An automated approach for binary classification on imbalanced data

Pedro Marques Vieira (pedro.pmv22@gmail.com)
Polytechnic Institute of Porto
Fátima Rodrigues
Polytechnic Institute of Porto

Research Article

Keywords: Imbalanced Classification, Resampling, Meta-learning, Automated Machine Learning

Posted Date: June 6th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3015970/v1

Version of Record: A version of this preprint was published at Knowledge and Information Systems on January 12th, 2024. See the published version at https://doi.org/10.1007/s10115-023-02046-7.

An automated approach for binary classification on imbalanced data

Pedro Marques Vieira[1*] and Fátima Rodrigues[1,2*]

[1]ISEP, Polytechnic Institute of Porto, Rua Dr. António Bernardino de Almeida, Porto, 4249-015, Porto, Portugal.
[2]ISRC, Interdisciplinary Studies Research Center.

*Corresponding author(s). E-mail(s): pedro.pmv22@gmail.com; mfc@isep.ipp.pt;

Abstract

Imbalanced data is present in various business areas and must be dealt with the appropriate resampling techniques and classification algorithms. However, there is a magnitude of multiple combinations of resampling and learning methods to handle imbalanced data that require specialised knowledge to be used correctly. In this paper, several approaches, ranging from more accessible and more advanced in the domains of data resampling and cost-sensitive techniques, will be considered to handle imbalanced data. The application developed delivers recommendations of the most suited combinations of techniques for a specific dataset, by extracting and comparing dataset meta-features values recorded in a knowledge base. It facilitates effortless classification and automates part of the machine learning pipeline with comparable or better results to a state-of-the-art solution and with a much smaller execution time.

Keywords: Imbalanced Classification, Resampling, Meta-learning, Automated Machine Learning

1 Introduction

Several current real-world datasets are imbalanced by nature, in that they have one or some classes underrepresented compared to the other class or classes.

The class imbalance problem arises in multiple areas, including telecommunication, bioinformatics, fraud detection, and medical diagnosis. The best approach to handle imbalanced data highly depends on the nature of the data. The methods and combination of methods proposed are abundant in various conceivable outcomes, and most times they require specialised knowledge to be used correctly.

As such, this project focuses on an open-ended current problem associated with machine learning tasks, being a new proposal to automate imbalanced classification, applied to different case study solutions.

Classification algorithms for imbalance scenarios applied without proper data resampling or a cost-sensitive approach, for instance, tend to perform better for well-represented classes and worse for underrepresented classes. In these cases, the underrepresented class tends to be the class with more interest to predict. Multiple strategies have been proposed to address class imbalance problems. However, there is no general guidance on when to use each technique.

In addition, combining different data resampling techniques, classification algorithms and multiple hyperparameter optimisation makes the possibilities to evaluate the desired solution endless. Thus, a solution to automate and facilitate these imbalanced classification tasks is needed, hence, to get better and faster results.

This project aims to develop a system to automatically prepare an imbalanced dataset to be used by a classifier. To accomplish that, this project includes a review of the state of the art on related solutions, an implementation of the most promising balance techniques and testing different combinations of them in several public datasets, using different classification algorithms. The best combination of the balance technique, with the best-performing classification algorithm and the appropriate meta-features values of the dataset, are recorded in a knowledge base to be recommended for new datasets.

The remainder of this paper is organised as follows. Section 2 reviews and discusses existing solutions for imbalanced classification. The developed solution that includes a learning module and a recommendation module is described in Section 3. In the learning module, it is presented the criteria for datasets selection to be used in the development of the solution, the meta-features extracted from the selected datasets, the resampling and classification algorithms used, and it is also explained the process of selection of the best combinations of resampling and classification algorithms to be considered in the learning module. The recommendation module describes the selection process of the best resampling and classification recommendations for a specific dataset. Section 4 presents an internal and external evaluation of the recommendation module. The internal evaluation compares the recommendation module with the best resampling and classification algorithms obtained with the learning module. The external evaluation compares the recommendation module results with a *TPOT* pipeline. The main conclusions and prospects of future work are disclosed in the final section.

2 State of the Art

There are several applications and services, capable of providing tools that can handle imbalanced classification. For instance, *Scikit-Learn* [1] is a general purpose machine learning *Python* library, which provides data preparation, machine learning algorithms, and model evaluation schemes and although not designed around imbalanced classification, it provides some useful tools for handling imbalanced datasets also. One *Python* library that directly addresses imbalanced classification is *imbalanced-learn* [2] which is related to *Scikit-Learn* and implements most of the necessary techniques. For *R* programming language, there exists the *ROSE* [3] and imbalanced [4] libraries, among various others, also specialised in imbalanced classification.

Numerous libraries automatically permit the creation of a predictive system with few steps capable of doing classification, even for imbalanced scenarios, with various results. For open-source software libraries ready to use when coding, the first concern to note is that most of them focus only on some parts of the automated machine learning (autoML) pipeline [5]. For instance, *Auto-sklearn* [6] is built on top of *Scikit-Learn* and formulated as a *CASH* problem capable of automatically trying different classifiers and hyperparameters, however, it only searches for traditional machine learning models [7]. *Auto-sklearn* does an automatic ensemble of the different models searched and applies a post-processing method, instead of discarding all the models searched [7]. It can do parallelisation on a single computer or in a cluster on a limited time budget [8].

Additionally, *AutoKeras* [9] based on *Keras* [10], supports multi-modal and multitask by searching for deep learning models [5]. *Neural Network Intelligence* (NNI) [11] developed by *Microsoft*, also integrates *Scikit-Learn* features that can automate feature engineering, hyperparameter optimisation, and *Neural Architecture Search*, becoming a powerful and lightweight toolkit for autoML. *TPOT* (Tree-based Pipeline Optimisation Tool) is an autoML tool specifically designed to efficiently construct optimal pipelines through genetic programming. *TPOT* is an open-source library and makes use of *Scikit-learn* components for data transformation, feature decomposition, feature selection and model selection [12].

In addition, there is also *Hyperopt-sklearn* [13] which supports various classifiers of *Scikit-Learn* and provides a fixed pipeline structure, to one classification algorithm to each processor, by adding a configuration space definition [8]. Finally, there is also *H2O AutoML* [14] that, instead of being built on *Python*, is programmed in *Java*, thus not using the *Scikit-Learn* library. It is able, without pre-processing, to select and tune each classification algorithm by a fixed order and create a final ensemble of them like *Auto-sklearn* [8]. Also, many big tech companies like *Microsoft*, *Amazon* and *Google* provide autoML services, such as *Microsoft Azure Automated Machine Learning* [15], *Amazon Web Services* (AWS) *SageMaker Autopilot* [16] and *Google Cloud Platform* (GCP) *AutoML* [17], correspondingly. All these services provide autoML tools by interacting on the website and without needing to code the implementation.

For *R* programming language, there is the proposed *Automated Imbalanced Classification* (ATOMIC) method implemented in the *autoresampling* package which applies autoML specifically for imbalanced classification becoming to their knowledge, the first approach that specialises in automating imbalanced classification [18]. It uses meta-learning therefore computationally complex to instantiate and on 101 imbalanced datasets tested, it got a predictive performance comparable to or better than similar state-of-the-art solutions. It is mainly for binary classification and only builds models using the *Random Forest* learning algorithm.

When analysing all these libraries/packages/frameworks, at this point there are not any advanced data pre-processing methods in the context of autoML, most methods combine predefined operators with features naively, and there are few flexible approaches to the autoML pipeline [8]. In addition, as most automate the creation of the pipeline, it is difficult to comprehend how a specific pipeline was created and introduce some hyperparameters to be used, it prevents the automation that autoML should automate in the first place. To make autoML truly available to inexperienced users in this domain, integration and deployment measures are necessary [8]. Moreover, there is sometimes a lack of scientific proof of why certain outcomes are achieved, and numerous papers do not cover all aspects of the implementation in detail, becoming complicated to reproduce the same outcomes [5].

Finally, when creating an autoML solution and addressing multiple datasets of different domains, it is also possible to remember previously learnt knowledge, however, the performance of the model on the previous datasets is substantially reduced [5]. For instance, there is the *learning without forgetting* method, which applies incremental learning and trains a model using only new data while preserving its original capabilities [19]. Then, in another work conducted, it is possible to only use a small proportion of old data for pretraining, and then escalate the proportion of a new class of data used to train the model [20].

Therefore, the contribution of this project is to implement a new easy-to-use application that automates the classification of imbalanced datasets even for less experienced users, mainly because that are few applications that specialise in imbalanced datasets.

3 Developed solution

The application was originally developed for a *Thesis of master's degree* [21]. It is built in *Python* and available, in a *GitHub* repository [22], as free and open-source software, licensed as *GPL 3.0* [23]. The built application implements two distinct modules, but they are interconnected, the learning module, which builds a knowledge base to be used by the second one, the recommendation module.

In the first module, the goal is to combine several resampling and classification algorithms, select the best combination of both to handle a dataset and save the dataset meta-features, the evaluation metrics, and the execution time of the best combination into the knowledge base.

In the second module, with the assistance of the previous knowledge base constructed, the application can recommend the best combination of resampling and classification algorithms to handle a new imbalanced dataset imported. This recommendation is made by finding in the knowledge base the most similar dataset in terms of meta-features.

To better understand this application, it was envisioned the architecture of the solution expressed as a component diagram in Figure 1.

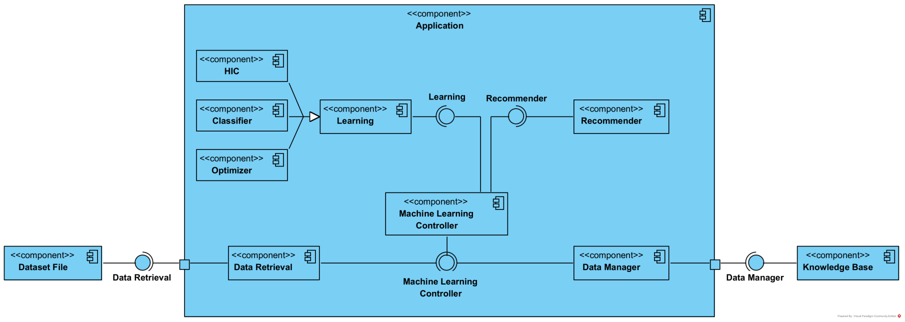


Fig. 1 Component Diagram.

Here, a dataset file should be loaded in the application using the data retrieval component that is responsible for reading the dataset file and that is called by the machine learning controller component. Then, the machine learning controller component communicates with the learning component, at the early stages of the application, and with the recommendation component, at the late stages of the application. The learning controller is composed of the handling imbalanced classification (HIC), classifier and optimiser components.

This first component applies different techniques to handle imbalanced classification, primarily in the pre-processing stage of the machine learning pipeline. The classifier component should select the most appropriate classification algorithm for the loaded dataset file, and then the optimiser component improves the selected classifier by optimising its parameters. When the model is prepared, the machine learning controller component uses the data manager component that is responsible for writing to the knowledge base.

3.1 Learning Module

To populate the knowledge base, several datasets were chosen from different business domains that have imbalanced data. The aim is to always choose publicly available datasets without needing to do specific data cleaning tasks before using them. In addition, it was also ensured to have a different ratio of proportions of imbalanced data across the diverse datasets.

In a summarised manner, first, it is read the imported dataset by file or by *OpenML* [24] dataset ID, then, it is extracted the meta-features information with the help of the *Meta-Feature Extractor* (MFE) library [25], next it is combined several resampling techniques and classification algorithms to train, test and validate, and then, the obtained results from the best combination of resampling and classification algorithm is written to the knowledge base of the application.

3.1.1 Datasets

Initially, it was analysed several candidate datasets from websites like *UCI Machine Learning Repository* [26], *KEEL – Knowledge Extraction based on Evolutionary Learning* [27], *OpenML*, *Kaggle* [28] and *Google Dataset Search* [29]. Then, it was selected to work with *KEEL* website because it listed the diverse datasets by the imbalanced ratio in an organised manner with key information. Afterwards, it was also selected to work with *OpenML* since it provides plenty of datasets to choose from, and it has an easy-to-use, and well-documented Application Programming Interface (API) [30] that simplified the different related datasets tasks.

At the time of this project development, the *OpenML API* provided 125 datasets when filtering the datasets that have an active status, for binary classification problems, with the number of instances (rows) between 200 and 10000, the number of features (columns) less than 500 and with an imbalance ratio above 2. Of these 125 datasets, some datasets were repeated since they have different versions of the same dataset, in this case it was selected the most recent one, discarding the older ones.

Other datasets were not possible to use because it was not conceivable to provide a decent enough evaluation metrics score. They needed major individual data pre-processing tasks that were not the point of this application to make. It was also selected datasets from the *KEEL* website, getting a total of 65 datasets to be used. For these 65 datasets, it was found that the imbalanced ratio ranges from 1.820 (minimum) to 85.880 (maximum), averaging 14.501 with a standard deviation of 19.301.

3.1.2 Reading and extracting knowledge from a dataset

To have a robust knowledge base to be used in the recommendation module, it was achieved 65 imported, executed, and documented datasets in the knowledge base. Most of these datasets were obtained with the help of the *OpenML API* and some by the *KEEL* website, as previously described. It was assumed in all these imbalanced datasets that the class with less representation is the class with more interest to predict, as it regularly occurs in imbalanced binary classification. Last, in some datasets, it is also needed to properly encode the existing categorical columns to integers/indicator values because some classification algorithms require it.

Regarding the *MFE* library to extract the meta-feature information from the datasets, it used the following groups of meta-features: *complexity*, *concept*, *general*, *itemset*, *landmarking*, *model-based* and *statistical*. Additionally, the summary function used was the *average/mean*, *standard deviation*, *kurtosis*, and *skewness*. It is important to note that some meta-features can have a distinct value, for example, the "*c2*" meta-feature of the group "*complexity*" which is the value of the imbalance ratio with no summary function values. Other ones are expressed with all (or some) of the summary functions defined, for example, the "*cov*" meta-feature of the group "*statistical*" which is the absolute value of the covariance of distinct dataset attribute pairs. All these meta-features used resulted in 257 values.

3.1.3 Sampling and classification algorithms used

This process started by executing 19 resampling techniques and 1 without any pre-processing technique combined with 11 classification algorithms, resulting in 220 different combinations. The 19 resampling techniques used, as of the time of writing, are all available in the Imbalanced Learn library [31].

Concerning resampling techniques, we considered 11 under-sampling techniques: *ClusterCentroids*, *CondensedNearestNeighbour*, *EditedNearestNeighbours*, *RepeatedEditedNearestNeighbours*, *AllKNN*, *InstanceHardnessThreshold*, *NearMiss*, *NeighbourhoodCleaningRule*, *OneSidedSelection*, *RandomUnderSampler* and *TomekLinks*; 6 over-sampling techniques: *RandomOverSampler*, *SMOTE*, *ADASYN*, *BorderlineSMOTE*, *KMeansSMOTE*, *SVMSMOTE*; and 2 combinations of over and under-sampling techniques: *SMOTEENN* and *SMOTETomek*.

The 11 classification algorithms used, available in *Scikit-Learn*, *LightGBM* [32] and *XGBoost* [33] libraries, are: *LogisticRegression*, *GaussianNB*, *SVC*, *KNeighborsClassifier*, *LGBMClassifier*, *XGBClassifier*, *RandomForestClassifier*, *ExtraTreesClassifier*, *AdaBoostClassifier*, *BaggingClassifier*, *GradientBoostingClassifier*.

It was used the default parameters of all resampling and classification algorithms, and for all the executions *random_state* equal to *42*, to guarantee reproducibility and *n_jobs* equal to *-1* to use all the processors of the machine during the cross-validation step.

When it was possible to specify that the dataset is binary or the *class_weight* is "*balanced*," it was appropriately indicated. The former specifies the learning objective function, and the latter stipulates, in "*balanced*" mode, to automatically adjust the class weights inversely proportional to class frequencies.

It was chosen the *RepeatedStratifiedKFold* function of the *Scikit-Learn* library that repeats a *Stratified K-Fold* cross-validator several times with different randomisation in each repetition, which assures an improved estimator performance. Here, it was used 10 folds with "*n_splits*" repeated 3 times with "*n_repeats*", which are common values for this case study.

Regarding the evaluation metrics to evaluate the solution, accuracy and error rate are not suited for imbalanced scenarios [34]. When the accuracy reflects the underlying class distribution, the accuracy paradox can occur. To rigorously evaluate each of these combinations, it was selected 5 adequate evaluation metrics to use in imbalanced binary classification, being: *Balanced Accuracy*, *F1 Score*, *ROC AUC*, *Geometric Mean* and *Cohen Kappa*.

3.1.4 Process of Discarding the Worst Performant Combinations

Testing 220 combinations of resampling techniques and classification algorithms on 65 datasets would be computationally very expensive, so, iteratively, we discarded some worst-performing combinations of resampling techniques and classification algorithms.

To do this selection, the 220 combinations of resampling techniques and classification algorithms were first applied to one dataset randomly chosen, which permitted to associate of each combination a *final score*, resulting from the average of the 5 metrics previously mentioned, and a corresponding ranking position, for example, position 22 from the 220 of total combinations. Next, two lists were initialised, one concerning the resampling techniques (*ResampTechRankList*) and the other with the classification algorithms (*ClassifierRankList*), both lists ordered from better to worse scores by the ranking position of resampling technique and classification algorithm, respectively.

Then, when some more datasets were randomly chosen and processed, the various positions of each combination were analysed by grouping all the different rank positions, first by the resampling technique and then by the classifier. Next, the combinations with the worst scores, with values above the third quartile (75% to 100%), were discarded for all the processed datasets.

In the first step, after 3 datasets were imported and processed, 5 resampling techniques and 3 classification algorithms were discarded, remaining 120 combinations. The algorithm was iteratively applied to several datasets, randomly chosen in each iteration. After five-time steps, it was discarded a total of 16 resampling techniques and 8 classification algorithms, with a total of 31 datasets processed. The rest of the datasets imported and processed no longer caused discarding more combinations because it was not found any worse performant resampling technique or classifier based on the previous explanation.

In the end, the remaining combinations were 12 with 4 resampling techniques and 3 classifiers, being:

- Resampling techniques:
  - RandomOverSampler,
  - SMOTE,
  - SVMSMOTE,
  - SMOTETomek.
- Classifiers:
  - LGBMClassifier,
  - XGBClassifier,

– GradientBoostingClassifier.

3.2 Recommendation Module

With this module, the goal is to deliver recommendations for the best combinations of resampling and classification algorithms to be used for a particular imported dataset.

For this, we started to get the best recommendation by developing a multi-classification model using the meta-feature values of each dataset as prediction features and the combination of resampling techniques and classification algorithms as the target. However, because of the limited size of the training set, 65 available datasets (number of instances/rows), each having 257 meta-features values, with 12 different target combinations, and the complexity of the classifiers, overfitting happened. Therefore, we calculated the best recommendations following an instance-based learning approach.

For that, it is computed the *Frobenius norm* (the *Euclidean distance* of two vectors) by using the "*linalg.norm*" function from the *NumPy* library [35], which, in this case, is the average of all *Euclidean distances* (vectors) of each meta-feature extracted between the current imported dataset and all the datasets in the knowledge base.

This takes into consideration the previously processed 257 meta-features in the learning module. The *Frobenius norm* can be expressed as Equation 1 and the *Euclidean distance* as Equation 2.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{i,j}|^2} \tag{1}$$

$$\|x\|_2 = \sqrt{\sum_{i}(u_i - v_i)^2} \tag{2}$$

Next, the three smaller average values are selected, since a smaller value means that those two datasets resemble the most in terms of the features used. By knowing the corresponding datasets, it is recommended the three combinations of resampling techniques and classification algorithms that are distinct and were recorded as the better performant ones, in the learning module, for those datasets.

3.2.1 Recommendation module exemplification

The user, in this module, interacts with the application through a GUI desktop application with the support of the *PySimpleGUI* library, as illustrated in Figure 2.

To better understand how this recommendation works, it will be exemplified with the following simple scenario. First, it is imported the "*car-good.dat*" dataset and submitted to the application, when it finalises all the calculations, it informs, in this example, that (*SVMSMOTE*, *GradientBoostingClassifier*),

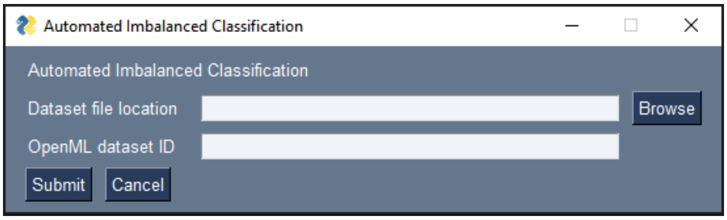
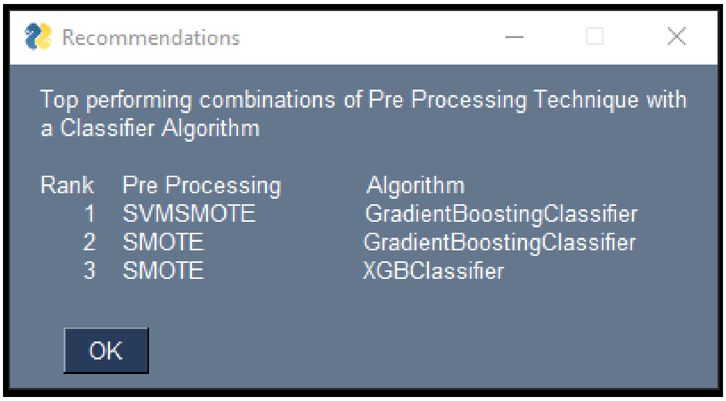

Automated Imbalanced Classification
Automated Imbalanced Classification
Dataset file location
Browse
OpenML dataset ID
Submit
Cancel

Fig. 2 GUI Application for Recommendation Module.

(*SMOTE*, *GradientBoostingClassifier*), (*SMOTE*, *XGBClassifier*) were the best three combinations of resampling techniques and classification algorithms correspondingly, as illustrated in Figure 3.

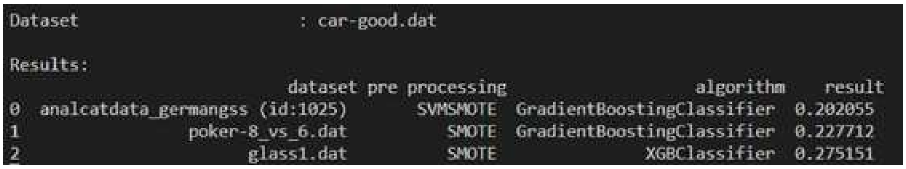


Top performing combinations of Pre Processing Technique with a Classifier Algorithm

Rank	Pre Processing	Algorithm
1	SVMSMOTE	GradientBoostingClassifier
2	SMOTE	GradientBoostingClassifier
3	SMOTE	XGBClassifier

Fig. 3 GUI recommendations example.

The application also outputs information to the console with more detailed and technical information, as illustrated in Figure 4.



```
Dataset                       : car-good.dat

Results:
                            dataset pre processing                   algorithm    result
0  analcatdata_germangss (id:1025)        SVMSMOTE  GradientBoostingClassifier  0.202055
1             poker-8_vs_6.dat           SMOTE  GradientBoostingClassifier  0.227712
2                   glass1.dat           SMOTE               XGBClassifier  0.275151
```

Fig. 4 GUI recommendations output example.

For this case, those recommendations were given because the "*analcatdata_germangss*" [36] (*OpenML ID*: 1025), "*poker-8_vs_6.dat*" [37] and "*glass1.dat*" [38] datasets had the lowest *Euclidean distances*, 0.202055, 0.227712 and 0.275151, respectively. Those datasets, in the learning module, had each of the best combinations of resampling techniques and classifiers. For instance, the "*analcatdata_germangss*" dataset, in the learning module, achieved the best *final score* (average of all evaluation metrics) with the (*SVMSMOTE*, *GradientBoostingClassifier*) combination.

4 Solution Evaluation

The evaluation of the solution is conducted with two distinct steps, an internal evaluation, and an external evaluation. The former is made by analysing and comparing the final recommended results, with the results that were acquired by the learning module. The latter can be made by analysing and comparing the final recommended results with other results from publicly available state-of-the-art papers, or by using autoML solutions. Find relevant state-of-the-art papers that could address these 15 selected datasets with a machine learning pipeline that adopted similar resampling techniques and/or classification algorithms, with a similar validation (*Stratified K-Fold* cross-validation) and with the same evaluation metrics selected it will be very difficult or even impossible. Consequently, the external evaluation will be done with one autoML application.

In the internal and external evaluation, the evaluation metrics used to evaluate the different solutions are the same used in the knowledge base construction: *Balanced Accuracy*, *F1 Score*, *ROC AUC*, *Geometric Mean* and *Cohen Kappa*. Additionally, it was assumed that the minority target class is the most relevant to predict. Also, it is important to mention that each evaluation task that needs to be executed, is executed with the same conditions of the same available local computer resources.

Concerning the datasets chosen to evaluate this application internally and externally, it was randomly selected 15 imbalanced datasets from the 65 used in the implementation of the application. The imbalanced ratio of these datasets ranges from 2.307 (minimum) to 67 (maximum), averaging 18.662 with a standard deviation of 21.998. The datasets, its dimension and their imbalance ratio are presented in table 1.

4.1 Internal Evaluation

It should be noted that the knowledge base records of the test datasets were not used in the internal evaluation, as this would not make sense, since the recommendation module is based on searching for the datasets closest to the dataset that is intended to find the best techniques to apply. It is important to mention that the values from the recommendation module are the ones got when executing the learning module to those 15 datasets, for the first recommended combination (of the three combinations available). All these values are expressed in the following Table 2 and Table 3, and the *final score* is the average of all metrics.

Consequently, for all 15 datasets, the recommendation module presented a *final score* smaller than the learning module and accomplished with all datasets

Table 1 Datasets selected to test the application.

ID	Dataset	lines x columns	IR
D1	*dis* (*OpenML ID*:40713)	3772 x 30	64.034
D2	*musk* (*OpenML ID*:1116)	2000 x 100	5.488
D3	*mfeat-fourier* (*OpenML ID*:971)	2000 x 77	9.000
D4	*Satellite* (*OpenML ID*:40900)	5100 x 37	67.000
D5	*arsenic-male-bladder* (*OpenML ID*:947)	5590 x 5	22.292
D6	*analcatdata_apnea2* (*OpenML ID*:765)	475 x 4	6.422
D7	*regime_alimentaire* (*OpenML ID*:42172)	220 x 20	3.744
D8	*page-blocks0.dat*	5473 x 10	8.789
D9	*dgf_test* (*OpenML ID*:42883)	3420 x 5	5.053
D10	*cpu_small* (*OpenML ID*:735)	8190 x 13	2.307
D11	*analcatdata_birthday* (*OpenML ID*:968)	365 x 4	5.837
D12	*optdigits* (*OpenML ID*:980)	5620 x 65	8.825
D13	*kr-vs-k-zero_vs_eight.dat*	1460 x 6	53.074
D14	*analcatdata_lawsuit* (*OpenML ID*:450)	264 x 5	12.895
D15	*JapaneseVowels* (*OpenML ID*:976)	9960 x 15	5.172

Table 2 Evaluation metrics values of the learning module.

Dataset	Balanced Accuracy	F1 Score	ROC AUC	Geometric Mean	Cohen Kappa	Final Score
D1	0.868	0.990	0.943	0.852	0.523	0.835
D2	0.998	0.996	1.000	0.998	0.996	0.998
D3	0.995	0.999	1.000	0.995	0.993	0.996
D4	0.875	0.752	0.993	0.860	0.749	0.846
D5	0.795	0.636	0.836	0.716	0.625	0.722
D6	0.936	0.833	0.972	0.934	0.804	0.896
D7	0.949	0.899	0.973	0.947	0.869	0.927
D8	0.946	0.883	0.990	0.945	0.870	0.927
D9	0.987	0.971	0.999	0.987	0.966	0.982
D10	0.916	0.947	0.979	0.916	0.827	0.917
D11	0.860	0.932	0.936	0.851	0.619	0.840
D12	0.979	0.997	0.999	0.979	0.971	0.985
D13	0.999	0.970	1.000	0.999	0.969	0.987
D14	0.970	0.916	0.993	0.965	0.909	0.951
D15	0.989	0.995	1.000	0.989	0.972	0.989

an average *final score* of 0.9087 ± 0.0837 and the learning module accomplished 0.9199 ± 0.0798. Thus, the *final score* of the recommendation module is smaller, on average, by 1.23% than the one attained by the learning module.

Regarding the evaluation metrics achieved from the recommendation module compared to the development module concerning the 15 datasets, the recommendation module will always return worse or, in the best scenario, equal to the best one got by the development module. This situation occurs because there is always one combination that returns the best score of the metrics used and all the remaining ones are worse, depending on the dataset used.

Concerning the execution time for all 15 datasets, the execution of the recommendation module was accomplished in 1073 seconds and the learning

Table 3 Evaluation metrics values of the recommendation module.

Dataset	Balanced Accuracy	F1 Score	ROC AUC	Geometric Mean	Cohen Kappa	Final Score
D1	0.787	0.995	0.915	0.747	0.614	0.812
D2	0.998	0.996	1.000	0.998	0.996	0.998
D3	0.990	0.999	0.999	0.990	0.986	0.993
D4	0.882	0.672	0.984	0.870	0.666	0.815
D5	0.795	0.636	0.836	0.716	0.625	0.722
D6	0.936	0.833	0.972	0.934	0.804	0.896
D7	0.940	0.876	0.977	0.938	0.840	0.914
D8	0.950	0.868	0.992	0.949	0.852	0.922
D9	0.987	0.971	0.999	0.987	0.965	0.982
D10	0.914	0.943	0.976	0.913	0.816	0.912
D11	0.800	0.937	0.944	0.778	0.576	0.807
D12	0.982	0.996	0.999	0.982	0.960	0.984
D13	0.980	0.947	0.998	0.977	0.945	0.969
D14	0.966	0.873	0.991	0.962	0.863	0.931
D15	0.978	0.987	0.998	0.978	0.925	0.973

module in 8237 seconds. Thus, for these 15 datasets, the recommendation module time was approximately 8 times smaller/faster than the learning module time execution. This was expected because it is usually faster to execute *Euclidean distances* on some meta-feature values than executing some combinations of resampling techniques and classification algorithms.

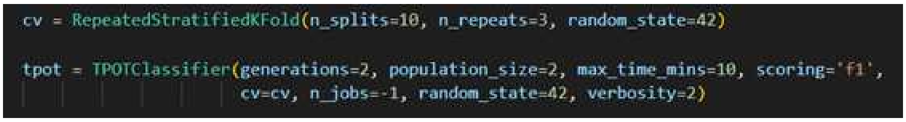
4.2 External Evaluation

In the external evaluation, the goal was to select one autoML application that could be executed for these 15 datasets with the same machine learning pipeline. It was explored all the different applications previously analysed in the State of the Art section. The first ones to be excluded from this choice were the autoML services, such as *Microsoft Azure Automated Machine Learning*, *AWS SageMaker Autopilot* and *GCP AutoML*, because they execute in different servers/machines than the one used in the implementation and evaluation of the developed application, this would cause a compromised comparison.

The autoML tool selected was the *TPOT*, a tree-based pipeline optimisation tool because it was noted to be the open-source tool that permits defining parameters that assure test conditions like those defined by our application. In this scenario, it was only needed to test higher or smaller values with a "try-error" approach for two parameters, as explained further. Additionally, it can export any produced pipeline directly to *Python* code.

It was executed the *Python* file to all the 15 test datasets selected which contain the "*TPOTClassifier*" function with the following parameters used, as illustrated in Figure 5.

First, the "*generations*" and the "*population_size*" parameters are, in this scenario, the parameters used as a "try-error" approach because specifying them with higher values usually results in higher scores/metrics values but with



```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)

tpot = TPOTClassifier(generations=2, population_size=2, max_time_mins=10, scoring='f1',
                      cv=cv, n_jobs=-1, random_state=42, verbosity=2)
```

Fig. 5 TPOTClassifier function used.

also increased times of execution. To have similar values of execution time as the recommendation module of the implemented application achieves, it was concluded that the value of "2" to the "*generations*" and the "*population_size*" was the most suited to these 15 datasets and the available local computer.

Then, it was used the "*max_time_mins*" parameter with "*10*" which sets the maximum time that *TOPT* must optimise the pipeline because it is a closer value to the maximum time that the learning module achieved in one of these 15 datasets. Next, the "*scoring*" parameter was set to "*f1*" because F1 Score is one of the metrics used, and this parameter only lets set one metric.

Afterwards, the "*cv*" parameter sets the cross-validation strategy to be used, and the "*n_jobs*" and "*random_state*" parameters were set the same as the implementation of the application.

4.2.1 Evaluation Metrics Comparison

Now, the evaluation metrics achieved from the recommendation module will be compared to the *TPOT* tool, previously explained, concerning the 15 datasets. The evaluation metrics values of the recommendation module were already presented previously in table 3, and the evaluation metric values of the *TPOT* tool are presented in Table 4.

Table 4 Evaluation metrics values of the TPOT tool.

Dataset	Balanced Accuracy	F1 Score	ROC AUC	Geometric Mean	Cohen Kappa	Final Score
D1	0.721	0.994	0.721	0.666	0.566	0.734
D2	0.998	0.991	0.998	0.998	0.989	0.995
D3	0.941	0.993	0.941	0.939	0.920	0.947
D4	0.875	0.857	0.875	0.866	0.855	0.866
D5	0.800	0.750	0.800	0.775	0.736	0.772
D6	0.528	0.105	0.528	0.236	0.091	0.298
D7	0.972	0.917	0.972	0.972	0.888	0.944
D8	0.906	0.868	0.906	0.902	0.853	0.887
D9	0.981	0.961	0.981	0.981	0.954	0.972
D10	0.890	0.939	0.890	0.888	0.784	0.878
D11	0.532	0.922	0.532	0.297	0.092	0.475
D12	0.958	0.992	0.958	0.957	0.924	0.958
D13	0.688	0.042	0.688	0.662	0.031	0.422
D14	0.742	0.600	0.742	0.701	0.569	0.671
D15	0.976	0.992	0.976	0.976	0.948	0.974

Consequently, for all 15 datasets, the recommendation module presented a *final score* greater than the *TPOT* tool and accomplished with all datasets an average *final score* of 0.9087 ± 0.0837 and the *TPOT* tool accomplished 0.7862 ± 0.2240. Thus, the *final score* of the recommendation module is greater, on average, by 15.6% than the one attained by the *TPOT* tool.

Concerning the execution time for all 15 datasets, the execution of the recommendation module was accomplished in 1073 seconds and the *TPOT* tool in 1381 seconds. Thus, for these 15 datasets, the recommendation module time was 29% smaller/faster than the *TPOT* tool time execution.

5 Conclusions

This application was successfully documented, designed, implemented, and evaluated. It can deliver recommendations of suited combinations of resampling techniques and classification algorithms to imbalanced datasets, therefore automating this step in the machine learning pipeline, and thus reducing the human effort placed in building accurate predictive models.

Such tasks are complicated and time-consuming because they require testing a significant number of possible solutions. The proposed application takes advantage of solutions already tested with previous datasets and provides recommendations to a newly imported dataset by using its meta-features values, to be compared with the most similar datasets already present in the knowledge base, thus helping to automate the development of efficient solutions to imbalance binary classification problems.

Additionally, it was used appropriate evaluation metrics to benchmark internally each combination, and externally to the overall recommendations delivered by this application compared to other autoML solution. The latter was achieved with small success but with smaller execution times, as it was evaluated to certain conditions established.

As it was analysed in the State of the Art, there are several autoML solutions. However, there are a few that focus specifically on handling imbalanced classification problems. Consequently, this project has a positive overview of the work done, especially when considering some limitations and future work needed, as it is going to be explained in the next section.

5.1 Limitations and Future Work

While the objectives were accomplished, there is still some improvement that should be adopted for this application. First, it should be evaluated if some hyperparameter optimisation techniques like *grid search*, *Bayesian optimisation* and others can improve the results achieved. Moreover, it can be experimented to run the application on the GPU instead of the CPU of the computer with libraries like *Apache Spark*, *Dask*, *Ray* or others, to improve the execution times.

Furthermore, it can also be applied a meta-feature selection like *principal component analysis* to the extracted meta-features. Additionally, it can be verified if the results obtained when recommending are improved by adding more datasets to the knowledge base of the application. Finally, in the future, this application should be extended to operate also with multi-class classification problems.

Ethical Approval

Not Applicable.

Availability of supporting data

The data used in this study is openly available from public sources as described in the text. The code developed is freely available at GitHub [22] and licensed as *GPL 3.0* [23].

Competing interests

The authors have no competing interests to declare.

Funding

Not Applicable.

Authors' contributions

Pedro Marques Vieira: application implementation, original manuscript preparation and writing. Fátima Rodrigues: conceptualisation, supervision and review/editing.

References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O. Scikit-learn: Machine learning in python. Journal of Machine Learning Research, 12, pp. 2825–2830, 2011.

[2] G. Lemaître, F. Nogueira, and C. K. Aridas, 'Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning', J. Mach. Learn. Res., vol. 18, pp.1–5, Jan. 2017.

[3] N. Lunardon, G. Menardi, and N. T. Maintainer, Package "ROSE" Type Package Title Random Over-Sampling Examples, 2021.

[4] CRAN - Package imbalance. https://cran.rproject.org/web/packages/imbalance/index.html accessed Feb. 14, 2022.

[5] He, X., Zhao, K., Chu, X. AutoML: A survey of the state-of-the-art. Knowledge-Based Systems, 2021, 212, p. 106622. https://doi.org/10.1016/j.knosys.2020.106622

[6] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, 'Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning', 2020, http://arxiv.org/abs/2007.04074 accessed: Feb. 13, 2022.

[7] Yao, Q., Wang, M., Chen, Y., Dai, W., Li, Y. F., Tu, W. W., Yu, Y., Taking Human out of Learning Applications: A Survey on Automated Machine Learning, 2018, https://arxiv.org/abs/1810.13306v4

[8] M. A. Zöller and M. F. Huber, Benchmark and Survey of Automated Machine Learning Frameworks, J. Artif. Intell. Res., vol. 70, pp. 409–472, 2021, https://doi.org/10.1613/jair.1.11854.

[9] AutoKeras https://autokeras.com/ accessed Feb. 13, 2022.

[10] Keras: the Python deep learning API https://keras.io/ accessed Feb. 13, 2022.

[11] Welcome To Neural Network Intelligence — An open source AutoML toolkit for neural architecture search, model compression and hyper-parameter tuning (NNI v2.6). https://nni.readthedocs.io/en/stable/ accessed Feb. 13, 2022.

[12] Olson, R.S., Bartley, N., Urbanowicz, R.J. and Moore, J.H., Evaluation of a tree-based pipeline optimisation tool for automating data science. In Proceedings of the genetic and evolutionary computation conference pp. 485-492, 2016. https://doi.org/10.1145/2908812.2908918

[13] hyperopt-sklearn by hyperopt https://hyperopt.github.io/hyperopt-sklearn/ accessed Feb. 13, 2022.

[14] AutoML: Automatic Machine Learning — H2O 3.36.0.2 documentation https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html accessed Feb. 13, 2022.

[15] Automated Machine Learning - Microsoft Azure, 2022. https://azure.microsoft.com/en-us/services/machine-learning/automatedml/#features accessed Feb. 07, 2022

[16] Amazon SageMaker Autopilot - Amazon SageMaker, 2022 https://aws.amazon.com/pt/sagemaker/autopilot/ accessed Feb. 07, 2022

[17] Cloud AutoML - Google Cloud, 2022 https://cloud.google.com/automl accessed Feb. 07, 2022.

[18] N. Moniz and V. Cerqueira, Automated imbalanced classification via meta-learning, Expert Systems with Applications, 178, 115011 https://doi.org/10.1016/j.eswa.2021.115011

[19] Z. Li, D. Hoiem, Learning without Forgetting, IEEE transactions on pattern analysis and machine intelligence 40, no. 12 (2017): 2935-2947, https://doi.org/10.1109/TPAMI.2017.2773081

[20] S. A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, iCaRL: Incremental Classifier and Representation Learning, In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 2001-2010

[21] P. Vieira, Automatic Handling of Imbalanced Datasets for Classification. http://hdl.handle.net/10400.22/22518 accessed Mar. 30, 2023

[22] P. Vieira, PedroVieira1160634/automated-imbalanced-classification: Automated Imbalanced Classification. https://github.com/PedroVieira1160634/automated-imbalanced-classification accessed Sep. 10, 2022

[23] GNU General Public License v3.0 - Project GNU - Free Software Foundation https://www.gnu.org/licenses/gpl-3.0.html accessed Sep. 10, 2022

[24] OpenML https://www.openml.org/search?type=data accessed Feb. 14, 2022

[25] The PyMFE example gallery — pymfe 0.4.1 documentation https://pymfe.readthedocs.io/en/latest/auto_examples/index.html accessed Aug. 20, 2022

[26] UCI Machine Learning Repository https://archive.ics.uci.edu/ml/index.php accessed Feb. 14, 2022

[27] KEEL: A software tool to assess evolutionary algorithms for Data Mining problems (regression, classification, clustering, pattern mining and so on) https://sci2s.ugr.es/keel/datasets.php accessed Feb. 14, 2022

[28] Find Open Datasets and Machine Learning Projects - Kaggle https://www.kaggle.com/datasets accessed Feb. 14, 2022

[29] Dataset Search https://datasetsearch.research.google.com/ accessed Feb. 14, 2022

[30] OpenML APIs - OpenML Documentation https://docs.openml.org/APIs/ accessed Jul. 30, 2022

[31] Imbalanced-learn documentation — Version 0.9.1 https://imbalancedlearn.org/stable/ accessed Sep. 10, 2022

[32] Python-package Introduction — LightGBM 3.3.2.99 documentation https://lightgbm.readthedocs.io/en/latest/Python-Intro.html accessed Sep. 10, 2022

[33] Python Package Introduction — xgboost 1.6.2 documentation https://xgboost.readthedocs.io/en/stable/python/python_intro.html accessed Sep. 10, 2022

[34] N. Japkowicz, Learning from Imbalanced Data Sets: A Comparison of Various Strategies, 2000, https://www.researchgate.net/publication/2628420_Learning_from_Imbalanced_Data_Sets_A_Comparison_of_Various_Strategies accessed: Feb. 06, 2022.

[35] NumPy https://numpy.org/ accessed Sep. 10, 2022

[36] OpenML: analcatdata_germangss dataset (ID: 1025), 2014. https://www.openml.org/search?type=data&status=active&id=1025 accessed Sep. 11, 2022

[37] KEEL: Poker Hand dataset https://sci2s.ugr.es/keel/dataset.php?cod=1340 accessed Sep. 11, 2022

[38] KEEL: Glass Identification dataset https://sci2s.ugr.es/keel/dataset.php?cod=142 accessed Sep. 11, 2022