Received September 28, 2020, accepted October 13, 2020, date of publication October 16, 2020, date of current version October 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031572

Creating a Recommender System to Support Higher Education Students in the Subject Enrollment Decision

ANTONIO JESÚS FERNÁNDEZ-GARCÍA[1], ROBERTO RODRÍGUEZ-ECHEVERRÍA[2], JUAN CARLOS PRECIADO[2], JOSÉ MARÍA CONEJERO MANZANO[2], AND FERNANDO SÁNCHEZ-FIGUEROA[2]

[1]Universidad Internacional de La Rioja, 26006 Logroño, Spain
[2]Quercus Research Group, University of Extremadura, 10003 Cáceres, Spain

Corresponding author: Antonio Jesús Fernández-García (antoniojesus.fernandez@unir.net)

This work was supported in part by the Ministerio de Ciencia, Innovación y Universidades (MCIU), Agencia Estatal de Investigación (AEI), and European Regional Development Fund (ERDF) under Project RTI2018-098652-B-I00; and in part by the European Regional Development Fund (ERDF) and Junta de Extremadura under Project IB16055, Project IB18034, and Project GR18112.

ABSTRACT Higher Education plays a principal role in the changing and complex world of today, and there has been rapid growth in the scientific literature dedicated to predicting students' academic success or risk of dropout thanks to advances in Data Mining techniques. Degrees such as Computer Science in particular are in ever greater demand. Although the number of students has increased, the number graduating is still not enough to provide society with as many as it requires. This study contributes to reversing this situation by introducing an approach that not only predicts the dropout risk or students' performance but takes action to help both students and educational institutions. The focus is on maximizing graduation rates by constructing a Recommender System to assist students with their selection of subjects. In particular, the challenge is addressed of constructing reliable Recommender Systems on the basis of data which are both sparse and few in quantity, imbalanced, and anonymized, and which might have been stored under imperfect conditions. This approach is successfully applied to create a Recommender System using a real-world dataset from a public Spanish university containing performance data of a Computer Science degree course, demonstrating its successful application in real environments. The construction of a support system based on that approach is described, its results are evaluated, and its implications for students' academic achievement, and for institutions' graduation rates are discussed. Through the construction of this decision support system for students, we intend to increase the graduation rates and lower the dropout rate.

INDEX TERMS Computer education, data mining, decision support system, machine learning, recommender systems, student dropout.

I. INTRODUCTION

A. IMPORTANCE OF MAXIMIZING GRADUATION RATES IN THE ENGINEERING AREAS

There is no doubt about the importance of Higher Education in Europe, and the relationship between education and inequality is very significant across the continent [1]. The number of students of Computer Science and Information Technologies has been increasing considerably recently due to the capacity of the field to transform our society [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Tallha Akram.

This fact poses a challenge that needs specific attention from universities and the main educational institutions. But this increasing number of Computer Science students is not enough. While the demand for Computer Science education is growing exponentially, the supply of researchers and specialists in the field is growing linearly, generating a large gap [3].

Therefore, measures should be taken to maximize the number of students who graduate in Computer Sciences and Information Technologies degree courses without deterioration in the quality of the teaching, so as to ensure that society's demands can be met. One such measure could be to lessen the high dropout rate in this field, especially in the first undergraduate year [4].

For this reason, beyond predicting whether or not a student will finish by graduating successfully, it makes sense to create a Recommender System capable of guiding them towards choosing the subjects best suited to them individually for the completion of their course. In this way, by creation a decision support system for students, the graduation rate can be optimized and each student's success maximized in accordance with their potential.

B. THE DIFFICULTY OF GATHERING QUALITY DATA

Creating a Recommender System or a predictive model requires a certain amount of data. Higher Education institutions produce large volumes of data related to their students' performance and background. Proper use of this data can be transformed into knowledge with which to improve the educational processes. The problem is that, in many cases, institutions do not save data properly (or do not collect it at all), and, if they do, sharing personal data is only facilitated for specified, legitimate, and necessary purposes. Even for these purposes, the data must be shared respecting the appropriate security protocols, which is understandable but not always easy. Indeed, in Europe, this is a very tedious process which often ultimately leads to data, at least partially, not being shared due to privacy policies.

The paucity of data available makes it harder to create highly reliable predictive models or recommendation systems. Fortunately, advances in the Data Mining field provide the opportunity of applying a wide variety of techniques and methods that can make the most of the data so as to infer knowledge. We shall take advantage of this to assist students in their educational development through a decision support system.

C. MAIN INNOVATION OF THE PAPER

In the light of the foregoing, the main innovation of the paper lies in:

- Creating a strategy that makes the most of small datasets to infer knowledge reliable enough to be deployed in educational institutions in form of decision support systems to assist their students in choosing the subjects best suited to each individual case.
- Making use of Recommender Systems instead of predictive models that are designed to forecast students' performance and success or their dropout risk. With Recommender Systems, it is possible to assist students in choosing the right subjects for them, maximizing the number of Computer Science and information technology graduates, and improving the students' experience individually.
- Proposing an approach that, making use of such Data Mining techniques as *Feature Encoding*, *Feature Engineering*, *Data Scaling*, or *Data Resampling*, and some well known Machine Learning algorithms such as *Random Forest*, *Support Vector Machine*, or *Logistic Regression* among others, may be reproduced by researchers or educational institutions with the aim of improving students' performance and successfully increasing the number of Computer Science graduates without the need for a large amount of data, and without accessing students' personal data.

D. RESEARCH QUESTIONS

The aforementioned facts led us to pose the following research questions:

RS1 *Is it possible, on the basis of a dataset with few instances, to create reasonably accurate models that can assist students in selecting the subjects best suited to them?*

RS2 *What impact and what capacity do Data Mining techniques such as Feature Encoding, Feature Engineering, Data Scaling, or Data Resampling have on generating knowledge from a dataset to help institutions predict the success of their students even in the absence of sufficient data?*

RS3 *Can one create a Decision Support System based on a Recommender System that assists students in selecting the subjects best suited to them, and thereby maximize the number of Computer Science graduates, on the basis of the models studied in Question* 1 *and the Data Mining techniques noted in Question* 2*?*

The rest of this communication is organized as follows. A student dropout and performance literature review and related projects are presented in Section II. Section III describes the proposed approach, providing a methodological overview and detailing the baseline data and the algorithms employed. Section IV addresses the creation of the experiments after carrying out each processing technique (*encoding*, *feature engineering*, *scaling*, and *resampling*) and presents their results. Section V discusses the results of the general approach. Finally, some conclusions and further considerations are given in Section VII.

II. LITERATURE REVIEW

This section comprises an analysis of the most commonly used algorithms for constructing recommender systems and then presents a review of the more important published work focused on the prediction of the students' academic performance and the prediction of the students' dropout.

A. ALGORITHMS

The first types of algorithm that come to mind when thinking about Recommender Systems are *Collaborative Filtering* and *Content-Based* algorithms. However, there are numerous Recommender Systems that are built using classical regression or classification algorithms. In this work, we opted for the second choice due to cold-start problems and the difficulty of inferring similarities between subjects or students in small datasets, in addition to lacking any additional personal data of the students.

Once the decision had been taken, there were still many algorithms in the literature that could be used to create recommender models. We selected well-known classification algorithms available in Scikit Learn [5], one of the most popular Machine Learning libraries.

The algorithms selected are:

- *Decision Tree*. This consists of building a tree structure in which each ramification represents a question about an attribute. New branches are created based on the answers to the question until reaching the leaves of the tree (where the structure ends). The leaf nodes indicate the predicted class.
- *Random Forest*. This is an improvement that creates several *Decision Trees*, using bagging or some other technique, and elects the most popular of their outputs. Usually, the outputs are not counted directly, Instead, the normalized frequencies of each output are summed to get the label with the greatest probability. It can be defined as: $\hat{f} = \frac{1}{T}\sum_{t=1}^{T} f_t(x)$.
- *Gradient Boosting Classifier*. This consists of ensembles constructed from *Decision Tree* models. Weak *Decision Tree* models are built and new *Decision Trees* are added one by one to the ensemble, trying to correct the errors of the existing ones. The concept is based on the idea of using a weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified [6].
- *Logistic Regression*. This uses a more complex cost function than the *Linear Regression* algorithm, which is the *Sigmoid* or *Logistic Function*. Input values are combined linearly using weights or coefficient values to predict an output value. A key difference with linear regression is that the output value being modeled is a binary variable (0 or 1) rather than a numeric one. The function can be defined as: $f(x) = \frac{1}{1+e^{-x}}$.
- *Support Vector Machine*. This algorithm classifies through a separator. It works by taking data to high multidimensional spaces where is possible to categorize data otherwise not linearly separable. Then, the *Support Vector Machine* algorithm finds a hyperplane who acts as a separator. As many possible hyperplanes can be drawn, it is selected the hyperplane that provides a greater margin between classes.
- *k Nearest Neighbours*. This algorithm is based on the idea that cases of similar observations must be close to each other in such a way that it is possible to classify new cases based on their similarity with other previously identified cases. Nearby cases are known as neighbours, and the parameter *k* refers to the number of closest individual observations used to classify a new observation.
- *Multilayer Perceptron*. Also known as *Feed-Forward Artificial Neural Network*, this algorithm consists of creating a set of levels, all of them interconnected. Each level consists of a set of perceptrons (neurons or nodes) receiving input and producing outputs. Weights are assigned to the outputs of the nodes. In this structure, layer 1 receive are the inputs, the nodes in between are "hidden nodes" and the output is given by the last layer. A *Multilayer Perceptron* can be defined as a DAG (Directed Acyclic Graph) with weighted nodes.

B. RELATED WORK

Currently, exploitation of the data that educational institutions have available is widespread, with the aim of generating valuable information mainly regarding two aspects: analysing their students' performance, and predicting the risk of early dropout. There is a growing body of literature in this respect. A summary of some of this literature addressing Data Mining and Machine Learning is given in the following paragraphs.

Regarding Higher Education students' performance, in [7], the factors having an impact on the success of university students are explored through a piece of knowledge discovery software created for that purpose. The software applies decision trees to a dataset that includes pre-processed student data containing private information such as family income. In [8], data that include student enrolment details as well as the activity generated from the university learning management system are used to predict the students' performance in an Australian university. Similarly, in [9], Machine Learning algorithms are used to predict graduation rates in a Colombian university. In [10], four types of mathematical models designed to predict students' academic performance in engineering dynamics (a Higher Education subject) are developed and compared.

Regarding Secondary Education, in [11] the students' performance is addressed through user-friendly decision support software which predicts a student's performance in the final examinations of the academic year. It is worth noting that the authors of this work mention that the dataset used is imbalanced but they took no action in that regard. In [12], Data Mining techniques are used to predict student success in placement test outcomes in Turkey.

In this paper, improved conditional generative adversarial network based deep support vector machine (ICGAN-DSVM) algorithm has been proposed to predict students' performance under supportive learning via school and family tutoring.

Regarding Higher Education again, in [13], the authors address the problem of predicting the student success rate with the novelty that they deal with small datasets, recognizing that this fact limits the scope of Data Mining techniques. In [14], the authors predict the student performance for select candidates in higher education based on a reliable criteria. In [15], conditional generative adversarial network based deep support vector machine (ICGAN-DSVM) is proposed to predict students' performance under supportive learning via school and family tutoring. In [16], the authors employed artificial intelligence models (Extreme Learning Machine and random forest), together with mathematically-based second order Volterra model to investigate the influence of continuous assessments on the first and second year of engineering mathematics.

As can be seen, the number of studies related to student performance is large. In addition to those noted above, [17] present an overview and comparison of many others.

There have also been many research studies addressing dropout prediction. In [18], the authors created methods for identifying at-risk students at a U.S. university early in the semester using performance data during that semester. In [19], the students' attendance, also in the first semester, is analysed to predict dropout risk. In [20], the authors applies the uplift modeling framework to the problem of student dropout prevention and estimating the effects of actions such as tutorial with the objective to improve the design of retention programs. In [21], well-known data mining classification algorithms, predicts students' overall academic performance based on the information available at the end of the first year of the students' academic path at the University of Porto (Portugal).

In contrast to the previous works, in [22], a decision support system is created to identify the dropout risk at registration time instead of during the semester. The authors include a case study from a Belgian University. In [23], the problem of predicting student failure is analysed in a Mexican university taking into consideration, as we shall do here, the limitation of using high-dimensional imbalanced datasets, which is very frequent in this field.

In [24], the authors identify students that are at risk of dropout or perform worse than they usually do by extracting features from their historical grading combining the prediction of different features groups and models. In [25], the authors presents two procedures for identifying dropout-prone and non-achievers students early on in an online university statistics course by means of tree-based classification models.

Works such as [26]–[29], and [30] can also be cited. They all have in common that they try to predict dropout from online courses using Learning Management System (LMS) data. The last three works focus on Massive Open Online Courses (MOOCs). The first of the five analyses the LMS tracking data from a Blackboard Vista supported course, identifying 15 variables strongly correlated with the student's final grade in a course of a Canadian university. The second work analyses the LMS data of a Chinese distance learning university, and provides two constraint strategies to obtain two classes of valuable features: learning features and temporal features. It also used students' personal data, and applied resampling techniques to reduce the imbalance of the datasets. The third work collects data from a MOOC in electronics on the Coursera platform. It focuses on analysing the learner's interactions, taking into consideration the sequence patterns derived from the self-paced characteristic of these courses. The fourth work applies feature engineering methods to generate significant features, and feature selection techniques to reveal the main factors affecting dropout. The last work, make use of the logit leaf model (LLM) algorithm because, although the predictive power is less effective than some others (such as ensemble random forest or BOOST models), it outperforms them in terms of interpretability. As a result, they can provide insights into student dropout behavior.

Table 1 presents a comparison of the work mentioned in this section in such aspects as:

a) Objective;
b) Number of Instances;
c) Number of Features;
d) Number of Classes of the Label Feature (we indicate 'Numeric' in the case that *Regression* algorithms are used);
e) Use of Personal Data;
f) Data Processing Methods Applied; and
g) Scope.

The main focus of the related work noted above is on predicting students' performance or dropout risk. Although prediction is very effective, it leaves future actions in the hands of the educational institutions if they want to develop specific policies to that end. With our approach, which is to create a Recommender System that is not focused on predicting or diagnosing student behaviour but acts directly, trying to improve the situation, we ensure that actions are taken to reverse any paths towards students' failure and to improve their performance.

Furthermore, we start from a very common situation in which one has no access to a large number of records or to personal data. Most of the studies analysed above consider ideal circumstances with access to all types of data. But, as some of them point out, these kinds of datasets usually have problems of high-dimensionality or imbalance that need to be resolved. Additionally, they also need to consider applying extensive data processing and feature engineering techniques.

For these reasons, it is reasonable the use of Data Mining techniques to assist students. We propose an approach that deals with real-world problems, and creates a robust Recommender System that shows satisfactory results.

III. OUR APPROACH

This section analyses the important parts of the work developed, and provides a general overview of the approach we take to achieve the goal of creating a Recommender System for suggesting subjects to students.

Firstly, we introduce an overview of the approach presenting the main parts that compound the whole approach, we outline the methodology behind our proposal, from handling the data to constructing the Decision Support System. After that, we describe in detail the main parts of the approach which includes describing the original data that has been provided to us, and detailing the *Feature Engineering*, *Encoding*, *Scaling*, and *Resampling* methods and techniques used in the approach.

TABLE 1. Summary of related work.

Research Work	Objective	# Instances	# Features	Label Classes	Personal Data	Data Processing Methods	Scope
Guruler et al. (2010) [7]	Performance	-	12	2	Yes	Feature Engineering	Higher Education
Helal et al. (2018) [8]	Performance	2648	27	Numeric	Yes	Discretization Data Transformation	Higher Education
Huang and Fang (2013) [10]	Performance	1907	8	Numeric	Yes	Discretization Scale and Normalize Data	Higher Education
Livieris et al. (2019) [11]	Performance	2260	10	4	No	Imbalanced Dataset	Secondary School
Şen et al. (2012) [12]	Performance	-	24	5	Yes	N/A	Secondary School
Mengash (2020) [14]	Performance	2039	-	5	No	N/A	Higher Education
Chui et al. (2020) [15]	Performance	1044	33	2	Yes	N/A	Higher Education
Deo et al. (2020) [16]	Performance	3545	-	5	No	Feature Extraction Feature Spaces	Higher Education
Natek and Zwilling (2014) [13]	Performance	-	12	Numeric	Yes	N/A	Higher Education
Marbouti et al. (2016) [18]	Dropout	3063	-	2	No	Feature Selection	Higher Education
Gray and Perkins (2019) [19]	Dropout	4970	32	5	Yes	Feature Selection	Higher Education
Olaya et al. (2020) [20]	Dropout	3362	60	2	Yes	Feature Engineering	Higher Education
Miguéis et al. (2018) [21]	Dropout	2549	19	2	Yes	N/A	Higher Education
Hoffait and Schyns (2017) [22]	Dropout	6845	9	2	Yes	N/A	Higher Education
Márquez Vera and Ventura (2013) [23]	Dropout	670	77	8	Yes	Feature Selection Resampling Methods	Higher Education
Polyzou and Karypis (2019) [24]	Dropout	-	-	4	No	Feature Selection Feature Extraction	Higher Education
Figueroa and Sancho (2020) [25]	Dropout	197	12	2	No	Feature Extraction Feature Selection	Higher Education
Macfadyen and Dawson (2010) [26]	Dropout	118	22	2	No	N/A	Higher Education
Chen et al. (2020) [27]	Dropout	19679842	-	2	Yes	Feature Engineering Feature Selection Resampling	Higher Education
Moreno-Marcos et al. (2020) [28]	Dropout	-	37	2	No	N/A	MOOC
Qiu and Liu (2018) [29]	Dropout	120542	212	2	No	Feature Engineering Feature Selection	MOOC
Coussement et al. (2020) [30]	Dropout	10554	122	2	No	N/A	MOOC
(Our proposal)	**Recommendation**	6948	55	2	No	Encoding Feature Engineering Feature Extraction Scaling Resampling	Higher Education

A. OVERVIEW

The steps we followed to create the Decision Support System based on a Recommender System are illustrated in Figure 1. With these steps, we do not intend to define a formal methodology. Instead, they represent the guidelines on which our approach is based, and we shall describe them in detail in the course of this communication in order to facilitate replication of the work and encourage its implementation.

The chain of steps that constitute this data-driven modeling pipeline are optional in all cases, and these guidelines may be adapted by the possible elimination, replacement, or adjustment of the steps in accordance with the specific data and problem specifications encountered.

As can be seen in Figure 1, our approach consists of four main steps in which we make use of different data processing techniques. Prior to passing through the main steps, a basic *Feature Engineering* process is applied to the data. Once this pre-processing has been done, the chain of steps that comprise our approach is initiated. After applying each of the main steps, a model is created and evaluated for each scenario using each one of the selected Machine Learning algorithms based on the status of the dataset at that particular moment. With this mechanism, at the end of the process we obtain a list of models that we can compare, and thus choose from among them the best model depending on the issues being considered to provide recommendations for Computer Science students.

- The first step (#1) consists of *Encoding* the features to make them properly interpretable by Machine Learning algorithms. Most algorithms require some sort of encoding of categorical features so that they can be processed. We explored two feature encoding strategies: *Label Encoding* and *One-Hot Encoding*.
- The second step (#2) consists of applying *Feature Engineering* to create significant features. The *Feature Engineering* process applied in this part of the work focuses on creating new meaningful features based on the existing ones to help compensate for the lack of data.
- The third step (#3) consists of *Scaling* the data. We perform this operation to ensure that different scale ranges in numeric features do not adversely affect the creation of the models. We explored four data *scaling* strategies: *MinMax*, *Standard*, *Robust*, and *Normalization*.
- Finally, the fourth step (#4) consists of *Resampling* data to correct any imbalanced data that the dataset may have. This pre-processing technique is necessary since an unequal distribution of classes within the whole

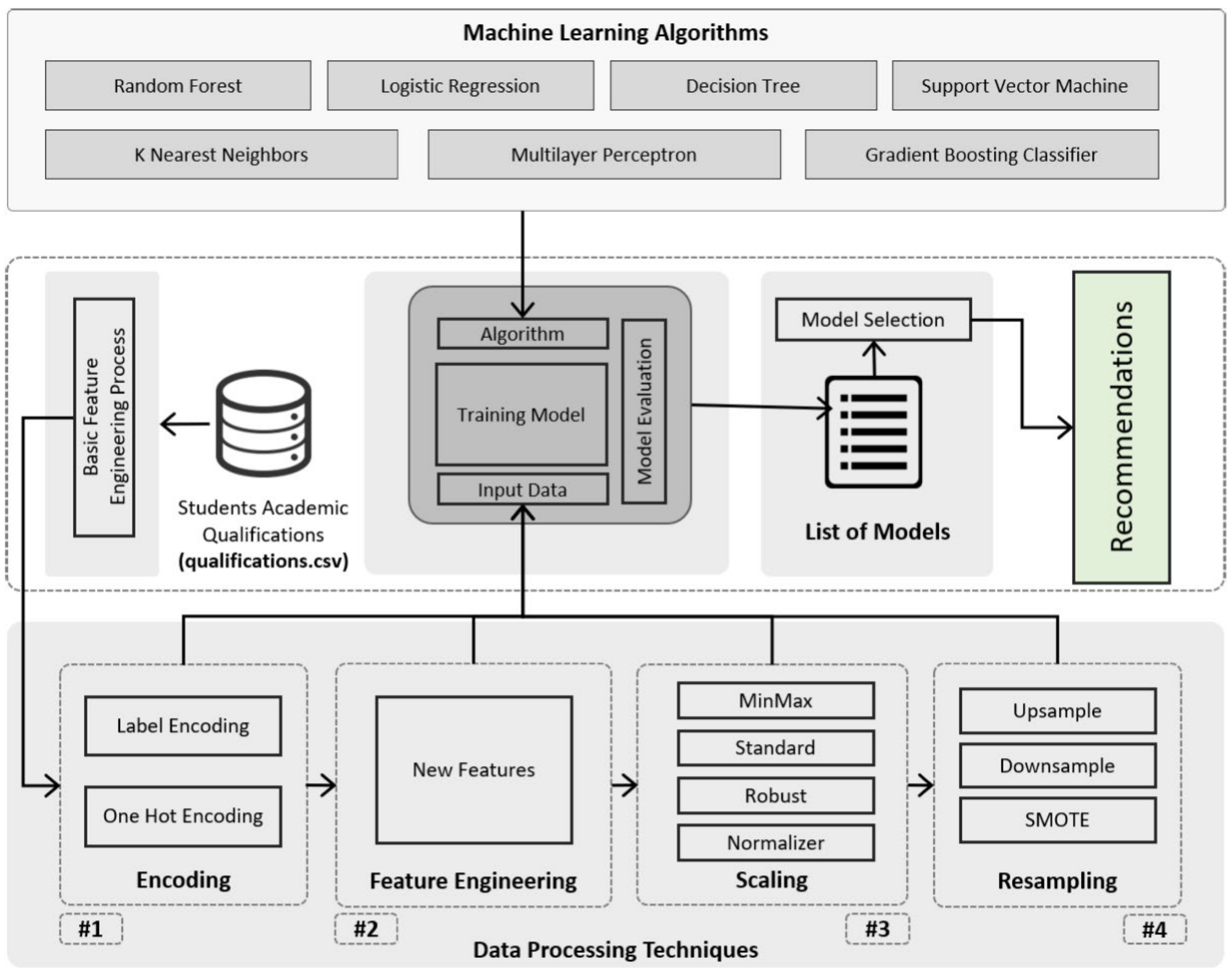


FIGURE 1. Set of processes followed to create the Decision Support System for Computer Science students.

dataset of observations can make it hard to predict the minority classes. We explored three *Resampling* strategies: *upsample*, *downsample*, and *SMOTE*.

B. PROCESSING TECHNIQUES

In this subsection, we shall describe in depth the chain of steps laid out above.

1) ORIGINAL DATA

The data supplied to us were in the form of a CSV file, `qualifications.csv`, containing the performance of a total of 6948 observations of 323 undergraduates taking the 45 subjects that comprise the Computer Science degree course at a public Spanish University from 2010 to 2018. The set of features (description, type, and classes) describing each observation is listed in Table 2.

As can be seen, the small number of observations aggravates the problem of creating a reliable Recommender System, and it is for that very reason that we had to face the challenge of developing a strategy to obtain an accurate system even though the data is limited.

TABLE 2. Set of features describing the `qualifications.csv` dataset.

Feature	Type	Classes
Degree	Categorical	1: {Computer Science Eng.}
Completion Year	Categorical	5: {13-14, 14-15, 15-16, 16-17, 17-18}
Subject	Categorical	45: {List of subjects}
Credits	Numerical	2: {6, 12}
Attempt Number	Numerical	6: {1, 2, 3, 4, 5, 6}
Degree Year	Numerical	4: {1, 2, 3, 4}
Academic Year	Categorical	8: {10-11, 11-12, 12-13, 13-14, 14-15, 15-16, 16-17, 17-18}
Call	Categorical	8: {JUN, FEB, JUL, JAN, SEP, JAX, FEX, NOV}
Mark	Categorical	7: {Not Taken, Fail, Compensation, Sufficient, Very Good, Outstanding, With Honours}
Pass/Fail	Categorical	2: {Pass, Fail}

2) FEATURE ENGINEERING: BASIC PRE-PROCESSING

To begin the data processing, basic *Feature Engineering* techniques are applied. *Feature Engineering* is a process that transforms data to create features that have better representation, and are therefore better suited to creating predictive models [31], [32]. There is a wide range of such techniques [33], and some have actually been applied to Recommender Systems [34].

In the present work, we use the Feature deletion and Class Reduction techniques. Feature Deletion consists of deleting certain features directly from the original dataset (Table 2). The reason is that they do not contribute to the generation of knowledge. Class Reduction is applied because there are features whose class fragmentation is high. Given its small number of instances, it makes sense to reduce the number of classes where possible so as to avoid high-dimensional spaces.

3) ENCODING

In order to work properly, many algorithms require some processing techniques to be applied to categorical data because they are not set up to work with non-numerical features. While it is true that many Machine Learning libraries and services deal with categorical features in a way that is transparent for researchers, Scikit Learn fails to do so. Whatever the conditions of the libraries and services, it is interesting to analyse the impact that this encoding may have on building the recommender model.

In this process, we shall transform categorical features into numerical features. To perform this transformation, two encoding techniques are employed: *Label Encoding* and *One-Hot Encoding*, each with its advantages and disadvantages. *Label Encoding* consists of assigning an integer value in the range $\{0, n-1\}$ to each feature class, with n being the number of classes of the feature. *One-Hot Encoding* consists of, given a feature with n classes, creating n new binary features.

4) FEATURE ENGINEERING: CREATING NEW FEATURES

This process consists of creating new features based on those we already have in order to improve the quality of the predictions. This is also a *Feature Engineering* process, but we deal with it in a specific subsection because we think it is an issue that requires detailed discussion.

The accuracy results so far are not good. We aim to improve them through creating new features. Since we cannot access new data, we need to make the most of the data we already have.

5) SCALING DATA

Some Machine Learning algorithms tend to perform better when the input features used to create the model are on similar scales and close to being normally distributed. Additionally, normally distributed data converge faster [5].

For these reasons, in the process of scaling data we apply different *scalers* to our dataset with the aim of improving the quality of the predictions. As in the previous step, we constructed predictive models associated with the transformed datasets for straightforward evaluation of the results.

We transformed the *One-Hot Encoding* dataset with the new features with 4 different *scalers*: *MinMax Scaler*, *Standard Scaler*, *Robust Scaler*, and *Normalizer Scaler*. We make use of these scalers because of the ease of setting them up using Scikit Learn, although they are easy to implement manually. We also appreciate the fact that is easy for readers access these scalers and reproduce the experiments reported in this paper. The selected scalers work as follows:

- *MinMax Scaler*. This scales each feature individually to a given range. We set a [0, 1] range. The shape of the distribution remains unchanged, and outliers remain unaffected.
- *Standard Scaler*. This scales features by subtracting the mean and scaling to unit variance, which makes individual features look like normally distributed data. It distorts the relative distances between the feature values.
- *Robust Scaler*. This subtracts the median and scales the data according to a specific quantile range. We defined the range to be between the 1st and the 3rd quartiles [25*th quantile*, 75*th quantile*]. By subtracting the median instead of the mean, the effect of outliers is reduced.
- *Normalizer Scaler*. This scales observations individually to unit norm applying Euclidean normalization. In contrast with the other scalers, this scaler does not work on features but on instances.

6) RESAMPLING METHODS

This process is involved in dealing with significant class imbalance that may causes difficulties in predicting the label class with few observations. To resolve this problem, specific techniques need to be deployed to address the imbalance. This process consists of applyng different *Resampling* methods to mitigate the impact of the dataset class imbalance on the predictive models' accuracy.

The methods used are *Upsample*, *Downsample*, and *SMOTE*, which can be defined as follows:

- *Upsample*. This consists of randomly duplicating instances from the minority class until we get a 1:1 ratio of the two classes.
- *Downsample*. This consists of randomly deleting instances from the majority class until we get a 1:1 ratio of the two classes.
- *SMOTE*. The Synthetic Minority Over-sampling Technique (SMOTE) [35] consists of synthesizing new instances from real instances. It takes examples from the minority class and looks for neighbours, *i.e.*, the observations closest in the space. Once the neighbours have been picked, lines are drawn between them, and new observations which are spatially located on these lines are created. The new observations are slightly different from the real existing observations in contrast to the

upsample method in which the new observations are identical to existing ones.

IV. EXPERIMENTAL SETUP AND EXPERIMENTATION

This section describes the implementation of the set of processes that transform the dataset. This set of processes, in the form of a sequential chain of steps, was discussed briefly in Section III along with the definition of the methods and techniques that comprise the approach. In this section, we shall describe in detail how they have been applied in order to convey the knowledge as clearly and accurately as possible to the reader. After the application of each step, a series of experiments based on the processes applied are carried out with the aim of evaluating its impact on the students' Decision Support System.

In addition, to ensure that the implementation and experimentation exposed on this manuscript can be easily replicated by other researchers (or data scientist) over their own data, we have incorporated further technical details in Appendix C with technical clarifications or additions facilitating and enhancing such work.

A. PRE-PROCESSING

This step consists of applying the set of initial feature engineering techniques Feature Deletion and Class Reduction.

The features that should in principle do not contribute to the generation of knowledge are deleted. These features are:

- Degree. This consists of one class only, the number of the degree.
- Completion Year. This indicates the year in which the student finished the degree, which has no direct relationship with the time of the observation.
- Credits. This appears as a numeric feature, but it does not add any numerical attributes. It may have them in other degree courses, but for the Computer Science subjects of the university being analysed there are only two classes: `{6, 12}`. Exploring this feature, we observed that all the subjects but one have 6 credits, which means that 98.2% of the observations belong to the `6` class. For this reason, given its inability to add meaning to the data, this feature is deleted.
- Academic Year. This indicates the year in which the student has taken the subject of the observation. It adds no relevant information from which to infer knowledge because academic years are treated in an isolated way, and we have no further features with which to link them.

The features that, due to their high fragmentation may provoke problems in high-dimensional spaces are:

- Call. Students have several opportunities to pass a subject during the academic year. The calls whose dates are set for the end of the semester are termed `ordinary` calls, and those spread over the rest of the academic year are termed `extraordinary` calls. The dataset had 8 classes for calls `{JUN, FEB, JUL, JAN, SEP, JAX, FEX, NOV}`. We reduced this number of classes to 2: `{Ordinary, Extraordinary}` where the `Ordinary` class grouped together the dataset's raw `{JUN, FEB, JAN}` classes, and the `Extraordinary` class grouped together the dataset's raw `{JUL, SEP, JAX, FEX, NOV}` classes.

Table 3 shows the dataset after applying this section's *Feature Engineering* process.

TABLE 3. Set of features describing the `qualifications.csv` dataset after applying the basic *Feature Engineering* pre-processing techniques.

Feature	Type	Classes
Subject	Categorical	45: {List of subjects}
Attempt Number	Numerical	6: {1, 2, 3, 4, 5, 6}
Degree Year	Numerical	4: {1, 2, 3, 4}
Call	Categorical	2: {Ordinary, Extraordinary}
Pass/Fail	Categorical	2: {Pass, Fail}

B. ENCODING

Table 3 lists the data types of the features that make up the `qualifications` dataset. As can be seen, the `{Subject, Call, Pass/Fail}` features are categorical. As commented above, this may cause problems when applying Machine Learning algorithms using the *sklearn* library. The `Pass/Fail` feature is already naturally label encoded and it is the target feature, but the other two features need to be encoded before creating the models. The *Label Encoding* and *One-Hot Encoding* transformations perform the following operations over these features:

- *Label Encoding*
  - The `call` feature classes are encoded into numerical values. The `Ordinary` class is encoded as `0` and the `Extraordinary` class is encoded as `1`.
  - The `subject` feature classes are encoded into numerical values. There exist 45 classes that are encoded in the `{0, 1, ..., 44}` range of values.
- *One-Hot Encoding*
  - The `call` feature is deleted, and two new features, corresponding to the two classes of the original feature, are created. These new binary classes are filled with 1s and 0s depending on the class that occurs in each instance, *e.g.,* if the class of a particular observation in the `call` feature is `Ordinary`, after the transformation the new `Ordinary` and `Extraordinary` classes are filled with `1`s and `0`s, respectively.
  - The `subject` feature is deleted, and 45 new features, corresponding to the 45 classes of the original feature, are created. These new binary classes are filled with 1s and 0s, depending on the class that occurs in each instance, *e.g.,* if the class of a particular observation in the `subject` feature is `Data Structures`, after the transformation the new `Data Structures` class is filled with `1`s and the rest of the classes with `0`s.

After the *Label Encoding* transformation, the number of features of the dataset remains the same, as does the number of classes of each feature. After the *One-Hot Encoding* transformation, the number of features of the dataset is 50, given that the `call` feature is transformed into 2 new features and the `subject` feature is transformed into 45 new features. In consequence, there are now 2 datasets being the outcomes of the corresponding encoding type, the *Label Encoding* dataset and the *One-Hot Encoding* dataset. They comprise the following features:

Label Encoding = `{Subject, Attempt Number, Degree Year, Call, Pass/Fail}`

One-Hot Encoding = `{Subject`$_0$`, ..., Subject`$_{44}$`, Attempt Number, Degree Year, Ordinary, Extraordinary, Pass/Fail}`

For each dataset, we constructed predictive models using the 7 classification algorithms described in Subsection II-A, making a total of 14 experiments. The results of these experiments are presented in Table 4.

TABLE 4. Experimental results comparing *Label Encoding* and *One-Hot Encoding*. (MLP: Multilayer Perceptron Neural Network, GBC: Gradient Boosting Classifier, LR: Linear Regression, SVM: Suppor Vector Machine, KNN: k Nearest Neighbour).

	Encoding	Algorithm	Accuracy	F1-Score
1	Label	GBC	0.712023	[0.455, 0.8043]
2	One-Hot	GBC	0.708423	[0.449, 0.8018]
3	One-Hot	LR	0.708423	[0.4841, 0.7968]
4	One-Hot	MLP	0.696904	[0.4128, 0.7957]
5	One-Hot	DT	0.686105	[0.2945, 0.7981]
6	One-Hot	KNN	0.676746	[0.5308, 0.7534]
7	Label	RF	0.673866	[0.5276, 0.751]
8	Label	DT	0.668826	[0.3801, 0.7741]
9	One-Hot	RF	0.665947	[0.5207, 0.7436]
10	Label	MLP	0.662347	[0.3153, 0.7759]
11	Label	KNN	0.660187	[0.5193, 0.7372]
12	Label	LR	0.652268	[0.2907, 0.7697]
13	One-Hot	SVM	0.650828	[0.0, 0.7885]
14	Label	SVM	0.650828	[0.0, 0.7885]

The results of the experiments show *One-Hot Encoding* to be the encoding type with generally better accuracy. Models created using *One-Hot Encoding* comprised four of the five top ranked. A curious point is that the encoding of the top ranked model in terms of accuracy was *Label Encoding*.

Furthermore, although *Label Encoding* leaves a dataset with fewer features, which is good, it causes some issues regarding the interpretability of some features, *e.g.*, when features such as `call` or `subject` are encoded following a *Label Encoding* strategy, a cardinality is imposed on the classes of the features. For instance, if three subjects are encoded as follows: `{Data Structures: 2, Mathematics: 4, Computer Organization and Design: 6}`, it may be assumed that `Mathematics` is the average of `Data Structures` and `Computer Organization and Design`. This is another reason why we choose *One-Hot Encoding* as the better encoding technique.

Analysing the accuracy, one can see that, at this point, the greatest accuracy achieved is `0.712023`. This may seem good, but there is a class imbalance in the dataset. The class frequency of the `Pass/Fail` feature is `{Pass:0.650828, Fail:0.349172}`, which means that a model always predicting `Pass` would have an accuracy of `0.650828`. This is precisely what is the case in the *Support Vector Machine* models. From this perspective, we can state that the results of the models created are not good.

C. NEW FEATURES

Several new (and richer) features have been created during the implementation of this process to improve the prediction capabilities. Based on the existing dataset, these new features are `{Category, Student Success Rate, Student Success Rate Category, Subject Pass Rate, Subject Requirements}`. They were created as follows:

- `Category`. Subjects are categorized. There are several ways to do this depending on the granularity of the classification. We opted for the simplest possible classification so as to keep the number of classes of the `Category` feature low. These classes are: `{Programming, Computer Architecture, Mathematics, Business}`.
- `Student Success Rate`. This feature is a metric of the student's performance up to the moment of taking the instance subject. It is calculated by dividing the number of subjects the student has passed by the total number of subjects. It is defined by the following equation:

$$\forall s \subseteq S : SSR_s = \frac{\sum_{j=1}^{m}(ps)_j}{\sum_{i=1}^{n}(ns)_i}, \tag{1}$$

where S is the set of students that make up the dataset, SSR is an abbreviation for `Student Success Rate`, ns with $card(ns) = j$ is the number of subjects taken by student s, and $ps \subseteq ns$ with $card(ps) = i$ is the number of subjects passed by student s.
It must be noted that this metric has a cold start problem, *i.e.*, for first-year students there are no previous records from which to calculate it. For situations such as this, the `Student Success Rate` feature is calculated by dividing the number of times subjects have been passed by the total number of times subjects have been taken, as indicated in the following equation:

$$\forall s \subseteq S : SSR_s = \frac{nps}{nts}, \tag{2}$$

where nps is the total number of successful attempts to pass subjects by all students and nts is the total number of attempts to pass subjects by all students.
When this happens therefore, a generic metric is used for all first-year students instead of a specific metric individually calculated for each one of them. The anonymized data supplied to us prevents us from calculating this metric individually for each first-year student. This decision is not risk-free because it clearly affects the recommendations to these first-year students by treating them all the same, but the fact is that, in their first year,

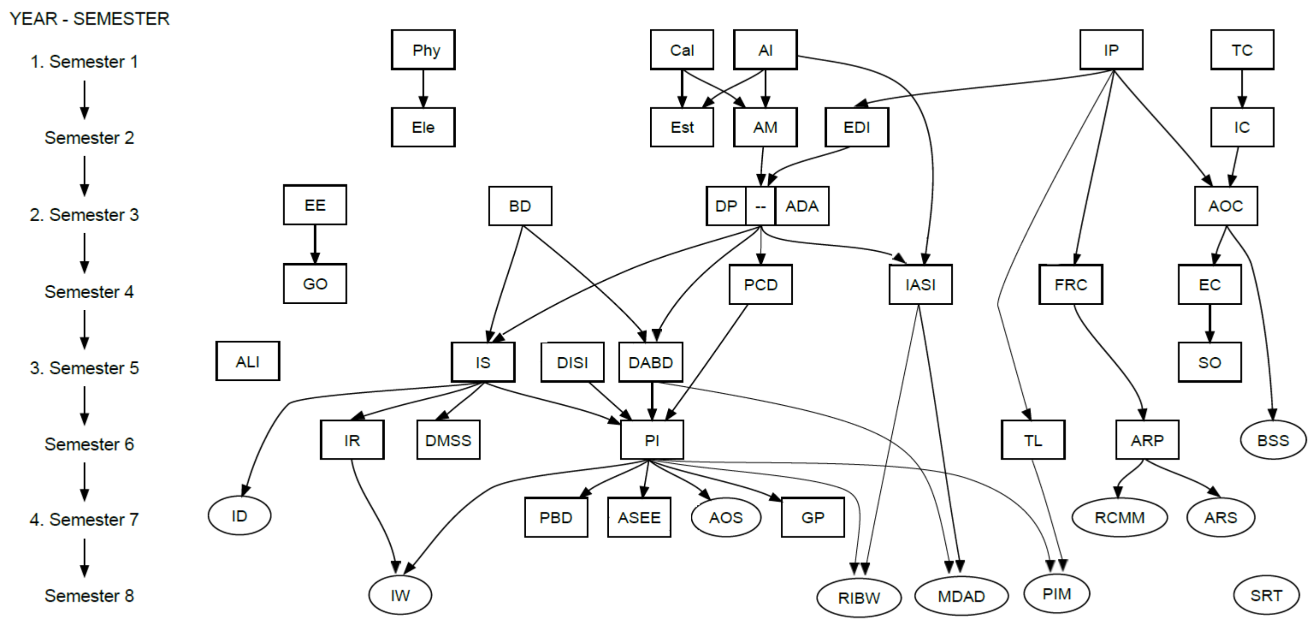


FIGURE 2. Required subjects that lecturers recommend to have successfully completed before taking their subject.

the students take the whole of that year's subjects which allows us to make decisions such as this.

- `Student Success Rate Category`. This feature is a metric of the student's performance in a specific category up to the moment of taking the instance subject in that category. It is calculated by dividing the number of subjects the student has passed in that specific subject category by the total number of subjects taken by the student in that specific subject category, as indicated in the following equation:

$$\forall s \subseteq S : \; SSRC_s = \frac{\sum_{j=1}^{m}(psc)_j}{\sum_{i=1}^{n}(nsc)_i}, \tag{3}$$

where S is the set of students that make up the dataset, $SSRC$ is as abbreviation for `Student Success Rate Category`, psc with $card(psc) = j$ is the number of subjects taken by student s in the instance category, and $nsc \subseteq psc$ with $card(nsc) = i$ is the number of subjects passed by student s in the instance category.
As was the case in the `Student Success Rate` feature, the `Student Success Rate Category` feature has the same cold start problem. The same reasoning is applied, and for first-year students this metric is calculated as follows:

$$\forall s \subseteq S : \; SSRC_s = \frac{npsc}{ntsc}, \tag{4}$$

where $npsc$ is the total number of successful attempts to pass subjects in the instance category by all students, and $ntsc$ is the total number of attempts to pass subjects in that category by all students.

- `Subject Pass Rate`. This feature is a basic metric of the overall students' performance in specific subjects. It is calculated by dividing the number of attempts in which students have passed the subject by the total number of attempts in which students have taken the subject, as indicated in the following equation:

$$\forall s \subseteq S : \; SPR_s = \frac{nsas}{nas}, \tag{5}$$

where $nsas$ is the total number of successful attempts to pass a subject s by all students and nas is the total number of attempts to pass that subject by all students.

- `Subject Requirements`. There are some subjects that have specific requirements that are set by the university lecturers. Even though it is not mandatory to successfully complete the required subjects in order to take a given subject, it is advisable in the sense that the knowledge acquired in the required subjects is useful in that it should increase the chance of success in the given subject.
A survey was conducted prior to the 2010/2011 academic year in which the lecturers responsible for each subject expressed their opinion about which other subjects were necessary to pass prior to taking theirs. As a result, the university prepared the diagram shown in Figure 2.
Based on that diagram, we created a new feature called `Subject Requirements` which is calculated by dividing the number of required subjects completed by the total number of required subjects. When a subject has no requirements we assume that all the requirements have been completed.

After creating these new features, we added them to the dataset transformed after the application of *One-Hot Encoding*. Using this new dataset, we constructed predictive models using the seven classification algorithms described in Subsection II-A. The results of these experiments together with those of the previous experiments are shown in Table 5 sorted by `Accuracy`. For the sake of readability of the paper, only the 20 best results are listed.

TABLE 5. Experimental results after creating new features. (NF: New Features, MLP: Multilayer Perceptron Neural Network, GBC: Gradient Boosting Classifier, LR: Linear Regression, SVM: Suppor Vector Machine, KNN: k Nearest Neighbour).

	Encoding	NF	Algorithm	Accuracy	F1-Score
1	One-Hot	True	MLP	0.746580	[0.6009, 0.8143]
2	One-Hot	True	GBC	0.742981	[0.5854, 0.8138]
3	One-Hot	True	LR	0.741541	[0.5859, 0.8121]
4	One-Hot	True	SVM	0.737941	[0.5439, 0.8162]
5	One-Hot	True	KNN	0.727142	[0.5708, 0.8]
6	One-Hot	True	RF	0.723542	[0.5789, 0.7942]
7	One-Hot	True	DT	0.712743	[0.5751, 0.783]
8	Label	False	GBC	0.712023	[0.455, 0.8043]
9	One-Hot	False	GBC	0.708423	[0.449, 0.8018]
10	One-Hot	False	LR	0.708423	[0.4841, 0.7968]
11	One-Hot	False	MLP	0.696904	[0.4128, 0.7957]
12	One-Hot	False	DT	0.686105	[0.2945, 0.7981]
13	One-Hot	False	KNN	0.676746	[0.5308, 0.7534]
14	Label	False	RF	0.673866	[0.5276, 0.751]
15	Label	False	DT	0.668826	[0.3801, 0.7741]
16	One-Hot	False	RF	0.665947	[0.5207, 0.7436]
17	Label	False	MLP	0.662347	[0.3153, 0.7759]
18	Label	False	KNN	0.660187	[0.5193, 0.7372]
19	Label	False	LR	0.652268	[0.2907, 0.7697]
20	One-Hot	False	SVM	0.650828	[0.0, 0.7885]

In light of the results of these new experiments, we can state that the models built using the datasets containing the new features get better results in all cases. This means that new features clearly contribute to increasing the accuracy of the predictions. The greatest accuracy at this point, `0.745860`, is achieved by the *Multilayer Perceptron* algorithm using *One-Hot Encoding*, with an improvement of more than 3 percentage points.

Although the results are clearly better, they are still not good enough to justify the creation of a Recommender System. They need to be improved to reach metrics which ensure that the creation of a Recommender System to assist students in choosing the right subjects will generate a notable impact on the students' progress in their course and on the faculty's capacity to increase the number of successful graduates.

D. SCALING

The implementation of this step consists of applying the 4 *scalers*: *MinMax, Standard, Robust, Normalizer* to the dataset obtained from the previous process.

After the application of this process we have 4 datasets, each one as a result of applying one of the *scaler* techniques. Subsequently, predictive models using the 7 classification algorithms discussed in Subsection II-A are constructed, yielding the results presented in Table 6. Note that the *Encoding* and *New Features* columns are omitted since in every dataset $\{NewFeatures = True\}$ and $\{Encoding = \text{“}One-Hot\text{”}\}$.

TABLE 6. Experimental results after applying *Scaling* techniques. (NF: New Features, MLP: Multilayer Perceptron Neural Network, GBC: Gradient Boosting Classifier, LR: Linear Regression, SVM: Suppor Vector Machine, KNN: k Nearest Neighbour).

	Scaler	Algorithm	Accuracy	F1-Score
1	MinMax	MLP	0.749460	[0.6142, 0.8145]
2	No	MLP	0.746580	[0.6009, 0.8143]
3	Standard	GBC	0.742981	[0.5854, 0.8138]
4	Robust	GBC	0.742981	[0.5854, 0.8138]
5	MinMax	GBC	0.742981	[0.5854, 0.8138]
6	No	GBC	0.742981	[0.5854, 0.8138]
7	Robust	LR	0.742261	[0.5876, 0.8126]
8	Standard	LR	0.742261	[0.5876, 0.8126]
9	No	LR	0.741541	[0.5859, 0.8121]
10	Robust	SVM	0.740821	[0.5745, 0.8137]
11	Robust	MLP	0.740821	[0.5881, 0.8109]
12	Normalizer	GBC	0.739381	[0.569, 0.8132]
13	MinMax	LR	0.739381	[0.582, 0.8107]
14	MinMax	SVM	0.739381	[0.5649, 0.814]
15	Standard	MLP	0.739381	[0.5987, 0.807]
16	Normalizer	MLP	0.738661	[0.6007, 0.8058]
17	Standard	SVM	0.738661	[0.5704, 0.8122]
18	No	SVM	0.737941	[0.5439, 0.8162]
19	Normalizer	SVM	0.734341	[0.5263, 0.8154]
20	No	KNN	0.727142	[0.5708, 0.8]

The greatest accuracy reached is `0.749460`. There are no major differences compared with the results obtained before applying *scaler* techniques. Indeed, the dataset used to build the second-ranked model was not scaled. While the individual models present results that are slight improvements, these improvements are still not enough, although they will form the basis for applying further pre-processing techniques.

Looking closely at Column `F1-Score` in Table 6, one can see where the major problem lies in predicting in this scenario, which needs to be more thoroughly analysed. The `F1-Score` is the harmonic mean of recall and precision, and is defined as follows:

$$F1\text{-Score} = \frac{2 * (precision * recall)}{precision + recall} \tag{6}$$

with the *precision* being defined as $TP/(TP + FN)$ and *recall* as $TP/(TP + FP)$, where TP stands for *True Positive*, FN for *False Negative*, and FP for *False Positive*, values that can be obtained from a *Confusion Matrix*, a table layout in which rows represent the actual class of instances and columns represent the class predicted by the model.

In the best model constructed so far, the `F1-Score` for the `Fail` class in the label feature is 0.6142 and 0.8145 for the `Pass` class. This indicates the difficulties that this model has at predicting the `Fail` class of the label. This problem is common to all the models. It makes sense because, as noted above, the label class frequencies are `{Pass:0.650828, Fail:0.349172}`, reflecting a major class imbalance in the dataset.

E. RESAMPLING

As noted above, the `qualifications` dataset presents a significant class imbalance. For that reason, the *Upsample*, *Downsample*, and *SMOTE* methods described for resampling in Subsection III-B are applied in order to deal with inbalanced datasets

Table 7 presents the results after completing the final experiments with the implementation of the *resampling* methods. *Encoding* and *New Features* columns are omitted since in every dataset $\{NewFeatures = True\}$ and $\{Encoding = ``One - Hot"\}$.

TABLE 7. Experimental results after applying *resampling* techniques. (NF: New Features, MLP: Multilayer Perceptron Neural Network, GBC: Gradient Boosting Classifier, LR: Linear Regression, SVM: Suppor Vector Machine, KNN: k Nearest Neighbour, RF: Random Forest, Alg: Algorithm).

	Scaler	Balance	Alg.	Accuracy	F1-Score
1	Robust	Upsample	RF	0.819879	[0.8308, 0.8075]
2	Robust	Upsample	GBC	0.761669	[0.779, 0.7414]
3	Robust	Upsample	MLP	0.757276	[0.774, 0.7378]
4	Robust	Upsample	SVM	0.756727	[0.7739, 0.7368]
5	Robust	SMOTE	RF	0.756178	[0.7638, 0.748]
6	Robust	SMOTE	GBC	0.752334	[0.7691, 0.733]
7	Robust	SMOTE	SVM	0.751236	[0.7754, 0.7212]
8	MinMax	No	MLP	0.749460	[0.6142, 0.8145]
9	Robust	SMOTE	MLP	0.748490	[0.7651, 0.7293]
10	No	No	MLP	0.746580	[0.6009, 0.8143]
11	Robust	Upsample	LR	0.744097	[0.7563, 0.7306]
12	MinMax	No	GBC	0.742981	[0.5854, 0.8138]
13	Standard	No	GBC	0.742981	[0.5854, 0.8138]
14	No	No	GBC	0.742981	[0.5854, 0.8138]
15	Robust	No	GBC	0.742981	[0.5854, 0.8138]
16	Standard	No	LR	0.742261	[0.5876, 0.8126]
17	Robust	No	LR	0.742261	[0.5876, 0.8126]
18	Robust	SMOTE	KNN	0.741900	[0.7587, 0.7226]
19	No	No	LR	0.741541	[0.5859, 0.8121]
20	Robust	No	SVM	0.740821	[0.5745, 0.8137]

It can be inferred from this table that *resampling* techniques clearly improve the accuracy of the models' predictions. Indeed, 8 of the 10 top-ranked models in terms of accuracy implement *resampling* techniques, and, in particular, the top 4 ranked models implement the *upsample* technique.

The model with greatest accuracy implements *One-Hot Encoding*, *Robust Scaler*, *Upsample resampling*, and uses the *Random Forest* algorithm, reaching an accuracy of `0.819879`, an increase of `7` percentage points over the best model created before implementing *resampling* techniques.

V. RESULTS

A. GENERAL RESULTS IN EVALUATING THE APPROACH

Over the course of Section IV, we have followed the evolution of the models' accuracies and *F1-Scores*. Our intention with this was to illustrate how, with the application of different pre-processing techniques, the resulting models get progressively better. The evolution of the evaluation metrics after the applications of the different pre-processing techniques can be appreciated in Table 8. We have considered the best model results at each step as reference values. Note that, in contrast with previous tables, the *Accuracy* is displayed as a percentage for the sake of readability, and the *F1-Score* shown is the average of the *F1-Score* class values shown previously so as to better analyse its overall evolution.

One observes in the table that the *creation of new features* and, above all, the application of *resampling* techniques considerably improved the results. However, *scaling* the data only slightly improved the best model. This might be because the data lie within a homogeneous range of values and there are few outliers, *i.e.*, extreme values which conform to unlikely observations. In spite of this, the application of *scaling* techniques is kept in the data-driven guidelines of the proposed approach for the reason that, given any dataset, its normalization might clearly benefit the models, and, in any event, it has no disadvantages beyond the computing time required for processing. Although this computing time may be relatively large, this should not be a problem given that the proposed approach is for small datasets.

The *resampling* process is an important part of our approach because it redresses the predictive errors deriving from imbalanced datasets, a potential characteristic of many small datasets in this area. If *resampling* techniques were not applied, the models' ability to predict the minority class would be poor. In cases in which the dataset is not imbalanced, this step could be omitted.

The basic application of *Feature Engineering* techniques to improve data representation and to create new features based on the existing data becomes crucial for constructing predictive models with an acceptable level of confidence and accuracy. Although this is not without controversy since adding new features may introduce biases and assumptions into the data, as long as it is done with due caution, we would encourage researchers and practitioners to consider applying such techniques.

B. RESULTS OF SEGMENTATION ACCORDING TO THE ACADEMIC YEAR

One of the steps of our approach consists of introducing new features based on the existing data so as to enhance the predictive power of the models. As shown in Subsection III-B4, the new `{Category, Student Success Rate, Student Success Rate Category, Subject Pass Rate, Requirements}` features were created. These new features are calculated based on each student's behaviour, which cannot be done for first-year students, thus creating biases between those students and the rest. For this reason, we studied the results according to segmentation of the academic year (see Table 9).

The results show that, as expected, the predictive capacity for the first year is lower than for subsequent years. The first-year accuracy is `0.785211` and the accuracy in subsequent years is `0.835594`. It is thus necessary to evaluate whether the model's first-year accuracy is good enough to validate deployment of the Recommender System to assist students in choosing suitable subjects. Notwithstanding the lower prediction capacity for first-year students, an accuracy of `78.5%` is still good, an improvement of `28.5` percentage

TABLE 8. Evolution of the results after the application of pre-processing techniques. FC: Frequent Class, NF: New Features.

	Random	FC	Encoding	NF	Scaling	Resampling
Accuracy	50%	65.08%	71.20%	74.65%	74.94%	81.98%
–Increment	-	-	*+6.12%*	*+3.45%*	*+0.29%*	*+7.04%*
F1-Score	-	-	0.6296	0.7076	0.7143	0.8191
–Increment	-	-	-	*+0.078*	*+0.0067*	*+0.1048*

TABLE 9. Results according to academic year segmentation (FY: First Year, SY: Subsequent Years.

Model	Accuracy	FY Accuracy	SY Accuracy
Best Model	0.819879	0.785211	0.835594

points over random guessing, and more than 13 points over the frequent class suggestion.

Whichever the case, in production it may result in the Recommender System suggesting the same subjects to every first-year student without attending to their individual characteristics, since for this stage we have no information in this sense. For this reason, the recommendations for this stage should be taken with caution, and in any case following the advice of professionals and respecting the students' own criteria. In the following stages, the Recommender System is more focused on the students' individual characteristics and abilities, which most often coincide with what they like. In these cases, the Recommender System attains an accuracy of 83.5%, which is an excellent result in accordance with the data available.

VI. DISCUSSION

In this section, several aspects of the approach will be discussed. One of the aims of this communication is to facilitate the replication of this work by researchers and practitioners. For this reason, it is worth discussing those characteristics of our approach that may impact or influence research findings. We can make a distinction between two levels of the characteristics of the decisions adopted: those corresponding to the scope of the work, and those corresponding to strategic decisions.

The characteristics of the work that correspond to its scope are:

a) **Small dataset**. As has been discussed above, the approach is specifically conceived for small datasets. We consider that a dataset with 6948 instances and 55 features (maximum number reached after the creation of new features) is too small to infer knowledge without proper data preprocessing, although it is true that many of these features were created after the application of the encoding methods. In Table 1 summarizing a comparison of related work, one can see that some of the studies analysed have a similar number of instances but few of them deal with this issue explicitly. In our experience, small datasets generate problems. Disturbance from outliers has a more direct effect. It is harder to deal with both *bias* and *variance* in trying to create models that generalize well from the training data. And problems with overfitting the training data arise easily if the dataset is consistent. We employed various techniques to avoid these problems and to generate as much knowledge as possible from the data we had.

b) **Imbalanced dataset**. The number of classes in our case study dataset are not equally represented. This is because each instance of the dataset corresponds to an attempt to pass a subject. Students with many fails abandon their studies, and students who tend to pass the subjects successfully continue with their studies with a high pass rate. This leads to the creation of an imbalanced dataset in which there are significantly fewer observations in the *fail* class than in the *pass* class. This is very common, as noted in [23]. The present approach assumes that the datasets are very likely to be imbalanced, and efforts are made to correct any potential negative impact that this might have. *Scaling* and *Normalizing Data* are used to avoid outliers, and *Resampling* methods are applied to handle imbalanced data. Additionally, the use of evaluation metrics such as *F1-Score* helps to better evaluate predictions in uneven class distribution datasets. In particular however, if the dataset is not imbalanced then the use of *Resampling* methods should be avoided.

c) **No use of personal data**. The proposed approach does not require the use of students' personal data. This ensures the students' privacy, a matter that is becoming more and more important every day. There is no doubt that access to personal data could clearly improve the model, as has been shown by studying previous work in the literature. Nevertheless, the present approach does not require any sharing of private data, which is often not easily accessible anyway, and would involve much time being spent on the required bureaucracy to deal with privacy policies. The final result is thus to actually facilitate the practical development and implementation of the approach.

The characteristics of the work that correspond to strategic decisions are:

d) **Instance $\neq$ Student**. In our approach, the instances of the dataset consist of students' attempts to pass subjects in a Computer Science degree course. Most of the studies found in the literature use students and their characteristics as instances. This is no minor assumption. With students considered as instances, the resulting models tend to "profile" the students' characteristics to make a prediction. With our approach, in addition to the students' overall characteristics, an individual history analysis is applied to calculate numerical values that correspond to students considered separately, thus considerably extending the overall understanding.

e) **Predict vs Take Action**. As noted before, one of the main objectives of this proposal was to take action and help both students and educational institutions – the former by their achieving better academic results, and the latter by maximizing the graduation rate and facilitating their decision-making processes. This represents a step forward beyond prediction of dropout and student performance which just identifies possible future events. Instead, it has a real impact on students and educational institutions.

VII. CONCLUSION AND FUTURE WORK

This communication has focused on researching into the prediction of students' performance and academic trajectories, a widely explored topic as was shown in Section II: *Related Works*. In contrast to most of those studies whose focus has been on predicting academic performance, success, or dropout, the novelty of the present work is that it has a bearing on students' progress by creating a Decision Support System that directly influences their academic path. The study has not only dealt with predictions, but has also aimed at helping students and faculty achieve better results by assisting them in their decision-making.

The present study has described the design of a full approach consisting of a set of guidelines for finding answers to the three research questions raised. Following these guidelines:

a) We have created models that assist students in selecting the subjects best suited to them with reasonable accuracy based on a dataset with few instances, answering affirmatively the first research question. This is possible as long as there are invaluable insights hidden in the scarce data available, and that the data is properly processed.

b) With respect to the second research question, we have empirically proven the capacity of Data Mining techniques such as *Feature Encoding*, *Feature Engineering*, *Scaling Data*, and *Resampling* to generate knowledge from an imbalanced dataset so as to help institutions predict the success of their students even in the absence of sufficient data.

c) We have created a Decision Support System based on a Recommender System using a dataset with few instances and imbalanced frequencies in the class label that is able to assist students in selecting the subjects best suited to them, and thereby maximize the number of those who graduate, thus answering affirmatively the third research question.

All of the experimentation done in this study was on the basis of real data of the Computer Science degree course in a public Spanish University, ensuring that, at all times, the work being developed involved real-life data, demonstrating the usefulness of the proposed approach in real environments.

In future work, the Decision Support System can be improved in the following ways:

a) Deploy the Decision Support System in the university enrollment system, and provide effective follow-up of students' academic path in comparison with the system recommendations. This would serve to test the system's success at assisting students beyond the test data.

b) Design a follow-up strategy that allows the Recommender System to be updated in accordance with changes in the degree course's organization. For example, a lecturer or lecturers assigned to teach a given subject are allocated to other subjects and replaced by other faculty members. It would be interesting to add to the dataset the lecturers that teach subjects for each specific academic year.

c) It would be interesting to calculate the new `Subject Requirements` feature on the basis of the exact mark earned by the students in the required courses instead of just `{Pass, Fail}`. This may be a better indication of the student's knowledge.

d) Provide a wider scenario, not only focused on Computer Science, for a better validation of the approach.

e) Additionally, given that students' capability to select subjects in the first year is limited, it could be interesting to work with the educational centers that supply the majority of students to the university to establish a joint strategy to improve the students' degree selection. Even though it implies access to external data and collaborative action beyond technical knowledge, it would be worth assessing.
will require the access to external data not always easy to acquire
-Authors focus on students in the first year since they argue that dropout rate is higher, however capability for students to select subjects in the first year is limited. Authors may think about describe an additional scenario?

APPENDIX

A. DETAILED DESCRIPTION OF THE ORIGINAL DATASET

A detailed description of the original dataset is provided in Table 10.

B. EXPERIMENTAL RESULTS

The complet list of experiment results is shown in Table 11 and Table 11b.

C. TECHNICAL DETAILS

This section is specifically created to provide specific content and accurate technical details. This will make it easy for other researchers or data scientists the replication of the work and the experimentation reflected in this manuscript whether for research purposes or for implementing this approach in Higher Educational Institutions real scenarios.

This appendix has been divided into 5 subsections. Each one of the subsection focus on a part of the experimentation and specific *source code* and discussion is provided to deal with concrete methods or techniques with the scope of the

TABLE 10. Experimental results after creating new features.

Feature	Type	Classes	Frequency
Degree	Categorical	*1 class*	{Computer Science Engineering:100%}
Completion Year	Categorical	*5 classes*	{2017-18: 35.0748%, 2016-17: 27.4180%, 2015-16: 20.2216%, 2014-15: 12.8670%, 2013-14: 4.4185%}
Subject	Categorical	*45 classes*	{Fundamentos de redes y comunicaciones: 0.055892, Física: 0.046096, Electrónica: 0.040046, Arquitectura de redes y protocolos: 0.035436, Análisis y diseño de algoritmos: 0.035004, Administración y Organización de Computadores: 0.034572, Diseño y Administración de BBDD: 0.031403, Estructura de datos y de la información: 0.030971, Programación Concurrente y Distribuida: 0.030539, Diseño y Modelado de Software: 0.029818, Programación en internet: 0.029386, Estructura de computadores: 0.029098, Teoría de lenguajes: 0.028954, Desarrollo de programas: 0.028810, Álgebra lineal: 0.027226, Cálculo: 0.026217, Tecnología de comptuadores: 0.024921, Introducción a los computadores: 0.023624, Gestión de los organizaciones: 0.022472, Bases de datos: 0.021896, Inteligencia Artificial: 0.021752, Sistemas Operativos: 0.021608, Arquitecturas software en entornos empresariales: 0.020887, Ampliación de Matemáticas: 0.020743, Estadística: 0.020743, Auditoría y legislación informática: 0.020311, Introducción a la programación: 0.020311, Ingeniería de software: 0.019735, Programación de BBDD: 0.018438, Ingeniería de requisitos: 0.018438, Diseño e interacción de sistemas de información: 0.018294, Proyecto fin de grado: 0.018006, Gestión de proyectos software: 0.018006, Economía y empresa: 0.015846, Imagen digital: 0.012532, Arquitecturas orientadas a servicios: 0.012532, Ingeniería Web: 0.012100, Recuperación de la Información y busqueda en la web: 0.011236, Prácticas externas: 0.010660, Minería de datos: 0.010372, Redes de comunicaciones: 0.006770, Seguridad en redes telemáticas: 0.006338, Biometría y seguridad de sistemas: 0.004610, Procesamiento de la información multimedia: 0.004177, Administración de redes y servicios: 0.003169}
Credits	Numerical	*2 classes*	{6: 98.2009%, 12: 1.7991%}
Attempt Number	Numerical	*6 classes*	{1: 84.5999%, 2: 12.3633%, 3: 2.4324%, 4: 0.5037%, 5: 0.0864%, 6: 0.0144%}
Degree Year	Categorical	*4 classes*	{1: 28.0656%, 2: 29.6200%, 3: 25.3454%, 4: 16.9689%}
Academic Year	Categorical	*8 classes*	{2010-11: 6.2464%, 2011-12: 11.6149%, 2012-13: 17.7893%, 2013-14: 20.4951%, 2014-15: 19.4876%, 2015-16: 14.3926%,k Nearest Neighbours 2016-17: 7.7001%, 2017-18: 2.2740%}
Call	Categorical	*8 classes*	{JUN: 36.8020%, FEB: 19.9914%, JUL: 19.0271%, JAN: 17.2855%, SEP: 5.5124%, JAX: 0.7484%, FEX: 0.3742%, NOV: 0.2591%}
Mark	Categorical	*7 classes*	{Not Taken: 0.124064%, Fail: 0.219919%, Sufficient: 0.291019%, Very Good: 0.260507%, Outstanding: 0.075993%, With Honours: 0.027634%, Compensation: 0.000864%}
Pass/Fail	Categorical	*2 classes*	Pass: 65.6016%, Fail: 34.3984%}

Python programming language and the *sklearn*, *pandas* and *numpy* libraries.

1) ENCODING

Due that many algorithms in *sklearn* are not set up to work with non-numerical features, it is necessary to preprocess categorical features before creating models with such algorithms. Throughout this manuscript, the *Label Encoding* and *One-Hot Encoding* encoding strategies have been discussed.

With respect to the source code, *sklearn* has the *LabelEncoder()* and the *OneHotEncoder()* methods in the *preprocessing* library. This methods can encode the whole dataset. In our implementation we have not used these methods and we have deal with individual features separately.

The following listing shows the source code that perform *One-Hot Encoding* on the `Call` feature:

```
df["Call"]=df["Call"].astype('category').cat.codes
```

Listing 1. Label Encoding Example

After the execution of this piece of code the {Ordinary, Extraordinary} classes of the `Call` categorical feature are encoded in numerical values: {0, 1}.

The following listing shows the source code that perform *One-Hot Encoding* on the `Call` feature:

```
import pandas as pd
dummy = pd.get_dummies(df["Call"])
df = pd.concat([df, dummy], axis=1)
df.drop("Call", axis=1, inplace=True)
```

Listing 2. One-Hot Encoding Example

After the execution of this piece of code the `Call` categorical feature with classes {Ordinary, Extraordinary} will be split into two new binary features: `Ordinary` and `Extraordinary`.

2) SCALING

Machine Learning algorithms tend to perform better and converge faster when input features are on similar scales and close to being normally distributed. To do so, throughout

TABLE 11. Complete list of experimental results. (Part 1/2).

	Encoding	NewFeatures	Scaler	Balance	Algorithm	Accuracy	F1-Score
1	One-Hot	True	Robust	Upsample	Random Forest	0.819879	[0.8308, 0.8075]
2	One-Hot	True	Robust	Upsample	Gradient Boosting Classifier	0.761669	[0.779, 0.7414]
3	One-Hot	True	Robust	Upsample	MLP Neural Network	0.757276	[0.774, 0.7378]
4	One-Hot	True	Robust	Upsample	Support Vector Machine	0.756727	[0.7739, 0.7368]
5	One-Hot	True	Robust	SMOTE	Random Forest	0.756178	[0.7638, 0.748]
6	One-Hot	True	Robust	SMOTE	Gradient Boosting Classifier	0.752334	[0.7691, 0.733]
7	One-Hot	True	Robust	SMOTE	Support Vector Machine	0.751236	[0.7754, 0.7212]
8	One-Hot	True	MinMax	No	MLP Neural Network	0.749460	[0.6142, 0.8145]
9	One-Hot	True	Robust	SMOTE	MLP Neural Network	0.748490	[0.7651, 0.7293]
10	One-Hot	True	No	No	MLP Neural Network	0.746580	[0.6009, 0.8143]
11	One-Hot	True	Robust	Upsample	Logistic Regression	0.744097	[0.7563, 0.7306]
12	One-Hot	True	MinMax	No	Gradient Boosting Classifier	0.742981	[0.5854, 0.8138]
13	One-Hot	True	Standard	No	Gradient Boosting Classifier	0.742981	[0.5854, 0.8138]
14	One-Hot	True	No	No	Gradient Boosting Classifier	0.742981	[0.5854, 0.8138]
15	One-Hot	True	Robust	No	Gradient Boosting Classifier	0.742981	[0.5854, 0.8138]
16	One-Hot	True	Standard	No	Logistic Regression	0.742261	[0.5876, 0.8126]
17	One-Hot	True	Robust	No	Logistic Regression	0.742261	[0.5876, 0.8126]
18	One-Hot	True	Robust	SMOTE	k Nearest Neighbour	0.741900	[0.7587, 0.7226]
19	One-Hot	True	No	No	Logistic Regression	0.741541	[0.5859, 0.8121]
20	One-Hot	True	Robust	No	Support Vector Machine	0.740821	[0.5745, 0.8137]
21	One-Hot	True	Robust	No	MLP Neural Network	0.740821	[0.5881, 0.8109]
22	One-Hot	True	Robust	SMOTE	Logistic Regression	0.740802	[0.7539, 0.7262]
23	One-Hot	True	Normalizer	No	Gradient Boosting Classifier	0.739381	[0.569, 0.8132]
24	One-Hot	True	Standard	No	MLP Neural Network	0.739381	[0.5987, 0.807]
25	One-Hot	True	MinMax	No	Logistic Regression	0.739381	[0.582, 0.8107]
26	One-Hot	True	MinMax	No	Support Vector Machine	0.739381	[0.5649, 0.814]
27	One-Hot	True	Standard	No	Support Vector Machine	0.738661	[0.5704, 0.8122]
28	One-Hot	True	Normalizer	No	MLP Neural Network	0.738661	[0.6007, 0.8058]
29	One-Hot	True	Robust	Upsample	k Nearest Neighbour	0.738056	[0.7545, 0.7192]
30	One-Hot	True	No	No	Support Vector Machine	0.737941	[0.5439, 0.8162]
31	One-Hot	True	Normalizer	No	Support Vector Machine	0.734341	[0.5263, 0.8154]
32	One-Hot	True	Robust	Downsample	Support Vector Machine	0.734310	[0.7524, 0.7133]
33	One-Hot	True	Robust	Downsample	Logistic Regression	0.731172	[0.7433, 0.7179]
34	One-Hot	True	No	No	k Nearest Neighbour	0.727142	[0.5708, 0.8]
35	One-Hot	True	Normalizer	No	Random Forest	0.727142	[0.5794, 0.7981]
36	One-Hot	True	Normalizer	No	k Nearest Neighbour	0.725702	[0.5656, 0.7996]
37	One-Hot	True	Standard	No	Random Forest	0.724262	[0.5805, 0.7946]
38	One-Hot	True	MinMax	No	k Nearest Neighbour	0.723542	[0.5646, 0.7975]
39	One-Hot	True	MinMax	No	Random Forest	0.723542	[0.5789, 0.7942]
40	One-Hot	True	No	No	Random Forest	0.723542	[0.5789, 0.7942]
41	One-Hot	True	Robust	No	Random Forest	0.723542	[0.5799, 0.794]
42	One-Hot	True	Normalizer	No	Logistic Regression	0.722822	[0.5133, 0.8062]
43	One-Hot	True	Robust	SMOTE	Decision Tree	0.722131	[0.7367, 0.7058]
44	One-Hot	True	Robust	Downsample	MLP Neural Network	0.721757	[0.7437, 0.6957]
45	One-Hot	True	Robust	Downsample	Gradient Boosting Classifier	0.720711	[0.741, 0.6969]
46	One-Hot	True	Robust	No	k Nearest Neighbour	0.718503	[0.566, 0.7917]
47	One-Hot	True	Robust	Downsample	k Nearest Neighbour	0.717573	[0.7278, 0.7065]
48	One-Hot	True	Normalizer	No	Decision Tree	0.714183	[0.5093, 0.7984]
49	One-Hot	True	Robust	Downsample	Random Forest	0.713389	[0.7215, 0.7047]
50	One-Hot	True	MinMax	No	Decision Tree	0.712743	[0.5751, 0.783]
51	One-Hot	True	Standard	No	Decision Tree	0.712743	[0.5751, 0.783]
52	One-Hot	True	Robust	No	Decision Tree	0.712743	[0.5751, 0.783]
53	One-Hot	True	No	No	Decision Tree	0.712743	[0.5751, 0.783]
54	One-Hot	True	Robust	Downsample	Decision Tree	0.712343	[0.7383, 0.6806]

TABLE 11. ***(Continued.)*** **Complete list of experimental results. Part (2/2).**

	Encoding	NewFeatures	Scaler	Balance	Algorithm	Accuracy	F1-Score
55	Label	False	No	No	Gradient Boosting Classifier	0.712023	[0.455, 0.8043]
56	One-Hot	True	Standard	No	k Nearest Neighbour	0.712023	[0.5413, 0.7901]
57	One-Hot	False	No	No	Logistic Regression	0.708423	[0.4841, 0.7968]
58	One-Hot	False	No	No	Gradient Boosting Classifier	0.708423	[0.449, 0.8018]
59	One-Hot	True	Robust	Upsample	Decision Tree	0.708402	[0.7475, 0.655]
60	One-Hot	False	No	No	MLP Neural Network	0.696904	[0.4128, 0.7957]
61	One-Hot	False	No	No	Decision Tree	0.686105	[0.2945, 0.7981]
62	One-Hot	False	No	No	k Nearest Neighbour	0.676746	[0.5308, 0.7534]
63	Label	False	No	No	Random Forest	0.673866	[0.5276, 0.751]
64	Label	False	No	No	Decision Tree	0.668826	[0.3801, 0.7741]
65	One-Hot	False	No	No	Random Forest	0.665947	[0.5207, 0.7436]
66	Label	False	No	No	MLP Neural Network	0.662347	[0.3153, 0.7759]
67	Label	False	No	No	k Nearest Neighbour	0.660187	[0.5193, 0.7372]
68	Label	False	No	No	Logistic Regression	0.652268	[0.2907, 0.7697]
69	Label	False	No	No	Support Vector Machine	0.650828	[0.0, 0.7885]
70	One-Hot	False	No	No	Support Vector Machine	0.650828	[0.0, 0.7885]

this manuscript, the *Standard Scaler*, *Robust Scaler*, *MinMax Scaler* and *Normalizer* scalers scaling strategies available in *Python* have been discussed.

The *sklearn* library have specific methods for dealing with this process. The following listing shows the source code that performs the different *Scaling* strategies that we have followed in our experimentation:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer

standard_scaler = StandardScaler()
Xtr_s = standard_scaler.fit_transform(xtrain)
Xte_s = standard_scaler.transform(xtest)

robust_scaler = RobustScaler()
Xtr_r = robust_scaler.fit_transform(xtrain)
Xte_r = robust_scaler.transform(xtest)

minmax_scaler = MinMaxScaler(feature_range=(0,1))
Xtr_mm = minmax_scaler.fit_transform(xtrain)
Xte_mm = minmax_scaler.transform(xtest)

normalizer_scaler = Normalizer()
Xtr_n = normalizer_scaler.fit_transform(xtrain)
Xte_n = normalizer_scaler.transform(xtest)
```

Listing 3. Scaling Strategies Example

```
from sklearn.utils import resample

# Separate majority and minority classes
df_majority = df[df['Pass/Fail']==1]
df_minority = df[df['Pass/Fail']==0]

# Upsample minority class
df_upsample = resample
    (df_minority,
    replace=True,      # sample with replacement
    n_samples=4552,    # to match majority class
    random_state=123) # reproducible results

# Combine majority class with upsampled class
df_upsample=pd.concat([df_majority, df_upsample])
```

Listing 4. Upsample Resampling Strategy Example

3) RESAMPLING

Models built using imbalance datasets may have difficulties predicting classes with few observations in the label class. Throughout this manuscript, *Upsample*, *Downsample* and *SMOTE* strategies available in *Python* have been used to mitigate such a problem.

The *sklearn* library have specific utilities to deal with this problem. The following pieces of code performs the different *Resampling* strategies that we have followed throughout the manuscript:

```
from sklearn.utils import resample

# Separate majority and minority classes
df_majority = df[df['Pass/Fail']==1]
df_minority = df[df['Pass/Fail']==0]

# Downsample majority class
df_downsample = resample
    (df_majority,
    replace=False,     # sample without replacement
    n_samples=2390,    # to match mainority class
    random_state=123) # reproducible results

# Combine minority class with downsampled class
df_downsample=pd.concat([df_downsample,
    df_minority])
```

Listing 5. Downsample Resampling Strategy Example

```
from imblearn.over_sampling import SMOTE

resample = SMOTE(
            sampling_strategy='minority',
            random_state=5,
            k_neighbors=5)

Xsmote, Ysmote = resample.fit_resample(
                df.drop('Pass/Fail',axis=1),
                df['Pass/Fail'])
```

Listing 6. SMOTE Resampling Strategy Example

4) ALGORITHMS

To create the recommender system models we have selected some well-known classification algorithms. Implementations of these algorithms are in the *sklearn* library.

```python
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(
                            criterion="entropy",
                            max_depth = 4)

#Random Forest
from sklearn.ensemble import
    RandomForestClassifier

model = RandomForestClassifier(
                            n_jobs=2,
                            random_state=0)

#Support Vector Machines
from sklearn import svm
model = svm.SVC(kernel='rbf')

#KNN
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 5)

#Multilayer perceptron (neural networks)
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(
                    hidden_layer_sizes=(8,8,8),
                    activation='relu',
                    solver='adam',
                    max_iter=500)

#GradientBoostingClassifier
from sklearn.ensemble import
    GradientBoostingClassifier
model = GradientBoostingClassifier()

#Logistic Regression
from sklearn.linear_model import
    LogisticRegression
model = LogisticRegression()
```

Listing 7. Algorithms instantiation and hyperparameters

The following pieces of code show the instantiation and application of these algorithms and the set of hyperparameters for each one of them.

Before creating the models the datasets have been properly split in *Train* and *Test*, as shown in the next piece of code.

In all cases, to create the model and make the prediction, the following code has been used.

```python
from sklearn.metrics import accuracy_score
X = df.drop('Pass/Fail',axis=1)
Y = df['Pass/Fail']
from sklearn.model_selection import
    train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X,
     Y, test_size=0.2, random_state=4)
```

Listing 8. Train / Test Split

```python
model.fit(Xtrain, ytrain)
prediction = model.predict(Xtest)
```

Listing 9. Fitting models and predicting

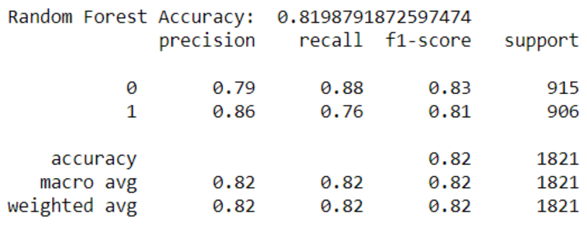
5) EVALUATION

Accuracy and *F1-Score* metrics have been used throughout the manuscript for evaluating the results of every experiment. To obtain this metrics, as well as other metrics such as *Precision*, *Recall* and *Support* and a graphic representation of the *Confusion Matrix* the following code have been used.

```python
from sklearn.metrics import classification_report
print (classification_report(ytest, prediction))

import itertools
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(ytest, prediction,
    labels=[0,1])
print(matrix)

accuracy = metrics.accuracy_score(ytest,
    prediction)

from sklearn.metrics import
    precision_recall_fscore_support
precision, recall, f1, _ =
    precision_recall_fscore_support(ytest,
    prediction, average=None)
```

Listing 10. Models Evaluation

The output of the code can be seen in Figure 3 and the *Confusion Matrix* in Figure 4.



```
Random Forest Accuracy:  0.8198791872597474
                 precision    recall  f1-score   support

              0       0.79      0.88      0.83       915
              1       0.86      0.76      0.81       906

       accuracy                           0.82      1821
      macro avg       0.82      0.82      0.82      1821
   weighted avg       0.82      0.82      0.82      1821
```

FIGURE 3. Best model metrics.

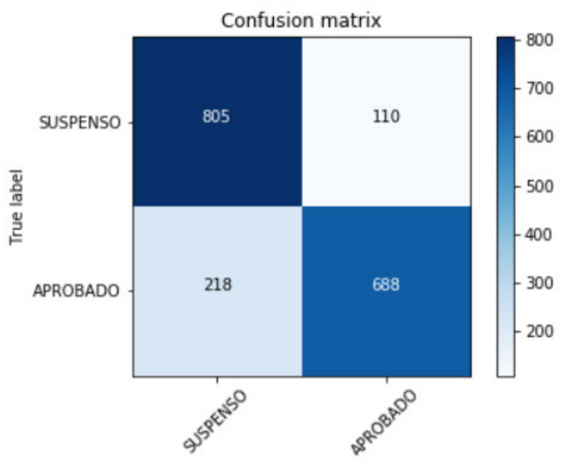


FIGURE 4. Best model Confusion Matrix.

REFERENCES

[1] P. Dolton, R. Asplund, and E. Barth, Eds., *Education and Inequality Across Europe*. Cheltenham, U.K.: Edward Elgar Publishing, 2009.

[2] *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*, The National Academies Press, Washington, DC, USA, 2018.

[3] P. Blikstein. (2018). *Pre-College Computer Science Education: A Survey of the Field*. Accessed: May 16, 2020. [Online]. Available: https://services.google.com/fh/files/misc/pre-college-computer-science%-education-report.pdf
[4] P. Kinnunen and L. Malmi, "Why students drop out CS1 course?" in *Proc. Int. workshop Comput. Edu. Res. ICER*, 2006, pp. 97–108.
[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[6] L. Valiant, *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. New York, NY, USA: Basic Books, Inc., USA, 2013.
[7] H. Guruler, A. Istanbullu, and M. Karahasan, "A new student performance analysing system using knowledge discovery in higher educational databases," *Comput. Edu.*, vol. 55, no. 1, pp. 247–254, Aug. 2010.
[8] S. Helal, J. Li, L. Liu, E. Ebrahimie, S. Dawson, D. J. Murray, and Q. Long, "Predicting academic performance by considering student heterogeneity," *Knowl.-Based Syst.*, vol. 161, pp. 134–146, Dec. 2018.
[9] Y. Nieto, V. García-Díaz, C. Montenegro, and R. G. Crespo, "Supporting academic decision making at higher educational institutions using machine learning-based algorithms," *Soft Comput.*, vol. 23, no. 12, pp. 4145–4153, Jun. 2019.
[10] S. Huang and N. Fang, "Predicting student academic performance in an engineering dynamics course: A comparison of four types of predictive mathematical models," *Comput. Edu.*, vol. 61, pp. 133–145, Feb. 2013.
[11] I. E. Livieris, T. Kotsilieris, V. Tampakas, and P. Pintelas, "Improving the evaluation process of students' performance utilizing a decision support software," *Neural Comput. Appl.*, vol. 31, no. 6, pp. 1683–1694, 2019.
[12] B. Şen, E. Uçar, and D. Delen, "Predicting and analyzing secondary education placement-test scores: A data mining approach," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9468–9476, Aug. 2012.
[13] S. Natek and M. Zwilling, "Student data mining solution–knowledge management system related to higher education institutions," *Expert Syst. Appl.*, vol. 41, no. 14, pp. 6400–6407, Oct. 2014.
[14] H. A. Mengash, "Using data mining techniques to predict student performance to support decision making in university admission systems," *IEEE Access*, vol. 8, pp. 55462–55470, 2020.
[15] K. T. Chui, R. W. Liu, M. Zhao, and P. O. De Pablos, "Predicting Students' performance with school and family tutoring using generative adversarial network-based deep support vector machine," *IEEE Access*, vol. 8, pp. 86745–86752, 2020.
[16] R. C. Deo, Z. Mundher Yaseen, N. Al-Ansari, T. Nguyen-Huy, T. Ashley Mcpherson Langlands, and L. Galligan, "Modern artificial intelligence model development for undergraduate student performance prediction: An investigation on engineering mathematics courses," *IEEE Access*, vol. 8, pp. 136697–136724, 2020.
[17] N. Tomasevic, N. Gvozdenovic, and S. Vranes, "An overview and comparison of supervised data mining techniques for student exam performance prediction," *Comput. Edu.*, vol. 143, Jan. 2020, Art. no. 103676.
[18] F. Marbouti, H. A. Diefes-Dux, and K. Madhavan, "Models for early prediction of at-risk students in a course using standards-based grading," *Comput. Edu.*, vol. 103, pp. 1–15, Dec. 2016.
[19] C. C. Gray and D. Perkins, "Utilizing early engagement and machine learning to predict student outcomes," *Comput. Edu.*, vol. 131, pp. 22–32, Apr. 2019.
[20] D. Olaya, J. Vásquez, S. Maldonado, J. Miranda, and W. Verbeke, "Uplift modeling for preventing student dropout in higher education," *Decis. Support Syst.*, vol. 134, Jul. 2020, Art. no. 113320.
[21] V. L. Miguéis, A. Freitas, P. J. V. Garcia, and A. Silva, "Early segmentation of students according to their academic performance: A predictive modelling approach," *Decis. Support Syst.*, vol. 115, pp. 36–51, Nov. 2018.
[22] A.-S. Hoffait and M. Schyns, "Early detection of university students with potential difficulties," *Decis. Support Syst.*, vol. 101, pp. 1–11, Sep. 2017.
[23] C. Márquez-Vera, A. Cano, C. Romero, and S. Ventura, "Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data," *Int. J. Speech Technol.*, vol. 38, no. 3, pp. 315–330, Apr. 2013.
[24] A. Polyzou and G. Karypis, "Feature extraction for next-term prediction of poor student performance," *IEEE Trans. Learn. Technol.*, vol. 12, no. 2, pp. 237–248, Apr. 2019.
[25] J. Figueroa-Canas and T. Sancho-Vinuesa, "Early prediction of dropout and final exam performance in an online statistics course," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 15, no. 2, pp. 86–94, May 2020.
[26] L. P. Macfadyen and S. Dawson, "Mining LMS data to develop an 'early warning system' for educators: A proof of concept," *Comput. Edu.*, vol. 54, no. 2, pp. 588–599, Feb. 2010.
[27] Y. Chen, Q. Zheng, S. Ji, F. Tian, H. Zhu, and M. Liu, "Identifying at-risk students based on the phased prediction model," *Knowl. Inf. Syst.*, vol. 62, no. 3, pp. 987–1003, Mar. 2020.
[28] P. M. Moreno-Marcos, P. J. Muñoz-Merino, J. Maldonado-Mahauad, M. Pérez-Sanagustín, C. Alario-Hoyos, and C. Delgado Kloos, "Temporal analysis for dropout prediction using self-regulated learning strategies in self-paced MOOCs," *Comput. Edu.*, vol. 145, Feb. 2020, Art. no. 103728.
[29] L. Qiu, Y. Liu, and Y. Liu, "An integrated framework with feature selection for dropout prediction in massive open online courses," *IEEE Access*, vol. 6, pp. 71474–71484, 2018.
[30] K. Coussement, M. Phan, A. De Caigny, D. F. Benoit, and A. Raes, "Predicting student dropout in subscription-based online learning environments: The beneficial impact of the logit leaf model," *Decis. Support Syst.*, vol. 135, Aug. 2020, Art. no. 113325.
[31] K. Ramasubramanian and A. Singh, *Feature Engineering*. Berkeley, CA, USA: Apress, 2017, pp. 181–217.
[32] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2018.
[33] M. Kuhn and K. Johnson, *Feature Engineering and Selection: A Practical Approach for Predictive Models*. Boca Raton, FL, USA: CRC Press, 2019.
[34] A. J. Fernández-García, L. Iribarne, A. Corral, J. Criado, and J. Z. Wang, "A recommender system for component-based applications using machine learning techniques," *Knowl.-Based Syst.*, vol. 164, pp. 68–84, Jan. 2019.
[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.

ANTONIO JESÚS FERNÁNDEZ-GARCÍA received the Ph.D. degree in computer science from the Universidad de Almería (UAL), in 2019. He is currently a Researcher and a Member of the Quercus Software Engineering Group, Department of Computer Science, University of Extremadura (UEX), and the Applied Computing Group, UAL. He is also an Assistant Professor with the Universidad Internacional de La Rioja (UNIR). He has published more than ten scientific publications in journals and international conferences. His research interests include recommender systems, machine learning, artificial intelligence, data mining, data engineering, and software engineering.

ROBERTO RODRÍGUEZ-ECHEVERRÍA is currently a Member of the Quercus Software Engineering Group and a Professor of computer languages and systems with the University of Extremadura, Spain. He has published more than 50 scientific publications in journals and international conferences. His research interests include software engineering, web engineering, model-driven engineering, legacy software modernization, and end-user development.

JUAN CARLOS PRECIADO received the Ph.D. degree in computer science from the University of Extremadura (UEX), in 2008. He is currently a Professor and a Member of the Quercus Software Engineering Group, Department of Computer Science, UEX. He was a Vicerrector of this university several years. His research interests include model-driven development and web and data engineering, where he has published around 100 articles in the software engineering field.

JOSÉ MARÍA CONEJERO MANZANO received the Ph.D. degree in computer science from the Universidad de Extremadura, in 2010. He is currently an Assistant Professor with the Universidad de Extremadura. He is the author of more than 50 articles of journals and conference proceedings and has also participated in different journals and conferences as a member of the program committee. His research interests include the aspect-oriented software development, requirements engineering, and model-driven development or ambient intelligence.

FERNANDO SÁNCHEZ-FIGUEROA received the Ph.D. degree in computer science from UEX. He is currently a Professor with the Department of Computer Science, UEX. He is the coauthor of more than 100 publications related to software engineering. His research interests include web engineering, big data visualization, and MDD.