Student Performance Prediction

Mathematics for Machine Learning Exam

Debora Caldarola

Prof. Francesco Vaccarino

Prof. Mauro Gasparini

Politecnico di Torino

a.y. 2019/2020

Contents

1 Introduction

Education is a key factor for achieving a long-term economic progress. During the last decades, the Portuguese educational level has improved. However, the statistics keep the Portugal at Europe's tail end due to its high student failure and dropping out rates. The education arena offers a fertile ground for Data Mining applications, since there are multiple sources of data. For instance, there are several interesting questions for this domain that could be answered using Data Mining techniques: Who are the students taking most credit hours? Who is likely to return for more classes? What type of courses can be offered to attract more students? What are the main reasons for student transfers? Is is possible to predict student performance? What are the factors that affect student achievement? This work will focus in the last two questions [1].

2 Student Performance Prediction

2.1 Introduction to Final Grade Prediction

Predicting student performance is really valuable as appropriate measures can be taken during the semester to help struggling students achieve a passing mark.

In Portugal, a 20-point grading scale is used, where 0 is the lowest grade and 20 is the perfect score. During the school year, students are evaluated in three periods and the last evaluation (`G3` in Table[1]) corresponds to the final grade. Dealing with the Mathematics and Portuguese grades, the output can be modeled using three supervised approaches:

1. Binary classification – *pass* if `G3` >= 10, else *fail*;

2. 5-Level classification – (16-20: excellent; 14-15: good 12-13: satisfactory 10-11: sufficient 0-9: fail);

3. Regression – on the final grade `G3` (numeric output between 0 and 20).

In this work we will focus on the first task, the binary classification.

2.2 Dataset and environment setup

The analysis and the experiments are performed on UCI Machine Learning Student Performance Data Set [2].

The dataset contains student achievements in Secondary education collected during the 2005-2006 school years from two public schools, from the Alentejo region of Portugal. with a lot of interesting social, age, gender information.

All results are generated using Python programming language and *Colab* environment. Mainly, `sklearn`, `imblearn` and `numpy` libraries are used.

All the code can be found on GitHub at `https://github.com/debcaldarola/Student-Performance-Prediction`.

3 Dataset Analysis

The Student Performance dataset contains two smaller datasets, one showing the grades achieved in Mathematics (with 396 entries) and one regarding the ones in Portuguese (with 650 entries).

In particular, the two datasets are a bit overlapped, which means that some students are present in both classes. There is no particular ID for each student, but it is possible to identify them by looking at the common features in both datasets, as suggested on the Dataset Description. It was possible to identify 382 students out of 1044 belonging to both courses.

The datasets show no missing values.

3.1 Features Description and Analysis

Each datum sample is characterized by the same 33 attributes, shown in Table[1]. They give additional information about the students, such as attended school, sex, family-related details, alcohol consumption, and so on. There are both binary, categorical and nominal features and they will be treated accordingly in Chapter 4.

The attribute that interests us most is `G3`, which represents students' final grades and is our target output. In order to perform binary classification, the variable is mapped from [0, 20] to [0,1], where 0 means *fail* and 1 *pass*, according to the following criteria: if `G3` >= 10, the student succeeded; otherwise, he failed.

As it can be seen in Figure[1], the number of students in both datasets is heavily skewed towards passing and this could make the employed algorithms skewed as well. Imbalanced datasets pose a challenge for predictive modelling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of examples for each class. That results in models having poor predictive performance, specifically for the minority class. This issue will be further discussed in Chapter 5.

In order to gain a better understanding of the features and to investigate the dependence between them, the correlation matrices are computed: each cell in the table shows the pairwise Pearson Correlation Coefficient r, calculated with the formula (3.1)

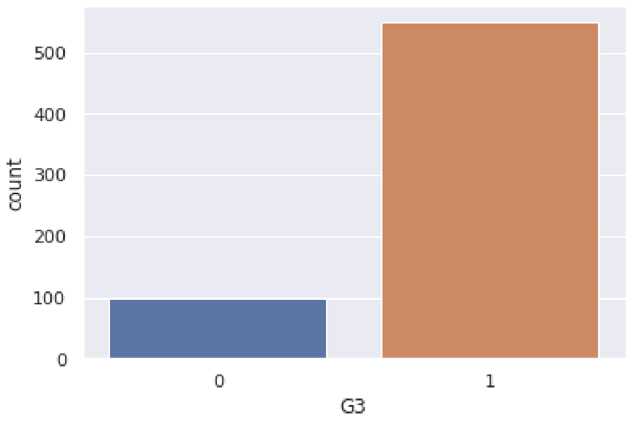
$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2} \tag{3.1}$$

where n is the sample size, x_i and y_i are the individual sample points indexed with i. The higher the absolute value of the coefficient, the more correlated the two variables are.

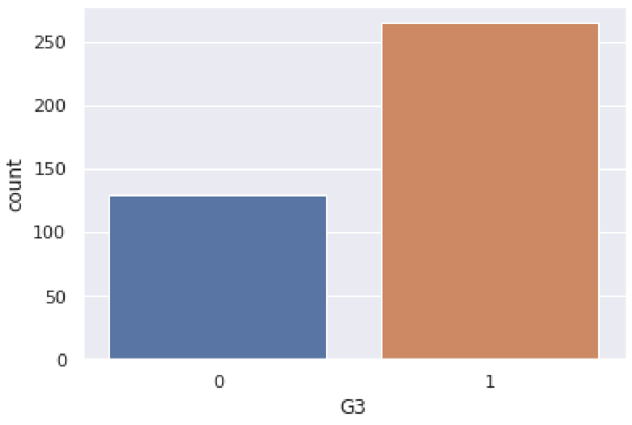
Figure [2] shows there are some low correlated features to the target variable `G3`, in both datasets. In particular, the attributes with a correlation belonging to the interval $[-0.07, 0.07]$ are dropped, so as to keep working only on relevant information. As for the Mathematics data, the attributes `famrel`, `freetime`, `Dalc`, `Walc`, `health`, `absences` are discarded, which leaves us with 27 remaining characterizing features; as for the Portuguese dataset, only `famrel` is loosely correlated to `G3` and therefore 32 attributes are kept.

Attribute	**Description**
school	student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
sex	student's sex (binary: 'F' - female or 'M' - male)
age	student's age (numeric: from 15 to 22)
address	student's home address type (binary: 'U' - urban or 'R' - rural)
famsize	family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
Pstatus	parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
Medu	mother's education (0-none, 1-primary education, 2–5th to 9th grade, 3–secondary education or 4–higher education)
Fedu	father's education (0-none, 1-primary education, 2–5th to 9th grade, 3–secondary education or 4–higher education)
Mjob	mother's job ('teacher', 'health' care related, civil 'services', 'at home' or 'other')
Fjob	father's job ('teacher', 'health' care related, civil 'services', 'at home' or 'other')
reason	reason to choose this school (close to 'home', school 'reputation', 'course' preference or 'other')
guardian	student's guardian (nominal: 'mother', 'father' or 'other')
traveltime	home to school travel time (numeric: 1 - 1 hour)
studytime	weekly study time (numeric: 1 - 10 hours)
failures	number of past class failures (numeric: n if $1 <= n < 3$, else 4)
schoolsup	extra educational support (binary: yes or no)
famsup	family educational support (binary: yes or no)
paid	extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
activities	extra-curricular activities (binary: yes or no)
nursery	attended nursery school (binary: yes or no)
higher	wants to take higher education (binary: yes or no)
internet	Internet access at home (binary: yes or no)
romantic	with a romantic relationship (binary: yes or no)
famrel	quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
freetime	free time after school (numeric: from 1 - very low to 5 - very high)
goout	going out with friends (numeric: from 1 - very low to 5 - very high)
Dalc	workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
Walc	weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
health	current health status (numeric: from 1 - very bad to 5 - very good)
absences	number of school absences (numeric: from 0 to 93)
G1	first period grade (numeric: from 0 to 20)
G2	second period grade (numeric: from 0 to 20)
G3	final grade (numeric: from 0 to 20, output target)

Table 1: Attributes description

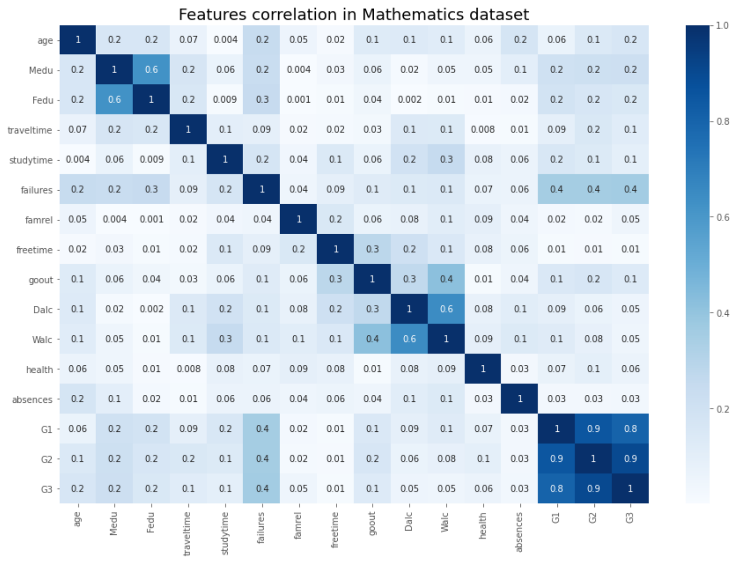


(a) *Portuguese*

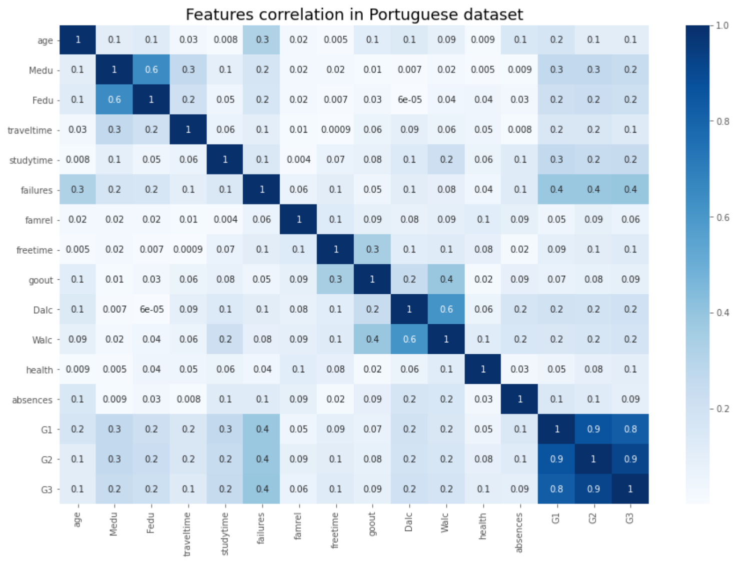


(b) *Mathematics*

Figure 1: Distribution of the target variable G3



(a) *Mathematics*



(b) *Portuguese*

Figure 2: Correlation Matrices

4 Data Preprocessing

4.1 Features encoding

Our data sets present both binary and categorical features. The latters introduce a challenge for many machine learning algorithms, that do not support text values, and therefore need to be translated into numerical ones.

The problem can be tackled with different approaches: *label encoding* converts each value in a column to a number, it is straightforward, but has the disadvantage the algorithms might misinterpret the numerical values (for example, a higher number could be interpreted as more significant information) [3]; in order to avoid such situation, *one-hot encoding* is preferable. In that case, each categorical value is converted into a new column, which is assigned the value 1 (True) or 0 (False). This has the benefit of not weighting a value improperly, but has the downside of adding more columns to the data set.

For the purpose of proceeding with features reduction and classification, our attributes are encoded with the following criteria:

- All binary features are processed with *zero-one encoding*
- For all categorical features, *one-hot encoding* is used. Moreover, unuseful features coming from the one-hot encoding application are dropped.

The result is a data set characterized by 36 features for Mathematics and 41 for Portuguese, all containing numerical values.

4.2 Bootstrap for Feature Scaling

The ***bootstrap method*** is a statistical technique for estimating quantities about a population by averaging estimates from multiple small data samples [4].

In particular, bootstrap addresses the issue of collecting information about a population, when we are provided only with a sample of it. So how can we be sure that sample is representative of the whole population? Namely, does the mean of our sample well approximate the true mean of the population? In our case, is the small sample of Portuguese students representative of the whole population?

So, the bootstrap approach proposes the idea of estimating our statistics many times, by resampling *with replacement* our original sample, instead of doing it only once on the obtained sample realization. It allows us to mimic the process of obtaining new data sets, so as to estimate the variability of our estimate, without generating additional samples. Each of those "bootstrap data sets" is the same size of the original data set and inside each of them some observations may appear more than once, whilst some not at all.

Specifically, bootstrap allows us to obtain **Standard Errors** (SEs) of estimators, without assuming any parametric form of the population in the presence of i.i.d. (independent and identically distributed) sampling.

Given the parameter to estimate θ, its estimator $\hat{\theta}$, the population P_x, $x_1, ..., x_n$ i.i.d. P_x, if we can simulate from P_x, then we can estimate the SE of any $\hat{\theta}$ with the following algorithm:

1. Simulate $x_1^*, ..., x_n^*$ i.i.d. P_x

2. Calculate $\hat{\theta}^*$ on $x_1^*, ..., x_n^*$
3. Iterate over 1. and 2. for a large number of times B
4. Compute $\hat{SE}^*(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(\hat{\theta}_i^* - \bar{\theta^*})}$.

By the Law of Large Numbers, for large values of B, $\bar{\theta^*} \to \theta$ and $\hat{SE}^*(\hat{\theta}) \to SE(\theta)$.

As in our case, most of the time P_x is unknown, so the bootstrap method suggests to use $\hat{P}_x$ instead, which is the empirical distribution of $x_1, ..., x_n$. As a consequence, the bootstrap estimate $\hat{SE}^*(\hat{\theta})$ is based on $\hat{P}_x$ and the simulation of $\hat{P}_x$ is easily accomplished by resampling $x_1, ..., x_n$ with replacement.

Since our data sets represent only a sample of the students coming from *Gabriel Pereira* and *Mousinho da Silveira* Secondary schools, the bootstrap approach was used to obtain estimates of mean and standard deviation values closer to the real ones and their related SEs. Figure [3] shows the difference between the Standard Error of the Mean (SEM) calculated with and without bootstrap.

The estimates of mean and standard deviation are used to **standardize** the data. Standardization is a scaling technique for centering the values around their mean with a unit standard deviation, following the formula 4.1:

$$X' = \frac{X - \mu}{\sigma} \tag{4.1}$$

where in our case X is the train or test set, μ and σ are respectively the estimated mean and standard deviation of the features values. Standardizing the features is important when we compare measurements that have different units. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. Moreover, the resulting scaled data has a Gaussian distribution and meets the requirements of those algorithms that assume the data has such distribution, such as Logistic Regression.

4.3 Principal Component Analysis (PCA)

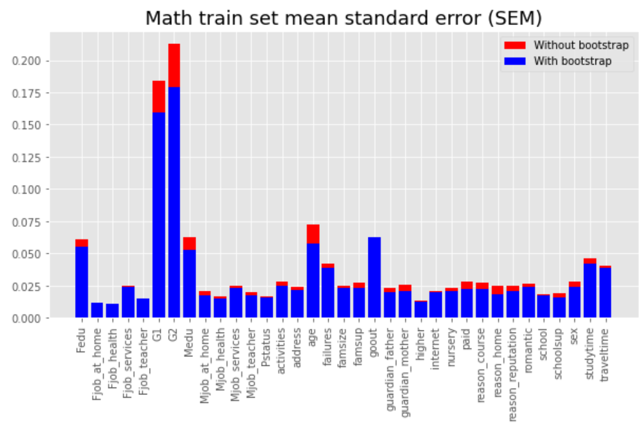
Principal Components Analysis (PCA) is an unsupervised method that aims to find the smallest subspace such that as much information about the original data as possible is preserved.

Given a fixed number of points, PCA has as its goal to find an orthogonal set of linear basis vectors such that the average reconstruction error is minimized. In the found optimal low-dimensional encoding of the data the mean squared distance between the data point and their projection is minimized and the variance of projected data maximized (the higher the value of the variance, the more information is stored).

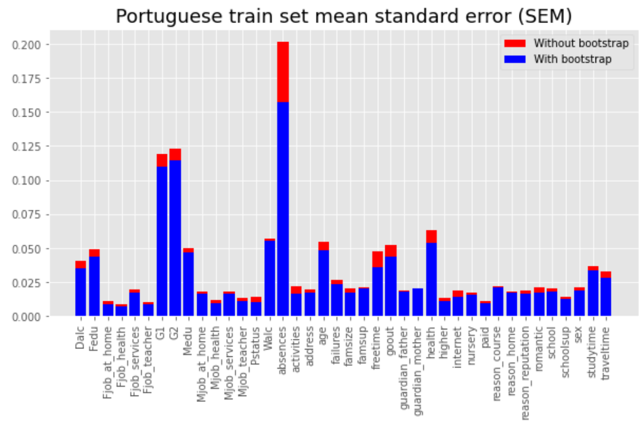
Given data described by D variables, we aim to find a reduced subspace of dimensions $d < D$, such that the most variability of the data is kept. The d variables describing the new space are called Principal Components (PCs) and each one of them is orthogonal to the previous one and points in the direction of the largest variance. We will denote them as $u_1, ..., u_d$.

Specifically, let PC_j be a linear combination of $x_1, ..., x_D$, defined by the weights

$$\mathbf{w}^{(j)} = (w_1^{(j)} ... w_D^{(j)})^T$$



(a) *Mathematics*



(b) *Portuguese*

Figure 3: Standard Error of the Mean on train sets, with and without application of bootstrap

i.e. $u_j = \mathbf{w}^{(j)^T}\mathbf{x}$.

The first PC u_1 is chosen as to have maximum variance:

$$Var(u_1) = Var(\mathbf{w}^{(1)^T}\mathbf{x}) = \mathbf{w}^{(1)^T} S \mathbf{w}^{(1)} \tag{4.2}$$

where S is the sample covariance. $Var(u_1)$ is maximized if the eigenvalue λ_1 is the maximum eigenvalue of S and u_1 is its corresponding eigenvector.

All the PCs are eigenvectors of S and their eigenvalues satisfy the condition

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D$$

so that

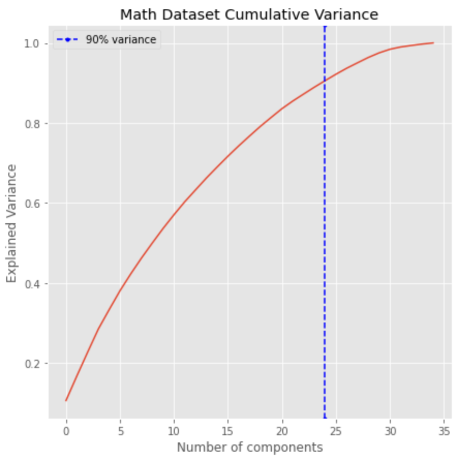
$$Var(u_1) \geq Var(u_2) \geq ... \geq Var(u_D)$$

The total variance of the data is consequently decomposed as:

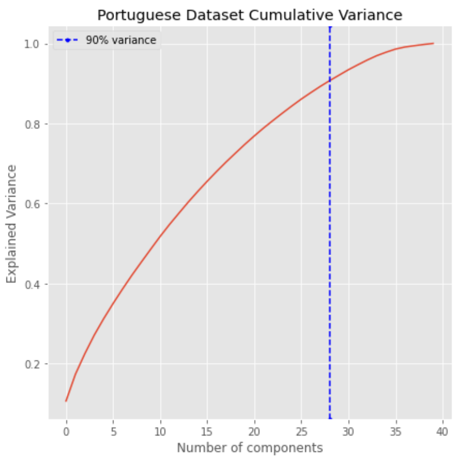
$$\sum_{i=1}^{D} Var(u_i) = \sum_{i=1}^{D} \lambda_i = \sum_{i=1}^{D} Var(x_i)$$

Those conditions allow us to better understand why it is highly recommended to standardize the data, before applying PCA: the principal directions are the ones along which the data shows maximal variance; this means PCA can be misled by directions in which the variance is high merely because of a measurement scale [5].

Here, PCA was applied to obtain the PCs retaining 90% of the cumulative variance. As Figure [4] shows, as for the Mathematics data set, 24 PCs were selected out of 35; in the Portuguese case, 28 out of 40.



(a) *Mathematics*



(b) *Portuguese*

Figure 4: Explained Cumulative Variance of Principal Components

4.4 Oversampling with SMOTE (Synthetic Minority Oversampling TEchnique)

When dealing with imbalanced data sets, two main solutions can be applied:

1. Usage of resampling techniques

2. Change in the evaluation metrics (discussed in Chapter 5).

As for the sampling techniques, they are used to create balanced datasets before fitting a classifier on it. They can be divided into three main classes:

- **undersampling**, consisting in sampling from the majority class in order to keep only a part of those points
- **oversampling**, consisting in replicating some points from the minority class in order to increase its cardinality
- **generating synthetic data**, that is to say, creating new synthetic points from the minority class to increase its cardinality (SMOTE, for example).

It is important to remark that resampling techniques should only be applied on the training set, so as not to distort testing outcomes.

The main disadvantage in using standard oversampling is the major probability of overfitting, due to the fact that the model sees the same data points more than once at training time. A possible solution to that problem is to create synthetic samples starting from the existing ones, instead of simply duplicating them: this method can be seen as a form of data augmentation for the minority class and the most widely used approach is the Synthetic Minority Oversampling Technique, or SMOTE.

SMOTE first selects a minority class instance a at random and finds its k=*5* nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b [6].

Starting from the imbalanced distribution presented in Figure [1], the data sets reach the following balanced configuration:

- *Mathematics* data set: from 104 to 212 *False* labels, reaching the same quantity of the *True* ones
- *Portuguese* data set: from 80 to 439 *False* labels, reaching the same quantity of the *True* ones.

5 Classification

5.1 Metrics

The metrics with which machine learning models are generally evaluated is **accuracy**, which is mathematically defined as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP stands for *True Positive* (True labels predicted as True), TN are the *True Negatives* (False labels predicted as False), FP are the *False Positives* (False labels predicted as True) and FN means *False Negative* (True labels predicted as False). So that implies accuracy represents the fraction of predictions got right by the model.

When using imbalanced datasets, accuracy turns out not to always be a good evaluation metrics: 67.09% of the *Math* dataset and 84.59% of the *Portuguese* one consist of students passing the year, so even if all students were classified as successful and, therefore, all failing students were misclassified, we would still have a total accuracy of 67% in the first case and of 85% in the second one. In order to understand if the model is classifying correctly even the minority class, the solution is to look at different evaluation metrics. In particular, in this notebook, we make use of the following ones:

- **confusion matrix**, a representation of the results divided into TP, TN, FP, FN
- **recall** $= \frac{TP}{TP+FN}$, so the percentage of passing students that were correctly identified by our model
- **precision** $= \frac{TP}{TP+FP}$, which tells us how many predicted samples are relevant i.e. our mistakes into classifying sample as a correct one if it's not true.
- **F1-score** $= 2 \cdot \frac{precision \cdot recall}{precision + recall}$, a balance between precision and recall, which is ideal in the case of imbalanced data.

5.2 K-fold Cross Validation and Hyperparameters Tuning

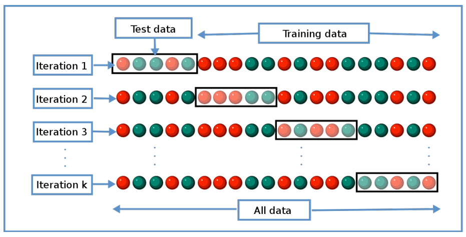


Figure 5: Example of K-Fold Cross Validation [7]

K-fold cross-validation is a common approach to estimate test errors. The idea is to randomly divide the data into K equal sized parts. For $k = 1, 2, ..., K$, the kth subset is left out, while $K-1$ ones are used to fit the model. Then, predictions are obtained for the left out kth part. Finally, results computed on all values of k are combined.

Specifically, let the K parts be $C_1, C_2, ..., C_K$, given n multiple of K, $n_k = \frac{n}{K}$ the observations contained in part k,

$$CV_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k \tag{5.1}$$

where MSE is the Mean Squared Error and is calculated as

$$MSE_k = \sum_{i \in C_k} \frac{(y_i - \hat{y}_i)^2}{n_k} \tag{5.2}$$

and y_i is the real response value and $\hat{y}_i$ is the fit for the observation i, computed on data when the $k-th$ part is removed.

If $K = n$, the method is called *leave-one-out cross-validation* (LOOCV).

In order to find the models parameters performing best with our data, a **Grid Search** is proposed: it performs an exhaustive search over specified parameter values for an estimator and returns the ones achieving the best results at training time according to a certain metrics, which is accuracy in our case. The parameters of the estimator are optimized by a 5-folds cross-validated grid-search over a parameter grid, so the total number of fits performed for each classifier on each set is

$$tot\_fits = \prod_{i=1}^{n\_parameters} n\_candidates_i \cdot 5$$

where $n\_parameters_i$ is the number of parameters to be optimized for the current classifier and $n\_candidates_i$ is equal to the possible candidates for that parameter.

5.3 Classification Algorithms

In machine learning and statistics, classification is a supervised learning problem with the goal of identifying to which set of categories (target label) a new observation belongs, on the basis of a training set of data whose labels are known.

In the following section, the classification task will be performed. Given the train data, our goal is to predict the outcome of the final exams (`G3`).

Different algorithms and their results will be introduced. We have at our disposal two different train sets: one resulting from the PCA transformation and one both reduced and then oversampled with SMOTE. The achieved performance will be compared, so as to find the best one.

5.3.1 Logistic Regression

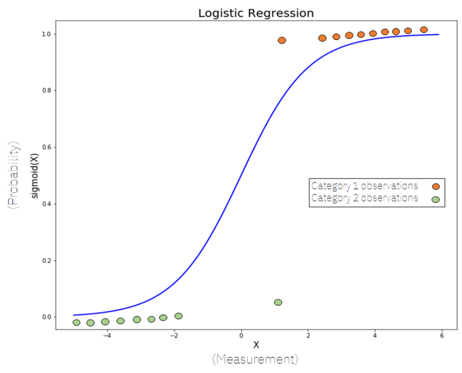


Figure 6: General Logistic Regression [8]

Logistic Regression (LR) is a binary classification model, corresponding to the following definition [5]:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = Ber(y|sigm(\mathbf{w}^T\mathbf{x})) \tag{5.3}$$

where y is the vector of training labels, $\mathbf{w}$ is the vector of regression weights and defines the normal to the decision boundary, $\mathbf{x}$ is the training data, Ber is the Bernoulli distribution (5.4) and $sigm(x)$ is the sigmoid function (5.5).

$$Ber(x|\mu) = \mu^x(1-\mu)^{1-x} = (1-\mu)\, exp(x\, log(\frac{\mu}{1-\mu})),\ x \in \{0,1\} \tag{5.4}$$

$$sigm(x) := \frac{1}{1+exp(-x)} = \frac{e^x}{e^x+1} \tag{5.5}$$

Let's denote $p(y|\mathbf{x}, \mathbf{w})$ as $p(y)$. The formula 5.3 becomes

$$p(y) = \frac{1}{1+e^{-(w_0+w_1x)}} \tag{5.6}$$

The logistic model is used to model the probability of a certain class or event existing. Using the Sigmoid function, each object being detected is assigned a probability between 0 and 1, with a sum of one.

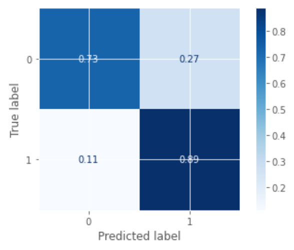
Our goal is to find w_0 and w_1, by maximizing the likelihood:

$$l(w_0, w_1) = \prod_{i;y_i=1} p(y_i) \prod_{i;y_i=0} (1-p(y_i)) \tag{5.7}$$

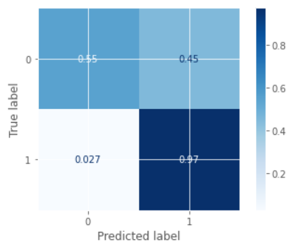
Logistic Regression is one of the most popular approaches to classification for several reasons, including the following ones: LR models are easy to fit to data (simple and fast algorithms), are easy to interpret and to extend to multi-class classification.

The tuned parameter is $C = \frac{1}{\lambda}$, the inverse of *regularization strength*. λ aims to find a trade-off between model simplicity (high λ value), which might lead to underfitting, and too high a complexity (low λ value), which might result in overfitting the data: as a consequence, lowering C implies strengthening the lambda regulator.

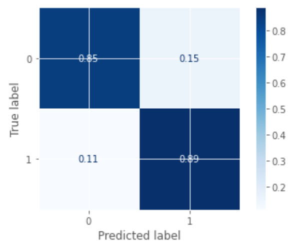
Figure [7] and [8] show the confusion matrices obtained with the Logistic Regression algorithm on all data sets. The resulting accuracies can be found in Table 2 and F1-scores in Table 3. Mostly, the model succeeds in predicting both True Positives and True Negatives.



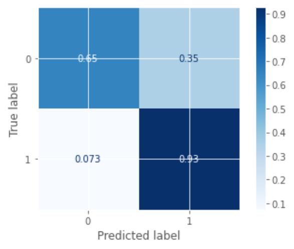
(a) *C=0.1*



(a) *C=10*



(b) *SMOTE oversampling - C=10*



(b) *SMOTE oversampling - C=10*

Figure 7: Confusion Matrices for Logistic Regression on *Math* Dataset

Figure 8: Confusion Matrices for Logistic Regression on *Portuguese* Dataset

5.3.2 K-Nearest Neighbors (KNNs)

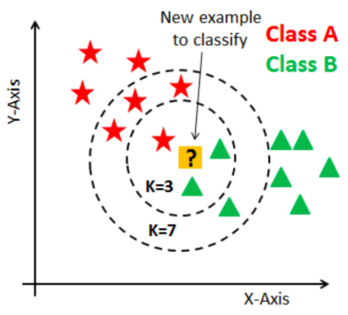


Figure 9: Example of KNN [9]

KNN is a multiclass classifier whose purpose is to find a predefined number K of training samples closest in distance to the new point, and predict the label from these (Figure 9). It looks at the K points in the training set that are nearest to the test input x, basing on distance metric, counts how many members of each class are in this set and returns the empirical fraction:

$$p(y = c|x, D, K) = \frac{1}{K} \sum_{i \in N_K(x,D)} \mathbb{I}(y_i = c) \tag{5.8}$$

where $N_K(x, D)$ are the indices of the K nearest points to x in D and $\mathbb{I}(e)$ is the indicator function, defined as:

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if e is true} \\ 0 & \text{if e is false} \end{cases} \tag{5.9}$$

This method is an example of **memory-based learning** or **instance-based learning**, because it does not have a learning process to construct a general internal model, but simply stores all the training samples.

The strong made assumption is that closest samples belongs to the same class, so they have to have the same labels.

The tuning of K parameter is important because if K is too small, our classifier will be sensible to noise; on the other hand, too high values of K will introduce samples belonging to other class.

Another key factor is the choice of the metric, that influences the computation of the distances between points in the given space (the KNN algorithm is based on the comparison of the distances).

In our case, the exhaustive search (using the function `GridSearchCV` from `sklearn.model_selection`) was done over the following specified parameters:

- **K**: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21;
- **weights**: 'uniform', 'distance';
- **metric**: 'euclidean', 'manhattan'.

where **Euclidean** distance in defined as:

$$dist(p_k, q_k) = \sqrt{}(\sum_{i=1}^{d}((p_k - q_k)^2)) \tag{5.10}$$

and **Manhattan** as:

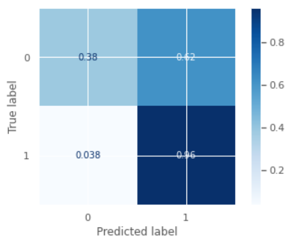
$$dist(p_k, q_k) = \sum_{i=1}^{d}(|p_k - p_k|) \tag{5.11}$$

where d is our dimensionality and p_k and q_k are, respectively, the k^{th} attributes.

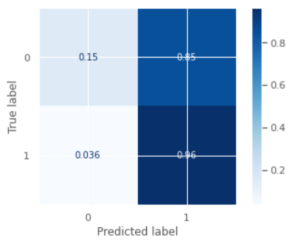
This classifier also uses a weight function for the predictions and its default value is **uniform**, which means all points in each neighborhood are weighted equally. Another weight function was tested too, the so called **distance**, which weights points by the inverse of their distance. In this case, closer neighbors will have a greater influence than neighbors which are further away.

Figure [10] and [11] show the confusion matrices obtained with KNN on all data sets and we can see how the initial model with no oversampling works quite well on classifying the true label, but it has problem with the negative class, probably because we don't have enough data belonging to the negative class to train satisfactorily the model. That problem is partially solved with the application of the SMOTE technique: now, the model can better recognise points belonging to the negative class.

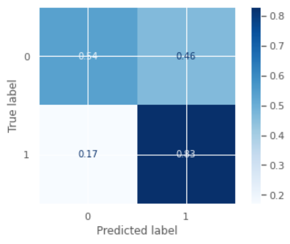
The accuracy values and F1 scores can be found in Table [2] and [3]. As we can understand from the first ones, it seems the SMOTE model works worse, but this is not coherent with what we see in the confusion matrices: this means that the accuracy is not the right metric to evaluate the model. F1 score, instead, shows a more realistic view, in which performances remains more or less the same.



(a) *K = 13, weights = 'uniform', metric = 'manhattan'*

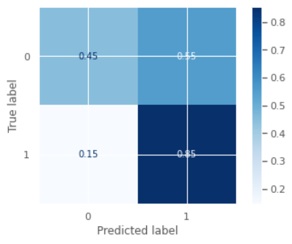


(a) *K = 17, weights = 'distance', metric = 'euclidean'*



(b) *SMOTE oversampling, K = 1, weights = 'uniform', metric = 'euclidean'*

Figure 10: Confusion Matrices for KNN on *Math* Dataset



(b) *SMOTE oversampling K = 1, weights = 'uniform', metric = 'euclidean'*

Figure 11: Confusion Matrices for KNN on *Portuguese* Dataset

Curse of dimensionality The KNN classifier is simple and can work quite well, provided a good distance metric and enough labeled training data. However, the main problem with KNN classifiers is that they do not work well with high dimensional inputs. The poor performance in high dimensional settings is due to the **curse of dimensionality**.

The phrase, attributed to Richard Bellman, was coined to express the difficulty of using brute force (a.k.a. grid search) to optimize a function with too many input variables [5].

This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient: the method is no longer very local. The trouble with looking at neighbors that are so far away is that they may not bee good predictors about the behavior of the input-output function at a given point.

One possible solution to this problem is to reduce the dimensionality of our dataset, as we did using PCA (see chapter 4.3).

5.3.3 Support Vector Machines (SVMs)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The problem SVMs tries to solve is to find the optimum hyperplane that divides the data and maximizes the distance between points belonging to the positive class (1) and the ones belonging to the negative class (-1). The first assumption is the classification problem we want to solve is linearly separable, which means it must exist an hyperplane that divides all data belonging to the two classes.

We define a real-valued function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ as:

$$f(x) = \langle \mathbf{wx} \rangle + b \tag{5.12}$$

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\mathrm{sgn}(f(\mathbf{x}))$.

We introduce two parallel hyperplanes that divide the samples, described by the following formulations:

$$f(x) = \langle \mathbf{wx} \rangle + b = 1 \tag{5.13}$$
$$f(x) = \langle \mathbf{wx} \rangle + b = -1 \tag{5.14}$$

Points belonging to positive class satisfy the relation $\langle \mathbf{wx} \rangle + b >= 1$, while the ones belonging to negative class satisfy $\langle \mathbf{wx} \rangle + b <= -1$.

We define the **margin** as the half distance between those two hyperplanes (dashed lines in Figure[12]):

$$margin = \frac{1}{\|\mathbf{w}\|} \tag{5.15}$$

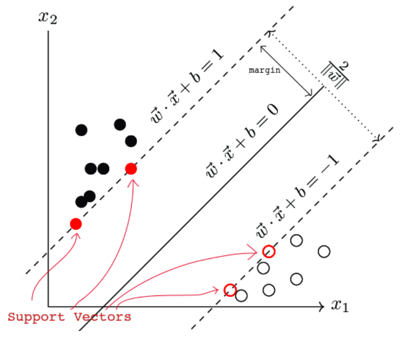

x_2
$\vec{w} \cdot \vec{x} + b = 1$
margin
$\frac{2}{\|\vec{w}\|}$
$\vec{w} \cdot \vec{x} + b = 0$
$\vec{w} \cdot \vec{x} + b = -1$
x_1
Support Vectors

Figure 12: Example of margin in SVM [10]

The points lying on the margin identify the distance we want to calculate: we need to find w and b such that the margin is maximized and at the same time all points are correctly classified. We are interested in the largest margin because it give us maximum robustness relative to uncertainty and independence from correctly classified instances.
Mathematically speaking, that formulation is equivalent to:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i\rangle + b) > 1 \tag{5.16}$$

where y_i is the true label and the prediction is the evaluated distance of the sample from the hyperplane. It means that if the label and the prediction have the same sign, the prediction is correct.

Once the algorithm converged, the model can be described using only the points on the margin, called **support vectors**. That is the reason why SVM can easily scale.

As we said before, being able to solve this problem means that our problem is linearly separable (**hard margin problem**). In real world, unfortunately, the hypothesis of linear separability may not hold and the algorithm may not converge: we need to relax this constraint, introducing the **soft margin problem**.

Soft Margin Problem We add to the previous formulation (5.16) a new therm:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i\rangle + b) > 1 - \xi_i \quad \text{and} \quad \xi_i >= 0 \tag{5.17}$$

where ξ_i, called **slack variable**, is the distance of x_i from the corresponding class margin, if x_i is on the wrong side of the margin, and 0 otherwise. C is a constant used to tune the level of misclassification that we are willing to accept.

Kernel Trick In some cases, problems are not well separable on the original feature spaces and a non-linear decision boundary is needed. A possible solution is to map our features in a higher dimensional space, usually a Hilbert space, in which, hopefully, a linear separation becomes possible.

A Hilbert space H is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product [11].

In the new space, the training samples become $((\phi(x_1), y_1), ..., (\phi(x_n), y_n))$, where $\phi : \chi \to H$ is a feature map, with $\chi \subset \mathbb{R}^d$ being the input space and H the new inner product space. Unfortunately, following this approach, the inner product is very hard to compute. To reduce the computational effort, we can use a symmetric function $K : \chi \times \chi \to \mathbb{R}$, implementing the inner product in the features space, which has lower computational cost. K is called *kernel function* and is such that:

$$K(x_i, y_i) = (\phi(x_i), \phi(y_i)) \tag{5.18}$$

In order to apply this mapping, the kernel function must satisfy **Mercer's theorem**, stating that: a symmetric function $K : \chi \times \chi \to \mathbb{R}$ implements an inner product in some Hilbert space, if and only if, for all $x_1, x_2...x_m$, the Gram matrix $G_{ij} = K(x_i, x_j)$ is positive semidefinite. A matrix is semidefinite if and only if $x^T G x \geq 0 \quad \forall(x) \in \mathbb{R}^n$.

Here, two different kernel functions were used: **Linear** kernel and **Radial Basis Function**.

Linear SVM The classic linear kernel assumes the different classes can be separated by a straight line.

Given two data points x and x', the linear kernel function k is defined as:

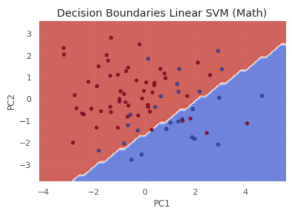
$$k(x, x') = < x, x' > \tag{5.19}$$

Given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new samples.

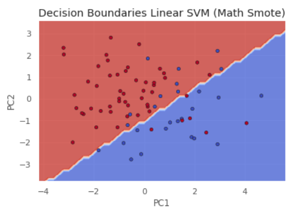
The parameter C was tuned to regularize the amount of misclassified training points allowed in the model. For large values of C, a smaller-margin hyperplane will be chosen, if that hyperplane gets all the training points classified correctly. Too high values of C might cause overfitting. Conversely, for a very small value of C, the optimizer will look for a larger-margin separating hyperplane, therefore for a simpler decision function, even if that choice brings to heavy misclassification. In this case, the chosen range is:

- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$.

Those values are used to train a linear SVM on the training set. The obtained decision boundaries are shown in Figure[13] and [14]. In all the cases, according to the accuracy values showed in Table [2], we can say the model is able to classify very well the true labels (red dots in Figure), while it demonstrates some uncertainty with the negative class (blue dots), due to the imbalance of the dataset. When SMOTE is applied, performances get worse: apparently, the model didn't need an oversampling of the training data and it was already able to well classify data. In this case, the oversampling brings the model to have an overfitting behaviour.

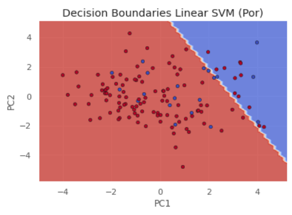


(a) *Non-oversampled dataset C = 0.01*

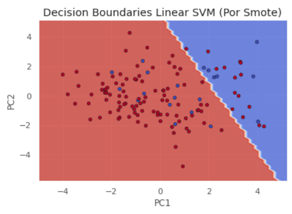


(b) *SMOTE oversampling C =100*

Figure 13: Decision Boundaries on test set (*Math* Dataset)



(a) *Non-oversampled dataset C =10*



(b) *SMOTE oversampling C = 1*

Figure 14: Decision Boundaries on test set (*Portuguese* Dataset)

RBF Kernel RBF kernel is used when the boundaries are hypothesized to be curve-shaped.

Given two data points x and x', the Radial Basis Function (RBF) kernel SVM k is defined as:

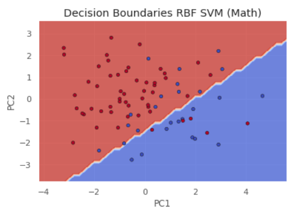
$$k(x, x') = \exp\left(-\gamma \|x - x'\|^2\right) \tag{5.20}$$

where γ must be greater than 0 and represents the inverse of the radius of the area of influence of samples selected by the model as support vectors, and $\|x - x'\|^2$ is the squared Euclidean distance between two data points x and x'.

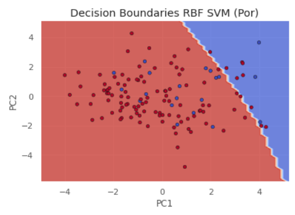
In this case, an exhaustive search (using the function `GridSearchCV` from `sklearn.model_selection`) over the following specified parameters was performed:

- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$;
- $\gamma = [10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 0.5, 1, 10, 10^2]$.

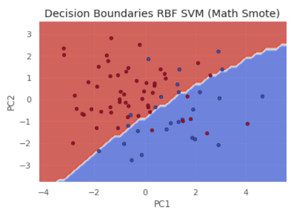
Figure[15] and [16] shows the decision boundaries on the test set and we can immediately spot the different shape with respect to the linear case: with RBF kernel, the decision regions tend to cover the spread of the data. The RBF kernel seems to work well on the *Math* dataset, according to accuracy and F1 scores shown in Tables 2 and 3, but the performance drops when it comes to the *Portuguese* set: especially when the oversampled training set is used, the model is not able to correctly classify the negative class (F1-score is 0.08).



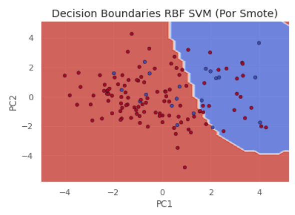
(a) Non-oversampled dataset
C = 1000, $\gamma = 10^{-5}$



(a) Non-oversampled dataset
C = 100, $\gamma = 10^{-3}$



(b) *SMOTE oversampling*
C = 1000, $\gamma = 10^{-3}$



(b) *SMOTE oversampling*
C = 10, $\gamma = 10^{-1}$

Figure 15: Decision Boundaries on test set (*Math* Dataset)

Figure 16: Decision Boundaries on test set (*Portuguese* Dataset)

5.3.4 Decision Trees

When using Decision Trees, the goal is to create a model predicting the value of a target variable by learning simple decision rules inferred from the data features. Their main advantage is interpretability, but they might easily lead to overfitting because of a poor generalization.

Decision Trees can be applied to both regression and classification problems. The steps to build a general DT are the following ones:

1. The set of possible input values $X_1, ..., X_p$, called the *predictor space*, is divided into J distinct and non-overlapping regions $R_1, ..., R_j$

2. For every observation that falls into R_j, the prediction is the mean of the response values for the training observations in R_j

3. The predictor space is divided into high-dimensional rectangles, called *boxes*. The goal is to find the boxes $R_1, ..., R_j$ that minimize a certain value: the *Residual Sum of Squares* (RSS) for regression problems (5.21); the *Classification Error Rate* (E) in the other case (5.22).

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{5.21}$$

where $\hat{y}_{R_j}$ the mean response for the training observations within the *jth* box,

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{5.22}$$

where $\hat{p}_{mk}$ is the portion of training observations in the *mth* region belonging to the *kth* class.

Since it is computationally infeasible to consider all possible partitions of the feature spaces into J boxes, the used approach is top-down and greedy, also known as *recursive binary splitting*: starting from the root of the tree, two or more branches are created at each split on the prediction space; at each step of the tree-building process, the best split is created, according to the chosen metrics, without taking into consideration suboptimal partitions that may lead to better results in the future.

The classification tree aims to predict the most commonly occurring class of training observations the current observation belongs to, inside its region. They are built using recursive binary splitting, based on the *classification error rate* E (5.22), the fraction of the training observations in that region that do not belong to the most common class. However, E is often not sensitive enough for tree-growing, so two other measures are preferred:

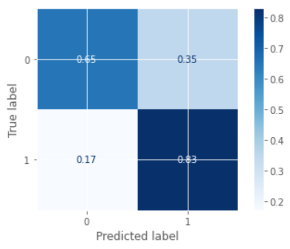
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.23}$$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk}) \tag{5.24}$$

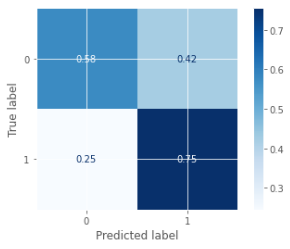
where G (5.23) is the *Gini index* and is a measure of the total variance across the K classes (often referred to as a measure of node purity), and D (5.24) is the *cross-entropy*.

In the proposed work, the tuned parameters are: *criterion*, which states if *Gini index* or *cross-entropy* is used to classify objects as belonging to a particular class; the maximum depth of the tree; the minimum number of samples required to split a node and per each leaf.

According to the results are shown in Figures 17 and 18, the model works well on classifying the `True` label. When it comes to the `False` target, it struggles with the *Portuguese* dataset, the most imbalanced one: performances improve a little bit when SMOTE is applied, but most of the predictions are False Positives. As for the *Math* set, 65% of the predictions are True Negatives. An example of Decision Tree can be found in Figure 19.

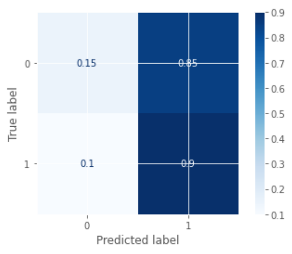


(a) *Cross-entropy*

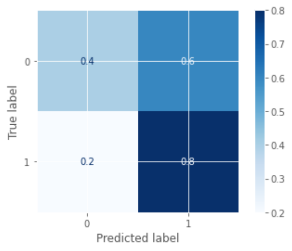


(b) *SMOTE oversampling - Cross-entropy*

Figure 17: Confusion Matrices for Decision Trees on *Math* Dataset



(a) *Gini index*



(b) *SMOTE oversampling - Cross-entropy*

Figure 18: Confusion Matrices for Decision Trees on *Portuguese* Dataset

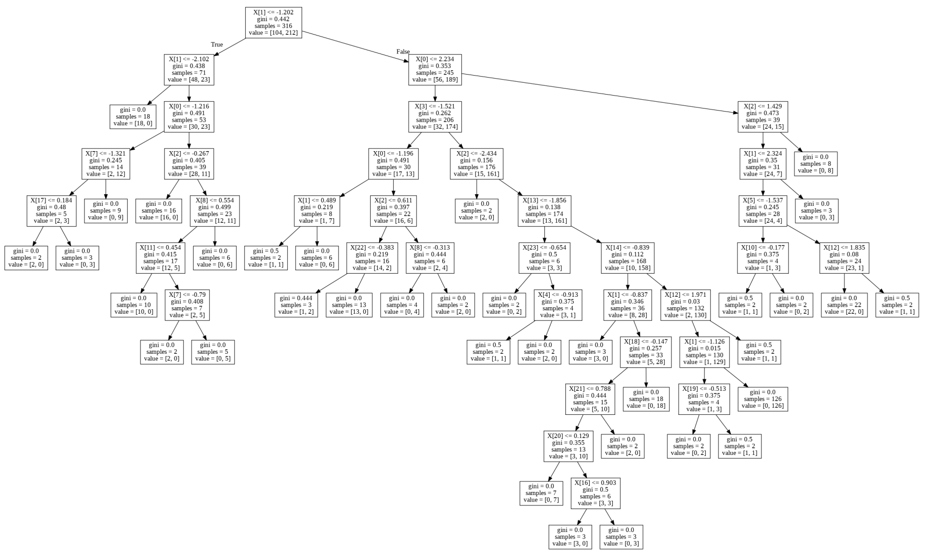

X[1] <= -1.202 gini = 0.442 samples = 316 value = [104, 212]
True
False
X[1] <= -2.102 gini = 0.438 samples = 71 value = [48, 23]
X[0] <= 2.234 gini = 0.353 samples = 245 value = [56, 189]
gini = 0.0 samples = 18 value = [18, 0]
X[0] <= -1.216 gini = 0.491 samples = 53 value = [30, 23]
X[3] <= -1.521 gini = 0.262 samples = 206 value = [32, 174]
X[2] <= 1.429 gini = 0.473 samples = 39 value = [24, 15]
X[7] <= -1.321 gini = 0.245 samples = 14 value = [2, 12]
X[2] <= -0.267 gini = 0.405 samples = 39 value = [28, 11]
X[0] <= -1.196 gini = 0.491 samples = 30 value = [17, 13]
X[2] <= -2.434 gini = 0.156 samples = 176 value = [15, 161]
X[1] <= 2.324 gini = 0.35 samples = 31 value = [24, 7]
gini = 0.0 samples = 8 value = [0, 8]
X[17] <= 0.184 gini = 0.48 samples = 5 value = [2, 3]
gini = 0.0 samples = 9 value = [0, 9]
gini = 0.0 samples = 16 value = [16, 0]
X[8] <= 0.554 gini = 0.499 samples = 23 value = [12, 11]
X[1] <= 0.489 gini = 0.219 samples = 8 value = [1, 7]
X[2] <= 0.611 gini = 0.397 samples = 22 value = [16, 6]
gini = 0.0 samples = 2 value = [2, 0]
X[13] <= -1.856 gini = 0.138 samples = 174 value = [13, 161]
X[5] <= -1.537 gini = 0.245 samples = 28 value = [24, 4]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 3 value = [0, 3]
X[11] <= 0.454 gini = 0.415 samples = 17 value = [12, 5]
gini = 0.0 samples = 6 value = [0, 6]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 6 value = [0, 6]
X[22] <= -0.383 gini = 0.219 samples = 16 value = [14, 2]
X[8] <= -0.313 gini = 0.444 samples = 6 value = [2, 4]
X[23] <= -0.654 gini = 0.5 samples = 6 value = [3, 3]
X[14] <= -0.839 gini = 0.112 samples = 168 value = [10, 158]
X[10] <= -0.177 gini = 0.375 samples = 4 value = [1, 3]
X[12] <= 1.835 gini = 0.08 samples = 24 value = [23, 1]
gini = 0.0 samples = 10 value = [10, 0]
X[7] <= -0.79 gini = 0.408 samples = 7 value = [2, 5]
gini = 0.444 samples = 3 value = [1, 2]
gini = 0.0 samples = 13 value = [13, 0]
gini = 0.0 samples = 4 value = [0, 4]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 2 value = [0, 2]
X[4] <= -0.913 gini = 0.375 samples = 4 value = [3, 1]
X[1] <= -0.837 gini = 0.346 samples = 36 value = [8, 28]
X[12] <= 1.971 gini = 0.03 samples = 132 value = [2, 130]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.0 samples = 22 value = [22, 0]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 5 value = [0, 5]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 3 value = [3, 0]
X[18] <= -0.147 gini = 0.257 samples = 33 value = [5, 28]
X[1] <= -1.126 gini = 0.015 samples = 130 value = [1, 129]
gini = 0.5 samples = 2 value = [1, 1]
X[21] <= 0.788 gini = 0.444 samples = 15 value = [5, 10]
gini = 0.0 samples = 18 value = [0, 18]
X[19] <= -0.513 gini = 0.375 samples = 4 value = [1, 3]
gini = 0.0 samples = 126 value = [0, 126]
X[20] <= 0.129 gini = 0.355 samples = 13 value = [3, 10]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 7 value = [0, 7]
X[16] <= 0.903 gini = 0.5 samples = 6 value = [3, 3]
gini = 0.0 samples = 3 value = [3, 0]
gini = 0.0 samples = 3 value = [0, 3]

Figure 19: Resulting Decision Tree for *Math* dataset

5.3.5 Random Forest

A Random Forest Classifier can be thought of as an ensemble of different Decision Trees, fit on various subsamples of the dataset, whose decisions are averaged in order to better generalise the model and therefore prevent overfitting.

Random Forests provide an improvement over *bagged trees*. In particular, *bagging* - or *bootstrap aggregation* - is a procedure for reducing the variance of a statistical learning method.

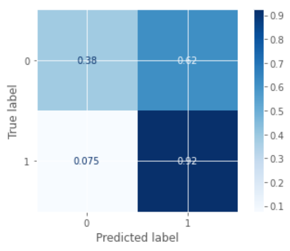
Given a set of independent observations $Z_1, ..., Z_n$, having variance σ^2, being $\bar{Z}$ the mean of those observations, the mean of $\bar{Z}$ is σ^2/n, which implies that averaging a set of observations reduces the variance. We usually don't have access to multiple trees, so bootstrap is our solution: having generated B different bootstrapped training data sets, the method is trained on the bth set and the prediction $\hat{f}^{*b}(x)$ at the point x is obtained; then, all predictions are averaged to compute the so called *bagging*, as follows:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{5.25}$$

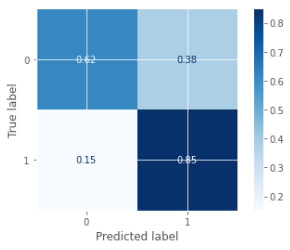
In the case of classification trees, the class predicted by each of the B trees is recorded and the overall prediction is the *majority vote*, the most commonly occurring class.

Random Forests improve bagged trees by decorrelating the trees, which results in a reduction of the variance. A number of decision trees is built on bootstrapped training samples and, each time a split has to be created, a random selection of m out of p predictors is chosen as split candidates. A fresh selection of m candidates is taken at each split. Typically, $m \approx \sqrt{p}$.

The tuned parameters are the number of predictors p and the criterion for classifying objects. According to the confusion matrices (Figure 20 and 21), the model achieves very high results when predicting the `True` label, which is the majority class, but the performance drops when the target is the minority class, especially when the training set is not oversampled. As for the accuracy values (Table 2), RF mostly outperforms decision trees. In terms of F1-score on the `False` label (Table 3), Random Forests do not work well on the most imbalanced dataset, which is the *Portuguese* one.

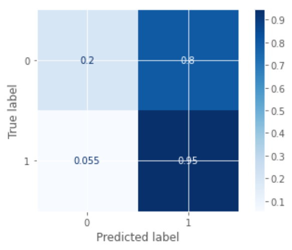


(a) *601 estimators, Cross-entropy*

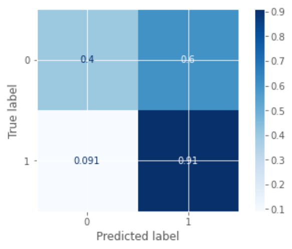


(b) *SMOTE oversampling - 901 estimators, Cross-entropy*

Figure 20: Confusion Matrices for Random Forests on *Math* Dataset



(a) *601 estimators, Gini index*



(b) *SMOTE oversampling - 301 estimators, Gini index*

Figure 21: Confusion Matrices for Random Forests on *Portuguese* Dataset

5.4 Results

Table 2 and 3 show the achieved results by all classifiers, respectively in terms of accuracy and F1-score on the `False` label. In the former case, Logistic Regression and SVM perform best on both *Math* and *Portuguese* data sets: the accuracy values range from 0.85 to 0.91. As for the latter, instead, we keep having good results on the *Math* dataset, where SMOTE oversampling always makes achieving better scores, whilst on the most imbalanced dataset, the *Portuguese* one, performances drop and SMOTE oversampling only helps in about half of the cases.

Comparing the results in Table 2 with the original case study [1], we can see how they match when SVM is used, but ours are lower when it comes to DTs and RFs, probably due to a different choice of hyperparameters.

	Math	*Math SMOTE*	*Portuguese*	*Portuguese SMOTE*
Logistic Regression	0.84	0.87	0.91	0.88
KNN	0.77	0.71	0.85	0.78
Linear SVM	0.85	0.84	0.89	0.88
RBF SVM	0.82	0.84	0.91	0.83
Decision Tree	0.80	0.70	0.78	0.78
Random Forest	0.77	0.80	0.82	0.83

Table 2: Classification Results in terms of accuracy

	Math	*Math SMOTE*	*Portuguese*	*Portuguese SMOTE*
Logistic Regression	0.77	0.80	0.69	0.60
KNN	0.54	0.56	0.41	0.39
Linear SVM	0.75	0.79	0.61	0.62
RBF SVM	0.71	0.78	0.65	0.08
Decision Tree	0.55	0.58	0.39	0.30
Random Forest	0.62	0.67	0.27	0.49

Table 3: Classification Results in terms of F1-score on the *False* label

6 Conclusions

The current thesis introduced an analysis performed on the UCI Machine Learning Student Performance Data Set [2].

In particular, starting from two different sample sets - one focused on results obtained by students in the *Mathematics* course and the other one in *Portuguese* - binary classification was performed, having students' final performance as target variable (*pass* or *fail* the final exam).

Before applying the classification algorithms, the data was preprocessed in different steps:

1. **Features encoding** to translate textual values into numerical ones and categorical attributes into boolean ones. Zero-one and one-hot encodings were used.

2. **Feature scaling**: data was standardized, according to the estimates of mean and standard deviation computed with the bootstrap method.

3. **Dimensionality Reduction** with **PCA** to decrease the number of features describing the dataset, retaining 90% of its cumulative variance.

4. **Oversampling with SMOTE** to balance the data sets, which were imbalanced on the negative class.

In order to find the best hyperparameters for each classification algorithm, a grid search with 5-fold cross validation was applied.

Models were trained on both oversampled and non-oversampled training sets. Results were evaluated in terms of accuracy and F1-score and then compared to the state-of-the-art [1].

The proposed methods were Logistic Regression, K-Nearest Neighbors, Support Vector Machines, Decision Trees and Random Forests. The most performing ones were Logistic Regression and SVM.

References

[1] Paulo Cortez and Alice Silva. Using data mining to predict secondary school student performance. *EUROSIS*, 01 2008.

[2] UCI Machine Learning. Student performance dataset. `http://archive.ics.uci.edu/ml/datasets/Student+Performance`.

[3] Practical Business Python. Guide to encoding categorical values in python. `https://pbpython.com/categorical-encoding.html"`, February 2017.

[4] Machine Learning Mastery. A gentle introduction to the bootstrap method. `https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/`, May 2018.

[5] Kevin P. Murphy. *Machine Learning A Probabilistic Perspective.* 2012.

[6] Yunqian Ma Haibo He. *Imbalanced Learning: Foundations, Algorithms, and Applications.* 2013.

[7] Wikipedia. Cross-validation (statistics). `https://en.wikipedia.org/wiki/Cross-validation_(statistics)`.

[8] Towards Data Science. Logistic regression explained. `https://towardsdatascience.com/logistic-regression-explained-9ee73cede081`.

[9] Medium. K nearest neighbour for classification on breast cancer data , results with preprocessing and w/o normalising. `https://medium.com/@madanflies/k-nearest-neighbour-for-classification-on-breast-cancer-data-results-with-preprocessing-and-w-o-e21b0cc98a2f`, May 2019.

[10] Research Gate. An example of svm classification. `https://www.researchgate.net/figure/An-example-of-SVM-classification-An-example-of-SVM-classification_fig1_336085357`, September 2019.

[11] Wikipedia. Hilbert space. `https://en.wikipedia.org/wiki/Hilbert_space#Definition`.