypes*

In this task we foresee the composition of the previously defined rules (i.e., for the pairs of transfor- mations), to generate the final set of rules that would allow to compose longer chains — consisting of



𝑁

𝑅

𝐸

𝐹

𝐼

𝐷

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*

Figure 6: Precedence graph generated from Table [5.](#_bookmark25) *E* - Encoding; *N* - Normalization; *D* - Discretization; *I* - Im- putation; *R* - Rebalancing; *F* - Feature Engineering.

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

- Feature Engineering.

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

more than two transformations. This is when we resolve the inconsistencies and also define precedences for the pairs of transformations that may not have any precedence defined already — in that case, we ba- sically take into account all the permutations. This step allows to finally generate the possible effective pipeline prototypes.

EXAMPLE 6. To generate the final pipeline prototypes, in this step we combine all the matrices generated by the previous steps. That is, we take the union of the edges (represented by 1’s) from the matrices in Table [2](#_bookmark8) (a,b,c), and create a new final adjacency matrix, shown in Table [5.](#_bookmark25) This is the matrix that will allow us to generate the final effective pipeline prototypes.

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

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

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

For a better reading, in Figure 6, we visualize Table [5](#_bookmark25) in form of a graph, where nodes represent the kinds of transformations and the directed edges represent a precedence order between them. Out of the graph, we generate the final pipeline prototypes by taking all the maximum length variations (ordered arrangements without repetition) of the nodes, respecting the precedence rules (i.e., not contradicting the direction of existing edges). The result is the set of five pipeline prototypes shown in Table 6. This set consisting of *compatible*, *meaningful* and *promising* pairs of transformations is the set of recommended *effective pipeline prototypes*.

* 1. *Meta-learning rules*

Once the pipeline prototype is constructed, that is, the order between the kinds of transformations is defined, what follows is the instantiation of transformations with the physical operators. For that, 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.](#_bookmark8) ***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.

NB KNN RF

30

Impact over the baseline

20

10

0

10

1 2 3 4 5 6 7 8 9 101112131415161718192021222324

Prototypes ID

1 2 3 4 5 6 7 8 9 101112131415161718192021222324

Prototypes ID

1 2 3 4 5 6 7 8 9 101112131415161718192021222324

Prototypes ID

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

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

However, given the way optimization algorithms work (e.g., SMBO) — successively finding better and better instantiations, there is a cold-start problem, where in the beginning, the algorithm does not have enough information in order to come up with the most promising initial instantiations, and a wrong choice may affect the optimization process.

* + 1. *Exploratory analysis*

Given the availability of the experimental SMBO executions (executed in an exhaustive manner, considering all the pipeline prototypes), one can perform an exploratory analysis with the aim of remov- ing useless prototypes, pipelines or operators. Hence, further tweaking the search space. In particular, starting from the highest level, that of prototypes, then going to the physical pipelines, and finally to the actual operators inside the pipeline, one can analyze if:

* + - * there exist some combination of transformations in the form of prototypes (see Table [7](#_bookmark35) for the exhaustive list of prototypes), that are generally useless (i.e., in terms of their impact to the final accuracy), and thus can be discarded a priori in order to reduce the search space,
      * there are some physical pipelines that are consistently chosen more often than others by the opti- mization algorithm, meaning that they are more useful than others,
      * within the physical pipelines, some transformations are chosen more often than others, meaning that they provide more positive impact.

EXAMPLE 7. We performed the above-mentioned analysis to our use case, but it did not lead to any conclusive or significant results. In particular, as shown in Figure [7,](#_bookmark26) we could not find any useless prototypes — not positively impacting the final accuracy, that could be discarded a priori from the potential list of prototypes. Actually, as we will show in Section [4.1,](#_bookmark34) all of them lead to the best in one case or another, which does not mean the epsilon improvement some provide is worth the search cost you incur in considering them (but this more in-depth analysis is done later).

Next, as shown in Figure [8,](#_bookmark27) there were no physical pipelines shown to be more useful — hence more often selected, than others. Even if *N → R* is clearly above, it barely reaches 30% in KNN.

Finally, observing Figure [9,](#_bookmark28) it is clear that some kinds of transformations are chosen more often, but looking closely (i.e., the shaded bars), it is not clear which operator brings more benefit. For instance, Normalization is present in 90% of the pipelines, but it is not easy to distinguish which kind of Normalization (i.e., actual operator) is more beneficial. For this, we need more complex rules or guidelines that may help in finding the right operator to use.

* + 1. *Meta-learning*

To mitigate the red cold-start problem, we propose to perform meta-learning (shown in Figure [3),](#_bookmark5) where we intend to use the knowledge extracted from historical data in order to devise rules that may help the optimization algorithm in its initial phase. Meta-learning is the process of ‘learning on top of



*D*

*D R*

*N*

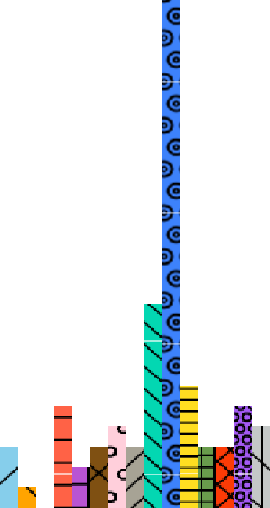
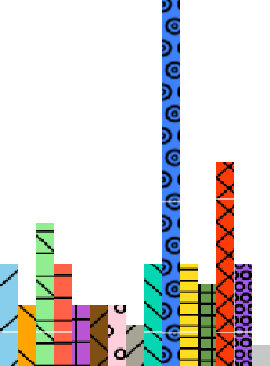
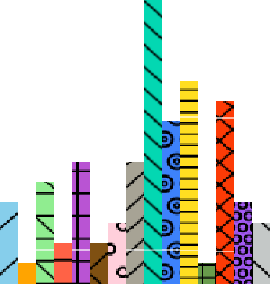
*N R*

*R D*

*D F D R F N F N R F R D F*

*D F R F R N F R R R F*

30%



25%

20%

15%

Usage

10%

5%

0%

NB KNN RF

Algorithms

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

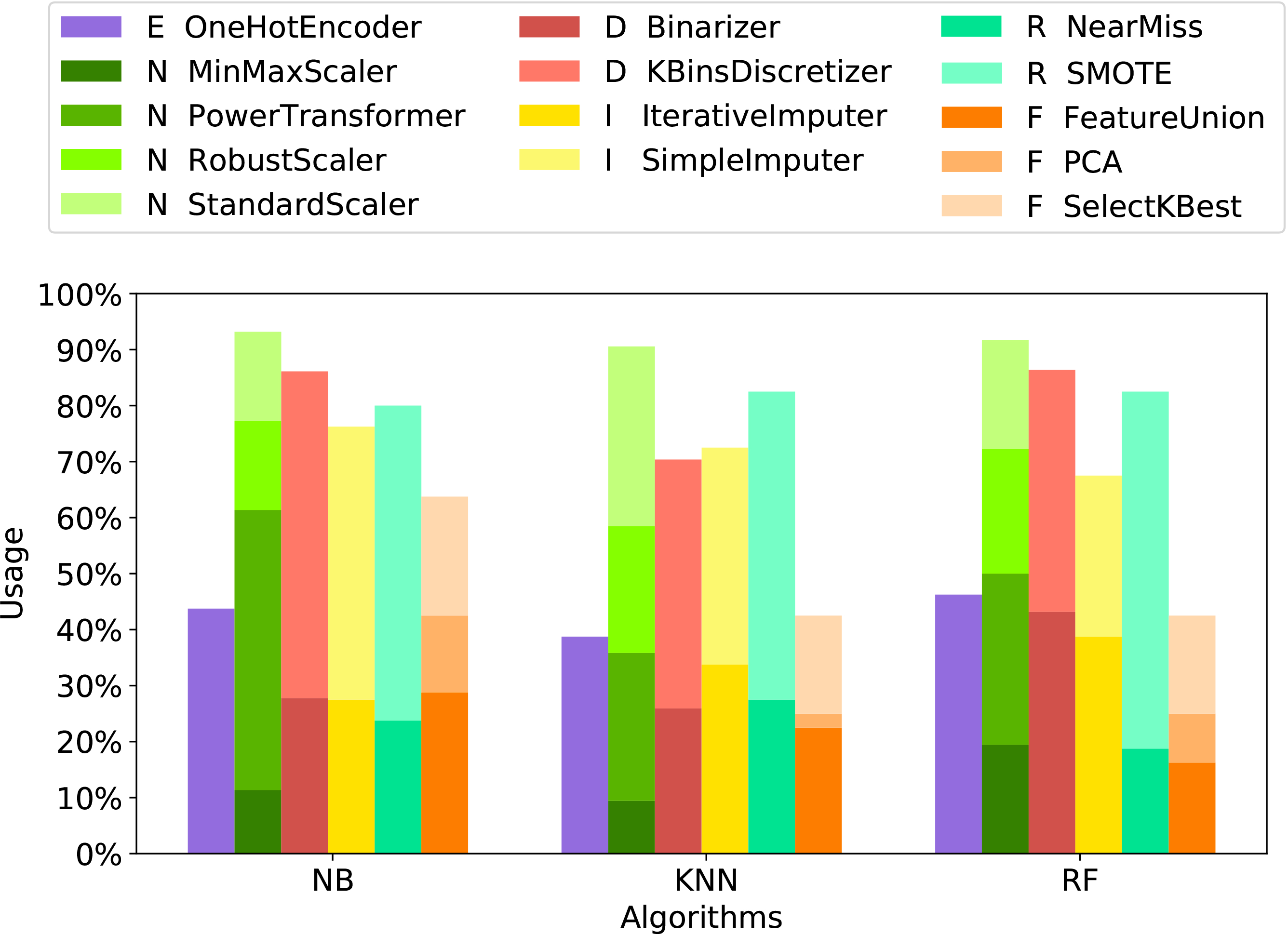


Figure 9: Percentage of use of a transformation in a physical pipeline.

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].](#_bookmark71) Lately, this technique

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

has been extended in order to predict the impact of transformations over the performance of classification algorithms and thus rank transformations based on their impact [[29,](#_bookmark72) [30,](#_bookmark73) [31].](#_bookmark74) The same idea can be applied for learning the best operator for a given transformation. That is, through meta-learning one can learn the intrinsic relationship between dataset characteristics and the operator performance, and thus come up with rules that are not obvious and are effective at the time of instantiating a transformation. The main idea is to build a model, that is able to predict the operator for a certain kind of transformation, given the meta-features extracted from the dataset considered for the optimization. This translates to answering the following question: “given that we know the dataset characteristics and having selected a certain kind of transformation (e.g., missing value imputation), what is the optimal physical algorithm (see Table [1)](#_bookmark7) we need to select, to obtain the highest improvement possible in terms of classification accuracy (i.e., when the classification algorithm is applied over the transformed dataset)?”. In particular, the model can generate a set of complementary rules that help in the optimization, providing a good starting instantiation for some of the transformations in the prototype.

To train the model we need a meta-dataset that can be (i) generated through optimization algorithms (e.g., SMBO executions), (ii) generated manually through simple evaluations of classification algorithms over transformed datasets, or (iii) assumed already given (e.g., OpenML).

Given a meta-dataset, we propose to learn to predict the best instantiation (operator) for a given transformation, where among the classes we can include the class None too. This means that one of the possible predictions is to not instantiate a transformation at all, hence remove it from the pipeline.

EXAMPLE 8. Our training dataset (sometimes referred to as ‘meta-database’ or ‘meta-dataset’) for the meta-learning is compiled through SMBO runs on the OpenML datasets (see Section [3.4.1).](#_bookmark16) That is, we first extract the dataset characteristics/profiles (i.e., number of features, number of instances, number of missing values, etc), and then by applying SMBO optimization, on classification algorithms and pre-processing pipelines (as explained in Section [3.5),](#_bookmark23) for each dataset, we retrieve the evaluations (i.e., predictive accuracy) of the algorithms over the optimized pipelines. This gives us the presumably optimal physical pipelines and their impact on the accuracy of the learning algorithms for each dataset at hand. Given such information, our aim is to now save time and improve the instantiation of the operators for each transformation considered in the prototype.

We trained several different Conditional Inference Trees [[32]](#_bookmark75) because they produce models that can be easily read and interpreted. Specifically, the independence of each variable (meta-features in our case) with the class (operator of a specific transformation) is tested through a statistical test. The split is made on the variable with the lowest p-value. We report the p-value too, so that it can be seen how strong the association is (i.e., why that variable was chosen). We stick with the p-value threshold of 0.05, and devise a rule from any branch of the tree that is within the threshold. In the following, we describe the rules obtained within the selected significance threshold.

Rules for Feature Engineering. The available operators in Scikit-learn for Feature Engineering

(see Table [1)](#_bookmark7) are: PCA (Principal Component Analysis), Feature Selection (Select K Best), Both (PCA + Select K Best), and None. The tree generated for the Feature Engineering trans- formation is shown in Figure [10.](#_bookmark32) The leaves show the selected operator frequency. For the sake of simplicity, we do not consider the union of PCA and Select K Best as an operator per se, instead we distribute that contribution to the two operators that compose it. Observe that there is a strong correlation between the Feature Engineering operator and the entropy of the class attribute. Indeed, such a meta-feature achieved a p-value smaller than 0.001. We can clearly read that if the Class Entropy is low, then Feature Selection is way more chosen than the other options (see Node 2). Recall that the entropy of an attribute is a measure of how much disorder there is among its instances. The less is that value, the easier is the classification problem. As a consequence, it is rea- sonable to think that the easier the classification problem is, the more likely is the fact that the class can be described by a low number of features. Hence, the Feature Selection technique can be successfully applied. Conversely, Node 5 shows that, when the Class Entropy is high, it is better to not apply any Feature Engineering operator. As a matter of fact, a high value of Class Entropy

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

involves a high number of classes and/or few instances per class, hence a really difficult problem. In such cases, reducing the dimensionality of the dataset does not lead to any improvement. Finally, when the Class Entropy is in between, there is no clear winner, and thus other non-obvious factors may affect the choice of the operator.

Rules for Rebalancing. As for Rebalancing, the operators considered from the imblearn[8](#_bookmark30) library

are: Near Miss, SMOTE, and None. The first is an undersampling algorithm which randomly eliminates the samples from the larger class. Instead, the second is an oversampling technique that creates samples of the minority class, as a linear combination of them. As shown in Figure [11,](#_bookmark33) the meta-feature Majority Class Percentage has a p-value of 0.014. This can be read as, in case of an unbalanced class problem (i.e., Node 3: Majority Class Percentage greater than 56), an oversampling of the minority class(es) is preferred to a downsampling of the majority one(s). However, when the Majority Class Percentage is smaller than 56%, the situation is not that clear, and there is no technique that is applied significantly more often than the rest; they are close to each other. Therefore, it is difficult to understand which problems (which dataset characteristics do they have) belong to Node 2. In summary, when the majority class has no more than 56% , it implies that it is an unbalanced class, and as mentioned above, SMBO tends to choose the same operator. However, when the majority class has less than 56%, it may imply that: (i) there are just two classes and the problem counts as a balanced problem, so no operator needs to be applied, or (ii) it is a multi-class problem, and thus there is no clear winner in terms of operators.

* 1. *Prototype instantiation*

The prototypes from the top flow and the meta-lerning rules from the bottom flow (if the optimization framework permits), are finally fed to the final step which deals with the instantiation and optimization of the prototypes. In this task we run an optimization algorithm that is executed until an optimal pipeline is found.

EXAMPLE 9. In our final execution, we run SMBO to find a suitable instantiation for the suggested prototypes. The simple but not obvious meta-learning rules, even though not included in our final execution, because of the implementation considered (i.e., HyperOpt), can potentially be used to 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).](#_bookmark34)

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).](#_bookmark37)

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).](#_bookmark39)

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).](#_bookmark16) Finally, the classification algorithms considered are *NB*, *KNN*, and *RF*. All the

585 experiments for a single algorithm, on average took approximately two weeks[9](#_bookmark31).

8<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)

100%

Class Entropy p<0.001

0.863 > 0.863

Class Entropy p<0.001

1.787 > 1.787

Node 2 (n=51)

100%

Node 4 (n=111)

100%

Node 5 (n=78)

80%

80%

60%

60%

40%

40%

20%

20%

0%

0%

80%

60%

Frequency

Frequency

Frequency

40%

20%

0%

None PCA SelectKBest

None PCA SelectKBest

None PCA SelectKBest

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

100%

MajorityClassPercentage p=0.014

0.56

> 0.56

Node 2 (n=150)

100%

Node 3 (n=90)

80%

60%

40%

20%

0%

80%

60%

Frequency

Frequency

40%

20%

0%

None NearMiss SMOTE

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,](#_bookmark35) we list the

589 exhaustive set of pipeline prototypes generated considering the compatible precedence graph in Table [2a](#_bookmark8)

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.](#_bookmark36) The enumerated prototypes are listed in the ordinate

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.](#_bookmark8) *E* - Encoding; *N*

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

20%

NB

KNN RF

Percentage of cases for which a prototype achieved the best performance

18%

16%

14%

12%

10%

8%

6%

4%

2%

0%

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)](#_bookmark35) to the optimization algorithm in order to get the best solutions out of them. 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:

1.2

1.0

Normalized distance

0.8

0.6

0.4

0.2

0.0

NB KNN RF

Algorithms

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

*Acc*(*deffective, a∗*) *− Acc*(*d, a*)

*normalized distance* =

*Acc*(*d*

*exhaustive*

*, a∗*) *− Acc*(*d, a*)

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*(*deffective, a∗*) is the accuracy of the optimized

616 algorithm *a∗* over the dataset *deffective* transformed using the optimized instantiation of the effective

617 set of prototypes (i.e., our approach). Finally, *Acc*(*dexhaustive, a∗*) is the accuracy of the optimized

618 algorithm *a∗* over the dataset *dexhaustive* 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.](#_bookmark38) 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 optimization[10](#_bookmark40).

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],](#_bookmark68) we split the budget 50-50 between the pre-processing pipeline optimization

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).

1

Normalized impact

0.5

Pre-processing + Hyper-parameter optimization NB KNN

Hyper-parameter optimization

RF

1

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.

*pp impact*

*Acc*(*deffective, a∗*) *− Acc*(*d, a*)

=

*max*(*Acc*(*d*

*effective*

*, a∗*)*, Acc*(*d, a∗*)) *− Acc*(*d, a*)

*hp impact*

=

*max*(*Acc*(*d*

*Acc*(*d, a∗*) *− Acc*(*d, a*)

*effective, a∗*)*, Acc*(*d, a∗*)) *− Acc*(*d, a*)

648

649

650

651

652

653

654

655

656

657

658

659

660

661

where, *Acc*(*d, a*) is the baseline accuracy (i.e., predictive accuracy of the algorithm *a* with default hyper-parameters over the original dataset *d*). *Acc*(*deffective, a∗*) is the accuracy of the optimized algo- rithm *a∗* over the dataset *deffective* transformed using the optimized instantiation of the effective set of prototypes obtained using our method. Finally, *Acc*(*d, a∗*) is the accuracy of the optimized algorithm *a∗* (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:

*normalized pp impact* = *pp impact*

*pp impact* + *hp impact*

We perform the same for the hyper-parameter impact and plot the results obtained for all the algo- rithms and datasets in Figure [14,](#_bookmark41) where each bar represents the results obtained for a single dataset. The different colors represent the impact values of pre-processing and hyper-parameter optimization.

Observing the bar-charts one can see that (i) dedicating a portion of the budget to pre-processing, brings benefit to the analysis in most of the cases (i.e., 73% of the cases), and (ii) the impact of hyper- parameter optimization, increases with the increase of the number of hyper-parameters of the ML al- gorithm (e.g., hyper-parameter optimization impacts more RF than NB). Overall, we can conclude that pre-processing is a critical step that once effectively applied may have a high positive impact on the final 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]:](#_bookmark76) distributed, cloud-

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,](#_bookmark77) [35,](#_bookmark78) [36,](#_bookmark79) [37].](#_bookmark80) 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 border line 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],](#_bookmark68) the DPSO problem, as we use it in this work, is formally defined. 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),](#_bookmark34) a single fixed prototype cannot perform best for

689 every dataset. Therefore, we build on top of [[25],](#_bookmark68) and instead of relying on a fixed prototype, we define

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

691 In PRESISTANT [[30,](#_bookmark73) [38,](#_bookmark81) [31],](#_bookmark74) we tackled the problem of recommending pre-processing operators

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],](#_bookmark82) authors define an algorithm that aims at prioritizing the cleaning of records that

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],](#_bookmark83) based on a reinforcement learning technique, for a given dataset, and an ML

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],](#_bookmark68) the pipeline prototype is fixed in advance.

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],](#_bookmark84) authors follow a similar approach to ours in that they define two steps for

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]](#_bookmark68)

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,](#_bookmark85) [43,](#_bookmark86) [44].](#_bookmark87) Since Bayesian optimization is randomized, meta-learning has been used

719 to find a good seed for the search [[45].](#_bookmark88) Most of these works however, only minimally consider the data

720 pre-processing step. Auto-WEKA [[43],](#_bookmark86) based on the Java machine learning library Weka, is the pioneer

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]](#_bookmark89) combines a hierarchical stacking architecture and an evolutionary algorithm (EA).

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]](#_bookmark90) exploits collaborative filtering for AutoML, choosing models that have performed well on

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]](#_bookmark85) is based on the popular Python library scikit-learn. 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]](#_bookmark87) is a tree-based pipeline optimization tool using genetic programming while requiring little

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]](#_bookmark46) uses hierarchical planning, a particular form of AI planning, to propose a solution to

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]](#_bookmark91) is a Python open-source tool. Like in ML-Plan [[3],](#_bookmark46) both pre-processing and mod-

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 timization is 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],](#_bookmark88) hence as immediate future work, we intend to extend an optimization framework (i.e.,

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 Innovacio´n 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 Polite`cnica 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, SIGKDD Explor. 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. Hu¨llermeier, Ml-plan: Automated machine learning via hierarchical planning, Machine Learning

815 107 (8) (2018) 1495–1515.

816 [4] L. Mun˜oz, J.-N. Mazo´n, 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. Zima´nyi, Data Warehouse Systems - Design and Implementation, Data-Centric Systems and Applica-

819 tions, Springer, 2014.

820 [6] B. Bilalli, A. Abello´, 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. Mun˜oz, 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. Abello´, 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. Zo¨ller, 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. Abello´, 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. Abello´, 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. Abello´, T. Aluja-Banet, R. Wrembel, PRESISTANT: learning based assistant for data pre-processing, Data

869 Knowl. Eng. 123 (2019).

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

1. T. Hothorn, K. Hornik, A. Zeileis, Unbiased recursive partitioning: A conditional inference framework, Journal of Com- putational and Graphical Statistics 15 (3) (2006) 651–674.
2. R. Elshawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges, arXiv preprint arXiv:1906.02287 (2019).
3. F. Geerts, G. Mecca, P. Papotti, D. Santoro, The llunatic data-cleaning framework, PVLDB End. 6 (9) (2013) 625–636.
4. Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiane´-Ruiz, N. Tang, S. Yin, Bigdansing: A system for big data cleansing, SIGMOD ’15, 2015, pp. 1215–1230.
5. X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, Y. Ye, Katara: A data cleaning system powered by knowledge bases and crowdsourcing, SIGMOD ’15, 2015, pp. 1247–1261.
6. Z. Jin, M. R. Anderson, M. Cafarella, H. V. Jagadish, Foofah: A programming-by-example system for synthesizing data transformation programs, SIGMOD ’17, 2017, pp. 1607–1610.
7. B. Bilalli, A. Abello´, T. Aluja-Banet, R. F. Munir, R. Wrembel, PRESISTANT: data pre-processing assistant, CAiSE Forum ’18, 2018, pp. 57–65.
8. S. Krishnan, J. Wang, E. Wu, M. J. Franklin, K. Goldberg, Activeclean: Interactive data cleaning for statistical modeling, PVLDB 9 (12) (2016) 948–959.
9. L. Berti-E´ quille, Learn2clean: Optimizing the sequence of tasks for web data preparation, WWW ’19, 2019, pp. 2580–

2586.

1. Z. Shang, E. Zgraggen, B. Buratti, F. Kossmann, P. Eichmann, Y. Chung, C. Binnig, E. Upfal, T. Kraska, Democratizing data science through interactive curation of ML pipelines, SIGMOD ’19, 2019, pp. 1171–1188.
2. M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, NeurIPS ’15, 2015, pp. 2962–2970.
3. C. Thornton, F. Hutter, H. H. Hoos, et al., Auto-weka: Combined selection and hyperparameter optimization of classifi- cation algorithms, in: KDD, 2013, pp. 847–855.
4. R. S. Olson, N. Bartley, R. J. Urbanowicz, J. H. Moore, Evaluation of a tree-based pipeline optimization tool for automat- ing data science, GECCO ’16, 2016, pp. 485–492.
5. M. Feurer, J. T. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, AAAI ’15, 2015, pp. 1128–1135.
6. B. Chen, H. Wu, W. Mo, I. Chattopadhyay, H. Lipson, Autostacker: A compositional evolutionary learning system, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 402–409.
7. C. Yang, Y. Akimoto, D. W. Kim, M. Udell, Oboe: Collaborative filtering for automl model selection, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1173–1183.
8. M. J. Smith, C. Sala, J. M. Kanter, K. Veeramachaneni, The machine learning bazaar: Harnessing the ML ecosystem for effective system development, CoRR abs/1905.08942 (2019).