# Genetic algorithms for AutoML in process predictive monitoring

Nahyun Kwon[1] and Marco Comuzzi[2]

Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea
{eekfskgus,mcomuzzi}@unist.ac.kr

**Abstract.** In recent years, AutoML has emerged as a promising technique for reducing computational and time cost by automating the development of machine learning models. Existing AutoML tools cannot be applied directly to process predictive monitoring (PPM), because they do not support several configuration parameters that are PPM-specific, such as trace bucketing or encoding. In other words, they are only specialized in finding the best configuration of machine learning model hyperparameters. In this paper, we present a simple yet extensible framework for AutoML in PPM. The framework uses genetic algorithms to explore a configuration space containing both PPM-specific parameters and the traditional machine learning model hyperparameters. We design four different types of experiments to verify the effectiveness of the proposed approach, comparing its performance in respect of random search of the configuration space, using two publicly available event logs. The results demonstrate that the proposed approach outperforms consistently the random search.

**Key words:** AutoML, genetic algorithm, predictive process monitoring, hyperparameter optimization

## 1 Introduction

Predictive process monitoring (PPM) is concerned with creating predictive models of aspects of interests of running process cases using the historical process execution data logged in so-called event logs [1]. Typical aspects predicted are the outcome of running cases or the next event to be executed in a running case.

PPM research has endured an exponential success in the last decade. However, the same cannot be said about the uptake of PPM solutions in practice. Existing commercial process mining tools, like Celonis or Apromore, have introduced simple PPM solutions only very recently. We argue that the main reason for such a limited uptake is the gap between the developer of the PPM models (a process mining expert) and the user of these models (a process analyst). The latter have the knowledge to interpret the insights given by PPM models, but they often lack the technical skills of the former to develop the PPM models effectively. This gap can be seen as an instance of a more general gap between machine learning experts, who develop models, and business analysts, who are in charge of using the insights returned by these models to take business decisions.

AutoML [2] is one prominent solution to bridge this gap. It aims at creating automated ways to support multiple aspects of the traditional machine learning model development pipeline, like data preparation, feature extraction, model selection, or hyperparameter optimisation. Specifically, given a dataset and a machine learning problem, the aim of an AutoML framework is to propose an optimal model to the user, hiding most of the inner details regarding the model development. AutoML solutions have proliferated in the last few years [3], even being touted to represent the "death of the data scientists".

In PPM, AutoML has received little attention. This is to some extent not surprising, since AutoML solutions for traditional machine learning problems cannot be directly instantiated into PPM problems. Besides the model hyperparameter optimisation, in fact, PPM requires to optimise other *parameters*, such as the type of trace encoding or bucketing used, which are PPM-specific and, therefore, cannot be directly understood by existing AutoML tools.

More broadly, the benchmark experiments for different PPM use cases [4, 5, 6] published in the literature provide only generic guidelines regarding the effectiveness of different ML techniques in specific PPM scenarios, but no automated solution. Nirdizati [7], i.e., a tool for automated development of PPM models, can develop different PPM models for a given PPM problem and show the results to the user. However, it has only limited facilities to optimise the models shown to the user. The only approach resembling AutoML is the one proposed by Di Francescomarino et al. [8] (also implemented within Nirdizati), in which genetic algorithms are adopted to optimise the parameters of outcome-based PPM models.

In this paper we propose a simple yet extensible AutoML framework for developing well-performing PPM models. The framework aims at optimising a set of PPM model parameters that comprise: the specific model used to create a predictive model (e.g., decision tree vs. random forest), the hyperparameters of the model, and other parameters specific to PPM, like the technique used to encode traces or the number of prefix buckets used to develop a model in the case of outcome prediction.

The presentation of the framework is split into two parts: (i) the solution space identification and (ii) a model optimisation method based on genetic algorithms (GA). In this paper, the proposed framework is instantiated in the case of outcome-based PPM. However, we argue that only little adaptation would be required for its instantiation in other PPM use cases, like next-activity prediction, that yield a machine learning classification problem. The framework is evaluated on two publicly available real world event logs and comparing different experiment configurations in respect of a baseline that involves random search of the configuration parameter space.

The paper is organised as follows. Section 2 briefly discusses the related work. Section 3 presents the parameter optimization space in the case of outcome-based PPM, while Section 4 presents the application of GA to solve the problem of finding a high performing model. The results of the evaluation are presented in Section 5, while conclusions are drawn in Section 6.

**Table 1.** PPM-specific model configuration space

| parameter | range |
|---|---|
| `drop_act` | {2,4,6,8} |
| `bucketing` | [1,2 * mean trace length] |
| `encoding` | {'aggregate', 'index'} |
| `model` | {'DT', 'RF', 'XGB', 'LGBM'} |

## 2 Related work

AutoML automates the process of developing the *best* model, e.g., the most accurate, to address a given machine learning challenge, speeding up the model development phase and facilitating the application of ML techniques even by non-experts. Different AutoML frameworks, such as Auto-sklearn, Tree-Based Pipeline Optimization Tool (TPOT), or H2O, provide different automated solutions for each different step of the typical machine learning pipeline [2, 3], such as data preparation or hyperparameter optimisation.

Predictive process monitoring [9, 1] concerns various prediction tasks such as predicting the outcome of a process [4], the next event of a running case [6], or time-related measures [5]. Approaches in the literature often define process outcomes as the satisfaction of service level agreements or the satisfaction of temporal constraints defined on the order and the occurrence of tasks in a case. Extensive efforts have been devoted to enhancing the performance of predictive monitoring models. Recently, deep learning is increasingly applied to solve the problem of outcome prediction [10]. However, deep learning-based approaches require extensive specialist skills by model developers to set the model hyperparameters effectively.

As mentioned in the Introduction, AutoML has been generally neglected by the PPM literature, with the exception of [8]. In respect of the work of Di Francescomarino et al. [8], the framework proposed in this paper considers different encoding and bucketing methods, additional parameters, such as the dropping of infrequent activities, a broader set of models, including boosting ensemble models, and different experiment configurations instead of a single one in which all the parameters are optimised using the GA at once.

## 3 A configuration space for predictive monitoring

We consider the PPM use case of outcome-based predictive monitoring, where the aim is to predict a binary categorical outcome of running cases. An analysis of the literature prompted us to design a configuration space that includes four PPM-specific parameters, which are shown in Tab. 1 and discussed next.

`Drop_act` : this parameter captures the process of removing low-frequency activities from an event log. It can give benefits of reducing the computational cost when creating a predictive model and it has been demonstrated to improve the model performance in some cases [11]. We consider a discrete gap-based

**Table 2.** Configuration space of hyperparameyters of classification models

| parameter | | range |
|---|---|---|
| DT | `max_depth` | (2,20) |
| | `min_samples_leaf` | (5,100) |
| | `criterion` | ['gini', 'entropy'] |
| RF | `n_estimators` | (10,1000) |
| | `max_depth` | (2,20) |
| | `max_features` | ['auto', 'log2'] |
| | `bootstrap` | [True, False] |
| | `criterion` | ['gini', 'entropy'] |
| XGB | `max_depth` | (2,20) |
| | `n_estimators` | (10,1000) |
| | `learning_rate` | [0.01, 0.05, 0.1] |
| LGBM | `max_depth` | (2,20) |
| | `num_leaves` | (10,500) |
| | `min_child_samples` | (2,10) |

scale for this parameters, which includes dropping the 2, 4, 6, or 8 less frequent activities in an event log.

`Bucketing` : When pre-processing an event log for outcome-based prediction, prefixes of each trace are extracted to construct a prefix log. In this paper, we consider prefix-length bucketing, which is concerned with grouping prefixes of the same length. A base strategy (zero-bucketing) groups all prefixes in a single bucket, thus training a single classifier. In prefix length bucketing, though, each bucket contains partial traces of a specific length, and one classifier is trained for each possible prefix length. Bucketing allows to group homogeneous prefixes, which is supposed to improve the performance of the trained models. For instance, if a lossless encoding that translates each event into a fixed of number features is adopted, then bucketing avoids the need to zero-pad prefixes of different length after encoding. Given an input event log, this parameter can assume values comprised between 1 (corresponding to zero-padding) up to two times the mean length of traces in an event log.

`Encoding` : The prefixes extracted from an event log must be numerically encoded to be fed into the model. The problem of encoding prefixes is one of complex symbolic sequence encoding [12] and can be approached in multiple ways. In this paper, we consider *aggregation* and *index-based* encoding. Aggregation is a lossy encoding, which represents entire event sequence attributes into a single entity, for example, based on frequency. Index-based is a lossless encoding that maintains the order of events in a prefix. In index-based encoding, each event in a prefix is encoded into a fixed number of numerical features.

`Model` : This parameter concerns the choice of the classification model to use for developing the predictive model(s). Even though any classification model can be used, the literature highlights that tree-based classifiers show good performance in outcome-based PPM [4]. Thus, in this work we consider four kinds of tree-based, including both individual and ensemble classifiers: Decision Tree (DT), Random Forest (RF), XGBoost (XGB), and LightGBM (LGBM).

Once a `model` is chosen, the hyperparameters of the model must be optimised. This is one of the typical functionalities of AutoML tools. In this work, we combine the optimisation of the model hyperparameters with the PPM-specific parameters mentioned above. Tab. 2 lists the domain of hyperparameters for each classifier that we consider in this work. Although several additional hyperparameters

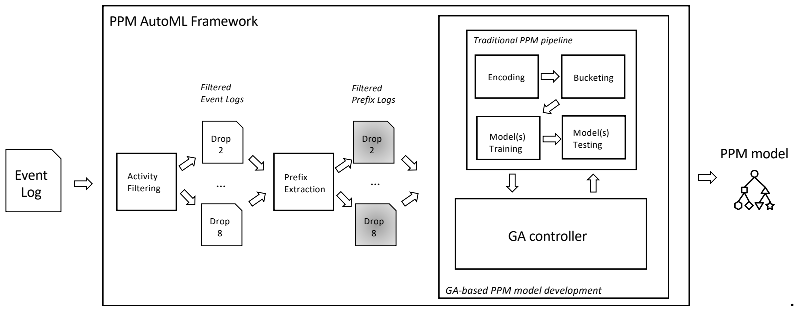

**Fig. 1.** GA-based framework for PPM model optimisation

can be considered for each classifier, in this work we consider a restricted set of hyperparameters that are shown to have significant effect on the model performance in the literature [13, 14, 15]. For the hyperparemters not mentioned in Tab. 2, we use the default settings of the Python implementation (more details about this in Section 5).

After having introduced the PPM model configuration space above, we can now introduce the architecture of the proposed PPM AutoML framework, which is depicted in Fig. 1. We assume that the PPM use case has been defined, so the input of the framework is simply an event log. First, several pre-processed filtered event logs in which the low-frequency activities are dropped are generated, i.e., one for each possible value of `drop_act`. Then, for each filtered event log, the prefixes are extracted for each trace, which yields a set of filtered prefix logs. The filtered prefix logs are the input of the GA-based PPM model development module. This comprises the *GA controller* — implementing the logic of the GA-based optimisation presented in the next section — and a traditional PPM pipeline, which is called by the GA controller to generate new PPM models for given values of the configuration space parameters. The output of the framework is one PPM model, i.e., the highest-performing one identified by the GA-based optimisation.

## 4 GAs for exploring the configuration space

GA was inspired from the Darwinian theory of evolution [16] according to which fitter individuals survive and their genes are passed onto the offspring. In a GA, every individual solution, i.e., a PPM model in our case, corresponds to a chromosome and each parameter represents a gene of a chromosome, which assume a certain value in the configuration space. GA evaluates the fitness of each individual in the population using a fitness function. A selection process is used at each iteration to select the best chromosomes. These then mate to produce an offspring using the crossover operation. In addition, at each iteration several chromosomes are mutated, i.e., their value is randomly changed. Ideally, as generations go on, the fitness value of the offspring increases, until a sufficiently fit individual is identified. We describe next how the typical elements of a GA are customised in our framework.

*Initial population:* The GA algorithm starts with creating an initial population. This population is generated by choosing parameter values randomly within a domain in configuration space. In our framework, the size of the initial population is 20 individuals.

*Evaluate fitness:* In this step, the GA computes the fitness value of each individual in the present population. Fitness is considered as an evaluation metric as well as objective function in GA. Individuals with high fitness value are likely to be selected, mutated and mated with another for crossover. In some simple GA implementations, fitness is defined as a single indicator, such as model accuracy. However, relying on only one metric can provide wrong insights. For example, when the distribution of classes is unbalanced, like in many PPM scenarios [4, 6], the accuracy is not sufficient to evaluate performance. In this context, it is helpful to use multiple measures rather than only one, as they compensate each other. Thus, we designed the fitness $f(i)$ of an individual $i$, i.e., an outcome-based PPM model, to combine different measures as follows:

$$f(i) = \frac{sc(i) + re(i) + tr(i) + se(i)}{4}$$

where:

$$sc(i) = \frac{AUC(i) + acc(i)}{2},$$

$$re(i) = 1 - failure\ rate(i),$$

$$te(i) = \frac{\max(time) - time(i)}{\max(time) - \min(time)}$$

$$se(i) = \frac{sc(i) - \min(sc)}{\max(sc) - \min(sc)}$$

In the formulas above, the score $sc(i)$ combines the average Area Under the receiving operator Curve $AUC$ and the average accuracy $acc$ obtained by the model $i$. AUC is a more balanced performance measure that is often considered in PPM problems.

The reliability $re(i)$ is a measure that computes the overall reliability of the predictions made by an individual $i$ over the test set. A classifier assigns to each observation in a dataset probabilities for each of the outcome labels. The label associated with the highest probability is chosen as the predicted one. When such a highest probability is less than a minimum threshold, we say that the prediction has *failed*, i.e. it is not reliable. In the GA, we compute the failure rate as 1 minus the fraction of observations (running cases) in a dataset for which the prediction failed. The minimum threshold value used in the experiments is 0.7, e.g., a prefix predicted with probability of 0.65 and 0.35 of having a positive or negative label, respectively, is considered a failed prediction.

The time efficiency $te(i)$ represents the relative amount of time required for a an individual to be trained and tested (in respect of the maximum and minimum times observed in the current population). One main purpose of AutoML is in fact to reduce the time for identifying a machine learning model. In this direction, the time efficiency represents how efficient the computation of a chromosome is compared to the other chromosomes in the same population. Similarly to the time efficiency, the score efficiency $se(i)$ represents the relative value of the score of the current individual $i$ in respect of all the other individuals of the current population. This term is introduced to consider also the magnitude of the performance improvement when evaluating the fitness of a new individual $i$.

*Selection:* The main objective of the selection is to give a higher chance of being a parent to the fittest individuals in order to pass on better genes to offsprings. In other words, the higher fitness an individual has, the higher opportunity of selection. In this context, we adopted the *roulette wheel* strategy in our framework, in which the best individual has the largest chance to be selected, while the worst individual has the lowest chance.

*Crossover and Mutation:* Crossover is implemented by selecting a random point (or points) in a chromosome where the exchange of parents' genes happens. The crossover then brings up a new offspring based on the exchange point chosen with particular parts of the parents. Since we consider a limiter number of parameters defining a chromosome, we use the *one-point* crossover, in which only one crossover point along the chromosome is randomly selected.

The purpose of the mutation is to encourage diversity in the population, thus alleviating the local-optima problem in a GA implementation. When mutation is applied, a few genes in the chromosome are randomly changed to produce a new offspring. As a result, this creates new adaptive solutions to avoid local optima. We decided to change one gene for each mutation step.

The GA is thus characterised by the parameter crossover rate $cr$ and the mutation rate $mr$. Both range between 0 and 1. The crossover rate indicates the chance that two chromosomes mate and exchange their genes, so that a new offspring is produced. If $cr = 1$ (100%), all the offspring are obtained applying the crossover. If $cr = 0$ then no mating at all occurs, i.e., a new generation is exactly the same as the previous one. The mutation rate determines how many chromosomes should be mutated in a generation. Setting $mr = 1$ (100%) results in mutating all the chromosomes in a population, while mutation never occurs when $mr = 0$. In the experiments, the values of these two parameters have been set experimentally through grid search (more details in the next section).

*New population:* A new population is generated by selection, crossover, and mutation. If the termination test (see next) is not passed, this population becomes the parent generation for the next population.

*Termination test:* A GA algorithm must stop, returning the best solution found as a result. Therefore, a termination condition is tested for every generation. Three conditions are tested and, if any of this is true, the algorithm stops: (i) the maximum number of iteration is reached, (ii) the number of times in which the average fitness of the new population is lower than the one of the previous

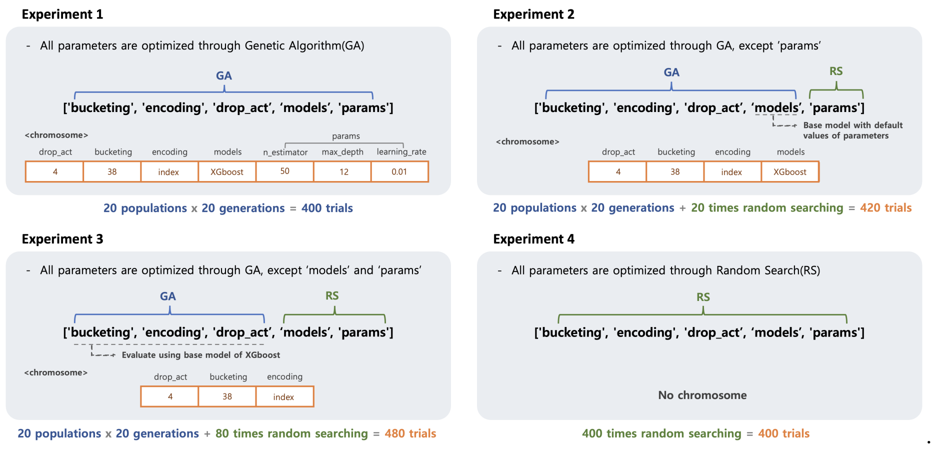
Experiment 1
- All parameters are optimized through Genetic Algorithm(GA)
GA
['bucketing', 'encoding', 'drop_act', 'models', 'params']
<chromosome>
drop_act bucketing encoding models params n_estimator max_depth learning_rate
4 38 index XGboost 50 12 0.01
20 populations x 20 generations = 400 trials
Experiment 2
- All parameters are optimized through GA, except 'params'
GA RS
['bucketing', 'encoding', 'drop_act', 'models', 'params']
Base model with default values of parameters
<chromosome>
drop_act bucketing encoding models
4 38 index XGboost
20 populations x 20 generations + 20 times random searching = 420 trials
Experiment 3
- All parameters are optimized through GA, except 'models' and 'params'
GA RS
['bucketing', 'encoding', 'drop_act', 'models', 'params']
Evaluate using base model of XGboost
<chromosome>
drop_act bucketing encoding
4 38 index
20 populations x 20 generations + 80 times random searching = 480 trials
Experiment 4
- All parameters are optimized through Random Search(RS)
RS
['bucketing', 'encoding', 'drop_act', 'models', 'params']
No chromosome
400 times random searching = 400 trials

**Fig. 2.** Experiment configurations: illustration

population exceeds a certain limit (5 in the experiments), and (iii) the difference between the average fitness of the new population and the last one is less than 0.001. Note that (i) guarantees that the GA algorithms eventually stops.

### 4.1 Experiment configurations

We designed four experiment configurations based on different ways of exploring the configuration space using GAs (see Fig. 2). In experiment 1, all the parameters in the configuration space are expressed by genes of the chromosomes and optimized using GA. In experiment 2, the hyperparameter values of the model are not part of the GA-based optimisation. First, the GA is run considering default parameter values for each model. Then, the hyperparameters of the model selected by the best individual using GA are optmised using random search. In experiment 3, only the PPM-specific parameters `bucketing`, `encoding` and `drop_act` are optimised using GA, considering XGB as the model with default hyperparmeter values. Then the model to be used and its hyperparameters are selected using random search. The fourth experiment is a totally random search (RS) baseline, in which all the values of all the parameters are optimised using random search. As can be seen in Fig. 2, all the experiments are configured to generate between 400 and 500 trials, i.e., models to train and test.

## 5 Experimental evaluation

First, we discuss the experimental settings (datasets, GA parameter settings, implementation details) and then we present the experimental results. The framework is implemented in Python and the code to reproduce the experiments is publicly available at `https://github.com/eekfskgus/GA_based_AutoML/`.

We consider 2 event logs publicly available at `https://data.4tu.nl/` published by the Business Process Intelligence Challenge in 2012 and 2017. The BPIC 2012 and BPIC 2017 event logs are from a process of managing loan requests at a Dutch financial institution. These logs have been chosen because they contain an outcome label and have been used by previous research on outcome-based process predictive monitoring. In the BPIC 2012 and BPIC 2017 event logs the outcome label captures whether a loan request is eventually accepted or not.

The design of GAs requires to set the values of several parameters. The value of the GA parameters can impact greatly on the solution found, even determining whether a solution is found at all by the algorithm [17].

**Table 3.** Grid search test for GA parameter setting

| parameter | best score | elapsed time(s) |
|---|---|---|
| cr = 0.9, mr = 0.1 | 0.76 | 4611 |
| cr = 0.9, mr = 0.05 | 0.75 | 4538 |
| cr = 0.9, mr = 0.01 | **0.82** | **3848** |
| cr = 0.8, mr = 0.1 | 0.75 | 3947 |
| cr = 0.8, mr = 0.05 | 0.71 | 4486 |
| cr = 0.8, mr = 0.01 | 0.76 | 4233 |
| cr = 0.7, mr = 0.1 | 0.77 | 4468 |
| cr = 0.7, mr = 0.05 | 0.73 | 5424 |
| cr = 0.7, mr = 0.01 | 0.7 | 4554 |

To find the best value of $cr$ and $mr$, we conducted a grid search experiment using the BPIC 2012 dataset, in which $cr \in [0.9, 0.8, 0.7]$ and $mr \in [0.1, 0.05, 0.01]$. It is known that high crossover rate and low mutation rate effectively works in GA, since the low crossover rates lead to low rates of exploration, whereas high mutation rates increase the randomness of the search [18, 19]. For every combination, we evaluated the best individual found and the elapsed time using the experiment 1 configuration. The results of this test are shown in Tab. 3 (the selected parameter values are in bold). For the other GA parameters, the initial population size is 20, with 5 individuals randomly generated for each of the 4 classification models considered. For the termination condition, the maximum number of iteration is 20.

For the training and testing of new individuals in a generation, the (training:test) ratio is set to (4:1). In addition, if the imbalance ratio of the minority class over the majority class is less than 0.33, then the dataset is automatically re-sampled using synthetic minority over-sampling, widely known as SMOTE. SMOTE sampling could lead to benefit the performance of classification in class imbalance problem, by improving class boundary region especially with extremely imbalanced datasets.

Tab. 4 compares the best solution obtained by the four types of experiments. Given the randomness intrinsic to the experiments, for each type of experiment we show the results of three different runs. We compare the execution time, the accuracy-AUC-based score ($sc$), and the values of the parameters of the configuration space. The proposed GA-based framework (adopted in experiments 1, 2, and 3) generally outscore the RS (experiment 4), on both execution time and quality of the solution (score). Interestingly, the classifiers XGB and LGBM

**Table 4.** Best solutions and corresponding parameter values

| | | experiment 1 | | | experiment 2 | | | experiment 3 | | | experiment 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | run 1 | run 2 | run 3 | run 1 | run 2 | run 3 | run 1 | run 2 | run 3 | run 1 | run 2 | run 3 |
| BPIC2012 | time(s) | 93532 | 95454 | 83139 | 84823 | 113432 | 51252 | 157937 | 123628 | 82579 | 113952 | 119033 | 117269 |
| | sc | 0.83 | 0.74 | 0.96 | 0.94 | 0.78 | 0.89 | 0.78 | 0.92 | 0.99 | 0.84 | 0.82 | 0.84 |
| | model | XGB | DT | RF | LGBM | RF | LGBM | LGBM | LGBM | XGB | RF | RF | RF |
| | drop_act | 8 | 8 | 4 | 4 | 6 | 2 | 8 | 4 | 8 | 6 | 8 | 6 |
| | bucketing | 30 | 29 | 13 | 22 | 34 | 26 | 39 | 23 | 6 | 2 | 2 | 1 |
| | encoding | index | index | index | index | index | index | index | index | aggregate | aggregate | index | index |
| BPIC2017 | time(s) | 77644 | 83140 | 101942 | 80839 | 54720 | 76564 | 102004 | 96954 | 109210 | 119351 | 93833 | 165861 |
| | sc | 0.83 | 0.96 | 0.96 | 0.94 | 0.84 | 0.73 | 0.76 | 0.99 | 0.96 | 0.75 | 0.74 | 0.78 |
| | model | RF | RF | XGB | LGBM | XGB | RF | LGBM | XGB | LGBM | DT | RF | DT |
| | drop_act | 6 | 4 | 6 | 8 | 8 | 8 | 4 | 4 | 4 | 6 | 8 | 8 |
| | bucketing | 25 | 13 | 6 | 7 | 28 | 34 | 37 | 18 | 9 | 1 | 1 | 3 |
| | encoding | index | index | aggregate | index | index | index | index | index | index | index | index | index |

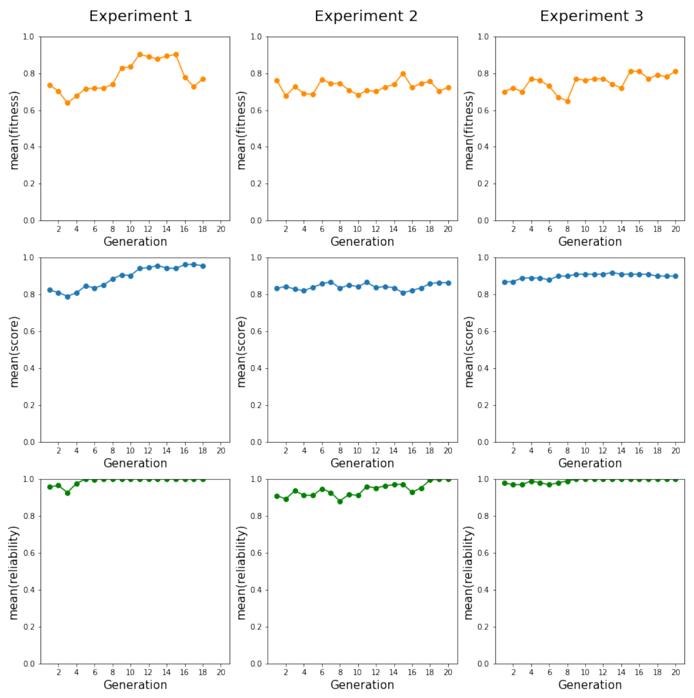

**Fig. 3.** Mean parameters values over generations in the GA-based experiments (BPIC 2017 event log)

(especially the latter) are frequently selected in the experiments that use the proposed framework, whereas RF or DT are often selected by the RS experiment.

The RS baseline in experiment 4 selects the best individual from 400 samples obtained using parameter values randomly selected. These 400 individuals are independent of each other, i.e., their selection is not affected by the constraints on fitness, execution time and failure rate of the proposed GA-based framework.

Therefore, a random search of the configuration space could work better for individual classifiers (like DT) or bagging-based classifiers (like RF), which do not try to improve iteratively the performance of the model. Another difference between XGB and LGBM when compared with DT and RF is that they use boosting. Boosting involves iterations, whereby the prediction results of a previous model affects the results of the next one. Based on the results shown in Tab. 4, the overlapping effect of boosting over generations improves the GA performance. In addition, LGBM is selected more frequently than XGB because of its superiority in terms of execution time. Being lightweight on execution time, LGBM is likely to lead to higher fitness of the solution found in a shorter time.

Regarding the other parameters, index encoding is more dominant in the solutions found than aggregation encoding. It appears also that dropping a higher number of infrequent activities leads to better results. Finally, Tab. 4 shows that the bucket size of the best chromosome tends to be higher when using the proposed framework when compared to the RS baseline.

To show the inner dynamic of the GA-based experiments, Fig. 3 shows the mean values across experiments 1, 2, and 3 of the three parameters fitness, score, and reliability over generations for the BPIC 2017 dataset. We can see that the value of each parameter tend to converge to 1 as the number of generation increases, which shows the suitability of the GA-based approach as an optimisation strategy for identifying a PPM model.

## 6 Conclusions

This paper has presented an AutoML framework for identifying a high-performing PPM model. The framework relies on genetic algorithms for exploring a solution space that includes both traditional and PPM-specific model hyperparameters. In the future we plan to extend the configuration space to more dimensions and use cases, e.g., next event prediction, and to compare the proposed GA-based approach with other bio-inspired heuristics, e.g. swarm or particle intelligence.

## References

1. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. Process Mining Handbook. LNBIP **448** (2022) 320–346
2. Yao, Q., Wang, M., Chen, Y., Dai, W., Li, Y.F., Tu, W.W., Yang, Q., Yu, Y.: Taking human out of learning applications: A survey on automated machine learning. arXiv preprint arXiv:1810.13306 (2018)
3. Karmaker, S.K., Hassan, M.M., Smith, M.J., Xu, L., Zhai, C., Veeramachaneni, K.: Automl to date and beyond: Challenges and opportunities. ACM Computing Surveys (CSUR) **54**(8) (2021) 1–36
4. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: review and benchmark. ACM Transactions on Knowledge Discovery from Data (TKDD) **13**(2) (2019) 1–57

5. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Teinemaa, I.: Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. ACM Transactions on Intelligent Systems and Technology (TIST) **10**(4) (2019) 1–34
6. Tama, B.A., Comuzzi, M.: An empirical comparison of classification techniques for next event prediction using business process event logs. Expert Systems with Applications **129** (2019) 233–245
7. Rizzi, W., Simonetto, L., Di Francescomarino, C., Ghidini, C., Kasekamp, T., Maggi, F.M.: Nirdizati 2.0: New features and redesigned backend. Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 **2420** (2019) 154–158
8. Di Francescomarino, C., Dumas, M., Federici, M., Ghidini, C., Maggi, F.M., Rizzi, W., Simonetto, L.: Genetic algorithms for hyperparameter optimization in predictive business process monitoring. Information Systems **74** (2018) 67–83
9. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Transactions on Services Computing **11**(6) (2017) 962–977
10. Rama-Maneiro, E., Vidal, J., Lama, M.: Deep learning for predictive business process monitoring: Review and benchmark. IEEE Transactions on Services Computing (2021)
11. Tax, N., Sidorova, N., van der Aalst, W.M.: Discovering more precise process models from event logs by filtering out chaotic activities. Journal of Intelligent Information Systems **52**(1) (2019) 107–139
12. Leontjeva, A., Conforti, R., Francescomarino, C.D., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: International Conference on Business Process Management, Springer (2016) 297–313
13. Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., Cox, D.D.: Hyperopt: a python library for model selection and hyperparameter optimization. Computational Science & Discovery **8**(1) (2015) 014008
14. Liang, W., Luo, S., Zhao, G., Wu, H.: Predicting hard rock pillar stability using gbdt, xgboost, and lightgbm algorithms. Mathematics **8**(5) (2020) 765
15. Kelkar, K.M., Bakal, J.: Hyper parameter tuning of random forest algorithm for affective learning system. In: 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), IEEE (2020) 1192–1195
16. Holland, J.H.: Genetic algorithms. Scientific american **267**(1) (1992) 66–73
17. Mills, K.L., Filliben, J.J., Haines, A.: Determining relative importance and effective settings for genetic algorithm control parameters. Evolutionary computation **23**(2) (2015) 309–342
18. Chiroma, H., Abdulkareem, S., Abubakar, A., Zeki, A., Gital, A.Y., Usman, M.J.: Correlation study of genetic algorithm operators: crossover and mutation probabilities. In: Proceedings of the International Symposium on Mathematical Sciences and Computing Research. (2013) 6–7
19. Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A., Prasath, V.S.: Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. Information **10**(12) (2019) 390