EDCA – An Evolutionary Data-Centric AutoML Framework for Efficient Pipelines
(Apendix)

Joana Simões

April 2025

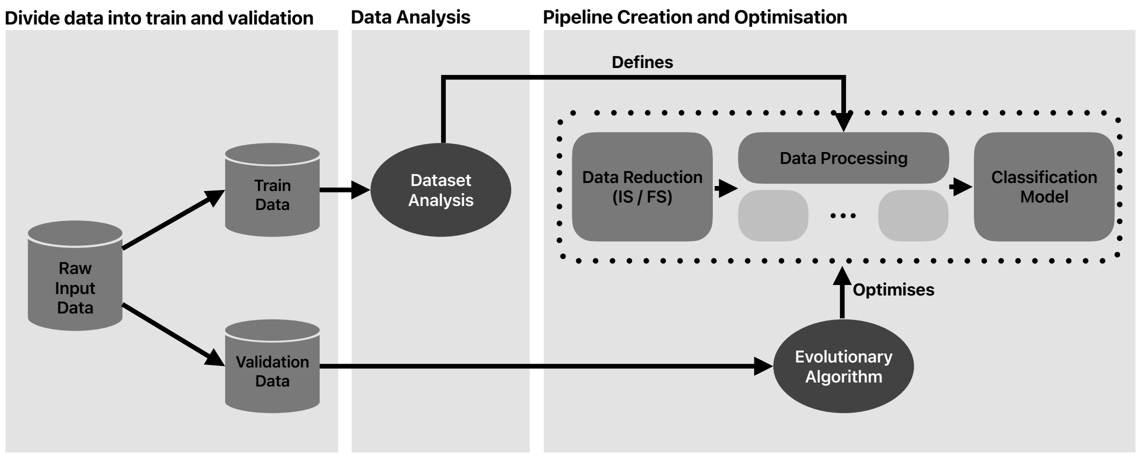

Figure 1: EDCA's overview

EDCA's Process

EDCA includes several components that create a framework capable of finding the most appropriate ML pipeline for a given task and dataset. The ML pipeline steps are created based on an initial dataset analysis. Different preprocessing sub-tasks are added to the pipeline depending on the data characteristics. Therefore, the pipeline structure is dynamic because it varies according to the received data, and some components can be turned on or off, but linear since the data has only one path to follow. At the end of the pipeline, there is always a predictive model for the problem. The pipeline can also include steps to select the most discriminatory data regarding instances and features. To find the best pipeline for the problem, an EA is implemented. For a given time budget, the EA tries to select the best configuration for the pipeline. After optimising, the best pipeline found is used in the ML application. Figure 1 presents an overview of the EDCA's process.

EDCA can include two DR steps for the instances and features, respectivly. Since using them may or may not be beneficial, EDCA uses an automatic data optimisation. This automatic data optimisation searches for the best combination of the DR techniques. It can be chosen that the best option is to use only one type of data selection or both or none of them. During the optimisation process, different combinations for the DR techniques are tested along with the other steps of the pipeline and the best at the end is selected. When a DR technique is activated, it says which indices, related to an instance or feature, should be used to train the pipelines. The indices vary between zero and the maximum length possible of the dataset, in terms of instances or features, i.e., $[0, max_{instances}[$ or $[0, max_{features}[$, respectively.

To create the data processing steps for the pipeline, the given dataset is analysed. This analysis aims to asses which processing techniques are required for the given data and cut the search space to the minimum. It asses the features, types, and characteristics of the data. Each feature in a dataset could belong to one of four groups: a binary column, a categorical column, a numerical column or an identifier. A binary feature, typically containing booleans, indicates that only two possible values exist. The numerical features could be composed of integers or float values, but they are assumed to be only numbers in this case. The categorical values include all the features that have data represented with text. They could be nominal or ordinal; however, the system considers them only categorical. The identifiers contain features where all the values are different.

With all the variables separated by their type, the analysis determines if the features have missing values. If they exist, the system annotates them. The output of the dataset analysis is several lists, each containing the features that belong to that group. These lists are necessary to understand the types of preprocessing needed for the problem, and each group of features will have a different treatment.

Based on the analysis of the dataset, a pipeline structure is created as described in Figure 2. In total, five different sub-steps could exist in the preprocessing pipeline. If the analysis indicates that the dataset contains numerical features, a normalisation step will be added to the pipeline. The columns with identifiers will be removed from the dataset since they do not add relevant information. An encoder step

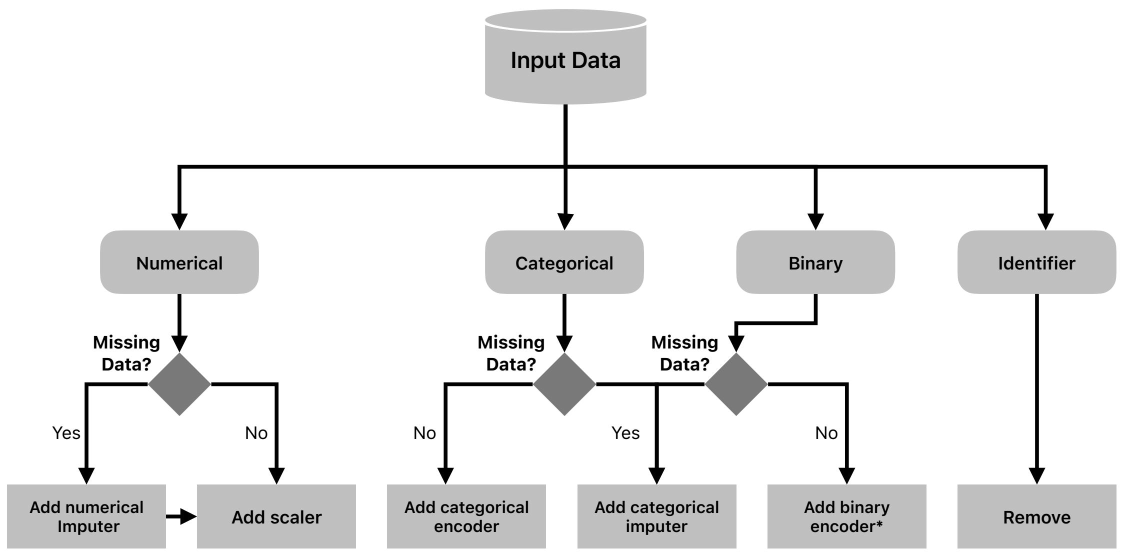

Figure 2: Dataset analysis to understand the data processing steps required to exist on the pipeline

should be added to the pipeline when the features are categorical, as the models only digest numerical features. When there are features with missing data, several imputation steps are added to the pipeline, one for each type of feature, since the hyperparameters will be different depending on the feature type.

Creating the pipeline's structure based on the data characteristics allows using only the strictly mandatory preprocessing steps. Therefore, reducing the preprocessing search space to the minimum required. If all the preprocessing steps were available to all datasets, it would be spent time tuning irrelevant methods for the problem ahead. For example, spending time optimising a numerical imputer when the data had no missing values. Thus, using an initial dataset analysis to define the preprocessing steps required for the problem and corresponding search space helps reduce the problem's complexity and only spend more time tuning the essential ML steps. Figure 2 presents the data analysis scheme to create the preprocessing pipeline.

The framework includes a final step on the pipeline to a predictive model, similar to other AutoML frameworks. A suitable model should be used at this stage since it would impact the final results. At this point in the process, diverse models can be applied, each offering its distinctive approach. Nevertheless, they share the objective of predicting the outcomes of the pipeline.

It can be concluded that the proposed framework will have a large search space, with different models to use at each phase of the pipeline and all of them with several hyperparameters that impact the final results. Therefore, an effective optimisation algorithm is essential to surpass the challenges of the problem.

EA have already been established to find good solutions in large and complex search spaces. EA is inspired by the process of biological evolution in natural selection, applying selection pressure to select the most apt individuals (solutions). It starts with an initial population that is evaluated. Then, the most qualified individuals are selected to reproduce. The generated offspring is then modified by variation operators such as crossover and mutation. The population and offspring are re-evaluated, and the most apt individuals pass to the next generation. This cycle repeats until a specific stop criterion is satisfied. The stop criterion can be reaching a particular performance, making a certain number of generations or using a time budget for the optimisation. In this work, we use a branch of EA, GA, where we represent the individuals as strings containing the ”genes”.

The variation operators allow for a trade-off between exploration and exploitation. Initially, with the crossover operator, the exploration allows searching over the entire search space to find good overall solutions. With already reasonable solutions, the mutation operator looks into the surroundings to see if they are the best or if it can find even better ones.

Individuals representation

Each individual is represented by a dynamic list, where each gene represents a step of the ML pipeline (see example in Figure 3). The data-reduction-related genes are composed of a set of integer numbers,

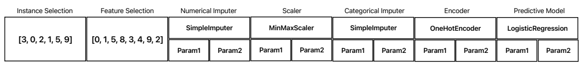

Figure 3: Example of one individual

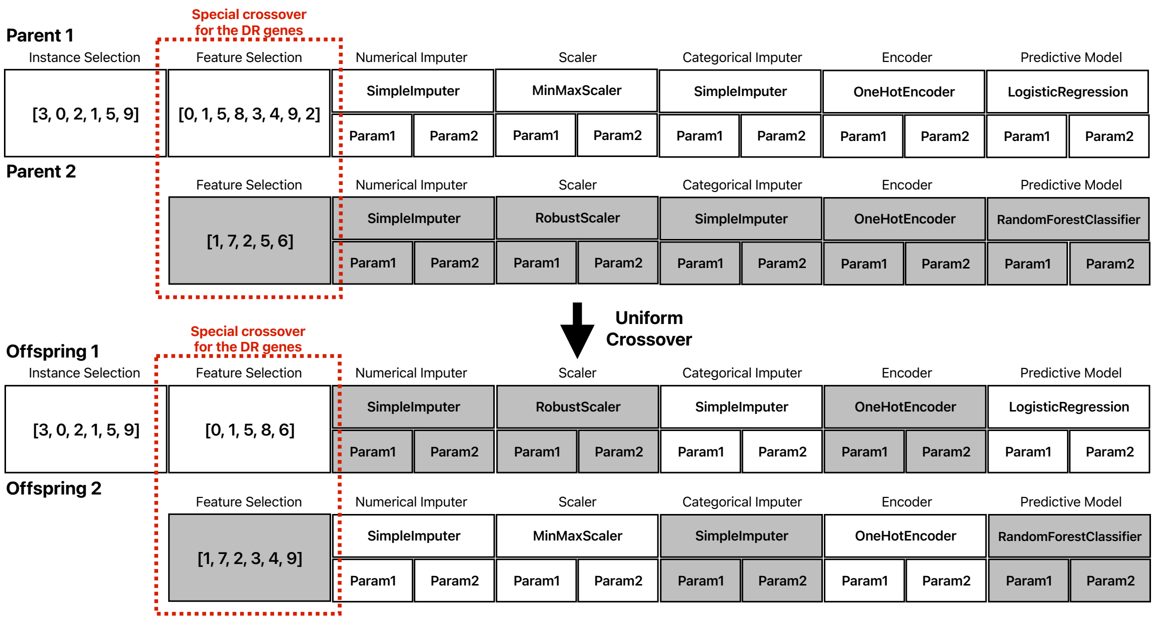

Figure 4: Uniform Crossover operator applied to the individuals to create the offspring. The DR genes use a special crossover due to the different representation. As can be seen, the crossover is not applied to the instance selection (IS) gene since it only appears for one individual. It passes directly to one offspring.

where each number represents an index in the training dataset in terms of instances of features, indicating that the data value of that index should be used to train the pipeline. The remaining genes correspond to the other pipeline steps previously stated and are composed of the model's name and its hyperparameters. Although referred to as a "model", it can be a preprocessor (scaler, encoder, etc.) or a classification model. The word "model" is only to be in line with the literature and frameworks' nomenclature.

The pipeline (individual) size depends on the initial analysis of the dataset, where the required preprocessing steps are defined. After the data analysis, the processing steps are always present on the individual. Yet, the pipeline can vary in size during the optimisation process since the data reduction techniques (instance and feature selection) may or may not be applied. The other genes, however, are always present in the pipeline, only varying in the number of hyperparameters.

Variation operators

The GA requires two different variation operators to create the offspring, i.e., the new individuals. The variation operators could be crossover or mutation.

Based on a given probability, the crossover operator receives two individuals and returns another to individuals. The genes of the received individuals are switched between them in a process called uniform crossover, except for the data-reduction-related genes. Since the data reduction genes include another set of values inside of them, these have a particular crossover operator. Considering that each gene might or may not be present in the individuals, the crossover operation can only be applied when it is present in both individuals. If it is present in both, one-point crossover is carried out at the gene level. Otherwise, the gene is passed on to the offspring corresponding to the initial individual. Figure 4 demonstrates the overall crossover operation in two individuals and Figure 5 shows an example of a possible one-point crossover for a data reduction gene. Both figures demonstrate the same situation to see the outcome.

The mutation operator introduces a new "genetic code" into the population. Given a certain probability of mutating the individual, only one gene per individual is changed. The general case works by changing the model and, consequently, its hyperparameters or changing only the hyperparameters

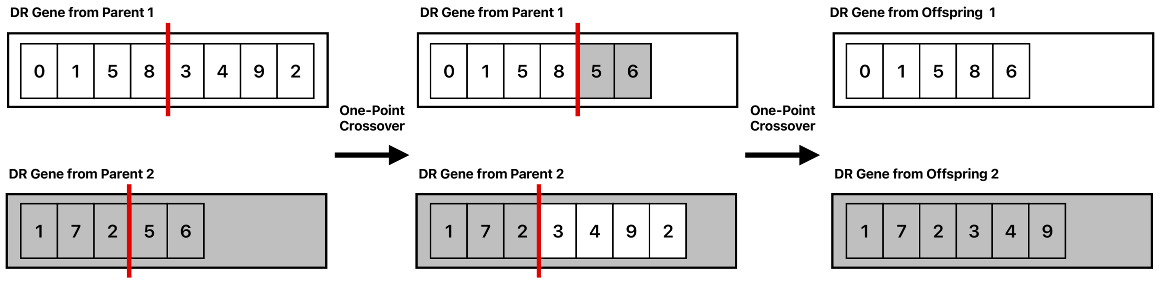

Figure 5: Crossover operator applied to the DR genes when the gene is present in both parents. To avoid data augmentation, we eliminate the repetitive indexes.

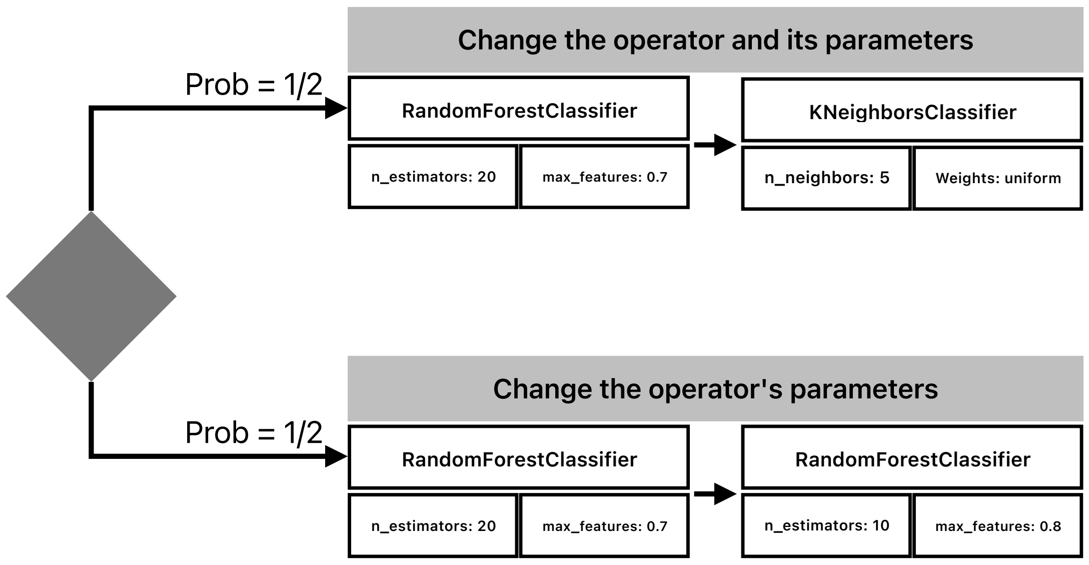

Figure 6: Mutation applied to the models' genes.

of a model (see Figure 6). By model, it is considered a method of the pipeline. Since automatic data optimisation is applied to the data reduction genes, depending on the presence or absence of the gene, different actions may be used. When it is chosen to mutate a DR gene, three different operations can be applied: change $\theta\%$ of the indices for other values, add $\theta\%$ for new values, and delete $\theta\%$ of the values. The user defines the percentage of change from the initial data size that can be changed, $\theta\%$. Figure 7 shows an example of a mutation to the data-reduction genes.

Fitness function to evaluate the individuals

The fitness function $f(x)$ (Equation 1) used in EDCA aims to optimise several components in the light of Green AutoML and to produce the best prediction results possible. Therefore, the fitness function applied is multi-objective and should minimise:

$$f(x) = \alpha * (1 - x_m) + \beta * x_d + \gamma * x_c \tag{1}$$

- $(1 - x_m)$: The prediction error, i.e., it should maximise the performance metric (x_m);
- x_d: The percentage of data used to train the models, calculated by the total size of the final selected dataset compared with the initial dataset size;
- x_c: The computational cost of training each model corresponds to the total CPU time needed to train the individual (pipeline). The component is normalised according to Formula 2 where t is the time spent evaluating the individual in seconds;

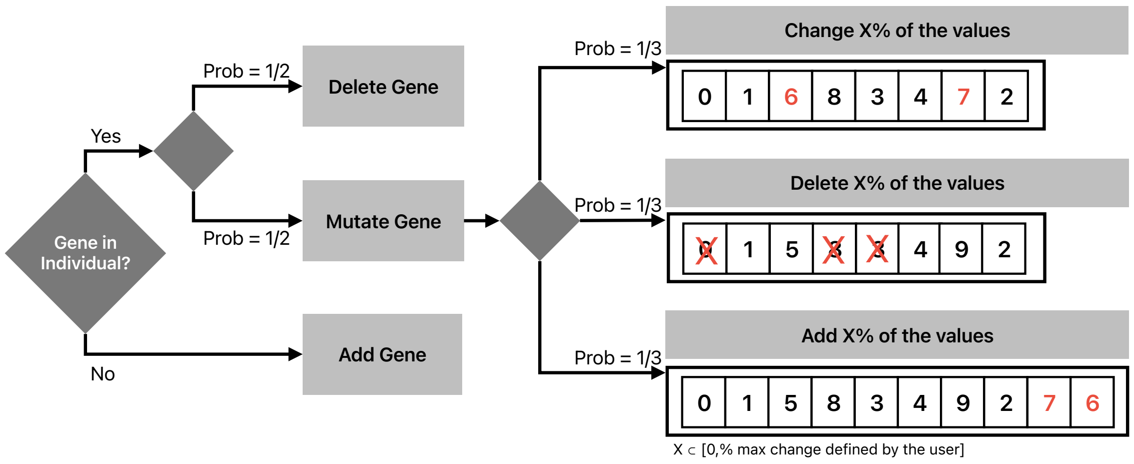

Figure 7: Mutation operator applied to the DR genes, it can add, change or delete the gene.

$$x_c = 1 - \frac{1}{1+t} \tag{2}$$

The fitness function is a weighted sum of its components. Each of the components' values ranges from 0 to 1 and have a weight associated, that adds up to 1. Therefore, the range of possible values belongs to the interval $[0, 1]$. 0 corresponds to the best possible individual and 1 to the least fitted individual since we use a minimisation problem. The weights associated with each component help change the necessary trade-off between the components. In some cases, it might be beneficial to turn off some components, i.e., assigning a zero weight to an element.

For the paper, the fitness function only contains the search metric prediction error, ranging between 0 (best case) and 1 (worst case). However, other fitness functions can be applied by given different importances to each component.

GA components

The optimisation with GA uses some techniques to save time and help the search process. The GA uses elitism to save promising solutions for one generation to the next, where the user defines the number of saved individuals. To maintain diversity and avoid performance stagnation, the optimisation process uses a patience mechanism to wait for best solutions. The population is restarted when the performance of the best individuals does not improve after a certain number of generations. The restart generates new individuals who substitute for the current population, except for the best individuals defined by elitism. The optimisation can also make an early stopping of the optimisation process if the performance stagnates even with the restarts.

Similar to other AutoML frameworks, EDCA offers parallel searching to evaluate multiple solutions simultaneously. This allows for the optimisation to be sped up and for more generations of the EA.

Results achieved in comparison with FLAML and TPOT

Visual analysis of the results

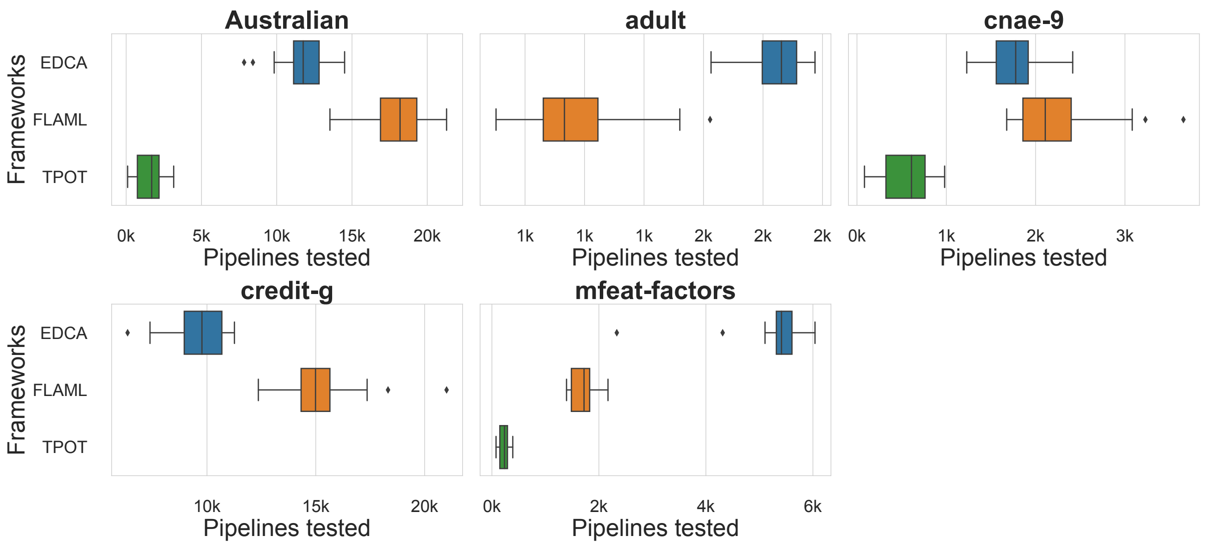

Figure 8: Boxplot with the distribution of the number of pipelines evaluated by each AutoML framework.

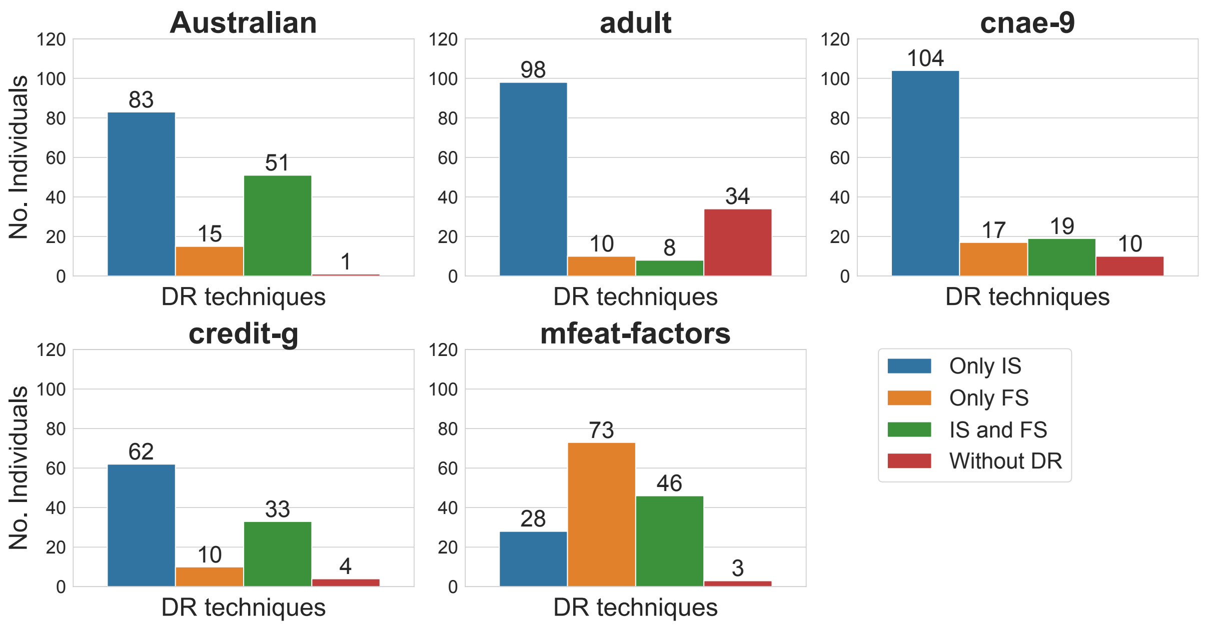

Figure 9: Histogram with the final DR techniques used in EDCA.

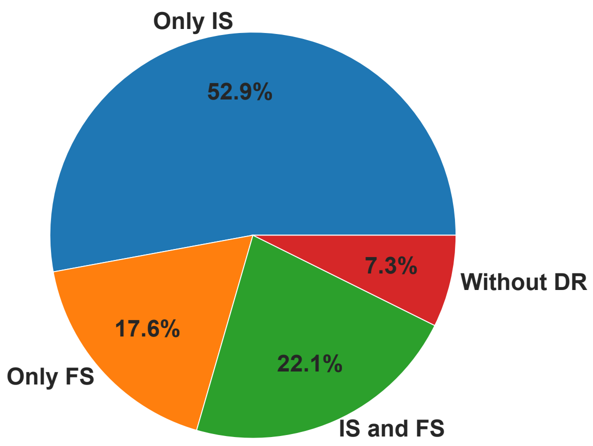

Figure 10: Distribution of percentage of each DR technique used in EDCA.

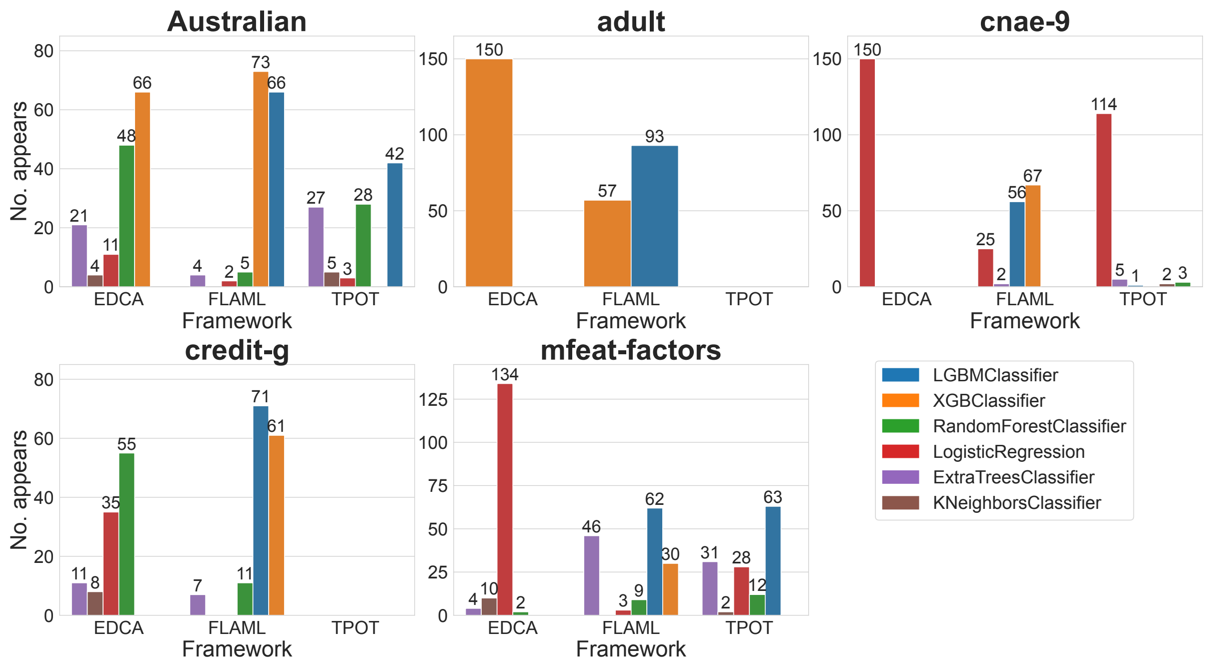

Figure 11: Histogram with the final classification models selected by each framework.

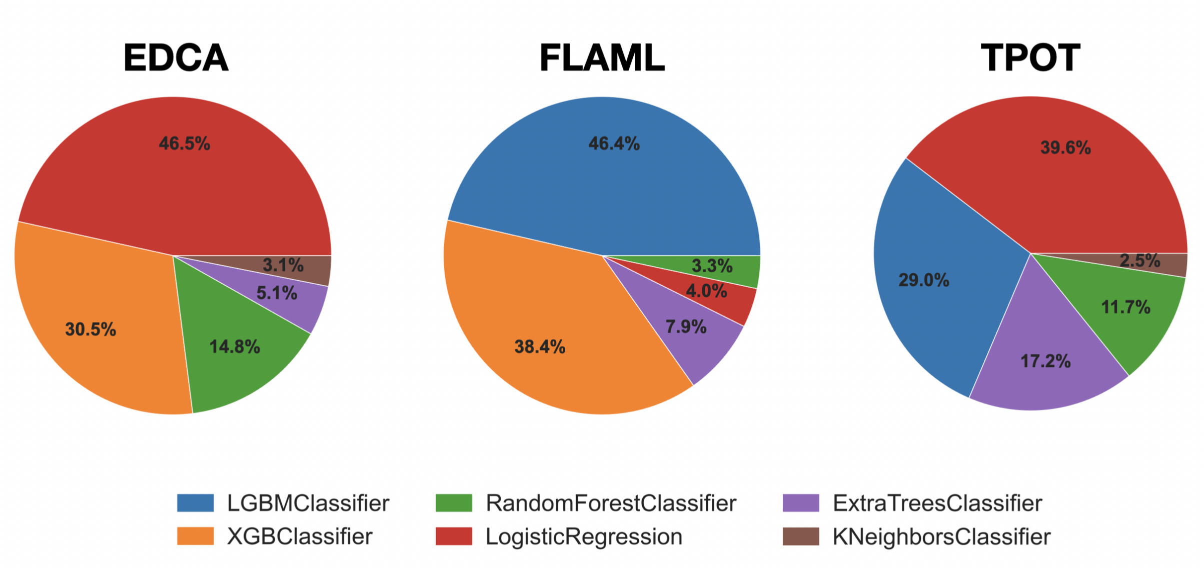

Figure 12: Overall distribution of the final classification models used by the frameworks.

Statistical comparison

The present subsection contains a statistical analysis of the results achieves by EDCA when compared with FLAML and TPOT.

Table 1: Average MCC values (± standard deviation) over 30 runs for EDCA, FLAML, TPOT, and FLAML + EDCA and TPOT + EDCA (i.e., FLAML or TPOT retrained with EDCA-selected data).. Results for EDCA and FLAML after retraining on all data are included, while TPOT always uses all data. Statistically significant differences compared to EDCA are in bold. The "*" reveals statistically significant differences from original framework's results. Arrows show changes after retraining.

Dataset	EDCA	FLAML	TPOT	EDCA-All	FLAML-All	FLAML+EDCA	TPOT+EDCA
Australian	0.72±0.01	0.72±0.02	0.72±0.02	**0.73±0.02**↑	0.72±0.02	**0.69±0.04***↓	**0.64±0.1***↓
adult	0.63±0.0	0.63±0.01	-	**0.63±0.0**	0.63±0.01	**0.63±0.0**	-
cnae-9	0.93±0.01	0.93±0.01	**0.94±0.01**	**0.94±0.01**↑	0.93±0.01	**0.91±0.01***↓	**0.92±0.03***↓
credit-g	0.32±0.04	0.34±0.03	-	**0.36±0.03**↑	0.34±0.02	**0.24±0.07***↓	-
mfeat-factors	0.97±0.0	**0.96±0.0**	0.97±0.0	**0.97±0.0**	0.96±0.0	**0.95±0.0***↓	**0.96±0.01***↓

Table 2: Average F-Score values (± standard deviation) over 30 runs for EDCA, FLAML, TPOT, and FLAML + EDCA and TPOT + EDCA (i.e., FLAML or TPOT retrained with EDCA-selected data).. Results for EDCA and FLAML after retraining on all data are included, while TPOT always uses all data. Statistically significant differences compared to EDCA are in bold. The "*" reveals statistically significant differences from original framework's results. Arrows show changes after retraining.

Dataset	EDCA	FLAML	TPOT	EDCA-All	FLAML-All	FLAML+EDCA	TPOT+EDCA
Australian	0.86±0.01	0.86±0.01	0.86±0.01	0.87±0.01↑	0.86±0.01	**0.85±0.02***↓	**0.81±0.06***↓
adult	0.87±0.0	0.87±0.0	-	0.87±0.0	0.87±0.0	0.87±0.0	-
cnae-9	0.94±0.0	0.94±0.01	**0.95±0.01**	**0.95±0.01**↑	0.94±0.01	**0.92±0.01***↓	**0.92±0.04***↓
credit-g	0.72±0.01	0.72±0.01	-	**0.73±0.01**↑	0.73±0.01↑	**0.69±0.03***↓	-
mfeat-factors	0.97±0.0	**0.96±0.0**	0.97±0.0	**0.98±0.0**↑	0.97±0.0↑	**0.96±0.0***	**0.96±0.01***↓

Table 3: Average percentage of data (± standard deviation) over 30 runs used by the frameworks. It includes the percentage of total, instances and features that were calculated by dividing the final data size by the original size. Statistically significant differences compared to EDCA are highlighted in bold.

(a) Percentage of data

Dataset	**EDCA**	**FLAML**	**TPOT**
Australian	0.35±0.06	**0.75±0.0**	**1.02±0.05**
adult	0.64±0.04	**0.75±0.0**	-
cnae-9	0.59±0.05	**0.75±0.0**	**0.98±0.06**
credit-g	0.4±0.09	**0.75±0.0**	-
mfeat-factors	0.47±0.06	**0.75±0.0**	**0.99±0.03**

(b) Percentage of instances

Dataset	**EDCA**	**FLAML**	**TPOT**
Australian	0.43±0.09	**0.75±0.0**	**1.0±0.0**
adult	0.66±0.05	**0.75±0.0**	-
cnae-9	0.6±0.06	**0.75±0.0**	**1.0±0.0**
credit-g	0.47±0.09	**0.75±0.0**	-
mfeat-factors	0.67±0.05	**0.75±0.0**	**1.0±0.0**

(c) Percentage of features

Dataset	**EDCA**	**FLAML**	**TPOT**
Australian	0.84±0.1	**1.0±0.0**	**1.02±0.05**
adult	0.98±0.03	**1.0±0.0**	-
cnae-9	0.98±0.03	**1.0±0.0**	0.98±0.06
credit-g	0.86±0.11	**1.0±0.0**	-
mfeat-factors	0.72±0.1	**1.0±0.0**	**0.99±0.03**