Politecnico di Torino

Facoltà di Ingegneria

DATA SPACES

REPORT

Machine learning classification algorithms for student performance prediction

Professor:
Francesco Vaccarino

Students:
Giuseppe Leoni
265170 IngMat
Luigi Pirisi
254989 IngInf

ACADEMIC YEAR 2020-2021

Contents

Chapter 1

Introduction

The dataset chosen for this analysis is *Student Performance Data Set*. It is splitted in two files each one relative to a specific subject: portuguese and mathematics. Each file is composed by 32 features. The target variable is *G3* and consist of the final grade obtained by the student. Actually, *G3* is strongly correlated with other two features *G1* and *G2* that represent the grades obtained at the end of the first and the second period respectively. The analysis was performed as a binary classification between students who pass or fail the course (Pass if *G3* is greater or equal than 10).
We tried to give an overall view of different classifiers, comparison metrics and oversampling techniques.

1.1 Features and target

The two datasets contain 32 features with information about students' performances, demographic and social information taken from school reports and questionnaires.
In particular the attributes are:

1. **school** - student's school, binary:
    - *GP* - Gabriel Pereira
    - *MS* - Mousinho da Silveira
2. **sex** - student's sex, binary:
    - *F* - female
    - *M* - male
3. **age** - student's age, numeric from *15* to *22*
4. **address** - student's home address type, binary:
    - *U* - urban
    - *R* - rural
5. **famsize** - family size, binary:
    - *LE3* - less or equal to 3
    - *GT3* - greater than 3
6. **Pstatus** - parent's cohabitation status, binary:

- *T* - living together
- *A* - apart

7. **Medu** - mother's education, numerical:
    - *0* - none
    - *1* - primary education (4th grade)
    - *2* – 5th to 9th grade
    - *3* – secondary education
    - *4* – higher education
8. **Fedu** - father's education, numerical, same values as *Medu*.
9. **Mjob** - mother's job, categorical:
    - *teacher*
    - *health* - care related
    - civil *services* - e.g. administrative or police
    - *at_home*
    - *other*
10. **Fjob** - father's job, categorical, same values as *Mjob*.
11. **reason** - reason to choose this school, categorical:
    - close to *home*
    - school *reputation*
    - *course* preference
    - *other*
12. **guardian** - student's guardian, categorical:
    - *mother*
    - *father*
    - *other*
13. **traveltime** - home to school travel time, numerical:
    - *1* - less than 15 min.
    - *2* - from 15 to 30 min.
    - *3* - from 30 min. to 1 hour
    - *4* - greater than 1 hour
14. **studytime** - weekly study time, numerical:
    - *1* - less than 2 hours
    - *2* - from 2 to 5 hours
    - *3* - from 5 to 10 hours
    - *4* - greater than 10 hours

15. **failures** - number of past class failures, numerical: n if $1 \leq n\_failures \leq 3$, else *4*.
16. **schoolsup** - extra educational support, binary: *yes* or *no*).
17. **famsup** - family educational support, binary: *yes* or *no*.
18. **paid** - extra paid classes within the course subject (Math or Portuguese), binary: *yes* or *no*.
19. **activities** - extra-curricular activities, binary: *yes* or *no*.
20. **nursery** - attended nursery school, binary: *yes* or *no*.
21. **higher** - wants to take higher education, binary: *yes* or *no*.
22. **internet** - Internet access at home, binary: *yes* or *no*.
23. **romantic** - with a romantic relationship, binary: *yes* or *no*.
24. **famrel** - quality of family relationships, numerical: from *1* - very bad to *5* - excellent.
25. **freetime** - free time after school, numerical: from *1* - very low to *5* - very high.
26. **goout** - going out with friends, numerical: from *1* - very low to *5* - very high.
27. **Dalc** - workday alcohol consumption, numerical: from *1* - very low to *5* - very high.
28. **Walc** - weekend alcohol consumption, numerical: from *1* - very low to *5* - very high.
29. **health** - current health status , numerical: from *1* - very bad to *5* - very good.
30. **absences** - number of school absences, numerical: from *0* to *93*.
31. **G1** - first period grade, numerical: from *0* to *20*.
32. **G2** - second period grade, numerical: from *0* to *20*.
33. **G3** - final grade, numerical: from *0* to *20*. This is the target variable.

The Portoguese dataset has informations about 649 students, while the Mathematics contains 395 rows. With a preliminary analysis we discovered that the two datasets contain the same students differing by few attributes such as grades (G1, G2, G3), number of absences, studytime and so on. For this reason we decided to analyze separately the two datasets.
In this relation we will report our results for Portoguese dataset taking the first 30 features as predictors, leaving out G1 and G2 and using G3 as output target. The output target is numeric so we decided to transform it in order to perform classification task on the dataset. We will use the following transformation:

- Pass: if G3 $\geq$ 10;
- Fail: if G3 $<$ 10.

Chapter 2

Dataset exploration

Firstly we proceed by analyzing the distribution of the outcomes.
The total number of instances are 649. We discovered that 85% of students passed while 15% of the students failed.
In order to find a sort of correlation between features and the target we chose to plot the categorical features and to build a correlation matrix with the numerical ones.

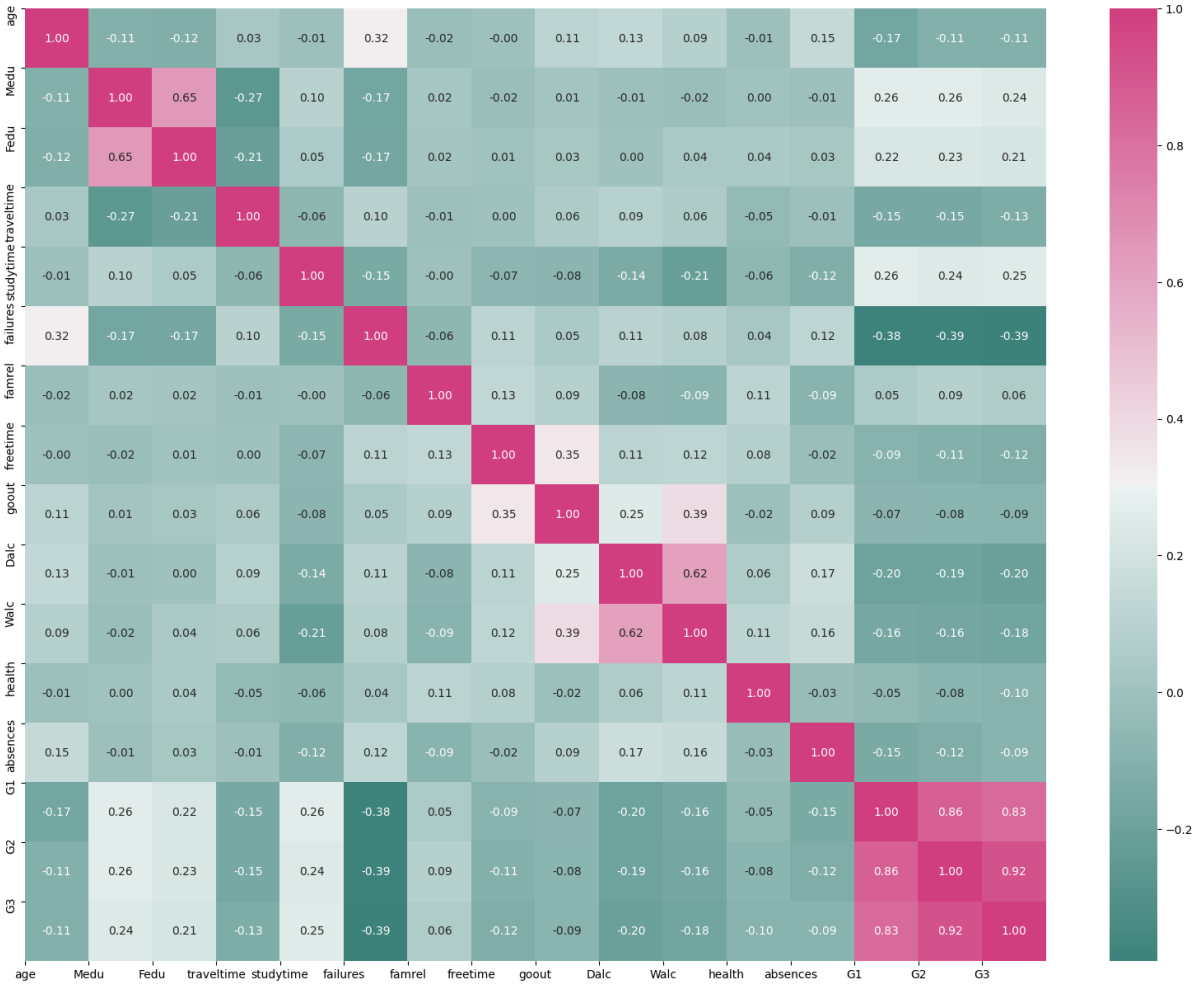


Figure 2.1: Correlation matrix of the numerical features

Figure 2.1 shows that the target feature *G3* (the final grade) is highly correlated with *G1* and *G2* (mid-term and last-term grade respectively). Such a strong correlation was predictable. Since our goal is to build a model not based on obvious information, we will left out *G1* and *G2* and, at most, we will use those features just for reference or for comparisons. Other interesting correlations with our target feature are:

- *failures*, −0.39. It is more likely to fail if you collected a certain number of failures in the past.
- *studytime*, 0.25.
- *Medu*, 0.24. Mother education level.
- *Fedu*, 0.21. Father education level.
- *Dalc*, −0.20. Workday alchool consumption.
- *Walc*, −0.18. Weekend alchool consumption.

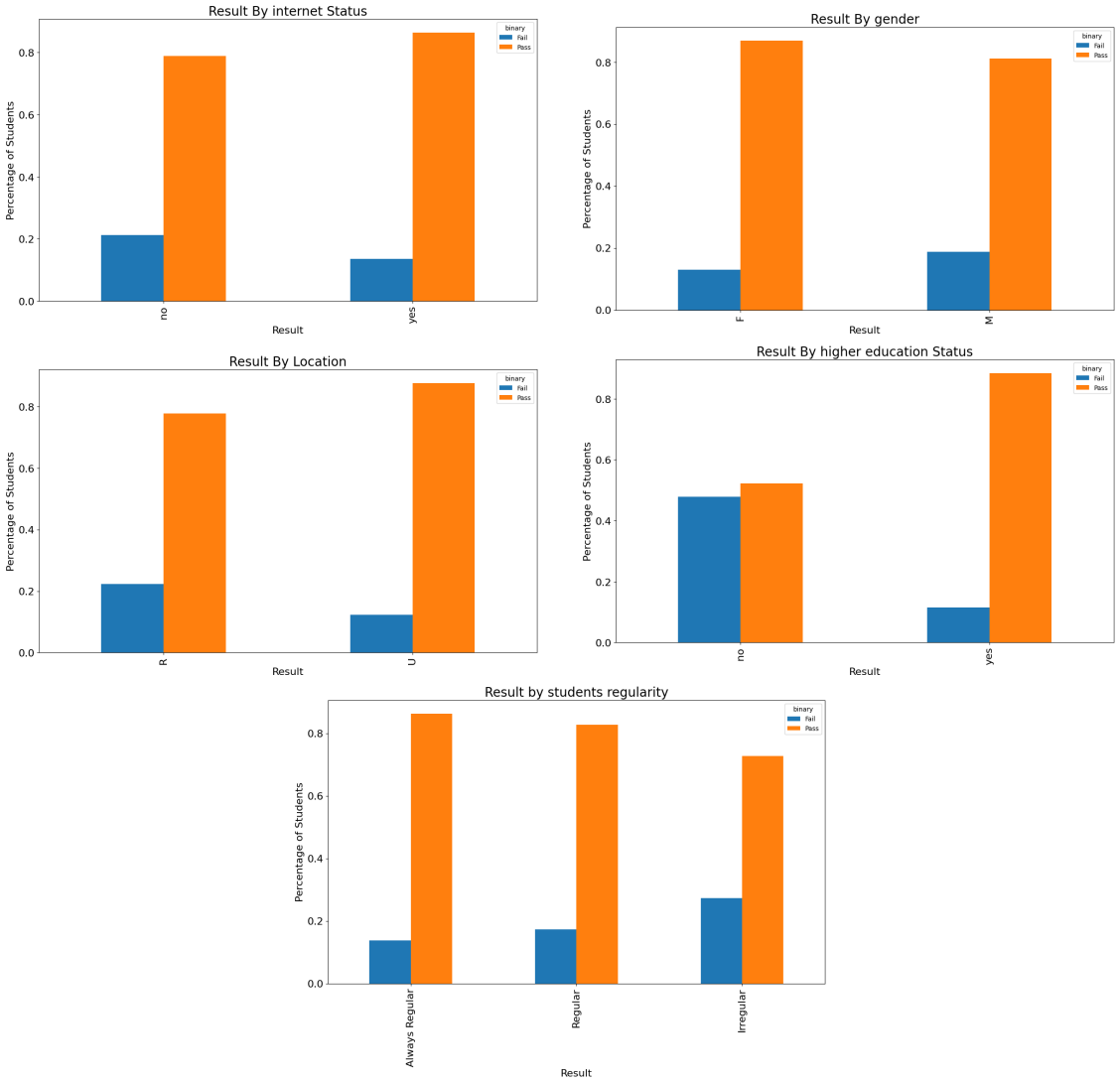


Figure 2.2: These histogram shows the most correlated categorical features against the target variable

Chapter 3

Data transformations

The features are mostly binary or categorical. It was then necessary to apply transformations on these features before using any classifier.
As we discussed in the previous chapter, the target variable is highly unbalanced and generating new samples with data augmentation algorithms as *SMOTE* can increase performances of the classification algorithms.
We also tried to apply *Principal Component Analysis* (PCA) on our data, to summarize our prediction variables in a lower dimensional space and evaluate how this affects the performances of the classifiers in terms of time and accuracy.

3.1 Preprocessing

The chosen classifiers accept numerical features only. So we had to make some adjustments.
In particular we did three main transformations:

- creation of the binary target variable.
- transformation of the binomial features to binary features.
- transformation of the multinomial features to multiple binary features.

Regarding the first one, we considered the attribute *G3* and we added a new feature called *binary* with value *pass* if $G3 >= 10$ and *fail* otherwise, according to the grading system.
The dataset contains 14 attributes (including the newly created *binary*) with binary value:

1. *school*
2. *sex*
3. *address*
4. *famsize*
5. *Pstatus*
6. *schoolsup*
7. *famsup*
8. *paid*

9. *activities*
10. *nursery*
11. *higher*
12. *internet*
13. *romantic*
14. *binary*

Using a dictionary and the function *pandas.DataFrame.replace* we replaced these with $\{0, 1\}$ valued variables with the same name.
We finally decided to create dummy variables to represent the multinomial attributes:

1. *Mjob*
2. *Fjob*
3. *reason*
4. *guardian*

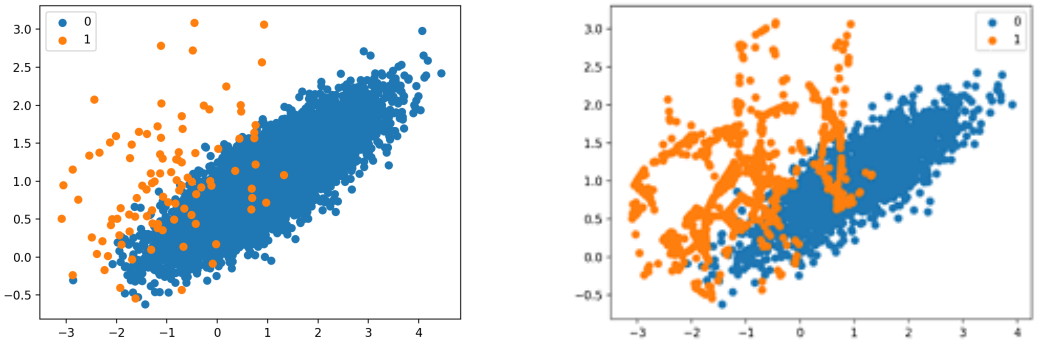
A dummy variable is one that takes 0 or 1 value if a certain condition is satisfied. If we consider the attribute *guardian*, using the function *pandas.get_dummies*, three new binary variables are created, corresponding to the values of the attribute: *guardian_father*, *guardian_mother* and *guardian_other*. In this way, the information is preserved but the dimensionality of the dataset increases.

3.2 SMOTE

SMOTE (*Synthetic Minority Over-sampling Technique*) is a data augmentation technique used to over-sample an unbalanced dataset by creating new records of the minority class.
An unbalanced dataset could be very problematic with classifiers. In fact, considering a dataset containing 90% of points belonging to a class, a greedy algorithm that classifies all points to the latter class will have a high accuracy but effectively poor performances. An over-sampling algorithm can help classificators overcome the issue and identify correctly minority class points. The algorithm first select a random sample of the minority class x_0 and identifies his neighbours $x_{n1}, x_{n2}, ..., x_k$. The number of neighbours is typically $k = 5$. The algorithm then choose randomly one of the points in the neighborhood x_{neigh} and generate the synthetic sample x_{smote} by a convex combination of the two chosen points.

$$x_{smote} = w * x_0 + (1 - w) * x_{neigh} \qquad w \in (0, 1) \tag{3.1}$$

In this way, the algorithm generates new sample points belonging to the minority class that are very close to the original points and share similar properties with them.



(a) Before applying SMOTE

(b) After applying SMOTE

Figure 3.1: SMOTE technique on an unbalanced dataset

3.3 PCA

Principal component Analysis is an unsupervised learning algorithm that projects data into a lower dimensional space preserving as much variance of the original points as possible.
The first principal component of a set of predictors $X_1, X_2, ..., X_p$ is the linear combination:

$$\begin{aligned} Z_1 &= p_{11}X_1 + p_{12}X_2 + ... + p_{1p}X_p \\ \text{s.t.} \quad & \sum_{j=1}^{p} p_{j1}^2 = 1 \end{aligned} \tag{3.2}$$

that has largest variance. The coefficients $p_{11}, p_{12}, ..., p_{1p}$ are the solution of:

$$\begin{aligned} \max_{p_{11},p_{12},...,p_{1p}} & \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} p_{j1}x_{j1} \right)^2 \right\} \\ \text{s.t.} \quad & \sum_{j=1}^{p} p_{j1}^2 = 1 \end{aligned} \tag{3.3}$$

The vector containing the directions p_{i1} defines the direction of the first principal component and all points are projected among these direction finding the principal component scores $z_{11}, z_{21}, ..., z_{p1}$.
The $i-th$ principal component direction is found as the direction that maximizes the variance of the projected data and is orthogonal to the first $i-1$ principal components.
Principal components corresponds to the eigenvectors of the covariance matrix of the data.
In fact, writing 3.3 in matrix form we find that:

$$p_1 = \arg\max_{\|p_1\|=1} \{\|Xp_1\|^2\} = \arg\max_{\|p_1\|=1} \{p_1^T X^T X p_1\} \tag{3.4}$$

It is a unit vector, so it also satisfies:

$$p_1 = \arg\max \{ \frac{p_1^T X^T X p_1}{p_1^T p_1} \} \tag{3.5}$$

The argument corresponds to the *Rayleigh quotient* and the solution is the largest eigenvalue of the covariance matrix when p_1 corresponds to his eigenvector.

3.3.1 PCA on the dataset

Before applying PCA on the dataset, it is important that the data is centered so we decided to *standardize* the dataset with the function *sklearn.preprocessing.scale()*.
The number of components shouldn't be determined a priori, but it should be determined depending to the variance explained.

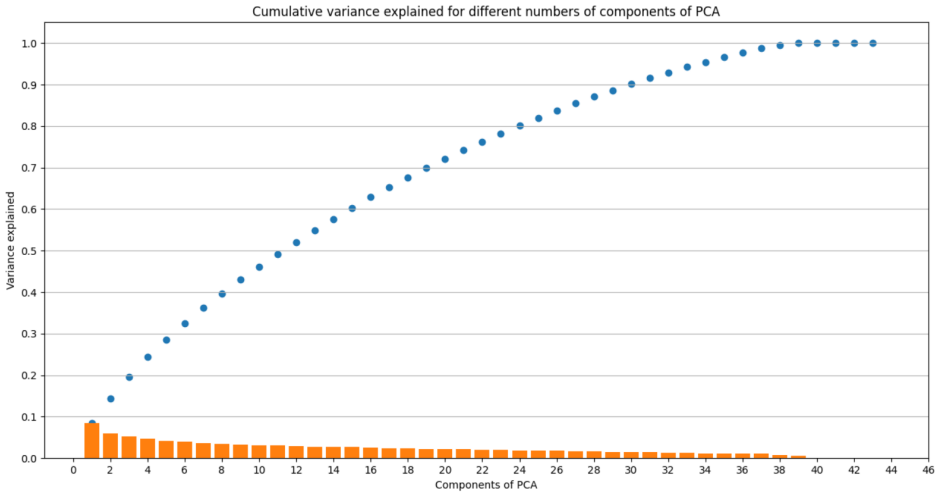


Figure 3.2: Variance explained with different number of components

Figure 3.2 plots in function of different numbers of principal components the variance explained by each one (orange bars) and the cumulative variance explained (blue points).
In particular we can report that the cumulative variance explained is:

- 85 % with 27 components;
- 90 % with 30 components;
- 99 % with 39 components.

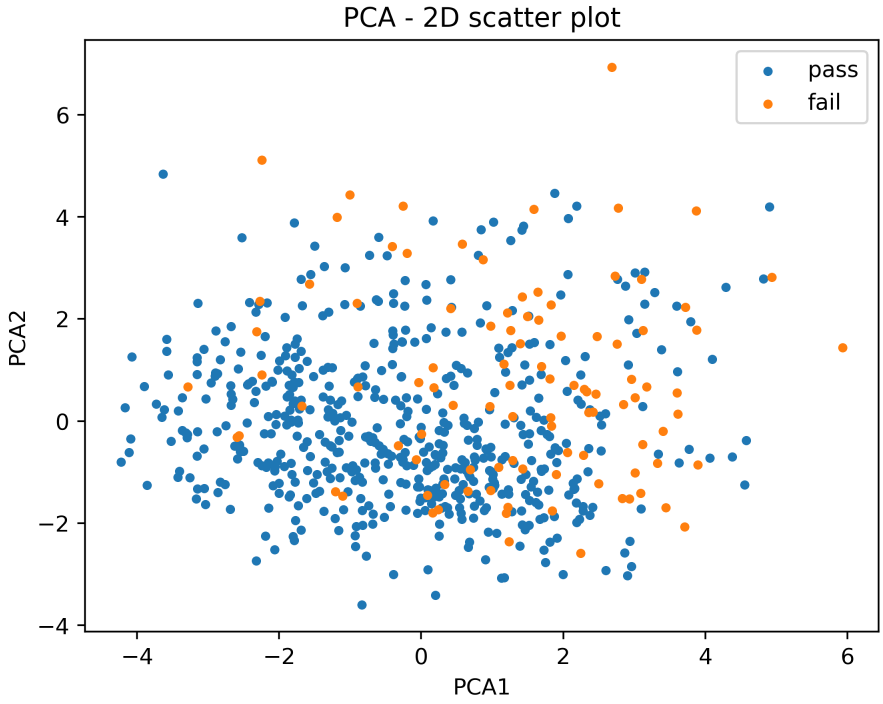


Figure 3.3: Scatter plot after applying PCA with 2 components as output

3.3.2 PCA influence on performance and time

We finally tested how PCA representation of the data influences performances of the classification methods and time spent by training and prediction of the test set.
We applied PCA with 30 components on the dataset and compared execution time and accuracy of KNN classifier and SVM classifier.

```
kNN classifier with pca
10-fold cross validation accuracy for k=5 is: 0.82432
Time elapsed:  0.12001538276672363

kNN classifier
10-fold cross validation accuracy for k=5 is: 0.82891
Time elapsed:  0.22002649307250977

SVM classifier with pca
10-fold cross validation accuracy for C=0.01 and linear kernel is: 0.84591
Time elapsed:  0.20399761199951172

SVM classifier
10-fold cross validation accuracy for C=0.01 and linear kernel is: 0.84591
Time elapsed:  0.3630502223968506
```

We can see that applying PCA, in general, we obtain a slightly worse accuracy but the execution time is faster. However, the analyzed dataset has not a high number of records and attributes, so that execution time is not an issue and our main objective is to obtain an high accuracy.
Another issue for PCA is that the components loose the information of the real attributes. In fact, it is also importart to investigate what are the most important attributes to predict the response variables.
For this reasons we decided to use the original dataset in the classification algorithms.

Chapter 4

Binary classification

We will perform binary classifications using as label the ad hoc created attribute *binary* which takes values *pass* or *fail* whether the value of *G3* is respectively ≥ 10 or < 10.
For this reason, our work is in the field of supervised classification. For each method, in order to tune the hyperparameters, we will split our dataset in two parts: train and validation set.
Due to the small dimension of the dataset we decided to compare different methods using 10-fold cross validation.

4.1 KNN

K-Nearest Neighbour is a non-parametric classification algorithm as it does not make any assumption on the distribution of the data.
Given a test point x_0 the algorithm computes the distance between the latter and the other training points, selecting the K closest to x_0. Due to the different scales of the attribute, it is required a normalization transformation preprocessing data.
We decided to use the Euclidean norm as distance function:

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.1}$$

The classifier compute the conditional probability as:

$$P(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \tag{4.2}$$

where N_0 represents the neighbours and

$$I(x = j) = \begin{cases} 1 & if \quad x = j \\ 0 & otherwise \end{cases} \tag{4.3}$$

Finally the test point is classified to the class with largest probability, corresponding to the most frequent label in the neighborhood.

The hyperparameter K identifies the number of neighbours to be considered in the classification and influences heavily the performances of the classifier. In fact, if K is very small the boundary is very flexible, corresponding in a high variance and low bias classifier. On the other hand, if K is high, the decision boundary is almost linear, resulting in a low variance and high bias classifier.

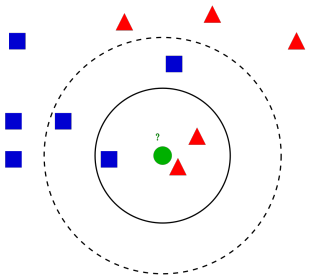


Figure 4.1: K-Nearest Neighbours classification

The KNN classifier lies under the assumption that similar points (in terms of distance) share similar labels.
However, this classifier is very costly in terms of space because it needs to store all the data and in terms of time because of the computation of the distance.
Moreover, the KNN classifier tends to perform poorly with high dimensional data and in particular with categorical data.

4.1.1 Hyperparameter tuning

To find the optimal number of number of neighbours we trained different models with K in a certain range and evaluated the accuracy in the validation set.

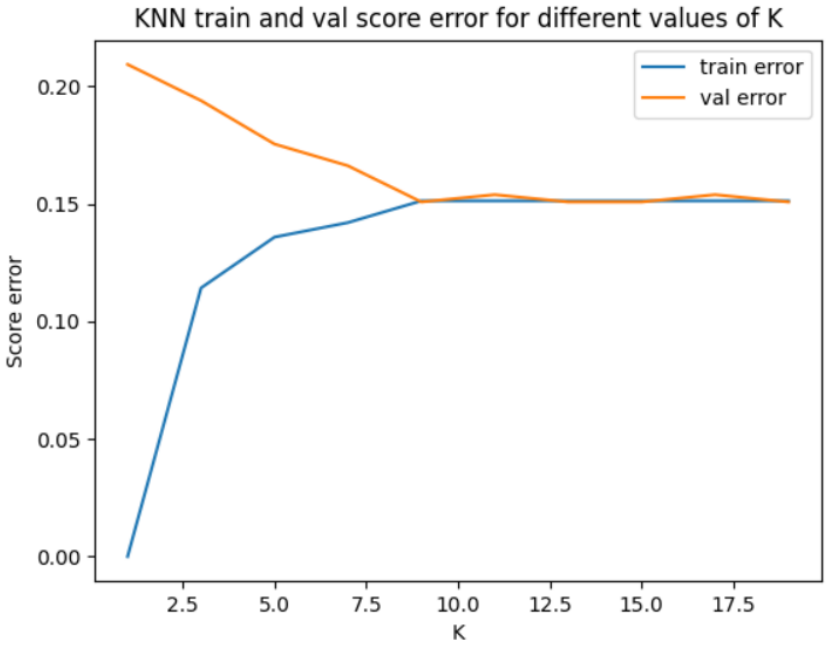


Figure 4.2: Knn classification train and validation error

As we can see in Figure 4.2 the optimal number for K is 5. In fact, we can observe a U-shape in the validation error. Using low values of K we can observe overfitting: the train error is very low as the decision boundary is very flexible, but the validation error is higher than the optimal value.

```
kNN classifier
                k=1       k=3       k=5       k=7       k=9       k=11      k=13      k=15      k=17      k=19
-----------  --------  --------  --------  --------  --------  --------  --------  --------  --------  --------
Train score  1         0.885802  0.864198  0.858025  0.848765  0.848765  0.848765  0.848765  0.848765  0.848765
Val score    0.790769  0.806154  0.824615  0.833846  0.849231  0.846154  0.849231  0.849231  0.846154  0.849231
```

Figure 4.3: K-Nearest Neighbours classification train and validation score

4.1.2 Cross validation accuracy

Training again the model with the chosen hyperparameter, we can compute the 10-fold cross validation to compare different classifiers.

```
10-fold cross validation accuracy for k=9 is: 0.8428365384615384
```

		Predicted	
		Fail	Pass
Actual	Fail	0.04	0.96
	Pass	0.004	0.996

Table 4.1: KNN - $k = 9$, Confusion matrix

It is also interesting to analyze the confusion matrix to better understand how the model is performing in the prediction of each class. We can see that the accuracy is quite high, but observing the confusion matrix, while 97,85 % of true pass are predicted correctly, only 15,21 % of fail are correctly predicted.
Applying *SMOTE*, we can increase the precision on the minority class. In fact, the total accuracy is way lower, but we can see that while the precision on the pass class is slightly lower at 79,57 %, the precision in the fail class is better at 58,70 %.

```
kNN classifier with SMOTE
10-fold cross validation accuracy for k=9 is: 0.6545913461538462
```

		Predicted	
		Fail	Pass
Actual	Fail	0.66	0.34
	Pass	0.356	0.644

Table 4.2: KNN with SMOTE - $k = 9$, Confusion matrix

4.1.3 Subset selection

Due to the poor performances of KNN using all features we decided to use only a subset of them. The main problem is that the distance function (and consequently the neighbourhood) looses significativity when working on high dimensional spaces.
Initially we greedily selected only numerical features. This model lead to better performances compared to the classifier considering all futures in terms of accuracy, while the confusion matrix is very similar to the previous case.

```
kNN classifier
Selecting only numerical attributes
10-fold cross validation accuracy for k=9 is: 0.8459134615384617
```

		Predicted	
		Fail	Pass
Actual	Fail	0.12	0.88
	Pass	0.022	0.978

Table 4.3: KNN on numerical features only - $k = 9$, Confusion matrix

4.2 Logistic regression

Logistic Regression is a classification algorithm where the response variable is categorical. It works very similar to linear regression, but with a binomial response variable.
Let be X the predictor and Y the outcome. In a two classes context the response of an outcome is often mapped to a generic 0/1 coding. The goal is to find a relationship between $p(X) = Pr(Y = 1|X)$ and X. The model used in Linear Regression is the following:

$$p(X) = \beta_0 + \beta_1 X.$$

This model performs poorly in a binary context because the predicted value may be lower or higher than 0 and 1 respectively. To avoid this problem, Logistic Regression models $p(X)$ using a function that gives output between 0 and 1 for all values of X called Logistic function:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{4.4}$$

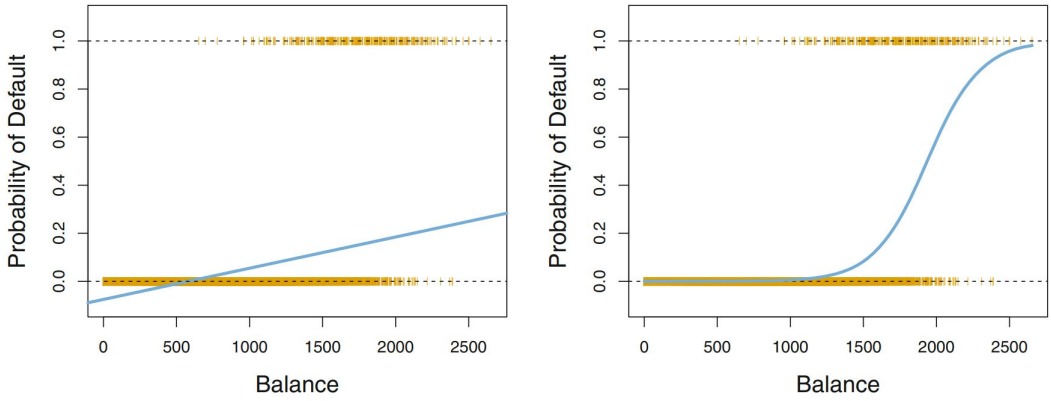


Figure 4.4: Left: Estimated probability of default using linear regression. Right: Predicted probabilities of default using logistic regression.

4.2.1 Model Fitting

Fitting the model means find appropriate β_0 and β_1 coefficients. To fit the model a method called *maximum likelihood* is used. There are other methods such as *least squares* approach but the chosen one is preferred since it has better statistical properties.
The goal is to find β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ corresponds as closely as possible to the known ground truth. The following equation is called *likelihood function*

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})) \tag{4.5}$$

In the fitting phase the model calculate $\hat{\beta}_0$ and $\hat{\beta}_1$ in order to maximize this likelihood function.
Once the coefficients have been estimated, the $\hat{p}(X)$ is easily calculated.

4.2.2 Hyperparameter tuning

In the *scikit-learn* implementation of the logistic regression there is a parameter representing the strength of the regularization factor. A regularization factor is a term added in the loss function in order to prevent overfitting. In our case, the parameter C represents an inverse regularization factor, so that the loss function can be seen as:

$$\min_f C \sum L(f(x_i), y_i) + R(f) \tag{4.6}$$

In our case, $L(f(x_i), y_i)$ is the logistic loss and $R(f)$ is the l_2 norm. In this way, as C grows larger, the regularization term is less important.
We trained different models with the hyperparameter C taking values $[0.001, 0.01, 0.1, 1, 10, 100]$. In Figure 4.5 and 4.6 we can see respectively how the train and validation error and score varies with the different values of C. The optimal value is then C=0.1.

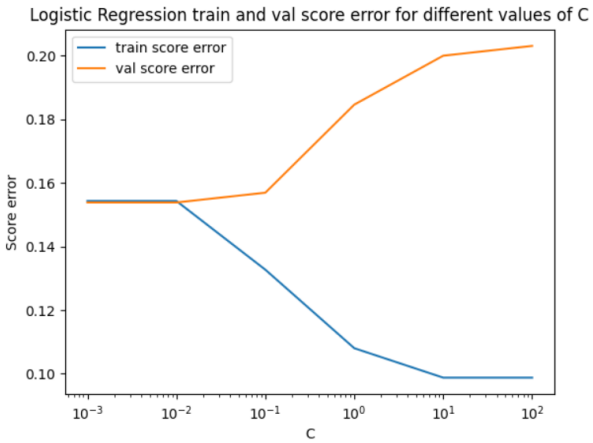


Figure 4.5: Logistic regression classification train and validation error

```
Logistic regression classifier
               C=0.001    C=0.01     C=0.1       C=1       C=10      C=100
-----------  ---------  --------  --------  --------  --------  --------
Train score   0.845679  0.845679  0.867284  0.891975  0.901235  0.901235
Val score     0.846154  0.846154  0.843077  0.815385  0.8       0.796923
```

Figure 4.6: Logistic regression classification train and validation score

4.2.3 Cross validation accuracy

The 10-fold cross validation accuracy of logistic regression with the optimal parameter is:

```
Logistic regression classifier
10-fold cross validation accuracy for C=0.1 is: 0.8427884615384617
```

We obtained a quite high accuracy, but the precision predicting the minority class is very low. After applying *SMOTE* to the train set, we obtained the following output:

```
Logistic regression classifier with SMOTE
10-fold cross validation accuracy for C=0.1 is: 0.7732932692307692
```

		Predicted	
		Fail	Pass
Actual	Fail	0.2	0.8
	Pass	0.04	0.96

Table 4.4: Confusion matrix for C=0.1

		Predicted	
		Fail	Pass
Actual	Fail	0.34	0.66
	Pass	0.167	0.833

Table 4.5: Confusion matrix for C=0.1 with SMOTE

In this case, although the overall accuracy is slightly lower, the precision of the *fail* class is way higher, equal to 60,87 %. The choice of the model could depend on the applications and the importance we give to each prediction: if we are aiming to detect students that may be problematic to give them extra support, the classification model with *SMOTE* is better.

4.3 LDA

LDA (*Linear Discriminant Analysis*) is a statistical supervised method that aims to reduce the dimensionality of feature space. More specifically, it projects the data onto a lower dimensional space by finding a linear combination of features that characterizes or separates two or more classes of objects. In this section the problem is studied in its binary form (LDA for two classes). It also can be generalized to a multiclass context.
Rather than the Logistic regression approach, where we model the conditional distribution of the response Y, given the predictor(s) X; in LDA approach we model the distribution of the predictors X separately in each of the response class (given Y). Then the Bayes' theorem is used to estimate $Pr(Y = k|X = x)$. This model performs very close to the Logistic Regression one but there are some differences:

- LDA performs better when the classes are well-separated while Logistic Regression shows instability issues.
- LDA performs better than Logistic Regression when dealing with reduced feature space and approximately normal distribution of the predictors.

4.3.1 Bayes' Theorem for Classification

Suppose that we want to classify an observation into one of $K \geq 2$ classes. So Y can assume K possible and unordered values. π_k represent the overall probability that a randomly chosen observation belongs from the kth class. Let be $f_k(x) = Pr(X = x|Y = k)$ the *density function* of X that comes from the kth class. The larger $f_k(x)$ will be, the more will the likelihood that the observation belongs to the class kth. The Bayes' theorem states that:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}. \tag{4.7}$$

Let $Pr(Y = k|X = x)$ be $p_k(x)$. $p_k(x)$ is the posterior probability that an observation $X = x$ belongs to the kth class. In general, estimating π_k and $f_k(x)$ (prior probability) is easier than estimating $p_k(x)$. Estimating π_k can be done by, given a random sample of elements, computing the fraction of observations that belong to class kth. On the other hand, estimating $f_k(x)$ tends to be more difficult unless some assumption about the form of the densities are taken.

4.3.2 Linear Discriminant Analysis for p = 1

Considering $p = 1$ means that we have only one predictor. $f_k(x)$ is assumed to be *normal* or *Gaussian*. This means that $f_k(x)$ will be:

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right), \tag{4.8}$$

where μ_k and σ_k^2 are the mean and the variance for the parameters for the kth class. Supposing that $\sigma_1^2 = ... = \sigma_K^2 = \sigma^2$ and plugging 4.8 into 4.7 we obtain

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)} \tag{4.9}$$

Taking the log of 4.9 and rearranging the terms we obtain:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \tag{4.10}$$

The Bayes classifier assigns to an observation the class for which 4.10 is largest.
When $p > 1$ equation 4.10 becomes:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k) \tag{4.11}$$

where Σ is the covariance matrix of X. The classifier assigns an observation to the class for which 4.11 is higher.

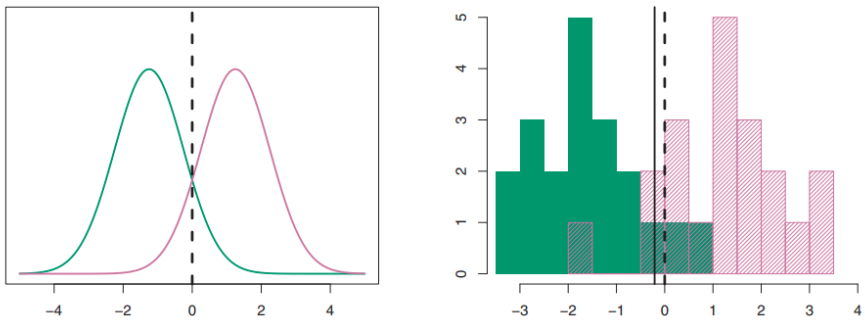


Figure 4.7: On the left: *Two one-dimensional normal density functions. The dashed line represents the Bayes decision boundary.* On the right: *20 observation of each class are shown as histogram. The dashed line represents the Bayes decision boundary. The solid black line represents the LDA decision boundary estimated from the training data*

4.3.3 Cross validation accuracy

The LDA classifier cross validation accuracy and confusion matrix is:

```
LDA classifier
10-fold cross validation accuracy is: 0.8334855769230771
```

		Predicted	
		Fail	Pass
Actual	Fail	0.4	0.6
	Pass	0.098	0.902

Table 4.6: LDA - Confusion matrix

In this case, the precision of the predictions of the minority class is the highest we found without SMOTE.
Applying SMOTE:

```
LDA classifier with SMOTE
10-fold cross validation accuracy is: 0.7916826923076923
```

		Predicted	
		Fail	Pass
Actual	Fail	0.34	0.66
	Pass	0.145	0.855

Table 4.7: LDA with SMOTE - Confusion matrix

The precision of *fail* class has a slightly improvement, but in this case, SMOTE algorithm is not so effective, and the original model could be better.

4.4 SVM

Support Vector Machines are a class of supervised learning linear classifiers. In particular it searches for a *(n-1)*-dimensional hyperplane that separates the two classes of points with a margin that is as large as possible.

4.4.1 Hard margin

Given the training points $(x_i, y_i)_{i=1}^n$ finding the decision boundary corresponds to the following optimization problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(\langle x_i, w\rangle + b) \geq 1 \quad \forall i \end{aligned} \tag{4.12}$$

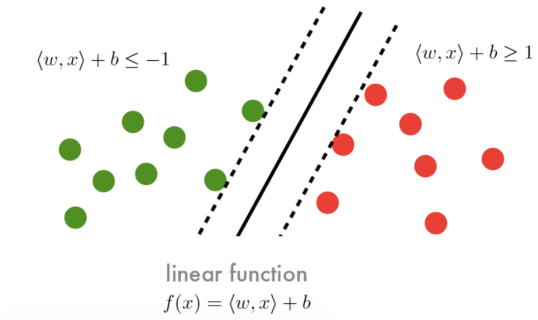


Figure 4.8: SVM classification

The SVM large margin classifier has several advantages:

- it uses the scalar product that allows the use of a kernel function;
- it is robust against uncertainty;
- it depends only on points on the margin, called *support vectors*;
- it is easy to find for easy problems.

However, the optimization problem 4.12 has at least one feasible solution only if data from the two classes are linearly separable. It is a strong requirement, that can be relaxed by allowing some misclassified points with the soft margin classifiers.

4.4.2 Soft margin

Relaxing the constraint, we find the following problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_i \epsilon_i \\ \text{s.t.} \quad & y_i(\langle x_i, w\rangle + b) \geq 1 - \epsilon_i \quad \forall i \\ & \epsilon_i \geq 0 \quad \forall i \end{aligned} \tag{4.13}$$

This problem is always feasible. In fact, we can see that now the constraint that makes each point correctly classified can be not met, but these situations are penalized in the objective function through the parameter C.
If C is low, the decision boundary is flexible and there could be many misclassified points. Otherwise, when we have high values of C, almost every point is correctly classified by the decision boundary as it is less flexible.

4.4.3 Hyperparameter tuning

We trained multiple SVM classifiers with linear kernel and parameter C varying in $[0.001, 0.01, 0.1, 1, 10, 100]$. Figures 4.9 and 4.10 reports train and validation errors and scores with different values of C.

The optimal value is $C = 0.01$.

4.4.4 Cross validation accuracy

The SVM classifier cross validation accuracy and confusion matrix is:

```
SVM classifier
10-fold cross validation accuracy for C=0.01 and linear kernel is: 0.8519951923076924
```

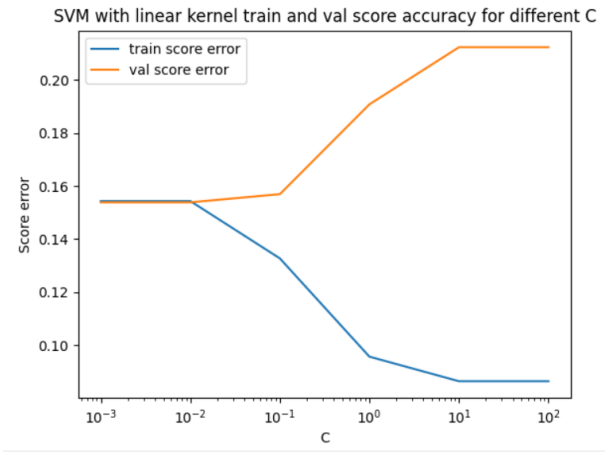


Figure 4.9: SVM classification train and validation error

```
SVM classifier
RBF kernel
                C=0.1     C=1.0     C=10.0    C=100.0    C=1000.0    C=10000.0
-----------  --------  --------  --------  ---------  ----------  -----------
Train score  0.845679  0.845679  0.861111  0.950617    0.996914     1
Val score    0.846154  0.846154  0.843077  0.8         0.775385     0.778462

Linear kernel
               C=0.001    C=0.01     C=0.1       C=1       C=10     C=100
-----------  ---------  --------  --------  --------  --------  --------
Train score   0.845679  0.845679  0.867284  0.904321  0.91358   0.91358
Val score     0.846154  0.846154  0.843077  0.809231  0.787692  0.787692
```

Figure 4.10: SVM classification train and validation score

		Predicted	
		Fail	Pass
Actual	Fail	0.14	0.86
	Pass	0.029	0.971

Table 4.8: SVM - Confusion matrix

The accuracy is the highest we achieved, but precision of the *fail* class is very low. Using SMOTE we have:

```
SVM classifier with SMOTE
10-fold cross validation accuracy for C=0.01 and linear kernel is: 0.794735576923077
```

		Predicted	
		Fail	Pass
Actual	Fail	0.4	0.6
	Pass	0.138	0.862

Table 4.9: SVM with SMOTE - Confusion matrix

Applying *SMOTE* algorithm, as usual, precision of *fail* class increase and overall accuracy

decreases. However, it is a discrete result, that could be compared to SVM for umbalanced classification problems below.

4.4.5 Kernel trick

Sometimes a linear separator is not suitable for certain data and SVM allows to map them in another space obtaining a non linear decision boundary. In fact the inner product can be generalized and replaced with a kernel function of the form:

$$K(x, x') = \sum_i \lambda_i \phi_i(x) \phi_i(x') \tag{4.14}$$

This function must satisfy some properties as symmetry and must be efficiently computable. Examples of kernel functions are:

Linear	$\langle x, x' \rangle$
Laplacian RBF	$\exp(-\lambda \|x - x'\|)$
Gaussian RBF	$\exp(-\lambda \|x - x'\|^2)$
Polynomial	$(\langle x, x' \rangle + c)^D$

In our case, our dataset contains many categorical attributes, that are converted into 0-1 dummy variables. For this reason we expected linear kernel to perform better, but we also tried Gaussian RBF and we compared the corresponding errors.

4.4.6 Cross validation accuracy

After tuning again the hyperparameter using the new kernel, the optimal value is C=10. The 10-fold cross validation accuracy and the confusion matrix is:

```
SVM classifier
10-fold cross validation accuracy for C=10 and rbf kernel is: 0.8443269230769233
```

		Predicted	
		Fail	Pass
Actual	Fail	0.1	0.9
	Pass	0.022	0.978

Table 4.10: SVM with RBF kernel - Confusion matrix

Comparing this result with the previous one we can see that the overall accuracy is slightly lower, but the prediction of the minority class is better.
Applying *SMOTE*, the optimal value for C is 100:

```
SVM classifier with SMOTE
10-fold cross validation accuracy for C=100 and rbf kernel is: 0.7732932692307692
```

		Predicted	
		Fail	Pass
Actual	Fail	0.38	0.62
	Pass	0.138	0.862

Table 4.11: SVM with RBF kernel and SMOTE - Confusion matrix

In this case, the SMOTE algorithm is very effective. In fact, the precision of *fail* class is 62,5 %, but the overall accuracy is slightly lower.

4.4.7 SVM for umbalanced classification problems

The function *SVC* (*Support Vector Classifier*) from package *sklearn* allows to assign weights for different classes when dealing with unbalanced data. Using the argument *class_weight* it sets the parameter C for the class i to $weight[i] * C$. In this way we can give more importance to a certain class and modify the decision boundary.
The resulting model is:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_i \epsilon_i + C * weight \sum_i \epsilon_i' \\ \text{s.t.} \quad & \langle x_i, w\rangle + b \geq 1 - \epsilon_i \quad if \qquad y_i = 1 \\ & \langle x_i, w\rangle + b \leq 1 - \epsilon_i' \quad if \qquad y_i = -1 \\ & \epsilon_i \geq 0 \quad \forall i \end{aligned} \tag{4.15}$$

In our case, we decided to give more importance to the class *fail* that includes 15% of the dataset.

4.4.8 Hyperparameter tuning

We decided to analyze the linear SVM, fixing the value of the hyperparameter C=0.01 and tuning the weight parameter among the values $[1, 1.2, 1.4, 1.6, 1.8, 2]$.
We find the results in Figure 4.11.
The optimal value is weight=2.

```
SVM unbalanced classifier
              weight=1    weight=1.2    weight=1.4    weight=1.6    weight=1.8    weight=2
-----------  ----------  ------------  ------------  ------------  ------------  ----------
Train score    0.845679      0.845679      0.845679      0.851852      0.854938    0.861111
Val score      0.846154      0.846154      0.846154      0.846154      0.849231    0.852308
```

Figure 4.11: SVM unbalanced classification train and validation score

4.4.9 Cross validation accuracy

After training the classifier with the weight=2, we can find the following:

```
SVM unbalanced classifier
10-fold cross validation accuracy for weight=2 is: 0.796514423076923
```

		Predicted	
		Fail	Pass
Actual	Fail	0.44	0.56
	Pass	0.135	0.865

Table 4.12: SVM unbalanced - Confusion matrix

The overall accuracy is equal to the standard SVM classifier, but we have a better prediction of the minority class. We also can obtain better precision of *fail* class by increasing the weight value, but in this way the overall precision decreases.

4.5 Decision trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Some **advantages** of decision trees are:

- Simple to understand and to interpret. Trees can be visualised. Uses a white box model. This means that the decisionmaking process is known to the user. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. *Scikit* models does not support missing values and categorical variables.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree
- Able to handle multi-output problems.

The **disadvantages** of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Decision-tree learning algorithms are based on heuristics to make the optimal decision locally. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

4.5.1 Structure

A decision tree is composed by *nodes* and *branches*. The nodes can be internal or external (also called *leaves*). Each internal node represent a condition based on a specific feature and splits the data in two branches. Each leaf node represents a class label (decision taken after computing all attributes).

4.5.2 Building a Decision tree

A decision tree splits the feature space into regions. there are two steps:

1. the predictor space is divided into J distinct and non-overlapping regions $R_1, R_2, ..., R_J$
2. For every observation that falls into region R_j the prediction is the mean of the response values for the training observation in R_j

The regions have, for simplicity, a box shape in an high dimensional space. The goal is to find boxes $R_1, R_2, ..., R_J$ that minimize the RSS (root sum squared) given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{Rj})^2 \tag{4.16}$$

where $\hat{y}_{Rj}$ is the mean response for the training observations within the jth box.

Since considering every possible partition of the feature space into J boxes is unfeasible, a *top-down, greedy* approach known as *recursive binary splitting* is taken into consideration. This process begins at the top of the tree and successively splits the predictor space each time in two new branches. At each step the best split is made at that specific step and the cutpoint s is chosen in order to lead to the greatest possible reduction in RSS. For each step all possible values of all predictors $X_1, X_2, ..., X_p$ are taken into account. The result of the split consist in two regions:

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_j \geq s\} \tag{4.17}$$

such that

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R2})^2 \tag{4.18}$$

This process continues until a stopping criterion is reached.

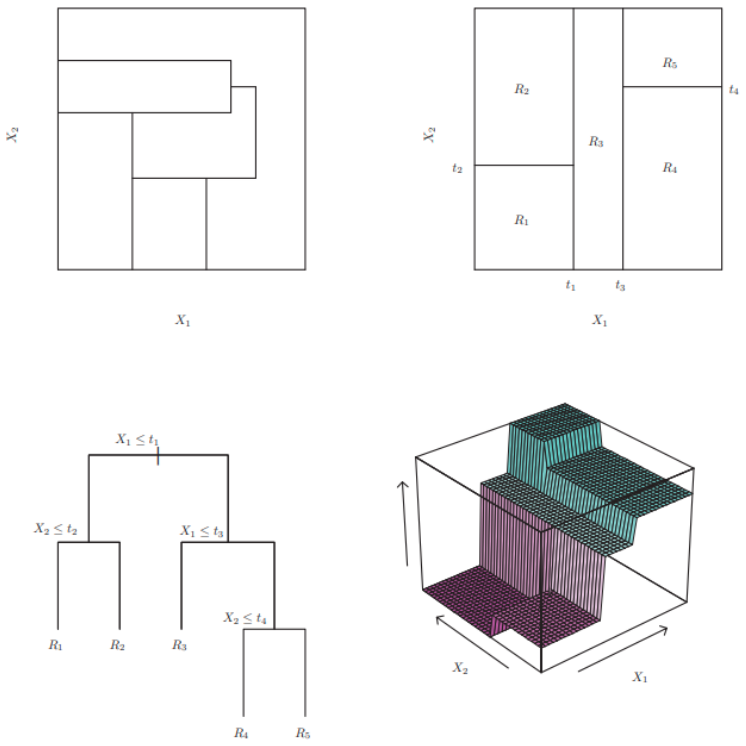


Figure 4.12: Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree*

4.5.3 Tree pruning

Although the process described above may produce good results on the train set, it is likely to overfit the data and lead to poor results on the test set. In addition the tree might be too complex and not easy to read. So it is better to build a less deeper tree with a cost of a little bias. Increasing the RSS threshold is not a solution because stopping the process at a certain step may prevent to reach a good but further split. A good strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*. The subtree is chosen in order to achieve the lowest test error rate. Given a Tree T, considering all possible subtrees would be unwieldy and time-consuming. Instead, a small set of subtrees is considered by using a criterion known as *Cost complexity pruning*, or *weakest link pruning*. Rather than considering every possible subtree, a sequence of trees indexed by a nonnegative tuning parameter α is considered.

For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{Rm})^2 + \alpha|T| \tag{4.19}$$

is as small as possible. Here $|T|$ is the number of leaves of the tree T, R_m is the rectangle corresponding to the mth leaf, and $\hat{y}_{R_m}$ is the predicted response associated with R_m. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0.

4.5.4 Classification Trees

A *classification tree* is very similar to a regression tree except that the goal is to predict a qualitative response rather than a quantitative one. The prediction is given by the *most occurring class* rather than the mean of the elements in that specific region. In order to determine the reliability of a prediction, a useful information is needed: the *class proportion inside* a region. RSS is not suitable as a criterion for making binary split, so the *classification error rate* is used:

$$E = 1 - \max_k(\hat{p}_{mk}). \tag{4.20}$$

Here $\hat{p}_{mk}$ is the proportion of training observations in the mth class that belongs to the kth class. As this method is not enough sensitive for tree-growing other two methods are used:

- The *Gini index*, given by

  $$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{4.21}$$

  is a measure of the total variance across the K classes. It is a measure of the node purity, in fact a small value of G means that the node contains mainly one class.

- *Cross-entropy* is defined by

  $$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \tag{4.22}$$

  The output of this measure is quite similar to the first one as it tends to zero when the node is pure.

Analyzing the performance of the two measures, we decided to use the entropy criterion that led us to slightly better accuracy.

4.5.5 Cross validation accuracy

The *sklearn.tree.DecisionTreeClassifier* accepts parameters to prevent overfitting. In particular we used *min_samples_leaf* that corresponds to:
"The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches."
We found the optimal parameter of this value at 15, resulting in a simple but effective model. One of the main advantages of the classification trees is that they are white box models and allows to represent the decision process and the attributes determining it. Training a classification tree in the train dataset we find the model in Figure 4.13.

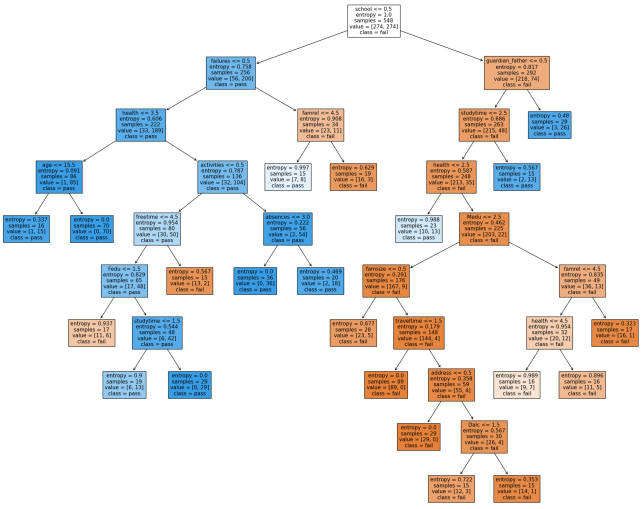


Figure 4.13: Trained decision tree

It is also possible to compute the importance of each feature in terms of (normalized) total reduction of the criterion brought by that feature.

```
The features sorting by descending importance are:
('failures', 0.4586454405996651)
('school', 0.3465696419902518)
('sex', 0.11082379807216879)
('freetime', 0.051782740645410205)
('Mjob_other', 0.02496168230867128)
('age', 0.007216696383832747)
```

As we could expect, the most important feature is the number of failures of the student. There is a big difference in the two schools regarding the number of *fail* and *pass*. The other factors are sex (male or female), if the student is in a romantic relationship and the health status.
The cross validation accuracy and the confusion matrix are:

```
Decision Tree Classifier
Cross validation accuracy: 0.8072355769230771
```

		Predicted	
		Fail	Pass
Actual	Fail	0.44	0.56
	Pass	0.12	0.88

Table 4.13: Decision tree - Confusion matrix

The accuracy is very high and we have also a good prediction of the minority class. Applying *SMOTE* we obtain:

```
Decision Tree Classifier with SMOTE
Cross validation accuracy: 0.7440625000000001
```

		Predicted	
		Fail	Pass
Actual	Fail	0.4	0.6
	Pass	0.175	0.825

Table 4.14: Decision tree with SMOTE - Confusion matrix

The model above is the best model we found using SMOTE. In fact we have that 70,8 % of the *fail* class are classified correctly, with an overall accuracy of 78,4 %.

4.5.6 Bagging

Since decision trees suffer from *high variance*, a procedure called *bootstrap aggregation* or *bagging* is often used for reducing the variance in this context.

Given a set of n independent observations $Z_1, ..., Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observation is σ^2/n. This means that *averaging a set of observations reduces variance*. So a way to reduce variance and increase accuracy could be to build a separate model for each training set and then to average the resulting predictions. Given $\hat{f}^1(x), \hat{f}^2(x), ..., \hat{f}^B(x)$ as models from B separate training sets the averaged result is given by

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x). \tag{4.23}$$

This is not feasible because, generally, it isn't possible to access to multiple training sets. A solution is to bootstrapping by taking repeated samples from the single training data set, this is called bagging:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x). \tag{4.24}$$

In order to apply bagging to decision trees, a simple approach could be the following. For a given test observation, each one of the B trees makes its prediction. The overall prediction will be the most frequent one.

4.5.7 Random forest

Random forests consist in an improvement over bagged trees through *decorrelating* the trees. As in bagging, we build a number of decision trees on bootstrapped training samples. But for each split a *random sample of m predictors* is chosen as split candidates from the full set of p

predictors. The split will use only one of those m predictors. Generally $m \approx \sqrt{p}$. This choice is made because, if all predictors were considered, the strongest or the most discrivinative predictor would always be preferred over the others and the advantages of a random choice would be lost. *Random forest* algorithms have lower variance comparing to classification trees. This prevents overfitting. However, in contrast to classification trees, this is a black-box model: the decision process is made averaging multiple trees and we can't identify a unique logic process and a graphic representation.

4.5.8 Hyperparamter tuning

When training a *Random forest classifier* we searched for the optimal value of two parameters:

- n_estimators: number of decision trees to be trained;
- criterion: the function to measure the quality of the split.

Training multiple models with different values of the parameters above led us to the results in Figure 4.14.

```
Random Forest Classifier
                                  n=10      n=50     n=100     n=200     n=500
--------------------------------  --------  --------  --------  --------  --------
Test score with entropy criterion  0.877301  0.877301  0.877301  0.877301  0.877301
Test score with gini criterion     0.877301  0.877301  0.877301  0.877301  0.877301
```

Figure 4.14: Random forest classifier validation scores

The results are very similar to each other and we decided to use *Gini criterion* that has slightly better accuracy and $n = 100$.
The choice of a value different from 1 of *min_samples_leaf* led us to worse result. In fact, it is not necessary a strict control of overfitting as random forest averages prediction on a set of decision tree classifiers to reduce it.

4.5.9 Cross validation accuracy

Training again the model with the optimal hyperparameters lead us to the following results:

```
Random Forest Classifier
10-fold cross validation accuracy for n=100 and gini criterion is: 0.8397115384615386
```

		Predicted	
		Fail	Pass
Actual	Fail	0.05	0.95
	Pass	0.028	0.972

Table 4.15: Random forest - Confusion matrix

We can see that the cross validation accuracy is quite high, but slightly worse than the decision tree one. The prediction of the minority class in terms of precision is very low, too.
Like the decision trees, we can compute the feature importances as the normalized total reduction of the criterion brought by the attribute, averaged on all the trees in the random forest.

```
The features sorting by descending importance are:
('failures', 0.10017109712366125)
('school', 0.06865525715741172)
('higher', 0.06523202488350203)
('absences', 0.05530116870496074)
('freetime', 0.04606938020622833)
('goout', 0.04394017553342254)
```

In this case, the first values are lower than the decision trees, because of the averaging. This allow us to detect more features that influences the response variable:

- *higher*: binary attribute that represents if the student wants to take higher education;
- number of absences;
- mother education;
- weekend alcohol consumption.

Applying SMOTE we obtain:

```
Random Forest Classifier with SMOTE
10-fold cross validation accuracy for n=100 and gini criterion is: 0.8010817307692306
```

		Predicted	
		Fail	Pass
Actual	Fail	0.2	0.8
	Pass	0.07	0.93

Table 4.16: Random forest with SMOTE - Confusion matrix

As expected, the overall accuracy is slightly lower but the precision of *fail* class is higher.

Chapter 5

Conclusion

In this chapter we aim to give a general point of view of our project, comparing the different classifiers considered with different configurations of the dataset.
In fact, we considered only the dataset without the *G1* and *G2* attributes, but it should be interesting to see how the accuracy change considering also these attributes.
The table below reports the cross validation accuracy of all the binary classifiers considered with the following dataset configurations:

1. without G1 and G2;
2. considering only G1;
3. considering both G1 and G2.

model	accuracy1	accuracy2	accuracy3
kNN	0.842	0.878	0.906
LDA	0.833	0.859	0.882
Logistic Regression	0.842	0.884	0.924
SVM	0.852	0.886	0.920
SVM unbalanced	0.797	0.874	0.903
decision tree	0.807	0.885	0.932
random forest	0.839	0.891	0.915

A powerful tool to compare different classifiers is the ROC curve in Figure 5.1.
The *receiver operating characteristic* (ROC) is a metric to evaluate predictions of a classifier varying the discrimination threshold. The plot of the ROC curve reports the true positive rate (in the y axis) against the false positive rate (in the x axis) varying the threshold value. Ideally we would like the curve to be as high as possible, corresponding in predictions with high true positive rate and low false positive rate. This means that a larger *area under the curve* (AUC) is better.

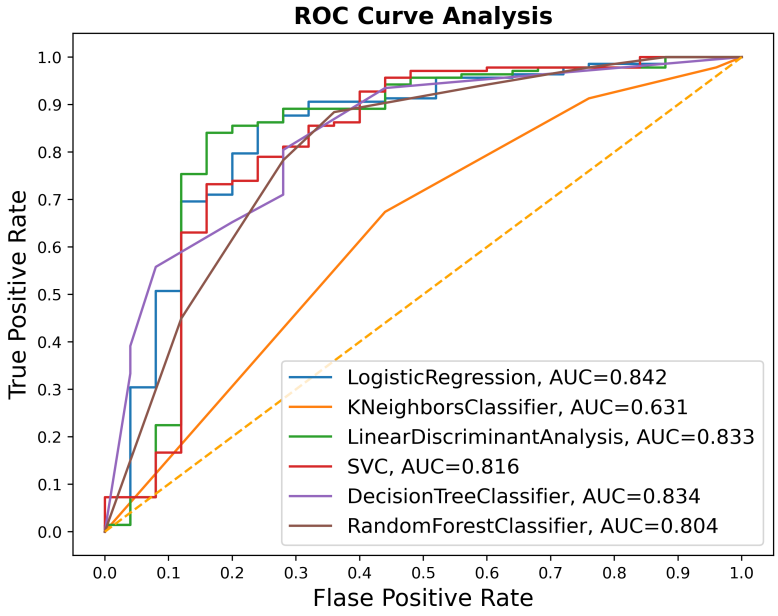


Figure 5.1: ROC curve

In this work, we had the chance to understand the underlying process and test many classification algorithms. We also learnt how to handle an unbalanced dataset, with oversampling methods and modifying the loss function of algorithms.
The choice of the best algorithms depends on:

- the information we have in terms of predictor;
- our research goal.

In particular, we can see from the table above that the information about the student demographic and social situation can lead to a good prediction in terms of accuracy, but it is obviously higher when we have also *G1* or both *G1* and *G2*. The performances of the classifiers vary with the configuration of the dataset: for example LDA slightly increases the accuracy, while decision trees and SVM perform way better with more informations.
The possible goals of this work could be:

- obtain the best accuracy possible: in this case, the best algorithms are Decision trees and SVM;
- obtain the best model in terms of ROC curve: the higher values of AUC are achieved by LDA and SVM classifier;
- investigate the link between the response variable and the predictors: the best classifiers are the tree based ones as they have as output the importance of the attributes in classification and Decision trees are visually understandable;
- try to detect *fail* class with the best accuracy: in this case, we can use the SMOTE algorithm with other classifiers to increase the true negative rate or SVM unbalanced with different weights to find an optimal tradeoff between overall accuracy and minority class prediction quality.

Bibliography

[1] P. Cortez and A. Silva. "Using Data Mining to Predict Secondary School Student Performance". In: *Proceedings of 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008)* (2008).

[2] *Pandas documentation*. URL: `https://pandas.pydata.org/docs/user_guide/index.html`.

[3] *Plot documentation*. URL: `https://matplotlib.org/`.

[4] *Scikit-learn documentation*. URL: `https://scikit-learn.org/stable/index.html`.

[5] *SMOTE oversampling*. URL: `https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/`.

[6] *Student performance Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/student+performance`.

[7] James Witten Hastie Tibshirani. *An introduction to Statitical Learning*. Springer.