INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




AUTOMATIC HANDLING OF IMBALANCED DATASETS FOR CLASSIFICATION

PEDRO MARQUES VIEIRA
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

AUTOMATIC HANDLING OF IMBALANCED DATASETS FOR CLASSIFICATION

Pedro Marques Vieira

A dissertation attending on the requirements for the master's degree in Computer Engineering, specialisation area in Information and Knowledge Systems

Advisor: Doctor Fátima Rodrigues

Porto, October 2022

I dedicate this work to my mother, my father, and my grandparents.

Abstract

Imbalanced data is present in various business areas and when facing it without proper knowledge, it can have undesired negative consequences. In addition, the most common evaluation metrics in machine learning to measure the desired solution can be inappropriate and misleading. Multiple combinations of methods are proposed to handle imbalanced data however, often, they required specialised knowledge to be used correctly.

For imbalanced classification, the desire to correctly classify the underrepresented class tends to be more important than the overrepresented class, while being more challenging and time-consuming. 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 the specific dataset imported, by extracting and comparing 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, Handling Imbalanced Data, Automated Machine Learning.

Resumo

Os dados não balanceados estão presentes em diversas áreas de negócio e, ao enfrentá-los sem o devido conhecimento, podem trazer consequências negativas e indesejadas. Além disso, as métricas de avaliação mais comuns em aprendizagem de máquina (*machine learning*) para medir a solução desejada podem ser inadequadas e enganosas. Múltiplas combinações de métodos são propostas para lidar com dados não balanceados, contudo, muitas vezes, estas exigem um conhecimento especializado para serem usadas corretamente.

Para a classificação não balanceada, o desejo de classificar corretamente a classe sub-representada tende a ser mais importante do que a classe que está representada em demasia, sendo mais difícil e demorado. Várias abordagens, desde as mais acessíveis até as mais avançadas nos domínios de reamostragem de dados e técnicas sensíveis ao custo vão ser consideradas para lidar com dados não balanceados.

A aplicação desenvolvida fornece recomendações das combinações de técnicas mais adequadas para o conjunto de dados específico importado, extraindo e comparando os valores de meta características registados numa base de conhecimento. Ela facilita a classificação sem esforço e automatiza parte das etapas de aprendizagem de máquina com resultados comparáveis ou melhores a uma solução de estado da arte e com tempo de execução muito menor.

Palavras-chave: Classificação Não Balanceada, Manipulação de Dados Não Balanceados, Automatização de Aprendizagem de Máquina.

Acknowledgements

First, I thank my mother, father, and grandparents who always believe in me and the rest of my family. Also, *Instituto Superior de Engenharia do Porto* provided a great workplace and my supervisor, Fátima Rodrigues, assisted me with the best intentions. Last, I am grateful to all people that have supported me on this journey.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Acronyms

ADASYN	*Adaptive Synthetic Sampling Approach*
API	*Application Programming Interface*
ATOMIC	*Automated Imbalanced Classification*
AutoML	*Automated Machine Learning*
AWS	*Amazon Web Services*
CASH	*Combined Algorithm Selection and Hyperparameter*
CPU	*Central Processing Unit*
FAST	*Features from Accelerated Segment Test*
FMS	*Full Model Selection*
FN	*False Negative*
FP	*False Positive*
GCP	*Google Cloud Platform*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
HIC	*Handling Imbalanced Classification*
MFE	*Meta-Feature Extractor*
NAS	*Neural Architecture Search*
NNI	*Neural Network Intelligence*
QFD	*Quality Function Deployment*
RH	Research Hypothesis
ROC AUC	*Area Under the Receiver Operating Characteristic Curve*
SMOTE	*Synthetic Minority Oversampling Technique*
SVM	*Support Vector Machines*
TN	*True Negative*

TOPSIS *Technique for Order of Preference by Similarity to Ideal Solution*

TP *True Positive*

1 Introduction

This chapter gives the reader an overview of the project, presents the Context, and then, it is explained the Problem, Goals and Approach for this project. Last, it presents the Document Structure where it is enumerated the chapters and sections that compose this document.

1.1 Context

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.

1.2 Problem

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, the combination of different data resampling techniques, classification algorithms and multiple hyperparameter optimization transforms the possibilities to evaluate the desired

solution to become endless. Thus, is needed a solution to automate and facilitate these imbalanced classification tasks, hence, to get better and faster results.

1.3 Goal

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 balance techniques, an implementation of the most promising ones 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.

1.4 Approach

First, it is necessary to survey the literature on balance algorithmic techniques, classification algorithms, and the automation of machine learning tasks that can be conducted for imbalance scenarios. Thus, this survey should be primarily aimed at the keywords "*automated imbalanced classification*," and then, it must be also analysed appropriate related solutions. Next, it should be conducted a value analysis of this new solution.

Later, it must be designed the architecture of the solution to then implement and evaluate the solution where it should be explained different choices made. Also, the selection of different algorithms and evaluation metrics should follow quantitative research methods with an experimental research design.

1.5 Document Structure

This document consists of seven chapters, each of which is divided into sections, and one part for the references. The first chapter, named Introduction, delivers the Context, Problem, Goals, and Approach to be conducted. Then, there is the State of the Art chapter with four sections, where is it explained and contextualised the Classification, Imbalanced Classification, Relevant Machine Learning Topics, and Related Solutions. Next, it follows the Value Analysis chapter with the Opportunity Identification, Opportunity Analysis, Value Offer, and TOPSIS Method sections.

Later, it is presented the Solution Design chapter, which is divided into four sections, Requirement Analysis, System Architecture, Datasets, and Technologies Choice. Next, there is the Solution Implementation chapter, which is split into two sections, the Development Phase, and the Recommendation Phase. Then, there is the Solution Evaluation chapter, which is divided into three sections, the Internal Evaluation, the External Evaluation, and the Final

Remarks. Finally, there are the Conclusions of this document with the Accomplished Objectives and the Limitations and Future Work sections.

2 State of the Art

This chapter starts with an explanation of Classification and examples of Classification Algorithms that can be applied. Next, it is made a contextualisation of the Imbalanced Classification, and then it is enumerated Causes and Examples and important Strategies for Handling Imbalanced Domains. After that, it is described Relevant Machine Learning Topics such as Meta-Features, Evaluation Metrics, Automated Machine Learning and Optimization. Finally, it is enumerated appropriate Related Solutions with Final Considerations included.

2.1 Classification

Classification is a supervised predictive modelling problem in machine learning that involves assigning a class label to each observation[1]. All the observations can be expressed on a training dataset file for uncomplicated data manipulation. Classification is in the supervised learning scope, where there is an output label for corresponding input features. Additionally, there is the binary classification where there are only two labels or multiclass classification where there are multiple (over two) labels.

2.1.1 Classification Algorithms

Classification algorithms can be split into linear, non-linear and ensemble algorithms. Linear algorithms, often mentioned as probabilistic algorithms, are those that are often drawn from the field of statistics and make strong assumptions about the functional form of the problem[2]. Examples can be logistic regression, linear discriminant analysis, and Naive Bayes. Nonlinear algorithms are drawn from the field of machine learning and make few assumptions about the functional form of the problem[2]. Examples can be decision trees, k-nearest neighbours, artificial neural networks, and support vector machines (SVM). Finally, ensemble algorithms are also drawn from the field of machine learning and combine the predictions from two or more models that can be bagging or boosting[2]. Examples can be bagged decision trees, random forests, extra trees, and gradient boosting.

Other algorithms that can be applied to classification are a cost-sensitive algorithm approach and the one-class algorithms. Cost-sensitive algorithms take the differing costs of misclassification into account when fitting the model to the training dataset[3]. Most of the linear, non-linear and ensemble algorithms can be adapted to this sensitive approach. On the other hand, One-Class Algorithms are more used in outlier detection and anomaly detection, with few examples of the positive class[3]. Examples can be one-class SVM, isolation forests, minimum covariance determinants, and local outlier factors.

2.2 Imbalanced Classification

A dataset becomes inherently imbalanced when one class is heavily underrepresented, in their instances, regarding the rest of the classes, in two-class or multi-class datasets[3]. The underrepresented class is designated as the minority class, which has few instances. Contrarily, the majority class has several instances. The minority class is typically the one with the most interest, which means it is more desired to predict the class label or probability for the minority class than the majority class or classes[2]. The minority class is represented as the positive one, which corresponds to the class where the correct prediction is more important. The imbalanced ratio can be defined as Equation 1[4], where N_- and N_+ are cardinalities of the minority and the majority classes, respectively.

$$IR = \frac{N_-}{N_+} \tag{1}$$

Although, this ratio can also be expressed, for example, in (1:50), which means that for every one example in a particular class, there are fifty examples in the other class, for an imbalanced binary classification problem. This imbalance property can be split into a slight imbalance and a severe imbalance[2]. The former applies when the distribution of examples is uneven by a small amount in the training dataset, for example, a distribution of (2:3); and the latter applies when the distribution of examples is uneven by a large amount in the training dataset, by (1:100) or more.

A slight imbalance of the classes is often not a concern, because it can be applied to regular classification predictive modelling problems without degradation of results[5]. A severe class imbalance and/or the existence of classes that are overlapped, one in another, may require the use of specialised techniques and can be challenging to model[5]. Although, sometimes, the less represented class is not the most relevant one depending on the aim of the work, or the existence of class imbalance does not become conceivably a challenging problem, for instance, when the classes are well separated[4].

2.2.1 Causes and Examples

The class imbalance causes origin mostly from two main groups, biased sampling, and measurement errors. The former can be applied when the data is collected in a way that does not correctly represent the entire distribution, and the latter, when the observations conducted have errors, for example, when applying the wrong label to some of the samples. Also, the imbalance can be a property of the domain problem, in which the presence of one class may dominate over other classes, because of cost, time, or computation. Some other important characteristics regarding these data intrinsic situations can be expressed as follows: "*i) the identification of areas with small disjuncts, ii) lack of density and information in the training data, iii) class-overlap, iv) the impact of noisy data, v) the significance of the borderline instances, and vi) changes between the distribution of train and test sets, i.e. data set shift*"[6].

Some of the most remarkable examples of imbalance classification are fraud detection, claim prediction, churn prediction, default prediction, spam detection, anomaly detection, outlier detection, intrusion detection and conversion prediction. Most of these examples are binary, for example, in fraud detection, the goal is to detect fraud and no-fraud transactions. The minority class is usually rare, extreme, or unusual in some capacity and faces abundant examples of the majority class. Consequently, the desire to detect or predict the minority class highlights the challenge of this problem. For imbalanced multiclass classification, the problem arises when there are multiple minorities and majority classes that cause skew data distribution, for example, "*a class may be a minority one when compared to some other classes, but a majority of the rest of them*"[7].

2.2.2 Strategies for Handling Imbalanced Domains

The different strategies for handling imbalanced domains can be summarized in Figure 1[8].

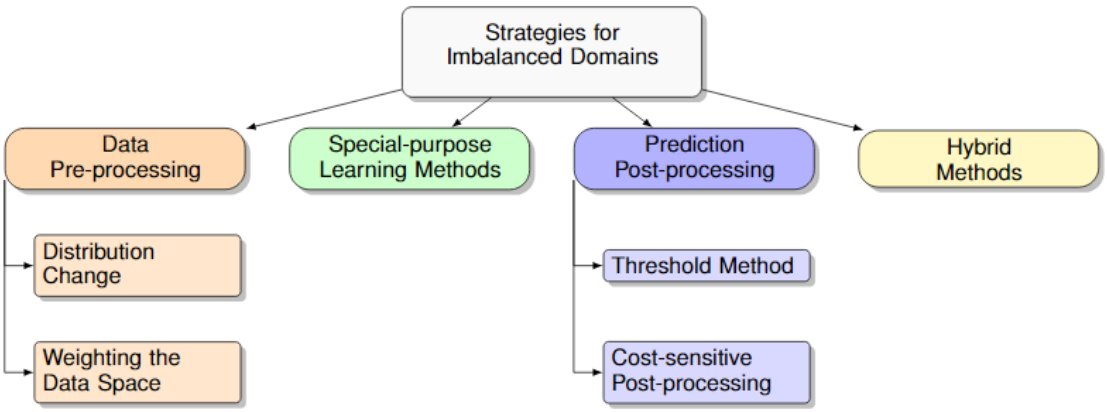

Figure 1: Strategies for Handling Imbalanced Domains.

Data pre-processing techniques can be split into distribution changes when resampling the data or weighting the data space when applying cost-sensitive procedures. First, distribution change occurs when it is changed the data distribution to better represent the more relevant and less represented cases[8]. Consequently, distribution change and more specifically data sampling algorithms change the composition of the training dataset to improve the performance of a standard machine learning algorithm on an imbalanced classification problem[5].

Data oversampling involves duplicating examples several times of the minority class or synthesising new examples from the minority class from existing examples[5]. Examples can be random oversampling, synthetic minority oversampling technique (SMOTE), borderline SMOTE, SVM SMOTE, k-means SMOTE, and adaptive synthetic sampling approach (ADASYN).

Under-sampling involves deleting examples from the majority class, such as randomly or using an algorithm to carefully choose which examples to delete[5]. Examples can be random under-sampling, condensed nearest neighbour, Tomek links, edited nearest neighbours, neighbourhood cleaning Rule, and one-sided selection. Also, it is possible to combine different combinations of oversampling and under-sampling techniques. Examples can be SMOTE and random under-sampling, SMOTE and Tomek links, and SMOTE and edited nearest neighbours.

When applying under-sampling there is a risk of losing important cases and when applying over-sampling there is a risk of overfitting because of the replication of certain cases[9].

For weighting the data space, it is changed the distribution of the data to process misclassification costs and avoid costly errors[8]. To this effect, cost-sensitive algorithms can be effective when used on imbalanced classification, where the cost of misclassification is configured to be inversely proportional to the distribution of examples in the training dataset[11]. However, there is a risk of model overfitting and not knowing the actual cost to properly apply.

For **special-purpose learning methods** like the approach of cost-sensitive algorithms, it is applied a modification of a selected algorithm in the preference criterion that directly incorporates costs in the learning process[8]. Additionally, ensemble methods can also be applied to this effect, using a cost-sensitive framework by integrating them into the learning phase.

For **prediction post-processing**, there is a threshold method that uses the ranking provided by a score, which expresses the degree to which an example is a member of a class and cost-sensitive post-processing that associates costs with prediction errors and minimises the expected cost[8].

Finally, there are also **hybrid methods** when combining different approaches to benefit from some of the main advantages of different solutions with a high variety of results[8].

2.3 Relevant Machine Learning Topics

There are many relevant machine learning topics to be addressed, thus it was selected a few, and they are explained in a summarised manner regarding some significant aspects.

2.3.1 Meta-Features

First, meta-learning refers to learning about learning and, in machine learning, to machine learning algorithms that learn from the output of other machine learning algorithms. Meanwhile, meta-features are measures used to characterise datasets and their relations with algorithm bias[10]. Meta-features are used in machine learning to represent and understand a dataset, to understand a certain learning bias, to provide the knowledge to create machine learning recommendation systems, and to create surrogate models and other scenarios.

Meta-features can be separated into the following meta-features groups[11]:

- Complexity: estimate the difficulty in separating the data points into their expected classes.
- Concept: estimate the variability of class labels among examples and the density of the examples.
- General: general information related to the dataset, also known as simple measures, such as the number of instances, attributes and classes.
- Itemset: compute the correlation between binary attributes.
- Landmarking: performance of simple and efficient learning algorithms.
- Model-based: measures designed to extract characteristics from simple machine learning models.
- Statistical: Standard statistical measures to describe the numerical properties of data distribution.

Finally, the meta-features can be expressed with descriptive statistics, when in a single value, or with a distribution, when in multiple values. Examples can be the mean, median, maximum, minimum, standard deviation, variance, kurtosis, and skewness.

2.3.2 Automated Machine Learning

Automated machine learning (AutoML) is "*a subfield of machine learning devoted to the development of approaches for automatically selecting and optimising predictive models*"[9]. AutoML usually involves the automatic selection of data preparation, machine learning model, and model hyperparameters for a predictive modelling task. It refers to techniques that quickly allow practitioners with modest technical skills to discover a suitable predictive model pipeline for the machine learning tasks, with little intervention in the different steps of the pipeline, other than providing a dataset and, in supervised classification, by choosing the target classes to aim[12].

Alternative approaches to automate tasks of the machine learning pipeline can be found in the fields of artificial neural networks. When applying neural networks to a classification problem, it is possible to use a Neural Architecture Search (NAS), transfer learning, and continual learning, among other novel approaches. NAS can automate artificial neural networks with sometimes excellent results nevertheless often demands increased computational resources[13]. Alternatively, transfer learning is a machine learning method where a model developed for a task, which can be pre-trained, is reused as the starting point for a model on a second task[14]. Finally, continual learning studies the problem of learning from an endless data source, with the ability to improve over time the acquired knowledge and use it for future learning[15].

2.3.3 Optimization

Based on the *No Free Lunch Theorem*, without having substantive information about the modelling problem, there is no single best machine learning algorithm for predictive modelling problems such as classification[16]. In addition, there is no definitive best path for machine learning algorithms to effectively predict and there is no clear way to know where to start or when to discard a model or when to perfect a model without proper knowledge of the problem[12]. Two viable solutions that address this situation are to approach the problem as a series of sequential decisions, iteratively, and by choosing popular machine learning algorithms that other researchers already used and that worked for similar cases. Another one is by implementing the Combined Algorithm Selection and Hyperparameter (CASH) Optimization, which addresses the selection of the data preparation technique, the learning algorithm, and the algorithm hyperparameters[17].

The CASH Optimization requires that the data preparation, the selection of the machine learning model, along with the corresponding model hyperparameters, must form the scope of the optimization problem and that the optimization algorithm must know all these dependencies[17]. This approach can also be referenced as Full Model Selection (FMS), being a challenging global optimization problem that must be aware of all these dependencies described as sequential global optimization algorithms with specific versions of Bayesian Optimization[18]. A Bayesian Optimization is an approach that uses the Bayes Theorem to direct the search to find the minimum or maximum of an objective function[19]. Also, there is the possibility of more advanced concepts like evolutionary optimization, or more simple ones like random search which is defined by a search space as a bounded domain of hyperparameter values and randomly sample points in that domain, or grid search, when evaluating every position in the grid of the hyperparameter values[2].

2.3.4 Evaluation Metrics

Regarding the evaluation metrics to evaluate a solution, accuracy and error rate are not suited for imbalanced scenarios[3]. When the accuracy is reflecting the underlying class distribution, the accuracy paradox can occur. There are more appropriate single-class measures for binary imbalanced classification like Precision, Recall, Specificity, Balanced Accuracy, F1 Score, and Geometric Mean, and some overall class metrics like Area Under the Receiver Operating Characteristic Curve (ROC AUC) and Cohen Kappa [20]. These metrics are calculated from the confusion matrix that informs what classes are predicted correctly, which were incorrectly predicted and what types of errors are being made[20], and is given in the following Table 1.

Table 1: Confusion Matrix.

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

The Precision or Positive Predicted Value summarises the fraction of examples assigned to the positive class that belongs to the positive class and the Recall, Sensitivity or True Positive Rate summarises how well the positive class was predicted. They range between 0 and 1 and are given by the following Equation 2 and Equation 3[20], respectively.

$$\boldsymbol{Precision} = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2}$$

$$\boldsymbol{Recall} = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3}$$

The Specificity or True Negative Rate summarises how well the positive class was predicted, which ranges between 0 and 1 and is given by the following Equation 4[20].

$$\boldsymbol{Specificity} = \frac{TrueNegative}{TrueNegative + FalsePositive} \tag{4}$$

The Balanced Accuracy is the arithmetic mean of the True Positive Rate and the True Negative Rate, which ranges between 0 and 1 and is given by the following Equation 5[8].

$$\boldsymbol{Balanced\ Accuracy} = \frac{True\ Positive\ Rate + True\ Negative\ Rate}{2} \tag{5}$$

The F1 Score or F-Measure is the harmonic mean between Precision and Recall, which ranges between 0 and 1 and is given by the following Equation 6[20].

$$\boldsymbol{F1\text{-}Score} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

The Geometric Mean or G-mean combines the Sensitivity and Specificity in a single score, which ranges between 0 and 1 and is given by the following Equation 7[20].

$$\boldsymbol{G\text{-}mean} = \sqrt{Sensitivity \times Specificity} \tag{7}$$

The ROC AUC is interpreted as the probability that the scores given by a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. It ranges between 0.5 and 1 and it is considered in the following Figure 2 and expressions when calculating[20].

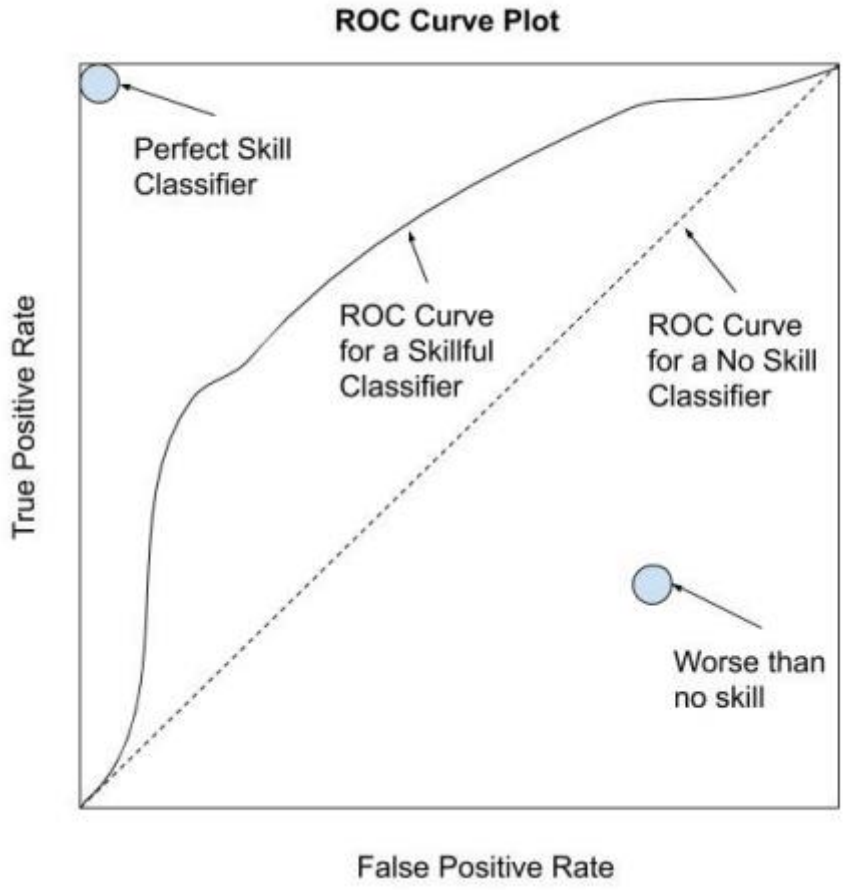

Figure 2: Depiction of a ROC Curve.

$$\boldsymbol{ROC\ Curve}: Plot\ of\ False\ Positive\ Rate\ (x)\ vs. True\ Positive\ Rate\ (y)$$

$$\boldsymbol{Perfect\ Skill}: A\ point\ in\ the\ top\ left\ of\ the\ plot$$

The Cohen Kappa (*k*) is the measure of the agreement between the model predictions and the actual class values as if happening by chance, which ranges between 0 and 1 and is given by the following Equation 8[8].

$$\boldsymbol{k} = \frac{2\ \times (TP \times TN - FN\ \times FP)}{(TP + FP) \times (FP + TN) + (TP + FN) \times (FN + TN)} \tag{8}$$

2.4 Related Solutions

There are several applications and services, capable of providing tools that can handle imbalanced classification. For instance, Scikit-Learn[21] 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*[22] which is related to Scikit-Learn and implements most of the necessary techniques. For R programming language, there exists the *ROSE*[23] and *imbalanced*[24] 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 autoML pipeline[25]. For instance, *Auto-sklearn*[26] 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[27]. *Auto-sklearn* does an automatic ensemble of the different models searched and applies a post-processing method, instead of discarding all the models searched[27]. It can do parallelization on a single computer or in a cluster on a limited time budget[28].

Additionally, *AutoKeras*[29] based on *Keras*[30], supports multi-modal and multi-task by searching for *deep learning* models[25]. *Neural Network Intelligence* (NNI)[31] developed by *Microsoft,* also integrates *Scikit-Learn* features that can do automated feature engineering, hyperparameter optimization, and *NAS*, becoming a powerful and lightweight toolkit for autoML[25]. *TPOT*[32] is a framework based on genetic programming able to only handle categorical parameters with the ability to create arbitrary complex pipelines makes it prone to

overfitting. To compensate for this, *TPOT* optimises a combination of high performance and low pipeline complexity[28].

In addition, there is also *Hyperopt-sklearn*[33] 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[28]. Finally, there is also *H2O AutoML*[34] 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 similar to *Auto-sklearn*[28]. Also, many big tech companies like *Microsoft*, *Amazon* and *Google* provide autoML services, such as *Microsoft Azure Automated Machine Learning*[35], *Amazon Web Services* (AWS) *SageMaker Autopilot*[36] and *Google Cloud Platform* (GCP) *AutoML*[37], 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[9]. 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[9]. It is mainly for binary classification and only builds models using the *Random Forest* learning algorithm[38].

2.4.1 Final Considerations

When analysing all these libraries/packages/frameworks, at this point there are not any advanced data cleaning methods in the context of autoML, most methods combine predefined operators with features naively, and there are few flexible approaches to the autoML pipeline[28]. 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[28]. 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[25].

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[25]. 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[39]. 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[40].

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 Value Analysis

This chapter is composed of four sections. First, is described the Opportunity Identification and Opportunity Analysis. Next, is presented the Value Offer with the Value for the Customer, the Perceived Value, and the Value Proposition. Then, it is applied the TOPSIS Method to compare similar related platforms that handle imbalance classification for different datasets domains.

3.1 Opportunity Identification

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 domains. Most of these domains are from the areas of fraud detection, claim prediction, churn prediction, default prediction, spam detection, anomaly detection, outlier detection, intrusion detection and conversion prediction. For example, in fraud detection, the goal is to detect fraud (rare cases) and no-fraud (abundant cases) transactions, with the desire to prioritise the classification of fraudulent transactions. This becomes a demanding problem that can be addressed from multiple novel balancing techniques that sometimes are challenging and time-consuming to obtain optimal results, as explained previously in the State of the Art chapter.

Many autoML solutions can be applied to imbalanced datasets to do classification. These solutions can be split into two groups, the code-free and the code-needed requirement to use. The former is a solution where the user does not need to code, it only needs to click on a few buttons on, for example, a webpage and the solution does automation of the machine learning classification pipeline in the loaded imbalanced datasets. Examples can be *Microsoft Azure Automated Machine Learning*, *AWS SageMaker Canvas*[41] (related to *AWS SageMaker Autopilot*) and *GCP AutoML*, among others. The latter is usually a software library for a particular programming language that needs to be imported for the project of the user that is

written in that programming language like *python* or *R*, for instance. Examples were already mentioned in the Related Solutions of the State of the Art chapter.

However, some of the autoML solutions already developed, either the code-free or the code-needed ones, sometimes may not be properly prepared for imbalanced classification because, for example, do not provide an appropriate metric to aim for imbalanced data, it only has a limited selection of the most used metrics, and they cannot be the more appropriate metrics to select to these scenarios, as it will be further discussed in the Solution Evaluation chapter. In addition, most of these autoML solutions can provide remarkable results for imbalanced data, nonetheless, they do not implement some specific techniques to handle imbalanced data and it can limit some greater results.

3.2 Opportunity Analysis

Several solutions address the automation of machine learning classification or handling imbalanced classification. However, when combining these two fields, there are a few solutions freely available. Furthermore, opportunity can be defined as "*an occasion or situation that makes it possible to do something that you want to do or have to do, or the possibility of doing something*"[42], therefore when this project combines the automation of machine learning with the specialisation on handling imbalanced data raises a segment in the market that has limited available tools to operate these multiple imbalanced scenarios[43]. The rising interest in these two areas can be noted in the number of *Google* search in *Google Trends* over the years, shown in Figure 3 and Figure 4.

Figure 3: Imbalanced Classification – Google Trends[44].

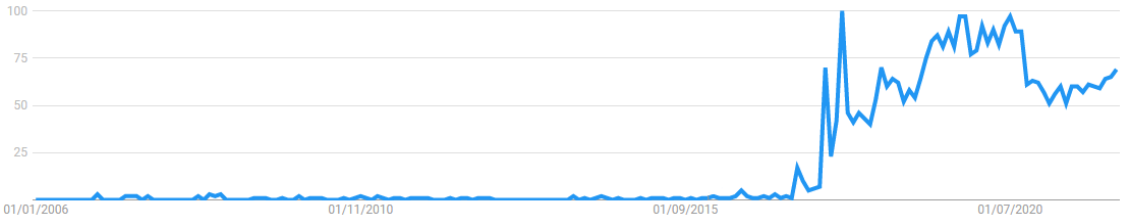

Figure 4: AutoML – Google Trends[45].

3.3 Value Offer

Value can be defined as the representation of the connection between customer satisfaction and cost, the relationship between the contribution of functions to the satisfaction of a need and the cost of functions[46]. In this project, customer satisfaction can be by obtaining remarkable results when properly analysing some relevant evaluation metrics and when comparing to state-of-the-art similar solutions. Oppositely, the cost can be the increasing time, computation resources or specialised knowledge needed to use the autoML solution.

As a small note, the *Quality Function Deployment* (QFD)[47] or the *Features from Accelerated Segment Test* (FAST)[48] are not appropriate to use on this project because the goal of this project is not to guarantee the specific requirements of each imbalanced dataset, rather make an open-ended solution that works for multiple datasets of different domains at the same time.

3.3.1 Value for the Customer

Customer value can be defined as the satisfaction that the customer has or expects to have when performing a certain action, considering the cost of that action[49]. Considering this definition, the potential client or rather the user of this free open-source software autoML solution, in this project, allows "*The freedom to run the program as you wish, for any purpose; The freedom to study how the program works and change it so that it does your computing as you wish; The freedom to redistribute copies so you can help others; The freedom to distribute copies of your modified versions to others – giving the whole community a chance to benefit from your changes*", established on the *four freedoms* of *Richard Stallman*, the founder of the *Free Software Foundation*[50].

3.3.2 Perceived Value

Perceived value can be defined as the overall assessment that the consumer makes of a product, weighing the relevant benefits and sacrifices[51]. It is expectable that any autoML solution should facilitate the tasks of doing, in this case, classification even for imbalanced datasets. The perceived value is affected by the easy-of-use, the evaluation metrics selection, the time to execute, the quality of the results and provided documentation of the autoML solution.

3.3.3 Value Proposition

Value propositions should not only express why the services, products, or solutions are better than the competition's, but also should be simple, clear, easy to absorb, credible, and adaptable to specific clients or segments[52]. In this context, the project aims to implement an autoML solution that can be applied specifically to handling imbalanced datasets, as previously mentioned.

Alexander Osterwalder proposed a value proposition framework[53]. This framework revolves around two larger entities, the customer profile, and the value map, which are visually presented on a canvas. The customer profile aims to identify the proposed gain of the system, the benefits which the customer expects and needs, pains, the risks that the customer may experience, and the customer's jobs, which represent the tasks that are trying to be done. As for the value map, it represents the gain creators, how the system creates and satisfies the gains of the customer, pain relievers, how the product or service alleviates customer pains, and the products and services, representing what functionalities and operations the system presents[53]. Next, it is presented the value proposition canvas for this project in Figure 5.

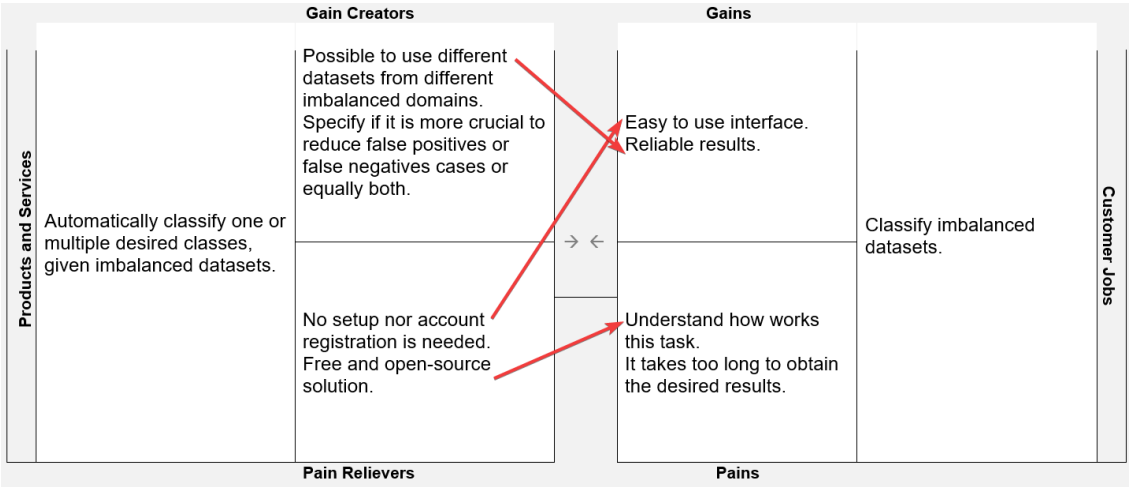

Figure 5: Value Proposition.

3.4 TOPSIS Method

The *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS) is a multi-criteria decision analysis method, which was originally developed by *Ching-Lai Hwang* and *Yoon*[54]. TOPSIS is based on the concept that the chosen alternative should have the shortest geometric distance from the positive ideal solution and the longest geometric distance from the negative ideal solution[54]. In this context, the aim is:

- To automatically classify two or more classes with imbalanced data provided.

The criteria that are going to be applied are:

- Easy-of-use
  - It measures if the solution is accessible to use or not.
- Metrics Selection
  - It measures if the metrics selection is appropriate.
- Results
  - It measures quantitively the quality of the results.
- Time of Execution
  - It measures how long it takes to execute the solution.
- Documentation
  - It measures the quality of the provided documentation after executing the solution.

It will be analysed some of the most used autoML solutions that can address this aim. The three autoML solutions selected to be compared are:

- Azure AutoML (short for Microsoft Azure automated machine learning)
- AWS Canvas (short for Amazon Web Service SageMaker Canvas)
- GCP AutoML (short for Google Cloud Platform AutoML)

The steps needed to apply the TOPSIS method are as follows[54]:

1. Create a matrix consisting of *M* alternatives and *N* criteria.

$$(a_{ij})_{M\times N}$$

2. Normalise the matrix.

$$\alpha_{ij} = \frac{a_{ij}}{\sqrt{\sum_{i=1}^{M}(a_{ij})^2}}$$

3. Calculate the weighted normalised decision matrix.

$$\chi_{ij} = \alpha_{ij} * \omega_j$$
$$\omega_j = \frac{w_j}{\sum_{j=1}^{N} w_j}$$
$$\sum_{j=1}^{N} \omega_j = 1$$

4. Determine the best and the worst alternative for each criterion.

$$\chi_j^b = \max_{i=1}^{M} \chi_{ij}$$
$$\chi_j^w = \min_{i=1}^{M} \chi_{ij}$$

5. Calculate the Euclidean distance between the target alternative and the worst alternative.

$$d_i^b = \sqrt{\sum_{j=1}^{N} (\chi_{ij} - \chi_j^b)^2}$$
$$d_i^w = \sqrt{\sum_{j=1}^{N} (\chi_{ij} - \chi_j^w)^2}$$

6. Calculate the similarity to the worst condition.

$$s_i = \frac{d_i^w}{d_i^w + d_i^b}$$

7. Rank alternatives according to the TOPSIS score in descending order.

3.4.1 Applying TOPSIS Method

The initial values attributed are expressed in Table 2.

Table 2: TOPSIS calculations.

weights	0.2	0.1	0.3	0.3	0.1
	Easy-of-use	**Metrics Selection**	**Results**	**Time of Execution**	**Documentation**
Azure AutoML	6	7	7	8	7
AWS Canvas	9	6	8	6	8
GCP AutoML	6	7	8	7	9

	Easy-of-use	**Metrics Selection**	**Results**	**Time of Execution**	**Documentation**
Azure AutoML	36	49	49	64	49
AWS Canvas	81	36	64	36	64
GCP AutoML	36	49	64	49	81
$\sum {X_{ij}}^2$	153	134	177	149	194
$(\sum X^2)^{1/2}$	12.37	11.58	13.30	12.21	13.93

	Easy-of-use	**Metrics Selection**	**Results**	**Time of Execution**	**Documentation**
Azure AutoML	0.49	0.60	0.53	0.66	0.50
AWS Canvas	0.73	0.52	0.60	0.49	0.57
GCP AutoML	0.49	0.60	0.60	0.57	0.65

	Easy-of-use	**Metrics Selection**	**Results**	**Time of Execution**	**Documentation**
Azure AutoML	0.098	0.060	0.159	0.198	0.050
AWS Canvas	0.146	0.052	0.18	0.147	0.057
GCP AutoML	0.098	0.060	0.18	0.171	0.065

A^*	0.146	0.060	0.18	0.198	0.065
$A^{'}$	0.098	0.052	0.159	0.147	0.050

The results obtained are:

- AWS Canvas with a score of 0.52
- Azure AutoML with a score of 0.47
- GCP AutoML with a score of 0.41

Based on these results, *AWS Canvas* is the best option. Consequently, is the best solution with the best score.

4 Solution Design

This chapter starts with the Requirements Analysis, which comprises the Functional and Non-functional Requirements that should be attained, and next, the System Architecture, which presents all the relevant documentation made for the solution. Then, the selected Datasets and the Technologies Choice to be used when developing and evaluating this solution.

4.1 Requirements Analysis

To understand the requirements analysis, first, requirements are capabilities and conditions to which the system and the project should reply. This analysis is split into two sections, first, the Functional Requirements and then the Non-functional Requirements.

4.1.1 Functional Requirements

The following functional requirements were identified in the analysis of the project and expressed in the following use cases:

- Use Case 1: Build a knowledge base with the dataset meta-features, the best combination of pre-processing technique and classification algorithm and the appropriate evaluation metrics of at least 50 datasets.
- Use Case 2: Recommend the best pre-processing techniques and classification algorithms for a certain dataset.

The following Figure 6 represents the use case diagram elaborated from the functional requirement previously defined.

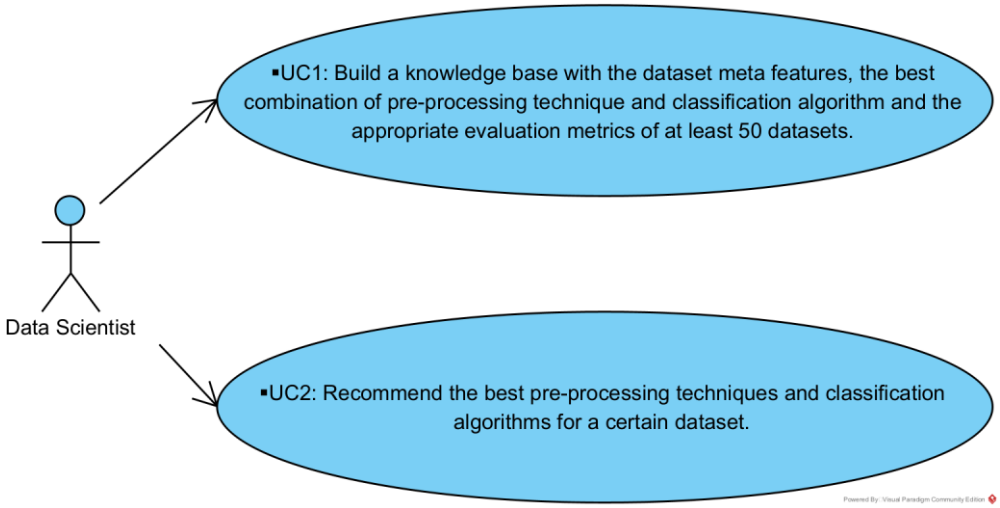

Figure 6: Use Case Diagram.

4.1.2 Non-functional Requirements

The non-functional requirements analysis was made according to the *FURPS+* Model[55]. Initially, this model was named *FURPS* which stands for Functionality, Usability, Reliability, Performance and Supportability requirements. Later, it was added the "plus," which extends into other requirements: Design, Implementation, Interface, and Physical requirements. It is also important to acknowledge that in the FURPS+ Model, all the different requirements that do not fit in the first category, the functionality requirements, are non-functional requirements.

The following non-functional requirements were identified in the analysis of the project.

- Usability requirements:
  - The user interface shall be simple and effective.
- Reliability requirements:
  - The user interface shall appropriately alert the user, even in unexpected situations.
- Performance requirements:
  - The user experience should not be degraded by the system, even with larger datasets loaded.
- Supportability requirements:
  - All components of the systems shall be modular, meaning that each component is not dependent on the technical specifications of another.
- "+" – Design, implementation, interface, and physical requirements:
  - Appropriate design patterns shall be used.

- A version control system shall be used.
- An incremental and iterative software development process shall be used.

4.2 System Architecture

This System Architecture is composed, initially, of the Activity Diagram to comprehend the initial architecture of the solution. Next, the Domain Model presents the different business concepts of the solution. Then, the alternative and final Component Diagram to comprehend the selected approach, and finally, the Sequence Diagram concludes the documentation of this system.

4.2.1 Activity Diagram

The solution should be able to receive an imbalanced dataset file prepared for classification, learn the data, and apply a classifier to predict with a low error margin. To better understand the high-granularity software architecture of the solution, Figure 7 presents an activity diagram of the solution.

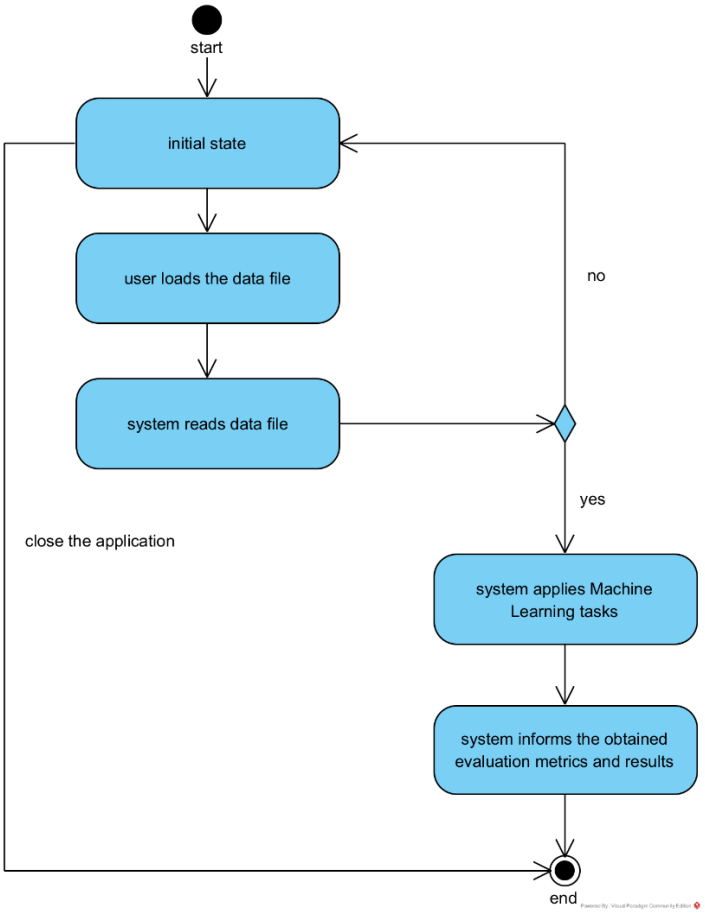

Figure 7: Activity Diagram.

First, the user starts the application and loads the desired imbalanced dataset file, then if the system reads it and is unsuccessful the system alerts and returns the user to the initial state. After, if succeeded, the system applies different machine learning tasks and when this step is concluded, the system informs the user of the obtained evaluation metrics and results. Finally, the user can quit the application.

4.2.2 Domain Model

Next, it is presented the Domain Model of the solution. Figure 8 represents the analysis of the identified Entities and their respective connections, thus explaining the logic underneath the different business concepts.

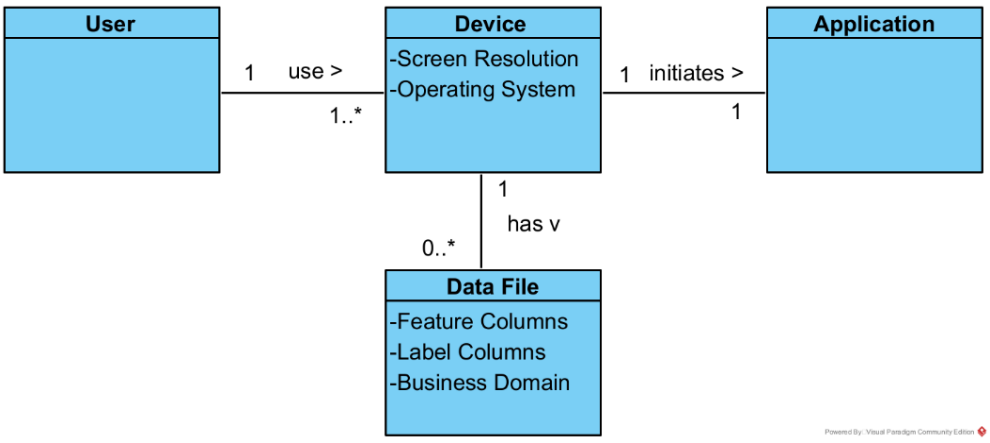

Figure 8: Domain Model.

The User uses one or more Devices, which can be a computer or a mobile device that has a certain screen resolution and an operating system. The Device can store multiple Data Files ready to be used by the Application. All Data Files should have the feature and label columns and represent a certain business domain.

4.2.3 Component Diagrams

An initial architecture of the solution was envisioned based on the previous requirements, and it is expressed as a component diagram in Figure 9.

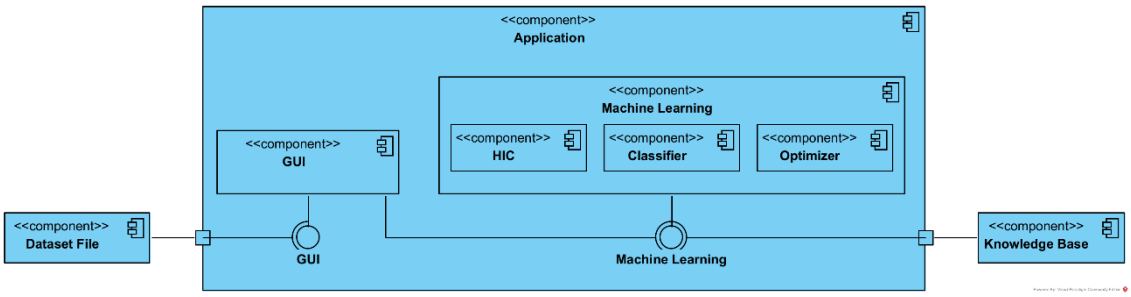

Figure 9: Initial Component Diagram.

A dataset file should be loaded in the application using the graphical user interface (GUI) of the application, and then, applied the necessary Machine Learning tasks where it is composed of the handling imbalanced classification (HIC), classifier and optimizer 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 optimizer component improves the previous results.

With the imbalanced techniques selected, the classification algorithm and the optimizations got, the machine learning component should write the results obtained to the knowledge base. However, as this initial architecture is simplistic and faces two problems, it is not appropriate to be the GUI responsible for reading the dataset file and neither the machine learning component is responsible for writing to the knowledge base. Therefore, a second, and more appropriate architecture was designed, the selected architecture, by addressing the two previous problems, present in Figure 10.

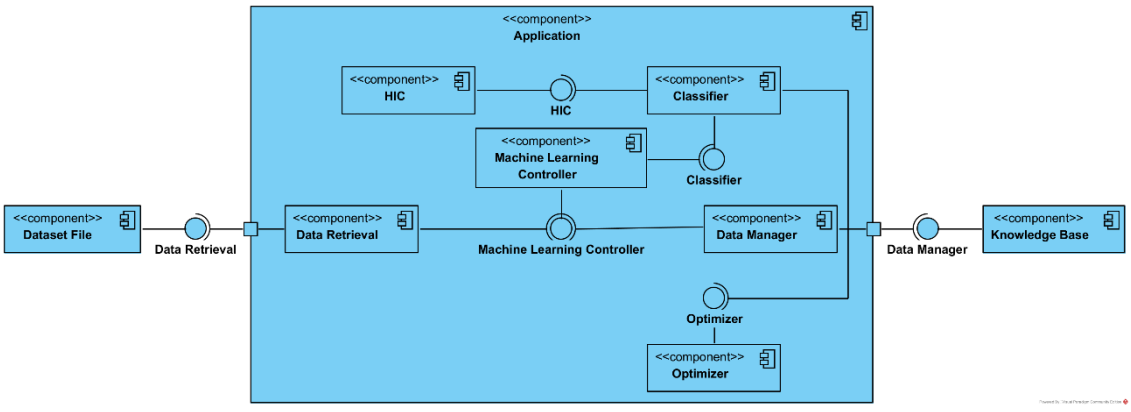

Figure 10: Final Component Diagram.

Now, there is a data retrieval component that is responsible for reading the dataset file and that is called by the machine learning controller component. The machine learning controller uses the classifier component that, in combination with the HIC and the optimizer, delivers the final classification model selected. When the model is prepared, the machine learning controller component uses the data manager component that is responsible for writing to the knowledge base.

4.2.4 Sequence Diagram

A final diagram was developed that presents the process view of the system that summarises all the previous documentation regarding this architecture in Figure 11.

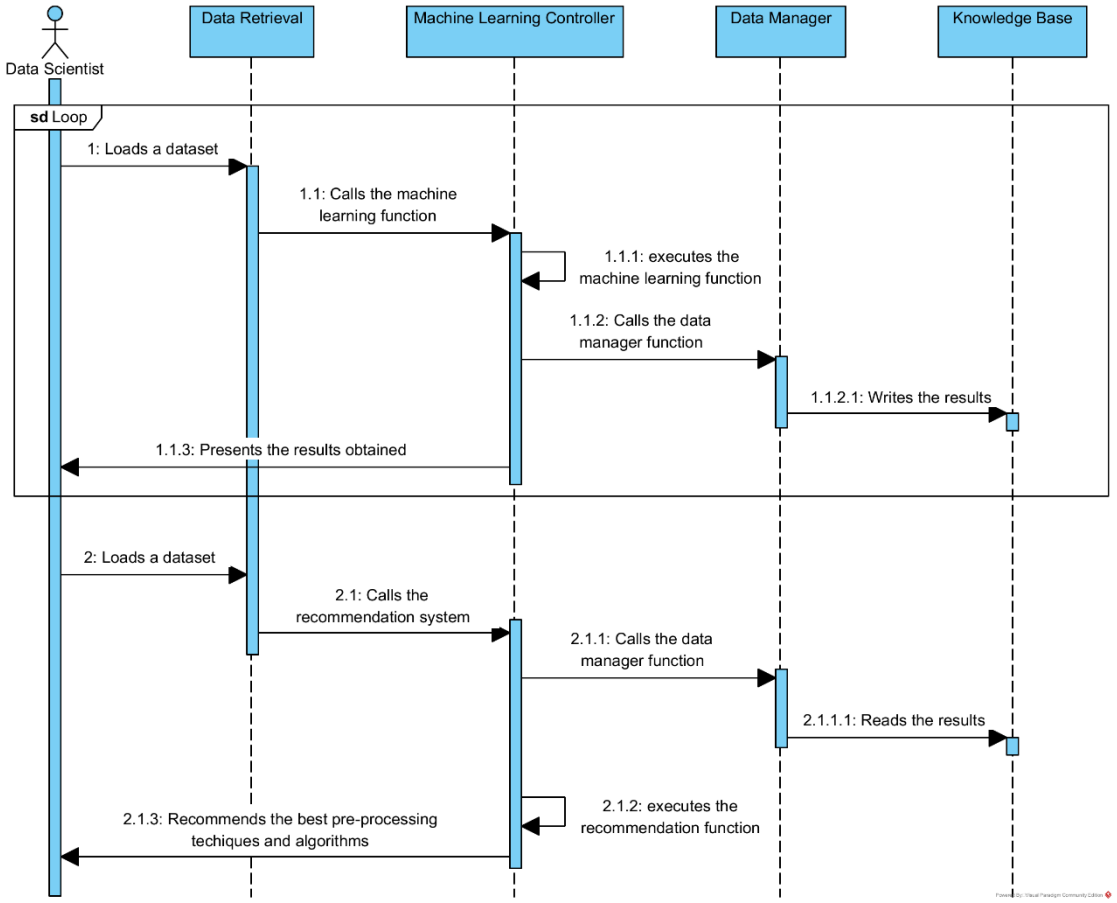

Figure 11: Sequence Diagram.

There are two different use cases that the user, identified as the data scientist, is envisioned to achieve with this system. The first one, regarding the initial construction of the knowledge base, is an iterative process with at least 50 different datasets that this user needs to make before the second use case. In this step, the goal is to have the knowledge base with enough meta-features and evaluation metrics results to properly be used by the recommendation engine.

In the second step, the user can now get recommendations from the system to a certain dataset that he loads. This recommendation is composed of a few combinations of pre-processing and classification algorithms that can be well adequate for the dataset that was loaded. This recommendation is possible by reading the previously obtained meta-features in the knowledge base and finding the previous similar dataset in terms of meta-features.

4.3 Datasets

The datasets are chosen from different business domains that have imbalanced data like fraud detection, claim prediction, churn prediction, default prediction, spam detection, anomaly detection, outlier detection, intrusion detection, conversion prediction and others. The aim is to always choose publicly available datasets without needing to do specific pre-processing tasks before using them. In addition, it was also ensured have a different ratio of proportions of imbalanced data across the diverse datasets.

Initially, it was analysed several candidate datasets from websites like *UCI Machine Learning Repository*[56], *KEEL – Knowledge Extraction based on Evolutionary Learning*[57], *OpenML*[58], *Kaggle*[59] and *Google Dataset Search*[60]. 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)[61] 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, for 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 pre-processing tasks that were not the point of this application to make. It was also selected datasets from the *KEEL* website getting 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.

4.4 Technologies Choice

For the selection of the programming language to use, there is *python*[62] and *R*[63], which are both free open-source and high-level programming languages that can be used for data manipulation and machine learning. Contrarily, the former is general-purpose object-oriented, and the latter is statistically computing but also with strong object-oriented capabilities. *Python* was selected because it was observed that it offers more support in developing autoML solutions and it has more versatility in developing desktop and web applications.

This application can be a GUI desktop application or a web application. For developing GUI applications, it was initially selected two easy-to-use *python* libraries, *PySimpleGUI*[64] and *DearPyGUI*[65]. *PySimpleGUI* wraps well-known GUI *python* libraries like *tkinter*[66], *Qt*[67], *WxPython*[68], and *Remi*[69] and simplifies developing the GUI. Similarly, *DearPyGUI* is another accessible GUI development library that is built on top of Dear ImGui[70], a *C++* library.

For developing web applications, it can be used two easy-to-use *python* libraries like *Flask*[71] a general-purpose web micro-framework or *streamlit*[72] which is intended for machine learning and data science applications. For deploying a web application, it can be used on cloud platforms like *Heroku*[73] or also *streamlit*. In the end, it was selected to do a desktop application with the assistance of *PySimpleGUI* because the objective of this project is to maximise the time of developing the machine learning core module and building and deploying a web application is more time-consuming than developing a simple GUI desktop application.

Another possible relevant technology to build the final application can be tools that help to parallelize the workload and utilise the graphics processing unit (GPU) for general-purpose processing instead of the central processing unit (CPU) and thus obtain faster results. To this effect, there is a multitude of options ranging from libraries that facilitate the distribution of workloads like *Apache Spark*[74], *Dask*[75], or *Ray*[76], or even more general-purpose libraries like *Tensorflow*[77] and *PyTorch*[78] albeit more tailored to *deep learning*.

Many of these libraries use the proprietary *Nvidia CUDA*[79] or open-source alternatives like *OpenCL*[80] and *OpenGL*[81]. The application is built in a *GIT*[82] repository that allows version control. To this effect, there is *Github*[83] and *Bitbucket*[84] that provide internet hosting for software development. Open-source applications like this project are more present at *Github*, and consequently, it was selected over *Bitbucket*, for this project.

Regarding the knowledge base, it can be built on an *SQL* or *NoSQL* database, or in *CSV* or *XSLX* files. The aim is to save the final evaluation metrics and results of the execution of the machine learning module on a certain dataset file, thus, if this was going to be stored in *SQL* or *NoSQL* the result was only one table or document correspondingly. Thus, the choice is to use *CSV* or *XSLX* files to store the learnt knowledge in the knowledge base that is used as guidance for future executions.

Finally, it can be used one *python* library, for instance, *PyInstaller*[85], capable of transforming a *python* program into an executable program that can be executed from the operating system

of the device without needing to have *python* installed. Thus, this application becomes not dependent on whether the device has *python* installed, to be executed.

5 Solution Implementation

The application is built in *python* and available, in a *GitHub* repository[86], as free and open-source software, licenced as *GPL 3.0*[87]. It is composed of two different functionalities, the development phase, and the recommendation phase, to address the two identified use cases reflected in the functional requirements stated in the previous Solution Design made. Consequently, it has also two distinct methods to interact with the user.

In the first functionality, the goal is to build the knowledge base by obtaining the meta-features of datasets, some adequate evaluation metrics, and the time of execution of selected classification algorithms in each dataset selected to be part of the knowledge base. In the second, with the assistance of the previous knowledge base constructed, the application can recommend the best combinations of pre-processing techniques and classification algorithms to handle a new imbalanced dataset imported.

These two phases are similarly engineered, as previously shown in the Sequence Diagram presented in the previous chapter. This recommendation should help the user to hopefully decide the best methods to correctly classify the imbalanced dataset imported.

The application has three main *python* files and two folders:

- "*ml.py*" file – where is made all the machine learning processing of the application.
- "*test_ml.py*" file – it is the file to be executed by the console application in the development phase.
- "*ui.py*" file – it is the user interface file to be executed in the recommendation phase.
- "*input*" folder – the place to put the datasets to be imported by the application.
- "*output*" folder – where are present the knowledge base files, the "*kb_characteristics.csv*," "*kb_results.csv*" and "*kb_full_results.csv*."

5.1 Development Phase

The user, in this phase, interacts with the application by executing, in the console, the "*test_ml.py*" file that has already presented the invocation of the "*execute_ml*" function, declared in the "*ml.py*" file. This function has two parameters to indicate the imported dataset, the former when importing a dataset by file and the latter when imported by *OpenML* dataset ID. If it is by file, it must preferably place the dataset file in the "*input*" folder and indicate the correct path of the dataset file.

Next, it is illustrated one example of importing a dataset in Figure 12, present in the "test.ml" file.

```
import sys
from datetime import datetime
from ml import execute_ml, execute_ml_test

print('\n\n-----------------------------------start -', datetime.now(), '---

#FILE OR OPENML

#FILE
dataset_name = "zoo-3.dat"
execute_ml(sys.path[0] + "/input/" + dataset_name, "")
#FILE

#OPENML
# execute_ml("", 975)
#OPENML

print('-----------------------------------finish -', datetime.now(), '------
```

Figure 12: *test.ml* code.

If it is by *OpenML* dataset ID, the user can go to the *OpenML* website and search for the dataset wanted and, if available, it should have the corresponding ID. One example of this is illustrated in Figure 13, in this case, the "*credit-g*" dataset has an ID of 31.

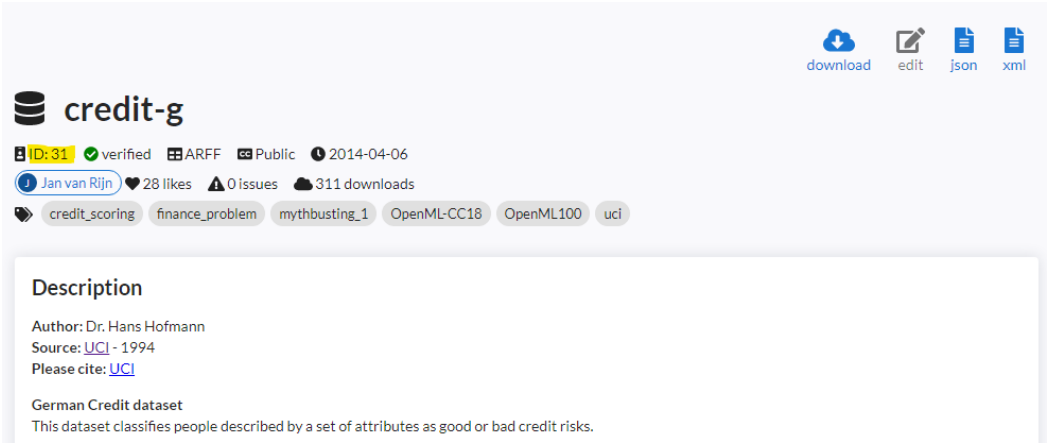

Figure 13: *credit-g* dataset in *OpenML*[88].

In a summarised manner and according to the activity diagram previously created, the "*execute_ml*" function, first, reads the imported dataset by file or by *OpenML* dataset ID, then, it extracts the meta-features information with the help of the Meta-Feature Extractor (MFE)[11] library, next it combines several different pre-processing techniques and classification algorithms to train, test and validate, and then, it writes the obtained results to the three knowledge base files of the application, and finally, it outputs to the console, the obtained results.

The last version of the "*execute_ml*" function, on the "*ml.py*" file, is presented in Figure 14.

```python
def execute_ml(dataset_location, id_openml):
    try:
        if dataset_location:
            df, dataset_name = read_file(dataset_location)
        elif id_openml:
            df, dataset_name = read_file_openml(id_openml)
        else:
            return False

        start_time = time.time()
        X, y, df_characteristics = features_labels(df, dataset_name)
        array_balancing = [
            "RandomOverSampler", "SMOTE", "SVMSMOTE",
            "SMOTETomek"
        ]
        resultsList = []
        i = 1
        for balancing in array_balancing:
            try:
                print("loading: ", i, " of ", len(array_balancing))
                i += 1
                balancing_technique = pre_processing(balancing)
                resultsList += classify_evaluate(X, y, balancing, balancing_technique, dataset_name)
            except Exception:
                traceback.print_exc()

        finish_time = (round(time.time() - start_time,3))
        best_result = find_best_result(resultsList)
        result_updated = write_results(best_result, finish_time)
        write_full_results(resultsList, dataset_name)
        write_characteristics(df_characteristics, best_result, result_updated)

        return dataset_name

    except Exception:
        traceback.print_exc()
        return False
```

Figure 14: *execute_ml* function on *ml.py* file.

One console output example is illustrated in Figure 15, where it is imported from the “car-good.dat” dataset[89].

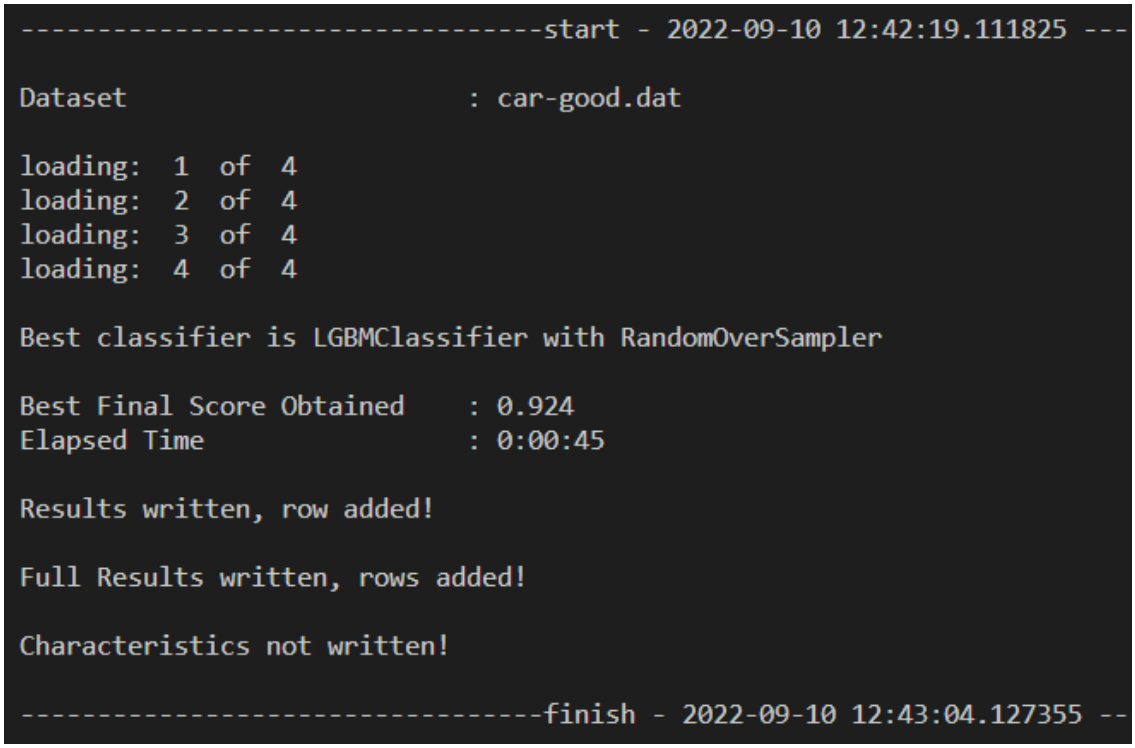

Figure 15: Console output of *execute_ml* function.

In this example, it is informed of the best combination of pre-processing technique and the corresponding classifier. In this case, it was the "*RandomOverSampler*" with the "*LGBMClassifier*." Then, it is displayed a final score that is the average of all evaluation metrics used and will be explained later, and the elapsed time in "*HH:mm:ss"* of that combination. Then, it ends by presenting if the application wrote or not in the three knowledge base files.

5.1.1 Reading and Extracting Knowledge from a Dataset

To have a robust knowledge base to be used in the recommendation phase and according to the first functional requirement, it was achieved 65 datasets imported, executed, and documented in the knowledge base files, surpassing the 50 datasets required. Most of these datasets were got with the help of the OpenML API and some by the *KEEL* website, as previously described in the Datasets section of the Solution Design. 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.

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. This can be observed in Figure 16.

```
mfe = MFE(random_state=42,
          groups=["complexity", "concept", "general", "itemset", "landmarking", "model-based", "statistical"],
          summary=["mean", "sd", "kurtosis","skewness"])

mfe.fit(X.values, y.values)
ft = mfe.extract()
```

Figure 16: The use of MFE Library.

These meta-features groups and summary functions are already described in the State of the Art chapter. 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.

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. This was attained with the function "*get_dummies*" of *Pandas* [90] library, as is present in Figure 17.

```
encoded_columns = []
for column_name in X.columns:
    if (X[column_name].dtype == object or X[column_name].dtype.name == 'category' or
        X[column_name].dtype == bool or X[column_name].dtype == str):
        encoded_columns.extend([column_name])

if encoded_columns:
    X = pd.get_dummies(X, columns=X[encoded_columns].columns, drop_first=True)
```

Figure 17: The use of the *get_dummies* function of *Pandas* library.

It is important to note that it was used the parameter "*drop_first*" with "*True*" gets the *k-1* indicators values out of *k* categorical levels by removing the first level. This means that it removes redundant columns that cause multi-collinearity. For example, in binary columns, the value of "1" in one column after obtaining the indicator values automatically implies "0" in the other column.

5.1.2 Pre-Processing Techniques and Classification Algorithms Used

This process started by executing 19 pre-processing techniques and 1 without any pre-processing technique combined with 11 classification algorithms, resulting in 220 different combinations. The 19 pre-processing techniques used, as of the time of writing, are all available in the *Imbalanced Learn* library [91], are:

- Under-sampling techniques:
  - *ClusterCentroids,*
  - *CondensedNearestNeighbour,*
  - *EditedNearestNeighbours,*
  - *RepeatedEditedNearestNeighbours,*
  - *AllKNN,*
  - *InstanceHardnessThreshold,*
  - *NearMiss,*
  - *NeighbourhoodCleaningRule,*
  - *OneSidedSelection,*
  - *RandomUnderSampler,*
  - *TomekLinks.*
- Over-sampling techniques:
  - *RandomOverSampler,*
  - *SMOTE,*
  - *ADASYN,*
  - *BorderlineSMOTE,*
  - *KMeansSMOTE,*
  - *SVMSMOTE.*
- Combination of over- and under-sampling techniques:
  - *SMOTEENN,*
  - *SMOTETomek.*

The 11 classification algorithms functions used with the help of the *Scikit-Learn, LightGBM*[92] and *XGBoost*[93] libraries are:

- LogisticRegression,
- GaussianNB,
- SVC,
- KNeighborsClassifier,
- LGBMClassifier,
- XGBClassifier,
- RandomForestClassifier,
- ExtraTreesClassifier,
- AdaBoostClassifier,
- BaggingClassifier,
- GradientBoostingClassifier.

The parameters used in all these pre-processing and classification functions, when available, were the "*random_state*" and the "*n_jobs*." In the former, when it is used the same value in all functions, it was chosen the widely used value of "42" which is the seed that guarantees reproducibility, meaning that executing the same machine learning functions more than once, will always result in the same scores. The latter, when equal to "-1", the machine learning function will use all the processors of the machine during the cross-validation step.

Specifically, in the classification algorithm function, the aim here was to not use any specific hyperparameter that can improve the scores got to a specific dataset but can degrade the performance of other datasets, thus the aim was to use the most generic and adequate hyperparameters that can work across all the datasets for this scenario.

Last, 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. Then, with the pre-processing task done, implicating that one class was under- or over-sampled, and the classification algorithm selected, the application does cross-validation with the use of the "*cross_validate*" function of the *Scikit-Learn* library, as illustrated in Figure 18.

```
model = make_pipeline(
    balancing_technique,
    classifier
)

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

scoring = {
    'balanced_accuracy': 'balanced_accuracy',
    'f1': 'f1',
    'roc_auc': 'roc_auc',
    'g_mean': make_scorer(geometric_mean_score, greater_is_better=True),
    'cohen_kappa': make_scorer(cohen_kappa_score, greater_is_better=True)
    }

scores = cross_validate(model, X, y.values.ravel(), scoring=scoring,cv=cv, n_jobs=-1)
```

Figure 18: The use of the *cross_validate* function.

To this effect, in the "*estimator*" parameter, the model chosen is a machine learning pipeline that includes the pre-processing task and the classifier selected, with the function "*make_pipeline*" of the *Scikit-Learn* library that guarantees that each step of the pipeline is constrained to the data available for the evaluation, such as the training dataset or each fold of the cross-validation procedure. Then it is passed the features "*X*" and target "*y*" columns of the

dataset and next in the "*scoring*" parameter it is specified the 5 evaluation metrics to be used, as will be further enumerated.

Then in the "*cv*" parameter, the cross-validation splitting strategy, it was chosen the "*RepeatedStratifiedKFold*" function of the *Scikit-Learn* library that repeats a Stratified K-Fold cross-validator several times with different randomization 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.

5.1.3 Process of Discarding the Worst Performant Combinations

Iteratively, it was discarded some worst-performing pre-processing techniques and classification algorithms. 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*,
- *Cohen Kappa*.

The distinction made to categorise each result of each combination was the average score of all these 5 selected metrics. As previously mentioned, the starting point of the application had 220 combinations with 19 pre-processing techniques and 1 more with no pre-processing technique, combined with 11 classification algorithms. It was iteratively discarded several combinations in five times/steps. Each combination for each dataset processed achieves a certain *final score*, the average of the 5 metrics, and a corresponding ranking position, for example, position 22 of 220 total combinations.

Then, when some datasets were randomly chosen and processed, it was analysed the various positions of each combination by grouping all the different rank positions, first by the pre-processing technique and then by the classifier. Next, it was examined the values above the third quartile (75% to 100%) of the distribution of the two lists of each mentioned group and it was studied the possibility of discarding the combinations that have a pre-processing technique or a classifier that is closer, in terms of rank position, to the maximum (100%) of the list than the third quartile value (75%) of the list. This analysis occurred after importing at least two datasets after the previous step or the beginning and examined if any pre-processing technique or classifier in the corresponding lists was to be discarded or not based on the previous explanation.

To better understand what happens when discarding one combination, two simple scenarios will be explained. In the beginning, when it starts with 220 combinations and it is found, for example, one pre-processing technique that is going to be discarded based on the previous explanation, when discarding it, it is discarded 11 combinations. This happens because there

are the initial 11 classifiers with that pre-processing technique. A comparable situation occurs when it is found, in the same situation, for example, one classifier that is going to be discarded, when discarding it, it is discarded 20 combinations. This happens because there are initially 19 pre-processing techniques and 1 with no pre-processing technique.

In the first step, after 3 datasets imported and processed, it was discarded all the combinations that have the following pre-processing techniques or classifiers:

- Pre-processing techniques:
  - SMOTEENN,
  - CondensedNearestNeighbour,
  - InstanceHardnessThreshold,
  - ClusterCentroids,
  - NearMiss.
- Classifiers:
  - LogisticRegression,
  - GaussianNB,
  - SVC.

After this first iteration, by removing 5 pre-processing techniques and 3 classifiers, the result was the remaining 14 pre-processing techniques, 1 with no pre-processing technique, and 8 classifiers resulting in 120 combinations.

Then, in the second step, after 2 datasets imported and processed, it was discarded all the combinations that have the following pre-processing techniques or classifiers:

- Pre-processing techniques:
  - OneSidedSelection,
  - “Without pre-processing,”
  - KMeansSMOTE.
- Classifiers:
  - AdaBoostClassifier,
  - KNeighborsClassifier.

After this second iteration, by removing 2 pre-processing techniques, 1 “without pre-processing” and 2 classifiers, the result was the remaining 12 pre-processing techniques and 6 classifiers, resulting in 72 combinations.

Then, in the third step, after 3 datasets imported and processed, it was discarded all the combinations that have the following pre-processing techniques or classifiers:

- Pre-processing techniques:
  - EditedNearestNeighbours,
  - AllKNN,
  - RepeatedEditedNearestNeighbours,

        - NeighbourhoodCleaningRule,
        - RandomUnderSampler.
- Classifiers:
    - BaggingClassifier,
    - ExtraTreesClassifier.

After this third iteration, by removing 5 pre-processing techniques and 2 classifiers, the result was the remaining 7 pre-processing techniques and 4 classifiers, resulting in 28 combinations.

Then, in the fourth step, after 14 datasets imported and processed, it was discarded all the combinations that have the following pre-processing techniques or classifiers:

- Pre-processing techniques:
    - BorderlineSMOTE,
    - TomekLinks.
- Classifier:
    - RandomForestClassifier.

After this fourth iteration, by removing 2 pre-processing techniques and 1 classifier, the result was the remaining 5 pre-processing techniques and 3 classifiers, resulting in 15 combinations.

Then, in the fifth and last step, after 9 datasets imported and processed, it was discarded all the combinations that have the following pre-processing technique:

- Pre-processing technique:
    - ADASYN.

After this fifth iteration, by removing 1 pre-processing technique, the result was the remaining 4 pre-processing techniques and 3 classifiers, resulting in 12 combinations.

The rest of the datasets imported and processed were not needed to have further steps of discarding more combinations, because it was not found any worse performant pre-processing technique or classifier based on the previous explanation and the result of 12 combinations is already relatively few combinations to discard further.

5.1.4 Write Results to Knowledge Base files

When it is time to write the obtained results, there are three knowledge base files. The application uses the functions "*write_characteristics*" to write in "*kb_characteristics.csv*," "*write_results*" to write in "*kb_results.csv*" and "*write_full_results*" to write in "*kb_full_results.csv*." In the "*write_characteristics*" and "*write_full_results*" functions, in this development phase, if there are already previous results present in the knowledge base files regarding a certain dataset, these functions only write if the newly obtained results are better

than previous ones. This distinction between better or worse results is made by the final score, meaning that is the average of all the 5 selected metrics score.

All these knowledge base files have the first column indicating the name of the dataset that was imported. For the "*kb_characteristics.csv*," then follows all the 257 meta-features obtained by the MFE library, and then, the best combination of pre-processing technique and classification algorithm that was achieved for each imported dataset.

Afterwards, the "*kb_results.csv*" also has the previously mentioned best combination, then the time, in seconds, that took to execute that combination, after, there are the 5 selected metrics and the corresponding standard deviation of the cross-validation of each one, and finally, there is a column to indicate the total elapsed time, in "*HH:mm:ss"* format, that counts the time of all the combinations that were executed.

Last, the "*kb_full_results.csv*" resembles the "*kb_results.csv*" however, here is written all the combinations to each imported dataset are to observe the time and metrics achieved in each combination. Additionally, the last column, instead of the total elapsed times, indicates the average of all the recorded metrics. Contrary to the two other knowledge base files, this one is guaranteed in order from the best global final score (average of the final metrics score) to the worst ones.

5.2 Recommendation Phase

The user, in this phase, interacts with the application by executing, in the console, the "*ui.py*" file that launches a GUI desktop application with the support of the *PySimpleGUI* library, as illustrated in Figure 19.

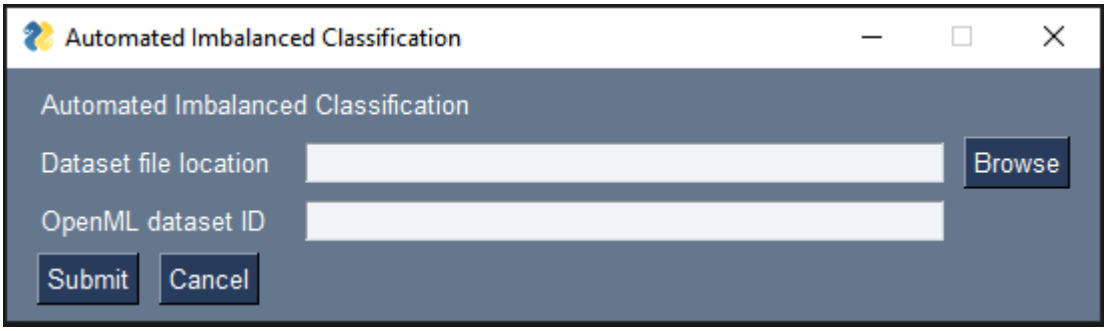

Figure 19: GUI Application for Recommendation Phase.

In this phase, the goal of the application is to deliver recommendations of the best combinations of pre-processing techniques and classification algorithms to be used in a certain imported dataset, as stated in the second use case addressed in the functional requirements.

5.2.1 Reading the Dataset

Similarly to the previous phase, it is possible to import the dataset by file, located in any place of the computer, or by *OpenML* dataset ID. However, if the user accidentally fills both fields or does not fill any, the application alerts it, as shown in Figure 20 and Figure 21, correspondingly.

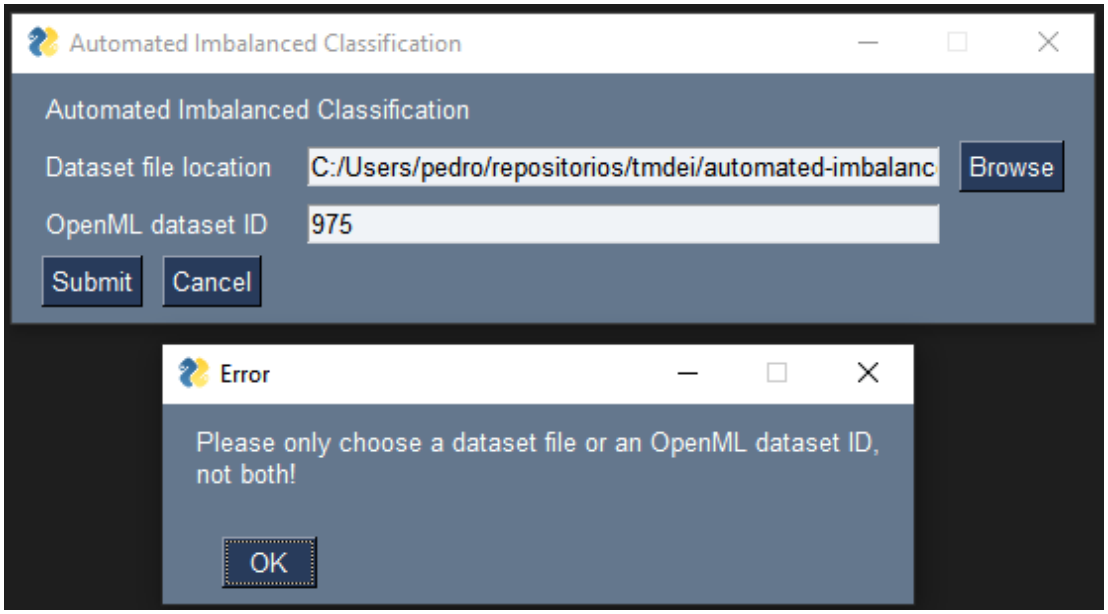

Figure 20: GUI Alert – fill both fields.

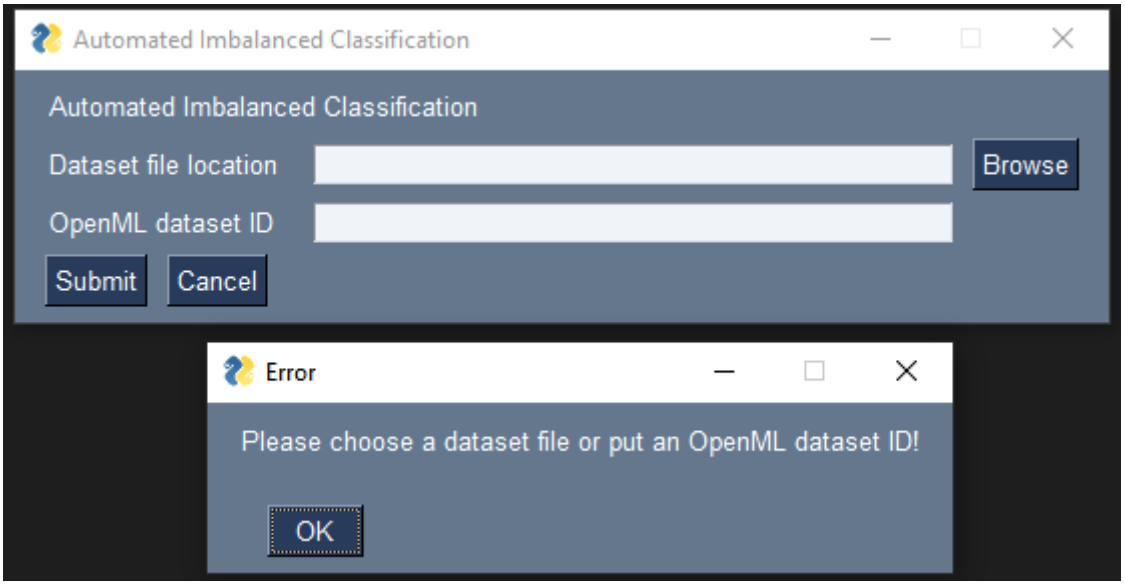

Figure 21: GUI Alert – not fill any field.

Then, when the user indicates only one option to import the dataset, of the two previous ones mentioned, he can "Submit," and the application executes the "*execute_byCharacteristics*" function present in the "*ui.py*" file, illustrated in Figure 22.

```
if values['file']:
    str_output = execute_byCharacteristics(values['file'], "")
elif values['omid']:
    str_output = execute_byCharacteristics("", values['omid'])

sg.Popup(str_output, keep_on_top=True, title='Recommendations')
```

Figure 22: Calling *execute_byCharacteristics* function in *ui.py*.

5.2.2 Calculating the Best Recommendations

The previously stated "*execute_byCharacteristics*" function is declared in the "*ml.py*" file and is presented in Figure 23.

```
def execute_byCharacteristics(dataset_location, id_openml):
    try:
        if dataset_location:
            df, dataset_name = read_file(dataset_location)
        elif id_openml:
            df, dataset_name = read_file_openml(id_openml)
        else:
            return False

        X, y, df_characteristics = features_labels(df, dataset_name)

        write_characteristics(df_characteristics, None, False)
        df_dist = get_best_results_by_characteristics(dataset_name)
        str_output = display_final_results(df_dist)

        return str_output

    except Exception:
        traceback.print_exc()
        return False
```

Figure 23: *execute_byCharacteristics* function of *ml.py* file.

First, the "*execute_byCharacteristics*" starts by extracting the meta-features and encoding the categorical columns, if there are any, of the dataset imported, equally to the development phase. Then, in the "*get_best_results_by_characteristics*" function it is computed the *Frobenius* norm (the Euclidian norm of two vectors) uses the "*linalg.norm*" function from the *NumPy*[94] library, which, in this case, is the average of all *Euclidian* distances (vectors) of each meta-feature extracted between the current imported dataset and all the previous imported datasets.

This takes into consideration the previously processed 257 meta-features in the development phase, in the "*kb_characteristics.csv*" file. Some meta-feature values can be null, negative, or positive infinity and those values were dropped from the vectors. The *Frobenius* norm can be expressed as Equation 9 and the *Euclidian* norm as Equation 10.

$$\|A\|_F = \left[\sum_{ij} abs(a_{ij})^2\right]^{1/2} \quad (9)$$

$$\|\boldsymbol{x}\|_2 = \left(\sum (\boldsymbol{w_i}|(\boldsymbol{u_i} - \boldsymbol{v_i})^2)\right)^{\boldsymbol{1/2}} \quad (10)$$

Next, it is selected the three smaller average values, 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 pre-processing techniques and classification algorithms that are distinct and were recorded as the better performant ones, in the development phase, for those datasets.

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 it to the application, when it finalises all the calculations, it informs, in this example, that (SVMSMOTE, GradientBoostingClassifier), (SMOTE, GradientBoostingClassifier), (SMOTE, XGBClassifier) were the best three combinations of pre-processing techniques and classification algorithms correspondingly, as illustrated in Figure 24.

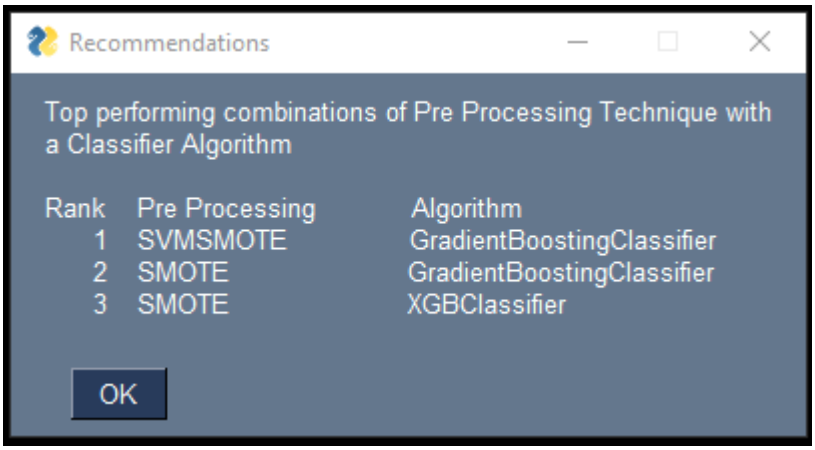

Figure 24: GUI recommendations example.

The application also outputs information to the console with more detailed and technical information, as illustrated in Figure 25.

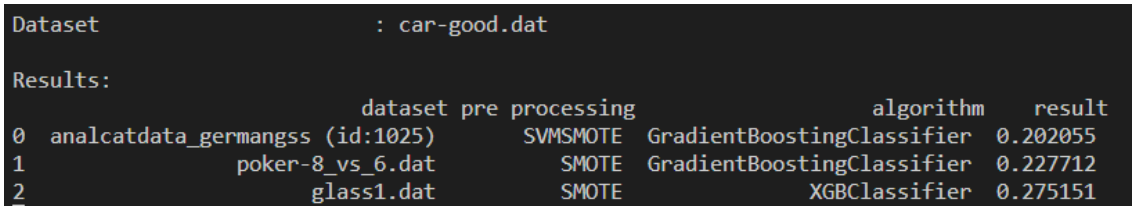

Figure 25: GUI recommendations output example.

For this case, those recommendations were given because the "*analcatdata_germangss*" (*OpenML* ID: 1025)[95], "*poker-8_vs_6.dat"*[96] and *"glass1.dat"*[97] datasets had the lowest *Euclidian* distances, it was obtained 0.202055, 0.227712 and 0.275151, correspondingly. Those datasets, in the development phase, had each of those best combinations of pre-processing techniques and classifiers. For instance, the "*analcatdata_germangss*" dataset, in the development phase, achieved the best final score (average of all evaluation metrics) with the (SVMSMOTE, GradientBoostingClassifier) combination.

It was also experimented to recommend the best combinations of pre-processing techniques and classification algorithms by multiclass classification instead of relying on the *Euclidean* distances from the meta-features values of each dataset. This multiclass classification task used the meta-features values of each dataset as features and the previously mentioned combination as the target. The problem occurs that for the 65 available datasets (number of instances/rows), by having 12 different target combinations with 257 meta-features values, this was proven to hinder the performance and take more processing time. Therefore, it was not the selected approach for these conditions.

6 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, by the recommendation phase, with the results that are acquired by the development phase. The latter is achieved by analysing and comparing the final recommended results with other results publicly available state-of-the-art papers or that is accomplished by using certain state-of-the-art autoML solutions.

This evaluation aims to know if the results achieved in the recommendation phase are at least comparable or better, first, to the development phase (internal evaluation) and then, to other similar state-of-the-art papers or autoML solutions (external evaluation). The results are considered better when comparing the same appropriate evaluation metrics, they are greater than other ones, or the execution time is smaller than the other ones.

Therefore, it was formulated four research hypotheses (RH) for this evaluation, the first two for the internal evaluation and the other two for the external evaluation, they are:

- RH1: The evaluation metrics achieved by the recommendation phase of the implemented application have comparable or greater values than the development phase of the implemented application.
- RH2: The time of execution achieved by the recommendation phase of the implemented application is comparable to or smaller than the development phase of the implemented application.
- RH3: The evaluation metrics achieved by the recommendation phase of the implemented application have comparable or greater values than similar state-of-the-art papers and/or autoML solutions.
- RH4: The time of execution achieved by the recommendation phase of the implemented application is comparable to or smaller than similar state-of-the-art papers and/or autoML solutions.

Thus, the goal of these two evaluation steps is to rigorously evaluate the implemented application, as well as the formulation of the four research hypotheses to clarify if the objectives previously defined are met or not. 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.

In the internal and external evaluation, the evaluation metrics used to evaluate the different solutions are 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.

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 are:

- dis (OpenML ID: 40713)[98],
- musk (OpenML ID:1116)[99],
- mfeat-fourier (OpenML ID:971)[100],
- Satellite (OpenML ID:40900)[101],
- arsenic-male-bladder (OpenML ID:947)[102],
- analcatdata_apnea2 (OpenML ID:765)[103],
- regime_alimentaire (OpenML ID:42172)[104],
- page-blocks0.dat[105],
- dgf_test (OpenML ID:42883)[106],
- cpu_small (OpenML ID:735)[107],
- analcatdata_birthday (OpenML ID:968)[108],
- optdigits (OpenML ID:980)[109],
- kr-vs-k-zero_vs_eight.dat[110],
- analcatdata_lawsuit (OpenML ID:450)[111],
- JapaneseVowels (OpenML ID:976)[112].

6.1 Internal Evaluation

To conduct the internal evaluation, first, in the development phase of the implemented application, it was observed the best combination of pre-processing technique and classification algorithm, using the average of all the selected evaluation metrics (final score) and the time of execution, of 15 datasets randomly selected.

Then, in the recommendation phase of the application, it was observed the three recommended combinations of pre-processing techniques and classification algorithms and the execution time, all these also to the same 15 datasets.

6.1.1 Research Hypothesis Number One

Regarding the evaluation metrics achieved from the recommendation phase compared to the development phase concerning the 15 datasets, the recommendation phase will always return worse or, in the best scenario, equal to the best one got by the development phase. 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.

Additionally, all the possible recommended combinations, by the recommendation phase, are the ones that the development phase previously calculated. Consequently, it is enough for the recommendation phase to return one combination that is not the best, to one of the 15 datasets that will cause the worst scores of metrics concerning the development phase.

To be certain of this effect, it was evaluated the 15 datasets by the two phases of the implemented application, and it was concluded that in some datasets it is not recommended the combination that would cause the best score, as explained further. Therefore, the scores accomplished by the recommendation phase are worse than the development phase, hence, the RH1 was not achieved.

However, since the recommendation phase was sentenced to fail, it was done a complementary analysis. For each dataset, it was investigated all three recommended combinations, by the recommendation phase, concerning the combination best scored by the development phase. All those combinations are on the following Table 3 and Table 4. The fields highlighted in green are the combinations that matched the best-scored combination by the development phase, and the fields highlighted in yellow are partially matched combinations where the match occurs only by the pre-processing technique or the classifier algorithm.

Table 3: Combination best scored by the development phase.

Dataset	Pre-processing technique	Classifier Algorithm
dis	RandomOverSampler	GradientBoostingClassifier
musk	RandomOverSampler	XGBClassifier
mfeat-fourier	SMOTE	GradientBoostingClassifier
Satellite	SMOTETomek	LGBMClassifier
arsenic-male-bladder	RandomOverSampler	LGBMClassifier
analcatdata_apnea2	RandomOverSampler	GradientBoostingClassifier
regime_alimentaire	SVMSMOTE	LGBMClassifier
page-blocks0.dat	RandomOverSampler	XGBClassifier
dgf_test	RandomOverSampler	XGBClassifier
cpu_small	SMOTETomek	LGBMClassifier
analcatdata_birthday	SVMSMOTE	XGBClassifier
optdigits	SVMSMOTE	XGBClassifier
kr-vs-k-zero_vs_eight.dat	RandomOverSampler	GradientBoostingClassifier
analcatdata_lawsuit	RandomOverSampler	LGBMClassifier
JapaneseVowels	SVMSMOTE	LGBMClassifier

Table 4: Recommended combinations by the recommendation phase.

Dataset	Recommendation number 1		Recommendation number 2		Recommendation number 3	
	Pre-processing technique	Classifier Algorithm	Pre-processing technique	Classifier Algorithm	Pre-processing technique	Classifier Algorithm
dis	RandomOverSampler	LGBMClassifier	RandomOverSampler	GradientBoostingClassifier	SMOTE	XGBClassifier
musk	RandomOverSampler	XGBClassifier	SMOTETomek	LGBMClassifier	SVMSMOTE	LGBMClassifier
mfeat-fourier	SVMSMOTE	GradientBoostingClassifier	SVMSMOTE	XGBClassifier	RandomOverSampler	GradientBoostingClassifier
Satellite	SMOTE	GradientBoostingClassifier	SMOTETomek	LGBMClassifier	RandomOverSampler	XGBClassifier
arsenic-male-bladder	RandomOverSampler	LGBMClassifier	SMOTETomek	GradientBoostingClassifier	SMOTETomek	LGBMClassifier
analcatdata_apnea2	RandomOverSampler	GradientBoostingClassifier	RandomOverSampler	LGBMClassifier	SVMSMOTE	LGBMClassifier
regime_alimentaire	SVMSMOTE	XGBClassifier	SVMSMOTE	GradientBoostingClassifier	RandomOverSampler	LGBMClassifier
page-blocks0.dat	SMOTE	LGBMClassifier	SMOTETomek	LGBMClassifier	SVMSMOTE	LGBMClassifier
dgf_test	SVMSMOTE	LGBMClassifier	SMOTETomek	LGBMClassifier	SMOTETomek	GradientBoostingClassifier
cpu_small	SMOTETomek	GradientBoostingClassifier	SMOTE	LGBMClassifier	SVMSMOTE	LGBMClassifier
analcatdata_birthday	RandomOverSampler	LGBMClassifier	RandomOverSampler	GradientBoostingClassifier	SMOTETomek	GradientBoostingClassifier
optdigits	RandomOverSampler	GradientBoostingClassifier	SVMSMOTE	GradientBoostingClassifier	SMOTE	XGBClassifier
kr-vs-k-zero_vs_eight.dat	SVMSMOTE	XGBClassifier	RandomOverSampler	GradientBoostingClassifier	SMOTE	XGBClassifier
analcatdata_lawsuit	SVMSMOTE	GradientBoostingClassifier	SVMSMOTE	XGBClassifier	RandomOverSampler	GradientBoostingClassifier
JapaneseVowels	RandomOverSampler	GradientBoostingClassifier	RandomOverSampler	XGBClassifier	SMOTETomek	LGBMClassifier

There are 3 datasets (*musk*, *arsenic-male-bladder* and *analcatdata_apnea2*) that, the recommendation phase, got right at the first recommendation according to the development phase. Then, there are also 3 datasets (*dis*, *Satellite* and *kr-vs-k-zero_vs_eight.dat*) that the best recommendation is on the second recommendation. Next, there are 6 datasets (*mfeat-fourier*, *regime_alimentaire*, *cpu_small*, *optdigits*, *analcatdata_lawsuit* and *JapaneseVowels*) that one part of the combination (pre-processing technique or classifier algorithm) was got right. Then, there are the remaining 3 datasets (*page-blocks0.dat*, *dgf_test* and *analcatdata_birthday*) that were not right in any of the combinations or partial combinations.

6.1.2 Research Hypothesis Number Two

Regarding the time of execution achieved from the recommendation phase compared to the development phase concerning the 15 datasets, all these values are expressed in the following Table 5, and all the values of time are in the "HH:mm:ss" format.

Table 5: Comparison of execution times of the recommendation and development phases.

Dataset	Execution time in the recommendation phase	Execution time in the development phase
dis	00:00:25	00:03:59
musk	00:04:08	00:55:34
mfeat-fourier	00:00:13	00:08:12
Satellite	00:00:57	00:10:30
arsenic-male-bladder	00:00:02	00:00:36
analcatdata_apnea2	00:00:01	00:00:17
regime_alimentaire	00:00:01	00:00:16
page-blocks0.dat	00:00:58	00:03:49
dgf_test	00:01:01	00:06:55
cpu_small	00:03:20	00:08:54
analcatdata_birthday	00:00:01	00:00:17
optdigits	00:01:19	00:11:49
kr-vs-k-zero_vs_eight.dat	00:00:06	00:00:48
analcatdata_lawsuit	00:00:01	00:01:40
JapaneseVowels	00:05:20	00:23:41

Consequently, for all the 15 datasets, the recommendation phase achieved 17 minutes and 53 seconds (1073 total seconds) of the total amount of execution time and the development phase achieved 2 hours 17 minutes and 17 seconds (8237 total seconds) of execution time. Thus, when dividing the total time achieved in the recommendation phase by the development phase, it is concluded that the execution time of the recommendation phase is smaller/faster with only 13.03% of the time of the development phase. Therefore, the RH2 was successfully achieved.

6.2 External Evaluation

To perform the external evaluation, it was first analysed relevant state-of-the-art papers that could address these 15 selected datasets with a machine learning pipeline that adopted similar pre-processing techniques and/or classification algorithms, with a similar machine learning validation used (Stratified K-Fold cross-validator) and with the same evaluation metrics selected. Crossing all these factors, especially these 15 datasets, was not found in any paper without ignoring relevant factors that would cause a compromised comparison of the attained results.

Consequently, the second step was to select one autoML application that could be executed for these 15 datasets with the same machine learning pipeline earlier mentioned. It was explored all the different applications previously analysed in the Related Solutions section of the State of the Art chapter. 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 that the one used in the implementation and evaluation of the developed application, this would cause a compromised comparison.

The autoML application/tool selected was the TPOT, a tree-based pipeline optimization tool, because it was noted to be the most easy-to-use open-source tool. 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 created the "*_test_TPOT.py*" file to execute the TPOT tool and the results are saved in the "results_TPOT.csv" file in the "output" folder of the project. Then, it was executed the *python* file to all these 15 datasets which contain the "*TPOTClassifier*" function with the following parameters used, as illustrated in Figure 26.

```
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)
```

Figure 26: *TPOTClassifier* function used.

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 also increased times of execution. To have similar values of execution time as the recommendation phase 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

development phase achieved in one of these 15 datasets. Next, the "*scoring*" parameter, the built-in scoring function to evaluate the quality of a pipeline, 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. At last, the "*verbosity*" parameter with the value "2" prints a progress bar with minimal information when TPOT is running.

6.2.1 Research Hypothesis Number Three

Now, the evaluation metrics achieved from the recommendation phase will be compared to the TPOT tool set in the "*_test_TPOT.py*" file, previously explained, concerning the 15 datasets. However, first, it is important to mention that the values from the recommendation phase are the ones got when executing the development phase of the implemented application to those 15 datasets, for the first recommended combination (of the three combinations available). All these values are expressed in the following Table 6 and Table 7, and the *final score* is the average of all metrics.

Table 6: Evaluation metrics values of the recommendation phase.

Dataset	Balanced Accuracy	F1 Score	ROC AUC	Geometric Mean	Cohen Kappa	Final Score
dis	0.787	0.995	0.915	0.747	0.614	0.812
musk	0.998	0.996	1.000	0.998	0.996	0.998
mfeat-fourier	0.990	0.999	0.999	0.990	0.986	0.993
Satellite	0.882	0.672	0.984	0.870	0.666	0.815
arsenic-male-bladder	0.795	0.636	0.836	0.716	0.625	0.722
analcatdata_apnea2	0.936	0.833	0.972	0.934	0.804	0.896
regime_alimentaire	0.940	0.876	0.977	0.938	0.840	0.914
page-blocks0.dat	0.950	0.868	0.992	0.949	0.852	0.922
dgf_test	0.987	0.971	0.999	0.987	0.965	0.982
cpu_small	0.914	0.943	0.976	0.913	0.816	0.912
analcatdata_birthday	0.800	0.937	0.944	0.778	0.576	0.807
optdigits	0.982	0.996	0.999	0.982	0.960	0.984
kr-vs-k-zero_vs_eight.dat	0.980	0.947	0.998	0.977	0.945	0.969
analcatdata_lawsuit	0.966	0.873	0.991	0.962	0.863	0.931
JapaneseVowels	0.978	0.987	0.998	0.978	0.925	0.973

Table 7: Evaluation metrics values of the TPOT tool.

Dataset	Balanced Accuracy	F1 Score	ROC AUC	Geometric Mean	Cohen Kappa	Final Score
dis	0.721	0.994	0.721	0.666	0.566	0.734
musk	0.998	0.991	0.998	0.998	0.989	0.995
mfeat-fourier	0.941	0.993	0.941	0.939	0.920	0.947
Satellite	0.875	0.857	0.875	0.866	0.855	0.866
arsenic-male-bladder	0.800	0.750	0.800	0.775	0.736	0.772
analcatdata_apnea2	0.528	0.105	0.528	0.236	0.091	0.298
regime_alimentaire	0.972	0.917	0.972	0.972	0.888	0.944
page-blocks0.dat	0.906	0.868	0.906	0.902	0.853	0.887
dgf_test	0.981	0.961	0.981	0.981	0.954	0.972
cpu_small	0.890	0.939	0.890	0.888	0.784	0.878
analcatdata_birthday	0.532	0.922	0.532	0.297	0.092	0.475
optdigits	0.958	0.992	0.958	0.957	0.924	0.958
kr-vs-k-zero_vs_eight.dat	0.688	0.042	0.688	0.662	0.031	0.422
analcatdata_lawsuit	0.742	0.600	0.742	0.701	0.569	0.671
JapaneseVowels	0.976	0.992	0.976	0.976	0.948	0.974

Consequently, for all 15 datasets, the recommendation phase accomplished an average *final score* of 0.9087 and the TPOT tool accomplished 0.7862. Thus, when dividing the average of the *final score* attained in the recommendation phase by the one in the TPOT tool, it is concluded that the *final score* of the recommendation phase is greater, on average, by 16% than the one attained by the TPOT tool. Therefore, the RH3 was successfully achieved in these conditions.

6.2.2 Research Hypothesis Number Four

Now, it is going to be analysed the time of execution achieved from the recommendation phase of the implemented application compared to the TPOT tool set in the "*_test_TPOT.py*" file, concerning the 15 datasets. All these values are expressed in the following Table 8, and all the values of time are in the "HH:mm:ss" format.

Table 8: Comparison of execution times of the recommendation phase to the TPOT tool.

Dataset	Execution time in the recommendation phase	Execution time in the TPOT tool
dis	00:00:25	00:00:40
musk	00:04:08	00:09:12
mfeat-fourier	00:00:13	00:01:19
Satellite	00:00:57	00:01:18
arsenic-male-bladder	00:00:02	00:00:17
analcatdata_apnea2	00:00:01	00:00:17
regime_alimentaire	00:00:01	00:00:28
page-blocks0.dat	00:00:58	00:01:34
dgf_test	00:01:01	00:01:01
cpu_small	00:03:20	00:02:40
analcatdata_birthday	00:00:01	00:00:27
optdigits	00:01:19	00:01:16
kr-vs-k-zero_vs_eight.dat	00:00:06	00:00:22
analcatdata_lawsuit	00:00:01	00:00:16
JapaneseVowels	00:05:20	00:01:54

Consequently, for all the 15 datasets, the recommendation phase accomplished 17 minutes and 53 seconds (1073 total seconds) of the total amount of execution time and the TPOT tool accomplished 23 minutes and 1 second (1381 total seconds) of execution time. Thus, when dividing the total time attained in the recommendation phase by the TPOT tool, it is concluded that the execution time of the recommendation phase is smaller/faster with 78% of the time of the TPOT tool. Therefore, the RH4 was successfully achieved in these conditions.

6.3 Final Remarks

To conclude the evaluation of this application, the RH2, RH3 and RH4 were successfully achieved to the tested conditions, and the RH1 was not achieved. Therefore, the outcome of this evaluation was mostly positive.

7 Conclusions

This chapter is composed of two sections, the Accomplished Objectives and the Limitations and Future Work. The former explains if the objectives were and how they were accomplished, and the latter, by guiding the potential future work to be addressed.

7.1 Accomplished Objectives

This application was successfully documented, designed, implemented, and evaluated. It can deliver recommendations of suited combinations of pre-processing techniques and classification algorithms to imbalanced datasets, therefore automating this step in the machine learning pipeline.

Moreover, it provides these recommendations to a newly imported dataset by using the meta-features values, previously extracted, of the most similar datasets already present in the knowledge base, from past executions. 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 is going to be explained in the next section.

7.2 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 optimization techniques like Grid Search, Random Search, Bayesian Optimization, CASH optimization and others can improve the results achieved. Then, other pre-processing techniques and classification algorithms can also be added to the application. 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 got when recommending are improved by adding more datasets to the knowledge base of the application. Then, this application should be compared to more autoML solutions. Finally, in the future, this application should operate also with multiclass classification problems.

References

[1] M. Kuhn and K. Johnson, 'Applied predictive modeling', *Appl. Predict. Model.*, pp. 1–600, Jan. 2013, doi: 10.1007/978-1-4614-6849-3.

[2] B. Krawczyk, 'Learning from imbalanced data: open challenges and future directions', *Prog. Artif. Intell.*, vol. 5, no. 4, pp. 221–232, Nov. 2016, doi: 10.1007/S13748-016-0094-0/TABLES/1.

[3] N. Japkowicz, 'Learning from Imbalanced Data Sets: A Comparison of Various Strategies *', 2000, Accessed: Feb. 06, 2022. [Online]. Available: www.aaai.org.

[4] M. Lango, 'Tackling the Problem of Class Imbalance in Multi-class Sentiment Classification: An Experimental Study', *Found. Comput. Decis. Sci.*, vol. 44, no. 2, pp. 151–178, Jun. 2019, doi: 10.2478/FCDS-2019-0009.

[5] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, 'Learning from Imbalanced Data Sets', *Learn. from Imbalanced Data Sets*, 2018, doi: 10.1007/978-3-319-98074-4.

[6] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, 'An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics', *Inf. Sci. (Ny).*, vol. 250, pp. 113–141, Nov. 2013, doi: 10.1016/J.INS.2013.07.007.

[7] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour, 'Boosting methods for multi-class imbalanced data classification: an experimental review', *J. Big Data*, vol. 7, no. 1, pp. 1–47, Dec. 2020, doi: 10.1186/S40537-020-00349-Y/FIGURES/5.

[8] P. Branco, I. Torgo, R. P. Ribeiro, and L. Torgo, 'A Survey of Predictive Modeling on Imbalanced Domains', *ACM Comput. Surv. 1, 1, Artic.*, vol. 1, 2016.

[9] N. Moniz and V. Cerqueira, 'Automated imbalanced classification via meta-learning', *Expert Syst. Appl.*, vol. 178, Sep. 2021, doi: 10.1016/J.ESWA.2021.115011.

[10] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, 'Metalearning', 2008, doi: 10.1007/978-3-540-73263-1.

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

[12] F. Hutter, L. Kotthoff, and J. Vanschoren, 'Automated Machine Learning: Methods, Systems, Challenges', 2019, doi: 10.1007/978-3-030-05318-5.

[13] B. Zoph and Q. V. Le, 'Neural Architecture Search with Reinforcement Learning', *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, Nov. 2016, Accessed: Feb. 09, 2022. [Online]. Available: https://arxiv.org/abs/1611.01578v2.

[14] I. Goodfellow, Y. Bengio, and A. Courville, 'Deep Learning', *Nature*, vol. 521, no. 7553, p. 800, 2016, Accessed: Feb. 07, 2022. [Online]. Available: http://goodfeli.github.io/dlbook/%0Ahttp://dx.doi.org/10.1038/nature14539.

[15] M. De Lange *et al.*, 'A continual learning survey: Defying forgetting in classification tasks', 2021, doi: 10.1109/TPAMI.2021.3057446.

[16] D. H. Wolpert and W. G. Macready, 'No free lunch theorems for optimization', *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.

[17] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, 'Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms', *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. Part F128815, pp. 847–855, Aug. 2012, doi: 10.1145/2487575.2487629.

[18] H. Jair Escalante, M. Montes, and L. Enrique Sucar ESUCAR, 'Particle Swarm Model Selection', *J. Mach. Learn. Res.*, vol. 10, no. 15, pp. 405–440, 2009, Accessed: Feb. 09, 2022. [Online]. Available: http://jmlr.org/papers/v10/escalante09a.html.

[19] E. Brochu, V. M. Cora, and N. de Freitas, 'A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning', Dec. 2010, Accessed: Feb. 09, 2022. [Online]. Available: https://arxiv.org/abs/1012.2599v1.

[20] 'Imbalanced Classification with Python: Better Metrics, Balance Skewed ... - Jason Brownlee - Google Livros'. https://books.google.pt/books?hl=pt-PT&lr=&id=jaXJDwAAQBAJ&oi=fnd&pg=PP1&dq=imbalanced+classification+metrics&ots=CfMx7PUX_Q&sig=p3LfN7xLTOwsXX6n9HZk_GMco8E&redir_esc=y#v=onepage&q=imbalanced classification metrics&f=false (accessed Aug. 20, 2022).

[21] 'scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation'. https://scikit-learn.org/stable/ (accessed Feb. 13, 2022).

[22] 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.

[23] N. Lunardon, G. Menardi, and N. T. Maintainer, 'Package "ROSE" Type Package Title Random Over-Sampling Examples', 2021.

[24] 'CRAN - Package imbalance'. https://cran.r-project.org/web/packages/imbalance/index.html (accessed Feb. 14, 2022).

[25] X. He, K. Zhao, and X. Chu, 'AutoML: A survey of the state-of-the-art', *Knowledge-Based Syst.*, vol. 212, p. 106622, Jan. 2021, doi: 10.1016/j.knosys.2020.106622.

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

[27] Q. Yao *et al.*, 'Taking Human out of Learning Applications: A Survey on Automated

Machine Learning', Oct. 2018, Accessed: Feb. 12, 2022. [Online]. Available: https://arxiv.org/abs/1810.13306v4.

[28] M. A. Zöller and M. F. Huber, 'Benchmark and Survey of Automated Machine Learning Frameworks', *J. Artif. Intell. Res.*, vol. 70, pp. 409–472, Jan. 2021, doi: 10.1613/JAIR.1.11854.

[29] 'AutoKeras'. https://autokeras.com/ (accessed Feb. 13, 2022).

[30] 'Keras: the Python deep learning API'. https://keras.io/ (accessed Feb. 13, 2022).

[31] '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).

[32] 'TPOT'. https://epistasislab.github.io/tpot/ (accessed Feb. 13, 2022).

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

[34] '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).

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

[36] 'Amazon SageMaker Autopilot | Amazon SageMaker', 2022. https://aws.amazon.com/pt/sagemaker/autopilot/ (accessed Feb. 07, 2022).

[37] 'Cloud AutoML - Google Cloud', 2022. https://cloud.google.com/automl (accessed Feb. 07, 2022).

[38] N. Moniz, 'Nuno Moniz: ATOMIC: an AutoML (and easy-to-use) solution for imbalanced learning', *dcc.fc.up.pt*, Apr. 04, 2021. https://www.dcc.fc.up.pt/~nmoniz/posts/2021-04-04-autoresampling-a-baseline-for-imbalanced-learning/ (accessed Nov. 15, 2021).

[39] Z. Li and D. Hoiem, 'Learning without Forgetting', *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Jun. 2016, doi: 10.1109/TPAMI.2017.2773081.

[40] S. A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, 'iCaRL: Incremental Classifier and Representation Learning', *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 5533–5542, Nov. 2016, doi: 10.1109/CVPR.2017.587.

[41] 'No-code machine learning - Amazon Web Services'. https://aws.amazon.com/pt/sagemaker/canvas/ (accessed Feb. 14, 2022).

[42] 'OPPORTUNITY | Significado, definição em Dicionário Inglês'. https://dictionary.cambridge.org/pt/dicionario/ingles/opportunity (accessed Feb. 14, 2022).

[43] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, 'Learning from class-imbalanced data: Review of methods and applications', *Expert Systems with Applications*, vol. 73. Elsevier Ltd, pp. 220–239, May 01, 2017, doi: 10.1016/j.eswa.2016.12.035.

[44] 'Imbalanced Classification - Google Trends'. https://trends.google.pt/trends/explore?date=2006-01-01 2022-02-27&q=imbalanced classification (accessed Feb. 15, 2022).

[45] 'AutoML - Google Trends'. https://trends.google.pt/trends/explore?date=2006-01-01 2022-02-27&q=automl (accessed Feb. 15, 2022).

[46] 'Repositório Aberto da Universidade do Porto: Uma reflexão sobre a análise do valor e o seu posicionamento no actual panorama da gestão da qualidade'. https://repositorio-aberto.up.pt/handle/10216/12568 (accessed Feb. 14, 2022).

[47] F. Franceschini and S. Rossetto, 'QFD: an interactive algorithm for the prioritization of product's technical design characteristics', *Integr. Manuf. Syst.*, vol. 13, no. 1, pp. 69–75, 2002, doi: 10.1108/09576060210411521/FULL/XML.

[48] Deepanshu Tyagi, 'Introduction to FAST (Features from Accelerated Segment Test) | by Deepanshu Tyagi | Data Breach | Medium', 2019. https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65 (accessed Feb. 14, 2022).

[49] 'Customer Value: What it Means & Why It's Important - Builtvisible'. https://builtvisible.com/understanding-customer-value/ (accessed Feb. 14, 2022).

[50] 'What is Free Software? - GNU Project - Free Software Foundation'. https://www.gnu.org/philosophy/free-sw.en.html (accessed Feb. 14, 2022).

[51] P. E. Boksberger and L. Melsen, 'Perceived value: A critical examination of definitions, concepts and measures for the service industry', *J. Serv. Mark.*, vol. 25, no. 3, pp. 229–240, May 2011, doi: 10.1108/08876041111129209.

[52] A. Whiting, 'Six Steps to Crafting Effective Value Propositions', 2012.

[53] 'Value Proposition Design Book - Preview & Download PDF', Oct. 30, 2014. https://www.strategyzer.com/books/value-proposition-design (accessed Feb. 14, 2022).

[54] H. S. Shih, H. J. Shyur, and E. S. Lee, 'An extension of TOPSIS for group decision making', *Math. Comput. Model.*, vol. 45, no. 7–8, pp. 801–813, Apr. 2007, doi: 10.1016/J.MCM.2006.03.023.

[55] R. B. Grady, 'Practical software metrics for project management and process improvement', p. 270, 1992.

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

[57] '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).

[58] ‘OpenML’. https://www.openml.org/search?type=data (accessed Feb. 14, 2022).

[59] ‘Find Open Datasets and Machine Learning Projects | Kaggle’. https://www.kaggle.com/datasets (accessed Feb. 14, 2022).

[60] ‘Dataset Search’. https://datasetsearch.research.google.com/ (accessed Feb. 14, 2022).

[61] ‘OpenML APIs - OpenML Documentation’. https://docs.openml.org/APIs/ (accessed Jul. 30, 2022).

[62] ‘Welcome to Python.org’. https://www.python.org/ (accessed Feb. 13, 2022).

[63] ‘R: The R Project for Statistical Computing’. https://www.r-project.org/ (accessed Feb. 13, 2022).

[64] ‘PySimpleGUI’. https://pysimplegui.readthedocs.io/en/latest/ (accessed Feb. 15, 2022).

[65] ‘Dear PyGui’s Documentation — Dear PyGui documentation’. https://dearpygui.readthedocs.io/en/latest/index.html (accessed Feb. 15, 2022).

[66] ‘tkinter — Python interface to Tcl/Tk — Python 3.10.2 documentation’. https://docs.python.org/3/library/tkinter.html (accessed Feb. 15, 2022).

[67] ‘Python UI | Design GUI with Python | Python Bindings for Qt’. https://www.qt.io/qt-for-python (accessed Feb. 15, 2022).

[68] ‘Welcome to wxPython! | wxPython’. https://www.wxpython.org/ (accessed Feb. 15, 2022).

[69] ‘dddomodossola/remi: Python REMote Interface library. Platform independent. In about 100 Kbytes, perfect for your diet.’ https://github.com/dddomodossola/remi (accessed Feb. 15, 2022).

[70] ‘ocornut/imgui: Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies’. https://github.com/ocornut/imgui (accessed Feb. 15, 2022).

[71] ‘Welcome to Flask — Flask Documentation (2.0.x)’. https://flask.palletsprojects.com/en/2.0.x/ (accessed Feb. 15, 2022).

[72] ‘Streamlit • The fastest way to build and share data apps’. https://streamlit.io/ (accessed Feb. 15, 2022).

[73] ‘Cloud Application Platform | Heroku’. https://www.heroku.com/ (accessed Feb. 15, 2022).

[74] ‘Apache Spark™ - Unified Engine for large-scale data analytics’. https://spark.apache.org/ (accessed Feb. 14, 2022).

[75] ‘Dask: Scalable analytics in Python’. https://dask.org/ (accessed Feb. 14, 2022).

[76] ‘Ray - Scaling Python made simple, for any workload’. https://www.ray.io/ (accessed Feb. 14, 2022).

[77] ‘TensorFlow’. https://www.tensorflow.org/ (accessed Feb. 14, 2022).

[78] ‘PyTorch’. https://pytorch.org/ (accessed Feb. 14, 2022).

[79] ‘CUDA Zone | NVIDIA Developer’. https://developer.nvidia.com/cuda-zone (accessed Feb. 14, 2022).

[80] ‘OpenCL Overview - The Khronos Group Inc’. https://www.khronos.org/opencl/ (accessed Feb. 14, 2022).

[81] ‘OpenGL - The Industry Standard for High Performance Graphics’. https://www.opengl.org// (accessed Feb. 14, 2022).

[82] ‘Git’. https://git-scm.com/ (accessed Feb. 15, 2022).

[83] ‘GitHub’. https://github.com/ (accessed Feb. 15, 2022).

[84] ‘Bitbucket’. https://bitbucket.org/ (accessed Feb. 15, 2022).

[85] ‘PyInstaller Manual — PyInstaller 4.9 documentation’. https://pyinstaller.readthedocs.io/en/stable/ (accessed Feb. 15, 2022).

[86] P. Vieira, ‘PedroVieira1160634/automated-imbalanced-classification: Automated Imbalanced Classification’. https://github.com/PedroVieira1160634/automated-imbalanced-classification (accessed Sep. 10, 2022).

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

[88] J. van Rijn, ‘OpenML: credit-g Dataset (ID: 31)’, 2014. https://www.openml.org/search?type=data&sort=runs&status=active&id=31 (accessed Sep. 10, 2022).

[89] ‘KEEL: Car Good dataset’. https://sci2s.ugr.es/keel/dataset.php?cod=1328 (accessed Sep. 11, 2022).

[90] ‘pandas - Python Data Analysis Library’. https://pandas.pydata.org/ (accessed Sep. 10, 2022).

[91] ‘imbalanced-learn documentation — Version 0.9.1’. https://imbalanced-learn.org/stable/ (accessed Sep. 10, 2022).

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

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

[94] ‘NumPy’. https://numpy.org/ (accessed Sep. 10, 2022).

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

[96] ‘KEEL: Poker Hand (Imbalanced: 8 vs 6) dataset’. https://sci2s.ugr.es/keel/dataset.php?cod=1340 (accessed Sep. 11, 2022).

[97] ‘KEEL: Glass Identification (Imbalanced: 1) dataset’. https://sci2s.ugr.es/keel/dataset.php?cod=142 (accessed Sep. 11, 2022).

[98] P. Gijsbers, ‘OpenML: dis Dataset (ID: 40713)’, 2017. https://www.openml.org/search?type=data&status=active&id=40713 (accessed Oct. 02, 2022).

[99] J. Vanschoren, ‘OpenML: musk Dataset (ID: 1116)’, 2014. https://www.openml.org/search?type=data&status=active&id=1116 (accessed Oct. 02, 2022).

[100] J. Vanschoren, ‘OpenML: mfeat-fourier Dataset (ID: 971)’, 2014. https://www.openml.org/search?type=data&status=active&id=971 (accessed Oct. 02, 2022).

[101] R. G. Mantovani, ‘OpenML: Satellite Dataset (ID: 40900)’, 2015. https://www.openml.org/search?type=data&status=active&id=1464 (accessed Oct. 02, 2022).

[102] J. Vanschoren, ‘OpenML: arsenic-male-bladder Dataset (ID: 947)’, 2014. https://www.openml.org/search?type=data&status=active&id=947&sort=runs (accessed Oct. 02, 2022).

[103] J. Vanschoren, ‘OpenML: analcatdata_apnea2 Dataset (ID: 765)’, 2014. https://www.openml.org/search?type=data&status=active&id=765&sort=runs (accessed Oct. 02, 2022).

[104] G. Ferrettini, ‘OpenML: regime_alimentaire Dataset (ID: 42172)’, 2019. https://www.openml.org/search?type=data&status=active&id=42172 (accessed Oct. 02, 2022).

[105] ‘KEEL: page-blocks0 Dataset - Page Blocks Classification’. https://sci2s.ugr.es/keel/dataset.php?cod=147 (accessed Oct. 02, 2022).

[106] P. Soriano, ‘OpenML: dgf_test Dataset (ID: 42883)’, 2021. https://www.openml.org/search?type=data&status=active&id=42883&sort=runs (accessed Oct. 02, 2022).

[107] J. Vanschoren, ‘OpenML: cpu_small Dataset (ID: 735)’, 2014. https://www.openml.org/search?type=data&status=active&id=735&sort=runs (accessed Oct. 02, 2022).

[108] J. Vanschoren, ‘OpenML: analcatdata_birthday Dataset (ID: 968)’, 2014. https://www.openml.org/search?type=data&status=active&id=968&sort=runs (accessed Oct. 02, 2022).

[109] J. Vanschoren, ‘OpenML: optdigits Dataset (ID: 980)’, 2014. https://www.openml.org/search?type=data&status=active&id=980&sort=runs (accessed Oct. 02, 2022).

[110] ‘KEEL: kr-vs-k-zero_vs_eight Dataset - Chess - King-Rook vs. King (Imbalanced: zero vs eight)’. https://sci2s.ugr.es/keel/dataset.php?cod=1335 (accessed Oct. 02, 2022).

[111] J. Vanschoren, ‘OpenML: analcatdata_lawsuit Dataset (ID: 450)’, 2014. https://www.openml.org/search?type=data&status=active&id=450&sort=runs (accessed Oct. 02, 2022).

[112] J. Vanschoren, ‘OpenML: JapaneseVowels Dataset (ID: 976)’, 2014. https://www.openml.org/search?type=data&status=active&id=976 (accessed Oct. 02, 2022).