AutoOC: A Python Module for Automated Multi-objective One-Class Classification

Luís Ferreira*[1], Paulo Cortez[1]
[1] – **ALGORITMI/LASI, Department of Information Systems, University of Minho, Guimaraes, Portugal**
* – **correponding author (luis.ferreira@dsi.uminho.pt)**

Abstract
AutoOC is an open-source Python module to efficiently automate the selection and hyperparameter tuning of quality OCC (One-Class Classification) learners. By using a GE (Grammatical Evolution) approach, AutoOC searches for five base learners, namely IF (Isolation Forest), LOF (Local Outlier Factor), OC-SVM (One-Class SVM), AE (Autoencoder), and VAE (Variational Autoencoder). The module provides a multi-objective search, where predictive performance and computational efficiency are simultaneously optimized. By providing a simple set of functions, AutoOC allows the user to easily generate OCC models for a dataset, being well-suited for anomaly detection tasks, where most of the data is composed of normal records.

Keywords
Automated Machine Learning
Deep Autoencoders
Grammatical Evolution
Multi-objective Optimization
One-Class Classification
Python.

Code metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.0.14
C2	Permanent link to code/repository used for this code version	https://github.com/luisferreira97/AutoOC
C3	Permanent link to Reproducible Capsule	https://codeocean.com/capsule/7689106/tree/v3
C4	Legal Code License	MIT License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python 3.8; graphviz
C7	Compilation requirements, operating environments & dependencies	AutoOC requires keras; matplotlib; mlflow; pandas; pydot; pygmo; numpy; scikit-learn; tensorflow; and tqdm
C8	If available Link to developer documentation/manual	https://github.com/luisferreira97/AutoOC/blob/main/README.md
C9	Support email for questions	luis.ferreira@dsi.uminho.pt

1. AutoOC: Automated One-Class Classification

While the area of AutoML is currently a hot topic, most of the current research works and software that tackles AutoML are focused on a supervised learning [1]. Moreover, often these works do not consider efficiency during the optimization, relying on random or grid search approaches [6].

In this paper, we present the AutoOC open-source Python module, an automated and computationally efficient GE (Grammatical Evolution) approach that optimizes the hyperparameters of OCC (One-Class Classification) learners. GE is a biologically inspired evolutionary algorithm for generating computer programs, widely used in both optimization and ML tasks [8]. In GE, the programs are defined using a formal grammar, which defines its syntax and structure. The grammar is used to generate solutions, which are then evaluated using a fitness function. The fitness function measures the quality of the programs and is used to guide the evolution process toward better solutions. AutoOC includes a multi-objective mode, optimizing both the OCC predictive performance and an efficiency measure (e.g., training time). By using GE, a biologically inspired evolutionary algorithm, AutoOC uses a formal grammar to select and tune the hyperparameters of five OCC base learners, namely IF (Isolation Forest), LOF (Local Outlier Factor), OC-SVM (One-Class SVM), AE (Autoencoder), and VAE (Variational Autoencoder). AutoOC also applies the NSGA-II algorithm to perform a multi-objective optimization. NSGA-II was proposed in 2002 [2] and is based on the concept of non-dominance, where the goal is to find a set of non-dominated solutions which represents the trade-off between the different objectives (known as the Pareto front). AutoOC uses NSGA-II to maximize the predictive performance of the OCC learners while minimizing their training time or other of the provided efficiency objectives (e.g., prediction time). Figure 1 shows a high-level overview of the AutoOC method.

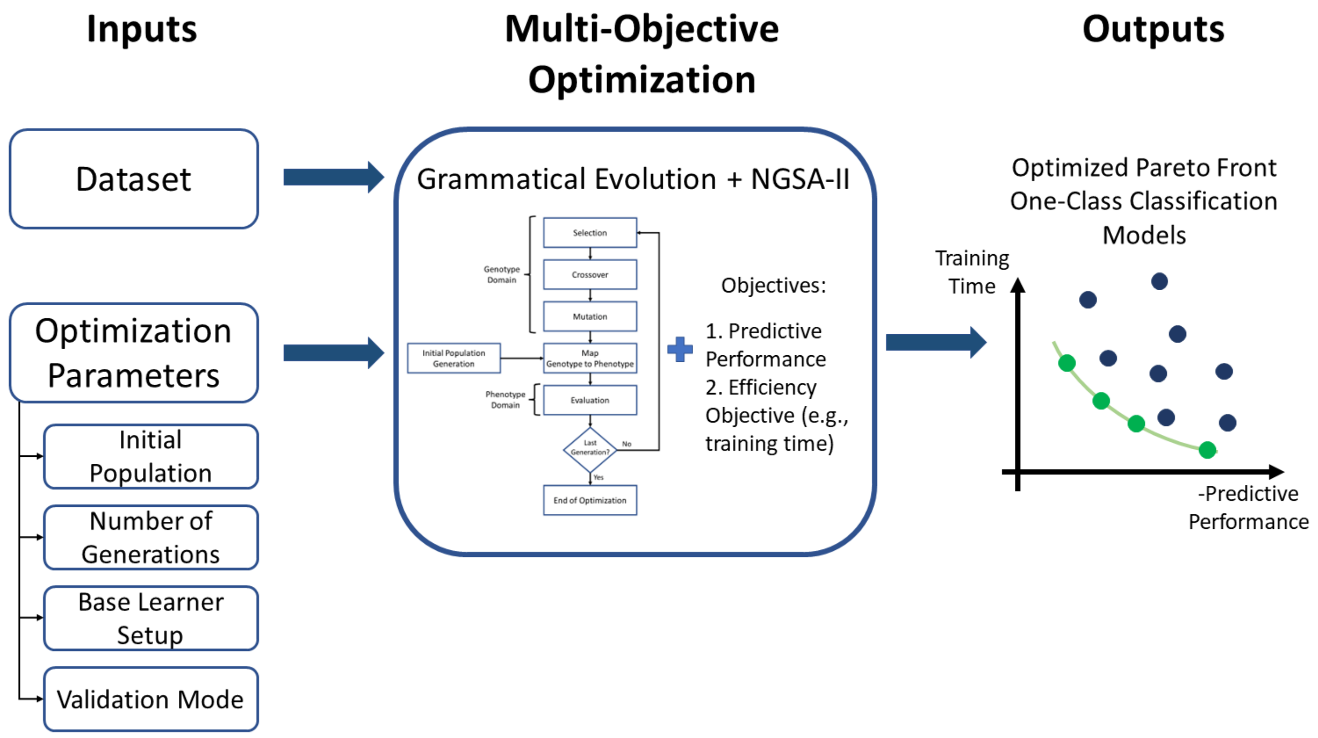

Figure 1: High-level overview of the AutoOC tool.

By instantiating its main Python class, AutoOC provides a set of functions to easily find the best and most efficient OCC models for a given dataset. The main steps carried out by AutoOC are problem formulation, data loading, model optimization, test set predictions, and model evaluation.

1.1 Problem definition

The first step when using AutoOC is to provide information about the dataset and ML problem context (e.g., performance metrics). The possible options for problem definition are:

- Class definition: the adopted dataset encoding for the “anomaly” and “normal” classes (`anomaly_class` and `normal_class` parameters).

- Algorithm: which OCC learners can be created during the AutoOC optimizations. Even though there is currently a fixed number of setups, it is possible to easily add new OCC learners or use any combination of the existing algorithms. The current learner setups (as of AutoOC version 0.0.13) are:

  - `autoencoders`: uses Deep AE, from the TensorFlow library [13].
  - `var`: uses VAE, from the TensorFlow library [14].
  - `iforest`: uses IF, from the Scikit-Learn library [10].
  - `lof`: uses LOF, from the Scikit-Learn library [11].
  - `svm`: uses OC-SVM, from the Scikit-Learn library [12].
  - `nas`: uses both AEs and VAEs, working as a NAS approach.
  - `all`: uses all five OCC base learners.

- Multi-objective optimization: using the `multiobjective` boolean parameter, it is possible to choose a single-objective or multi-objective optimization. In a single-objective mode, only the predictive performance will be optimized (the adopted metric will depend on the usage of labeled or unlabeled validation data); for the multi-objective mode, an additional objective will be optimized, related to computational efficiency.

- Efficiency metric: when choosing the multi-objective optimization mode, it is possible to select which measure will be used as the efficiency objective. The possible options for the `performance_metric` parameter are:

  1. Training time: minimizes the training time of the individuals (OCC models).
  2. Predict time: minimizes the time it takes to predict one record from the validation data.
  3. Number of parameters: minimizes the number of parameters of the ANN model, given by Keras `count_params()` function. Only available when using Deep Learning (DL) OCC learners (algorithm setups `autoencoders`, `var`, or `nas`).
  4. BIC: minimizes the value of the BIC (also known as Schwarz Information Criterion), a criterion used for model selection of a finite set of models [9].

- Multicore: when set to True, the optimization will run in a multicore manner, using all available processors. When set to False, AutoOC will only use a single core.

1.2 Data loading

AutoOC is designed to optimize OCC learners, where the training data is composed only of "normal" records. Depending on the type of the provided validation data, the AutoOC optimization can work with two validation manners: unsupervised validation, where the model performance is evaluated using only unlabeled data (e.g., through an anomaly score); or supervised validation, where there is access to a labeled validation set to assess the model performance using supervised learning metrics, such as the AUC of the ROC curve classification measure [3]. Table 1 summarizes the type of data used for each AutoOC validation setup.

Table 1: Required data for each AutoOC validation mode. Adapted from [4].

Validation Mode	**Training Data**	**Validation Data**	**Test Data**
Supervised	Unlabeled Data (`X_train`)	Labeled Data (`X_val`, `y_val`)	Labeled Data (`X_test`, `y_test`)
Unsupervised	Unlabeled Data (`X_train`)	Unlabeled Data (`X_val`)	Labeled Data (`X_test`, `y_test`)

AutoOC provides the method `load_example_data()` to load the popular ECG dataset, provided by the Google API[1]. This method returns a dataset already preprocessed and split into training, validation, and test data that can be used to run AutoOC optimization.

[1]http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv

1.3 Model optimization

The execution of the AutoOC GE optimization is performed by the `fit()` function. This function accepts the following parameters:

- Supervised or unsupervised validation: the `fit()` function requires the user to provide the training data (`X_train`) and validation data. If the user only provides the validation set features (`X_val`), AutoOC will adopt the unsupervised mode; when the user provides a labeled set of targets (`y_val`), the supervised mode will be adopted by the optimization.
- Population size: the `pop` parameter allows the definition of the GE initial population size (defaults to 100). In general terms, the population size represents the number of individuals (or candidate solutions) that will be created in each generation. The higher the value of `pop`, the more OCC models will be trained in each generation. For example, a `pop` equal to 100 means that AutoOC will optimize 100 OCC models in each GE generation.
- Number of generations: the `gen` parameter allows the definition of the GE number of generations (defaults to 100). A generation represents an iteration of the GE, where the individuals are replaced by new solutions (also named offspring). The higher the number of generations, the more iterations there will be during AutoOC optimization. For example, if `pop` is equal to 100 and generations equal 50, it means that the GE optimization will run during 50 iterations, training 100 OCC in each one (total of 5,000 models).
- Epochs: internal Keras parameter, representing the maximum number of epochs that each model based on DL learners (AE or VAE) will be trained on (defaults to 100).
- Early stopping: AutoOC provides two parameters related to early stopping of the GE optimization. The `early_stopping_tolerance` allows the definition of the tolerance value t used for early stopping (defaults to 0.01). The `early_stopping_rounds` parameter, when set to an integer value r, will stop the optimization if the performance does not improve by value t after r consecutive GE generations (defaults to False). We note that, for single-objective optimization, the early stopping value will be based on the chosen predictive metric (e.g., AUC); for multi-objective, the performance will be based on the hypervolume.
- "Always at hand": The boolean parameter `always_at_hand`, when set to True, keeps the current best OCC model (or the best Pareto front models for multi-objective optimization) during the execution of the GE optimization. This feature allows the usage of the current best OCC models, even if the GE optimization is still running. Defaults to False.
- Results path: all the metadata related to each AutoOC run is exported to the path related to parameter `results_path`. The metadata that is saved includes all the individual models generated during the optimization (all generations), leaderboards files including the evaluation of each generated solution, PDF reports with the evolution of predictive performance and efficiency across generations, and other metadata associated with the OCC models (e.g., images with the AEs and VAEs architectures).
- MLFlow integration: AutoOC is integrated with MLFlow [7], an open-source solution for experiment tracking and registry of ML models. AutoOC provides three parameters, to personalize the tracking URI (`mlflow_tracking_uri`), experiment name (`mlflow_experiment_name`), and run name (`mlflow_run_name`).

1.4 Test set predictions

The AutoOC `predict()` function uses the generated execution results to predict the labels on test data. The only required parameter is the data containing the test inputs (`X_test`). The optional `mode` parameter can be changed to select which individuals from the last generation are used to predict. The default value for this parameter is "all", which uses all individuals (OCC models) from the last GE generation; "best" – uses the model from the last generation that achieved the best predictive performance metric (e.g., highest validation AUC); "simplest" – uses the model from the last generation with the best efficiency metrics (e.g., lowest training time), only available when using multi-objective optimization; "pareto" – uses the non-dominated solutions (Pareto front) from the last generation (these are the models that achieved simultaneously the best predictive metric and efficiency metric), only for multi-objective optimization.

Additionally, the `threshold` parameter (only used for AEs and VAEs) can be used to set the threshold used for the prediction. The possible values for this parameter are: "mean": for each individual (DL model), the threshold value is the sum of the mean reconstruction error obtained on the validation data and one standard deviation (this is the default value for the `threshold` parameter); "percentile": for each model, the threshold value is the 95th percentile of the reconstruction error obtained on the validation data (there is an additional `percentile` parameter to change the percentile); "max": for each model, the threshold value is maximum reconstruction error obtained on the validation data; it is also possible to use a fixed value for all models, by passing an Integer or Float value to the `threshold` parameter. In this case, the threshold value will be the same for all the models.

1.5 Model evaluation

After making the predictions with the `predict()` function it is possible to manually calculate performance measures (e.g., AUC, accuracy). However, AutoOC provides the `evaluate()` function as a more convenient way to do it. It is possible to use the `mode` and `threshold` parameters (similarly to the `predict()` function) to directly predict and evaluate the performance of the AutoOC execution best models. Currently, the `evaluate()` function supports five predictive metrics from the Scikit-Learn library: "roc_auc", "accuracy", "precision", "recall", and "f1".

1.6 Code example

A full example using the AutoOC Python package is shown below[2].

```
from autooc.autooc import AutoOC

# Define the problem
aoc = AutoOC(anomaly_class=0,
    normal_class=1,
    multiobjective=True,
    performance_metric=``training_time'',
    algorithm=``autoencoder'')

# Load the Data
X_train, X_val, X_test, y_test = aoc.load_example_data()

# Run AutoOC optimization
run = aoc.fit(
    X=X_train,
    X_val=X_val,
    pop=3,
    gen=3,
    epochs=100,
    mlflow_tracking_uri=``../results'',
    mlflow_experiment_name=``codeocean_experiment'',
    mlflow_run_name=``codeocean_run'',
    results_path=``../results''
)

# Make Predictions
predictions = aoc.predict(X_test, mode=``all'', threshold=``default"")
```

[2]This example code is also presented at the CodeOcean reproducible capsule: https://codeocean.com/capsule/7689106/tree/v3

```
# Evaluate Predictions
score = aoc.evaluate(X_test,
    y_test,
    mode=''all'',
    metric=''roc_auc'',
    threshold=''default'')
print(f''Scores: {score}'')
```

2. Impact on academic research

Two versions of AutoOC have been used in previous research works, namely two journal articles related to ML applications. Additionally, AutoOC has been published as a Python module[3] in June 2023, having more than 2,700 downloads[4].

In [5], a preliminary version of the software (named AutoOneClass) was proposed, associated with an Industry 4.0 PdM project. The goal was to predict the number of days until the next failure of an equipment and also determine if the equipments will fail in a fixed amount of days. In this first version, AutoOneClass adopted three OCC learners (AEs, IF, and OC-SVM) and it was applied to a recently collected dataset from a Portuguese software company. The results from AutoOneClass were compared with ten recent open-source AutoML technologies focused on a Supervised Learning and with two manual ML approaches. The AutoOneClass proposed method revealed competitive results, especially when compared with gloing.

In [4], a second version of the software (named AutoOC) solution was developed and a robust benchmark with eight public OpenML datasets was performed to assess the performance of several scenarios, all focused on a multi-objective optimization. In this new work, AutoOC already considered five distinct OCC base learners (AEs, IF, LOF, OC-SVM, and VAEs) and included includes two execution speedup mechanisms: a periodic training sampling and a multi-core fitness evaluation. AutoOC provided predictive results with high quality, outperforming a baseline IF for all the studied datasets and surpassing the best supervised public human modeling for two datasets.

3. Future work

AutoOC is a method aimed at automating the creation of OCC ML models by using multi-objective optimization based on a GE. It is worth noting that the current version of AutoOC already implements new features that were not available in the two previous versions that were used in the research works detailed in Section 2 (e.g., distinct efficiency objectives, early stopping). Nevertheless, in the future, we aim to further extend the AutoOC capabilities by providing additional features. First, we intend to enhance the efficiency capabilities of AutoOC, by using distributed data preprocessing (e.g., Apache Spark), or allowing the usage of GPUs to preprocess the data and train the models. Also, we intend to add even more efficiency objectives, such as model size, or extend the number of parameters objective to the non-DL algorithms (e.g., as a "model complexity" metric). Finally, we aim to explore AutoML technologies to automate the phases of feature engineering and feature selection.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We wish to thank the anonymous reviewers for their helpful comments.

[3]https://pypi.org/project/autooc
[4]https://pepy.tech/project/autooc

References

[1] Maroua Bahri, Flavia Salutari, Andrian Putina, and Mauro Sozio. AutoML: State of the Art with a Focus on Anomaly Detection, Challenges, and Research Directions. *International Journal of Data Science and Analytics*, 14(2):113–126, 2022. doi:10.1007/s41060-022-00309-0.

[2] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi:10.1109/4235.996017.

[3] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. doi:10.1016/j.patrec.2005.10.010.

[4] Luís Ferreira and Paulo Cortez. AutoOC: Automated Multi-objective Design of Deep Autoencoders and One-class Classifiers Using Grammatical Evolution. *Applied Soft Computing*, 144:110496, 2023. doi:10.1016/j.asoc.2023.110496.

[5] Luís Ferreira, André Pilastri, Filipe Romano, and Paulo Cortez. Using Supervised and One-Class Automated Machine Learning for Predictive Maintenance. *Applied Soft Computing*, 131:109820, 2022. doi:10.1016/j.asoc.2022.109820.

[6] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowl. Based Syst.*, 212:106622, 2021. doi:10.1016/j.knosys.2020.106622.

[7] MLFlow. MLflow - A platform for the machine learning lifecycle, 2023. URL: https://mlflow.org/.

[8] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001. doi:10.1109/4235.942529.

[9] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978. doi:10.1214/aos/1176344136.

[10] Scikit-Learn. Isolation Forest, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html.

[11] Scikit-Learn. Local Outlier Factor, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html.

[12] Scikit-Learn. One-Class SVM, 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[13] TensorFlow. Convolutional Variational Autoencoder, 2023. URL: https://www.tensorflow.org/tutorials/generative/cvae.

[14] TensorFlow. Intro to Autoencoders, 2023. URL: https://www.tensorflow.org/tutorials/generative/autoencoder.