# Table of Contents

**1 Framing the problem: Motivation and Objectives .................... 6**

**2 Literature Review .................... 7**

**3 Bridging Research and Application .................... 9**

**4 Dataset Overview and Preprocessing .................... 9**

4.1 Context and Initial Data Exploration .................... 9
4.2 Data Refinement and Target Definition .................... 10
4.3 Feature Analysis and Selection .................... 12

**5 Methodology and Model Development .................... 13**

5.1 Dataset Splitting and Validation Strategy .................... 13

5.2 Supervised Machine Learning Models .................... 14
5.2.1 Logistic Regression (Baseline Model) .................... 15
5.2.2 Support Vector Machine .................... 17
5.2.3 K-Nearest Neighbours .................... 18
5.2.4 Naïve Bayes .................... 20
5.2.5 Decision Tree .................... 21

5.3 Ensemble Learning Approaches .................... 23
5.3.1 Bagging: Random Forest .................... 24
5.3.2 Boosting: Gradient Boosting, AdaBoost and CatBoost .................... 26
5.3.3 Stacking .................... 30
5.3.4 Voting: Majority Vote .................... 31

**6 Summary of Findings and Conclusions .................... 32**

6.1 Comparative Analysis of predictive models .................... 32
6.2 Interpretation and Final Reflections .................... 33
6.3 Study Limitations and Future Research .................... 35

**7 Bibliography .................... 36**

**8 Appendix – Python code .................... 38**

# 1 Framing the problem: Motivation and Objectives

The key to the future is undoubtedly the youth. This implies that, as a community, It's in our best interest to ensure they perform as well as possible, not only because of moral reasons or of the personal satisfaction that strengthens our collective sentiment, but also for practical considerations tied to our future, as explained in several educational economics treatise [1]. Indeed, it's a logical assumption that better academic performance reflects greater abilities, which in turn increases the likelihood of professional success. As a consequence, it would result in having a more capable workforce and, as such, better outcomes for our society. This reasoning highlights the importance of understanding which factors influence academic success, in order to be able to identify those that can, to some extent, be shaped to our advantage. Through our analysis, we will also analyse deeper the ongoing debate: is our future merely a matter of chance, or, as the philosopher Seneca previously said, "luck is what happens when preparation meets opportunity"?

Moreover, the ability to predict not only students' academic success but, more importantly, their unsuccess, has always been a hot topic [2] and represents a really valuable tool for educational institutions, enabling a fast intervention to support students who are struggling more and, therefore, improving overall outcomes. In that way, students of the same class or school were provided with the same means, adapted to their needs, and a more equitable learning environment is fostered.

In summary, this study pursues a twofold objective: first, to develop predictive models through a data-driven approach that classifies students according to their academic performance; second, to identify and analyse the factors that mostly influence the aforementioned predictions, in order to draw meaningful social insights and conclusions.

Aiming to delve deeper into the mechanisms just shown, we will begin by reviewing the main findings of previous scientific research on this topic.

Then, the focus will shift to the specific dataset selected for this study, followed by a detailed explanation of the methodology and the various predictive models developed. Once this other section is concluded, we will present the results and draw the insights and conclusions. Finally, we will reflect on the limitations of the study and explore further the potential directions for eventual future research.

# 2 Literature Review

At this point, before delving into our analysis, it's useful to review the most relevant previous studies on the topic of this thesis. There can be identified three key contributions, all of notable significance since they examine different, yet equally relevant, dimensions and factors influencing students' academic performance.

The first noteworthy study is *"Predicting academic success: machine learning analysis of student, parental, and school efforts"* by Xin Jin (2023) [3]. This recent research provides a detailed investigation, restricted to the Asian continent, considering variables strictly related to the personal and academic sphere. The questions used to measure the effects on academic performance and to actually build the models consisted in whether the students complete assigned homework, take extra courses or are closely supervised by their parents. In substance, this study focuses almost exclusively on student and environmental factors, with little attention paid to the role that luck and chance play.

Jin's work builds upon previous results found by Gamboa & Waltenberg (2012) [4], Golley & Kong (2018) [5] and Broer et al. (2019) [6], who highlighted the crucial roles of student, parental and school effort in shaping academic outcomes. Xin Jin takes this to a further point by investigating how much each of these three factors contributes more and, in general, which weight we should associate them with in order to predict academic success. The conclusion obtained still reinforces the idea that these are all linked and that it’s their combined influence that is essential to reduce educational inequality.

The second relevant study, *”Predicting student success: Considering social and emotional skills, growth mindset, and motivation”* conducted by Nola Daley in the first months of 2025 [7], shifts the focus to the psychological dimension of learning, changing also the geographical area which now becomes the United States. Unlike the previous research, this one examines the importance of mental factors too.
Moreover, the main finding is that students who adopt the so called “growth mindset”, a positive objective-oriented mentality, tend to show greater perseverance when faced with challenges, improving consequently their academic outcomes.

However, what both of these papers lack is attention to what happens at a successive stage of academic life (so, older students), but this gap has been addressed by the third and final study we’re going to mention: *“Predicting academic success in higher education”* by Eyman Alyahyan et al [8].
Being a systematic review of the literature, and not an empirical study per se, this paper extends the geographic focus as well, comprising Asia, Africa and Europe. The conclusions retrieved by Alyahyan are relevant also since the factors considered include both demographic variables and academic ones, such as prior academic transcript, learning environments and engagement in learning activities (measured mostly thanks to the involvement in digital Q&A forums). The results show that the most significant predictor of academic success is prior performance, followed closely by the demographic factors taken into account.

It was precisely this last insight that inspired the direction of our study, motivating us to further investigate how these variables interact with each other and contribute to academic success within different contexts, finally aiming to address our main question: to what extent the environment where students grow impact future outcomes?

# 3 Bridging Research and Application

Starting from the point where existing literature left, since it seemed that social factors were not explicitly and empirically isolated before, our research aims to investigate more those.

After having laid the foundations of our study, we now turn our attention to its empirical component, beginning with a complete description of the dataset we used for our predictive analysis and exploring why it suited best our objectives. In fact, the following section presents the characteristics of the chosen data, including its structure, variables and scope.

# 4 Dataset Overview and Preprocessing

## 4.1 Context and Initial Data Exploration

To tackle this kind of analysis, it was essential to rely on a dataset that included both a large number of observations and a broad set of relevant features. After a careful search among publicly available datasets, the most suitable option was a dataset developed by Enrique De La Hoz and Rohemi Zuluaga [9], researchers at the Technological University of Bolívar (UTB). It was constructed by systematically merging multiple databases from the Colombian Institute for Educational Evaluation (ICFES) and was made accessible in CSV format via the Mendeley Data Repository.

The original purpose of this dataset was to analyse academic efficiency among university-level engineering students in Colombia, with a specific focus on their performance in national standardized assessments. Since the aim of that project was strongly aligned with the goals of ours, this dataset was selected as it provided not only a sufficiently large sample but also a rich variety of variables spanning both academic and social dimensions.

An initial exploration of the dataset was performed through Python (fully documented in the Appendix) revealed that it contains information on

12411 students, each one identified by an anonymized code to ensure privacy, and includes 45 distinct variables. Among these factors, 18 are numerical (e.g. test scores and GPA), while the remaining ones are categorical (e.g. gender, type of school, parental education).

As aforementioned, one of the main strengths of this dataset lies in the wide variety of variables it includes, covering both academic and personal dimensions of the students' lives. Moreover, this allows for a more comprehensive understanding of the factors that may influence academic performance. Without listing every single variable, it is useful to extract and explain the most relevant ones.

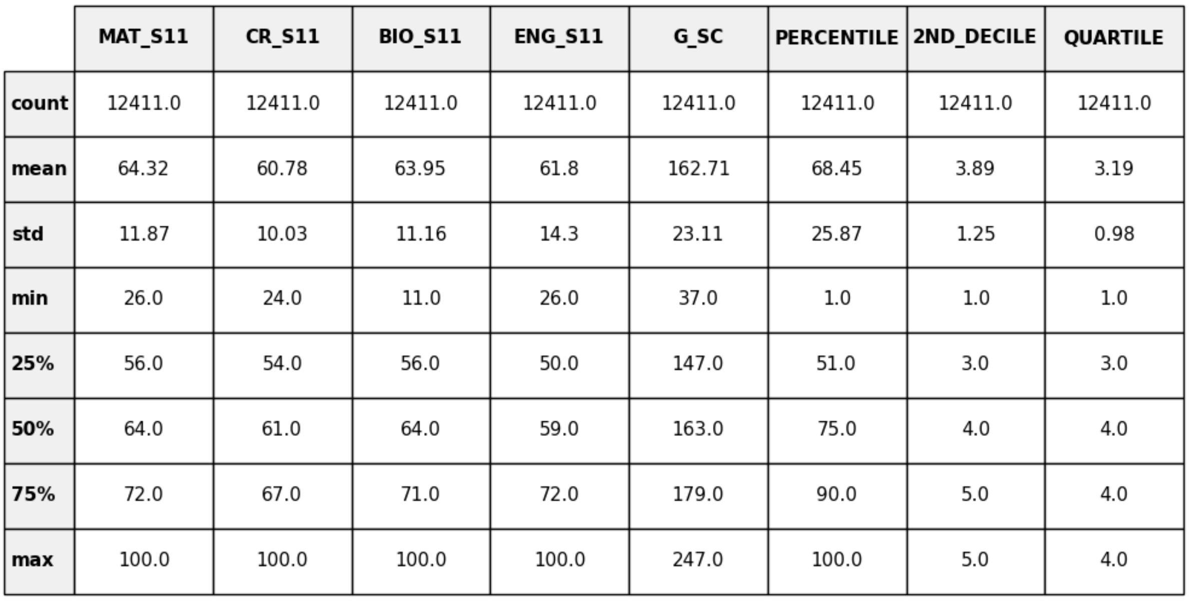
In terms of the factors related to the academic sphere, we can identify students' performance percentile, their average scores in key subjects (MAT_S11, BIO_S11, ENG_S11,…), the university and the nature of the school they attend (SCHOOL_NAT, public/private).

Regarding the personal sphere, the dataset provides information on the students' gender, income, employment status and access to technological devices and household appliances (e.g. internet connection, mobile, phone, computer, tv, dvd player, car, washing machine and microwave).

Lastly, there are also valuable contextual and family-related insights. Some examples are the educational background of the parents (EDU_FATHER, EDU_MOTHER), their occupations (OCC_FATHER, OCC_MOTHER), the socioeconomic level (STRATUM) and the number of people living in the same house (PEOPLE_HOUSE).

## 4.2 Data Refinement and Target Definition

Before proceeding to build our machine learning prediction models, we need to refine the data, in order to have a cleaner and easier analysis. Therefore, the first profiling operation is checking missing values. We are able to state that almost all fields are complete, except for an entirely empty column present in the Excel file, which has been removed since it doesn’t provide any meaningful insight. For all numerical features, descriptive statistics have been generated *(Table 1)*, in order to ensure that the values are positive and fall within reasonable ranges.



| | MAT_S11 | CR_S11 | BIO_S11 | ENG_S11 | G_SC | PERCENTILE | 2ND_DECILE | QUARTILE |
|---|---|---|---|---|---|---|---|---|
| **count** | 12411.0 | 12411.0 | 12411.0 | 12411.0 | 12411.0 | 12411.0 | 12411.0 | 12411.0 |
| **mean** | 64.32 | 60.78 | 63.95 | 61.8 | 162.71 | 68.45 | 3.89 | 3.19 |
| **std** | 11.87 | 10.03 | 11.16 | 14.3 | 23.11 | 25.87 | 1.25 | 0.98 |
| **min** | 26.0 | 24.0 | 11.0 | 26.0 | 37.0 | 1.0 | 1.0 | 1.0 |
| **25%** | 56.0 | 54.0 | 56.0 | 50.0 | 147.0 | 51.0 | 3.0 | 3.0 |
| **50%** | 64.0 | 61.0 | 64.0 | 59.0 | 163.0 | 75.0 | 4.0 | 4.0 |
| **75%** | 72.0 | 67.0 | 71.0 | 72.0 | 179.0 | 90.0 | 5.0 | 4.0 |
| **max** | 100.0 | 100.0 | 100.0 | 100.0 | 247.0 | 100.0 | 5.0 | 4.0 |

*Table 1. Descriptive statistics for the most relevant features of the dataset.*
We reported the main numerical indicators describing each one of the following factors: evaluation of the student’s skills in Mathematics (MAT_S11), Critical Reading (CR_S11), Biology (BIO_S11) and Engineering (ENG_S11); Global Score (G_SC), which considers the academic performance overall; indicators of how successful student is with respect to others (PERCENTILE, 2ND_DECILE, QUARTILE).

At this point, we ensure that the dataset doesn’t contain any problematic values that could compromise our analysis (such as 0s, Nas or similar anomalies) and, subsequently, we create ordinal encoder by manually defining a meaningful order for the entries of each categorical variable.

After developing the individual encoders, we integrate them into a unified preprocessing pipeline and we move on to define the target variable. Our objective is to predict students’ percentile scores as accurately as possible. However, since guessing the exact value of a continuous ranking is both difficult and potentially unstable, we consider two alternatives: designing a custom loss function or discretizing the percentile into broader categories. The best approach seems to be the second, so we introduce a new variable (PERFORMANCE_CLASS), which classifies students into three groups based on their original percentile score: "Low", "Medium" and "High". Also, we observe that the target behaviour *(Figure 1)*, which seems to have a quite balanced distribution, although the Medium class is slightly overrepresented (note that this is completely reasonable considering the dynamics that rule the "real" world).

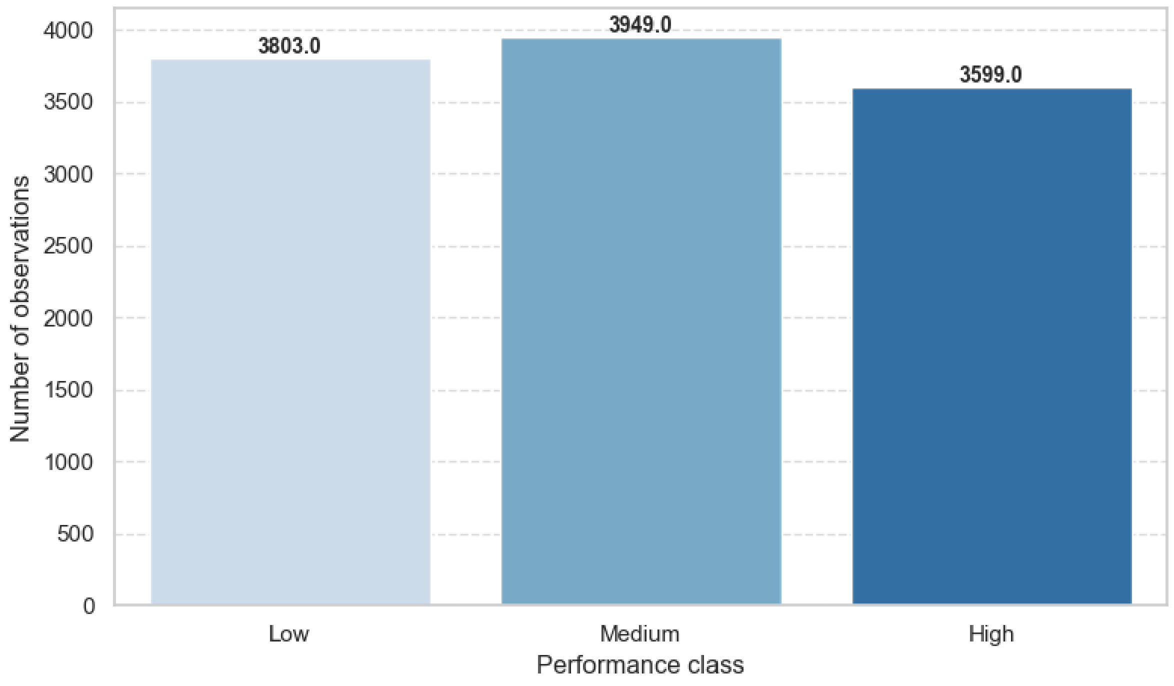


*Figure 1. Distribution of PERFORMANCE_CLASS (target)*
We plot the target variable's distribution in order to check that it's evenly balanced. The three new categories, based on the percentile value are "Low" (< 58), "Medium" (58-83) and "High" (> 83).

## 4.3 Feature Analysis and Selection

Given the variety of the indicators (both academic and social), the next step consists in selecting and actually identifying the most accurate ones. The criteria used for the variable selection operation are following a logical conceptual reasoning and choosing the more normally or evenly distributed features. To do this last thing, it's very useful to plot histograms (as done below) for each variable in order to have a better understanding of their behaviour and to eventually identify outliers *(Figure 2)*. The key observation is that the most well-distributed variables among the factors contained by the dataset are gender distribution (59% male, 41% female) and school nature (53% private, 47% public institutions) *(Figure 2a-b)*.

Looking at other variables, access to technology is obviously more common in general, with 81% of students having access to internet, 83 % having a computer at home and 71% a mobile phone.

For what concerns parental education and stratum (socioeconomic status), instead, we can observe that these are the most widely distributed factors showing an uneven distribution but still pretty interesting to study.

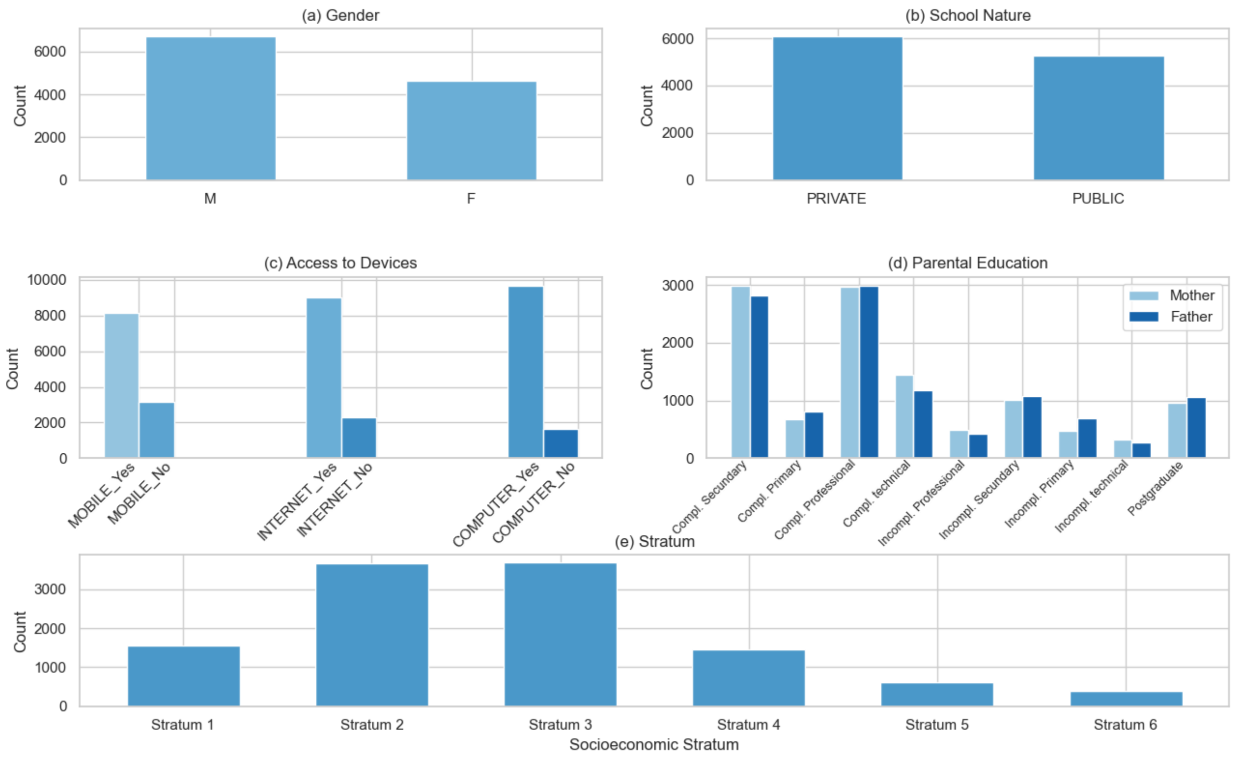


*Figure 2. Distributions of the main features*
(a) Distribution of the gender of the students in the dataset; (b) Nature of the school in which the students are enrolled; (c) Students' access to technological devices and tools, specifically mobile phone, internet and computer; (d) Education of both parents, with mother and father taken into account separately (Complete Secondary, Complete Primary, Complete Professional, Complete technical, Incomplete Professional, Incomplete Secondary, Incomplete Primary, Incomplete Technical, Postgraduate); (e) Socioeconomical situation of the students (i.e. stratum), divided in 6 categories.

# 5 Methodology and Model Development

## 5.1 Dataset Splitting and Validation Strategy

One last step before proceeding with the actual modelling is preparing the dataset for the training and testing phase. Moreover, our aim is to have to different subsets of our data in order to use one to train the models that are going to be designed and the other one to test their performance.

Because of the dataset’s nature and dimension, we choose a 80/20 proportional splitting, in such a way to have a sufficiently big batch of values to train the models, but also enough data to check their accuracy.

## 5.2 Supervised Machine Learning Models

The prediction of student performance is conducted through many trials with different classification models to see how those behave. Each one is evaluated using accuracy (total correct outputs out of all predictions made), precision (true positive outputs over all predictions), recall (true positives correctly identified over all predicted positives) and F1-score (harmonic mean of precision and recall, balancing the two in a single metric).

Building upon the paths traced by previous research in the field [10]–[12], the supervised machine learning models we choose to perform are the following: Logistic Regression (as our Baseline Model), Support Vector Machine, K-Nearest Neighbours, Naïve Bayes and Decision Tree.

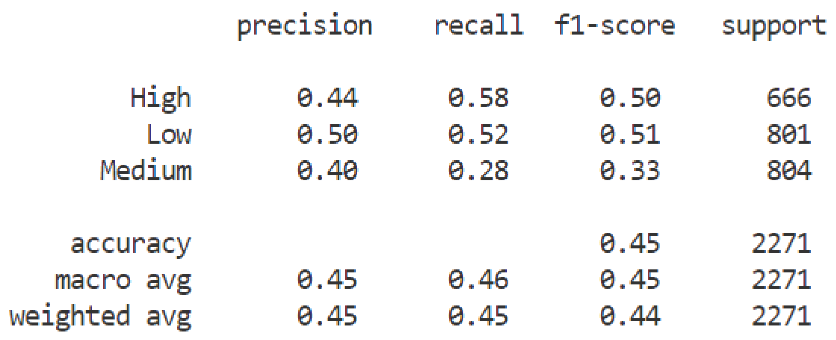
In order to better understand how we construct these prediction models, we define a systematic process to follow for each of those. This consist in creating the complete pipeline first (defining the preprocessor and the classifier), explicit the parameter grid that is going to be used to tune the model itself, find the hyperparameters that give the best performance and, lastly, evaluate the model (both through the metrics defined earlier and thanks to confusion matrices). Another tool that we’re going to use to have a better visual description of the learner’s performance is the ROC curve analysis. In this regards, since we’re not dealing with a simple binary classification task, it’s interesting to remark that it’s not possible to do it in the classical way, so, for each category (“Low”, “Medium” and “High”), we performed a One-To-Rest ROC curve plot. As a side note, there is also a focus on extracting the pivotal features in order to understand which one influence more the results and gain even more insights from our analysis.

### 5.2.1 Logistic Regression (Baseline Model)

Our starting point for the predictive modelling is the implementation of a baseline model, which we choose to be the Logistic Regression classifier for its simplicity and interpretability. Using this as a baseline means that it's useful for comparison with other more complex algorithms.

After the main steps aforementioned have been done, as preprocessing and optimization of hyperparameters, we obtain the configuration that seems to perform the best. Keeping in mind that we train the model setting, as variables, the regularization strength, the solver and the number of iterations, the best combination seems to be: strength 10 (strong regularization), meaning that the model is penalized more heavily for large coefficients; 100 maximum iterations, which is reasonable since the dataset is big so if it doesn't converge the process simply gets interrupted; l2 classifier penalty (ridge regularization, the most common when dealing with logistic regressions), that penalizes the square of the coefficients; lbfgs solver, an efficient solver for medium-sized problems.

The overall performance obtained by the best model, which can be retrieved by the classification report below *(Figure 3)*, is 45%, slightly better than a random classifier (with accuracy 1/3 ≈ 33%). We also report the confusion matrix as well as the ROC curve plot *(Figure 4 – 5)*, two clear ways to picture easily and represent graphically the performance of this first classifier.



```
              precision    recall  f1-score   support

        High       0.44      0.58      0.50       666
         Low       0.50      0.52      0.51       801
      Medium       0.40      0.28      0.33       804

    accuracy                           0.45      2271
   macro avg       0.45      0.46      0.45      2271
weighted avg       0.45      0.45      0.44      2271
```

*Figure 3. Classification report (Logistic Regression)*
This specific tool shows how the model performs throughout the various categories, thanks to the precision value, and describes generally the classifier through the other metrics we can display.

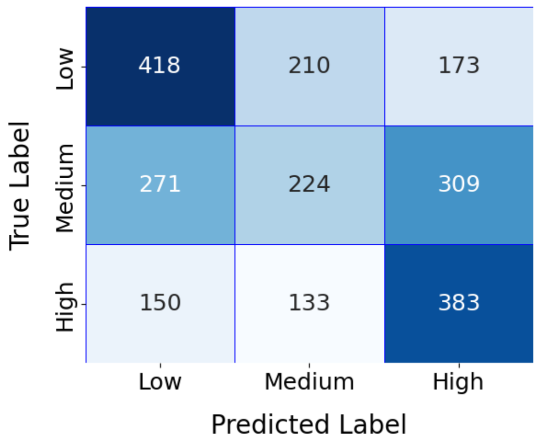


*Figure 4. Confusion Matrix (Logistic Regression)*
The darkest the color is, the better the performance is. Indeed, the logistic classifier happens to be more precise overall handling low and high scores.

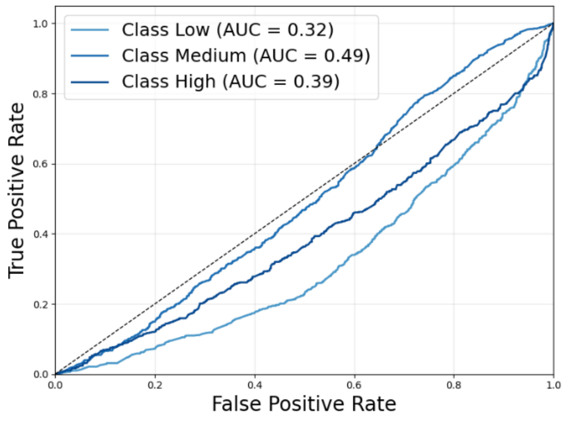


*Figure 5. One-To-Rest ROC Curve Analysis (Logistic Regression)*
The three values corresponding to the AUCs are not optimal but still better than the random guess (which would return 0.33).

As preannounced in the previous section, we also perform a feature importance analysis *(Figure 6)*, to obtain more information about the underlying reasoning for our objective. Since we are categorizing the students in three different classes, our feature analysis is going to be separate for each one. It's very interesting to observe that stratum (the socioeconomic situation of the students) is way more relevant in the chances to have success academically speaking, which could make sense since better economic positioning often results in greater possibilities and education, which subsequently leads to a better preparation and, as such, higher grades. If we shift our focus to unsuccessful students, instead, it appears that being enrolled in a public school and belonging to a low stratum group increase the chances to have a low percentile score, which is consistent with the feature importance of the High category.

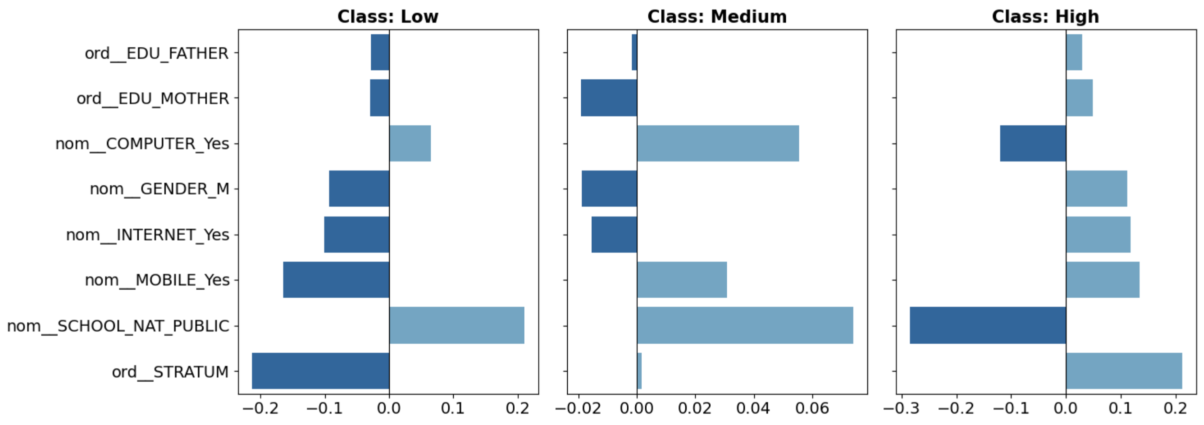


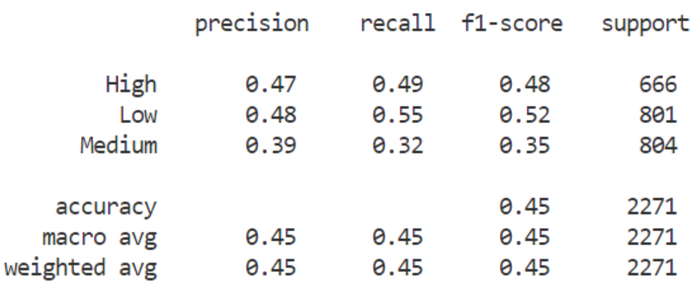
*Figure 6. Features analysis per class (Logistic Regression)*
The dark histograms correspond to a negative correlation, while the light ones give a positive one.

### 5.2.2 Support Vector Machine (SVM)

This second technique is considerable as a more advanced model, since it’s able to handle non-linear decision boundaries [13]. We still follow the same protocol, with an updated GridSearch adapted to our new model.

Moreover, we experiment with both linear and radial basis function (RBF) kernels and the kernel coefficient (gamma), while still varying the regularization parameter as before. The coefficients that give us the best results are: regularization strength 1 (intermediate level), equilibrated between underfitting and overfitting; “auto” coefficient gamma (value: 1 / number of features), which means that every datapoint has a heavy influence on the model; RBF kernel, which allows the model to find curved decision boundaries (good for non-linearly separable and more complex data). Additionally, note that since the best model is characterized by a non-linear structure, the feature importance is no more meaningful and, as such, we avoid that part of the analysis.

Speaking of evaluating the learner itself, the accuracy of the model is retrievable from the classification report below *(Figure 7)* and it’s not outstanding with respect to the previous one. Indeed, we obtain the same performance, while having a better precision when it comes to higher percentile scores. In addition to that, the confusion matrix reported *(Figure 8)* confirms the results of the report, showing that the precision is consistently higher for the “Low” category (443 correct guesses over 919 samples classified as such during the prediction phase).



```
              precision    recall  f1-score   support

        High       0.47      0.49      0.48       666
         Low       0.48      0.55      0.52       801
      Medium       0.39      0.32      0.35       804

    accuracy                           0.45      2271
   macro avg       0.45      0.45      0.45      2271
weighted avg       0.45      0.45      0.45      2271
```

*Figure 7. Classification Report (SVM)*
The best precision registered is the one reached with the lower class, as also *Figure 8* (on the right) shows.

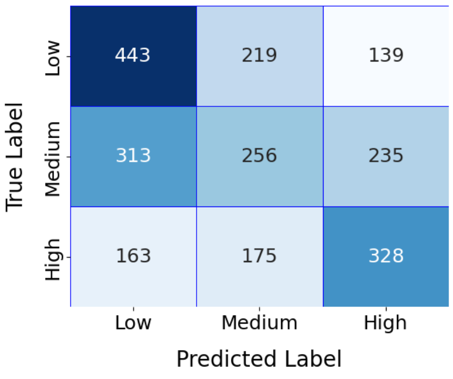


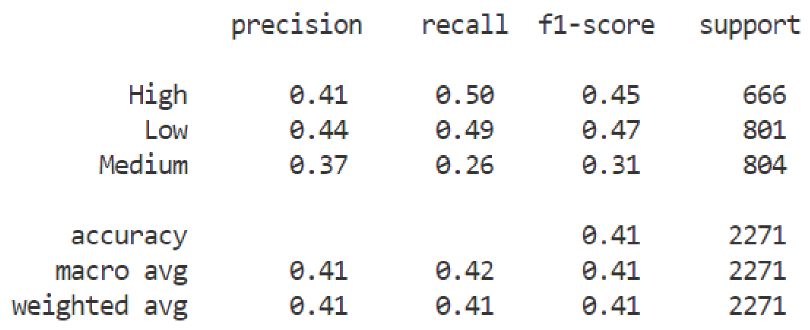
*Figure 8. Confusion Matrix (SVM)*

### 5.2.3 K-Nearest Neighbours

The third model we want to build is K-Nearest Neighbours (KNN), an non-parametric model that is able to classify datapoints based on proximity in the feature space.

The grid used to tune the hyperparameter needs to be adapted again to our new algorithm and we choose to the following variables: number of neighbours, weighting schemes and distance metrics.
All the hyperparameters' definitions are pretty straightforward, but the one regarding the measurement of distance deserves a closer look. In particular, the two possible metrics taken into account are Euclidean, which measures the straight-line distance between two points (in two-dimensional space, the classical square root), and Manhattan distance (or city block distance), that calculates the sum of the absolute differences across dimensions.

As you can see in the classification report *(Figure 9)*, the best accuracy reached is unfortunately 41%. The final hyperparameters are: 10 neighbours, which was the highest among the possible values in the grid; uniform weights, completely reasonable to give even importance; Manhattan metric, which makes the model more robust to outliers.



```
              precision    recall  f1-score   support

        High       0.41      0.50      0.45       666
         Low       0.44      0.49      0.47       801
      Medium       0.37      0.26      0.31       804

    accuracy                           0.41      2271
   macro avg       0.41      0.42      0.41      2271
weighted avg       0.41      0.41      0.41      2271
```

*Figure 9. Classification report (KNN)*
Comparing this report with the previous one, we can state that the performance of this last learner is definitely worse than the previous one (both focusing at the precision for each category and looking at the overall accuracy).

Since we're working with the same usual protocol, defined in the first place before we started to build all the various classifiers, we proceed by graphically displaying both the confusion matrix and the ROC curve, still adapting the technique to multiple categories *(Figure 10 – 11)*.

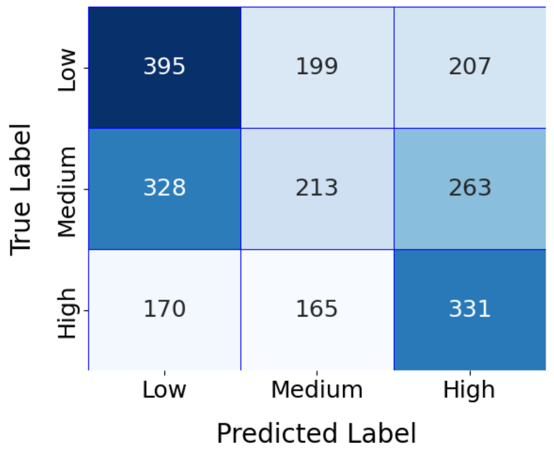


*Figure 10. Confusion Matrix (KNN)*
The highest counter of correct predictions seems to be lower scores as previous cases (with 395 correctly identified samples).

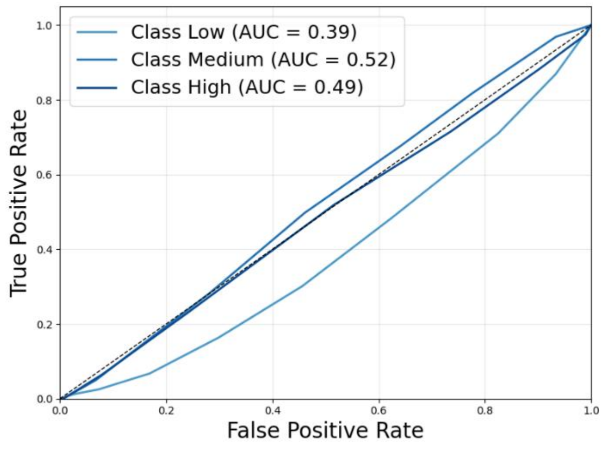


*Figure 11. One-To-Rest ROC Curve Analysis (KNN)*
In this case, the medium's class AUC reaches 0.52, closely followed by the other two categories.

Looking at the feature importance analysis *(Figure 12)*, conducted separately per class as before, the most relevant factor seems to be the stratum to which the student belongs. However, all the weights are close to zero, implying that no variable has a meaningful impact on the classification. The reason why this situation is encountered is that KNN doesn't learn weights or coefficients and the technique chosen, Permutation Feature Importance, gives just an estimated relevance by measuring how much model performance drops when a feature is randomly shuffled.

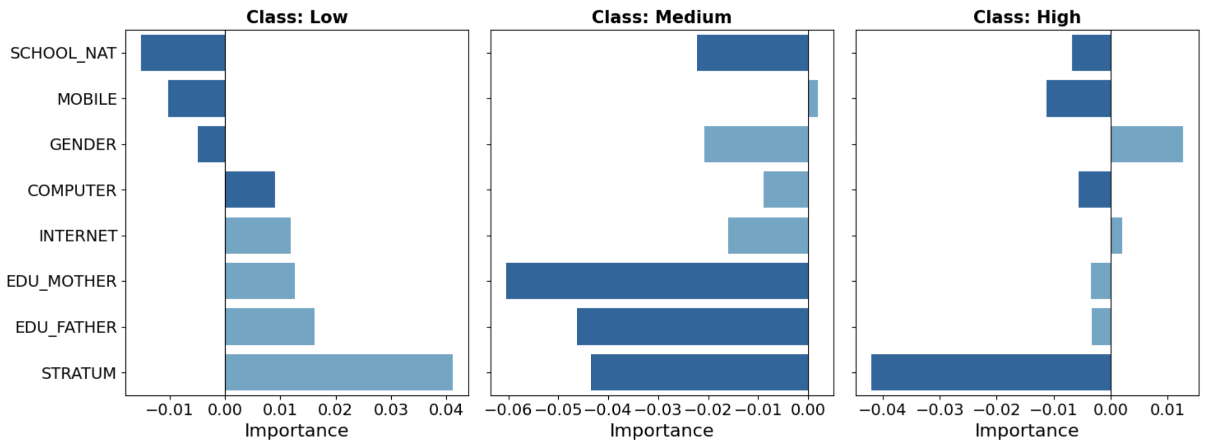


*Figure 12. Feature analysis per class (KNN)*
One can easily observe that stratum influencing negatively the percentile score results in high relevance for the unsuccessful students and, as a consequence, negative impact on successful ones. Still, all the features have a small impact on the results, being not that meaningful for the analysis.

### 5.2.4 Naïve Bayes

Among all the approaches we're pursuing, we consider also a probabilistic method, Naïve Bayes, which assumes conditional independence between features. Despite the unrealistic assumptions on which this algorithm is founded, it often performs surprisingly well when having to classify data.

The Naïve Bayes model has also a great advantage of being particularly fast and computationally efficient, at the expense of prediction accuracy. In fact, even after having found the best hyperparameters, as you can see in the image below *(Figure 13)*, the performance is 42%.

```
              precision    recall  f1-score   support

        High       0.39      0.67      0.50       666
         Low       0.49      0.50      0.49       801
      Medium       0.35      0.14      0.20       804

    accuracy                           0.42      2271
   macro avg       0.41      0.44      0.40      2271
weighted avg       0.41      0.42      0.39      2271
```

*Figure 13. Classification report (Naïve Bayes)*
The report it's sufficiently consistent with the previous ones we saw, not bringing anything new to the study.

Additionally, this model doesn't excel in any of the three categories. Indeed, this insight can be easily retrieved from both the confusion matrix and the ROC curve analysis *(Figure 14 – 15),* which shows that the area under the curve (AUC) is less than 0.5 for all the categories (poor output accuracy).

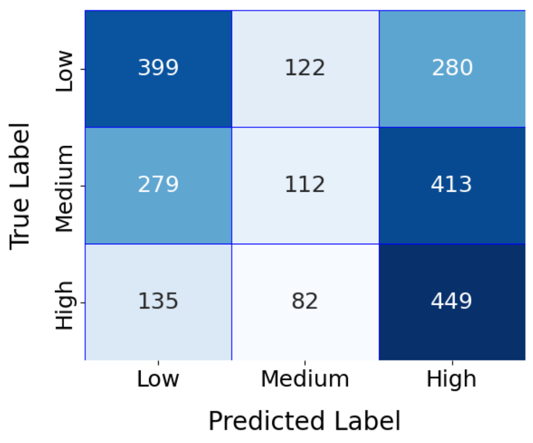


*Figure 14. Confusion Matrix (Naïve Bayes)*
In this case, differently from the others, the most precise predictions are the ones regarding high scores (and no more low ones).

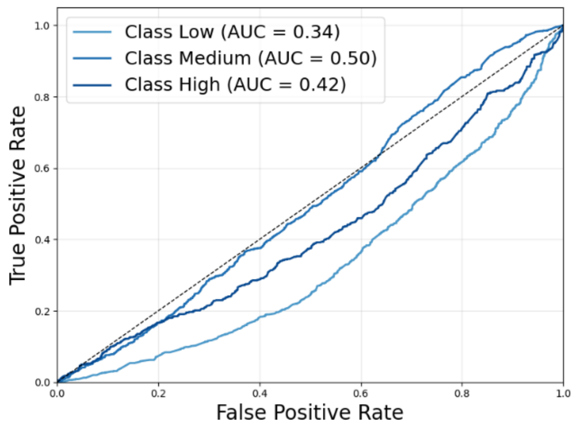


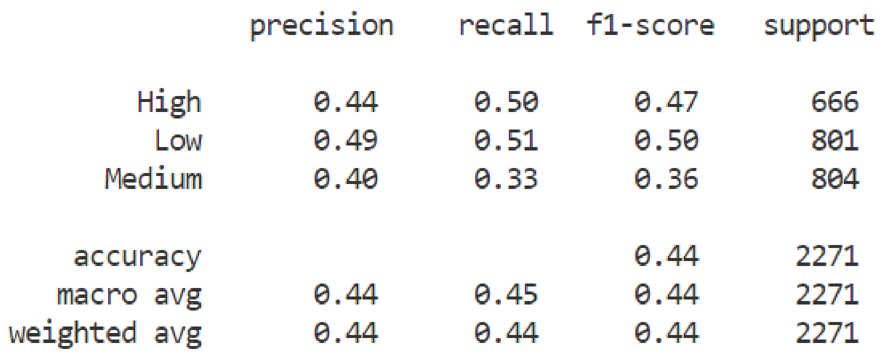
*Figure 15. One-To-Rest ROC Curve Analysis (Naïve Bayes)*

Since this learner assumes probabilistic independence between the factors considered, then importance analysis is no more applicable. In fact, isolating the variables, so considering them one at a time, will lead to not having to focus on feature relevance.

### 5.2.5 Decision Tree

The last supervised machine learning model is the Decision Tree classifier, worth of mention because of its interpretability and ability to capture non-linear relationship. The reason why we considered also this tool is its adaptability to both categorical and numerical variables at the same time and without requiring any type of scaling, still providing a precise analysis of the relevance of each feature [14]–[15]. As always, after having defined the pipeline, we create the parameters grid in order to have it set for the tuning. Through the usual GridSearch technique, we aim to find the maximum depth of the tree, the minimum number of samples required to split an internal node and the minimum number of samples that must be present in a leaf node, that helps to control the size of the tree and prevent splits that result in too small leaves (which may lead to overfitting).

After having tuned the hyperparameters, we check the evaluation metrics *(Figure 16 – 18)* and see that the accuracy is now 44%, definitely closer to the one of our Baseline Model.



```
              precision    recall  f1-score   support

        High       0.44      0.50      0.47       666
         Low       0.49      0.51      0.50       801
      Medium       0.40      0.33      0.36       804

    accuracy                           0.44      2271
   macro avg       0.44      0.45      0.44      2271
weighted avg       0.44      0.44      0.44      2271
```

*Figure 16. Classification report (Decision Tree)*
Other than the usual precision results, still in the 40-55% range, here we have to underline the good recall registered especially in the two extremal cases.

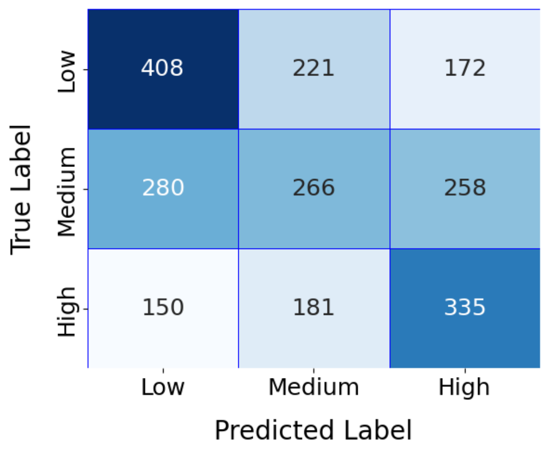


*Figure 17. Confusion Matrix (Decision Tree)*

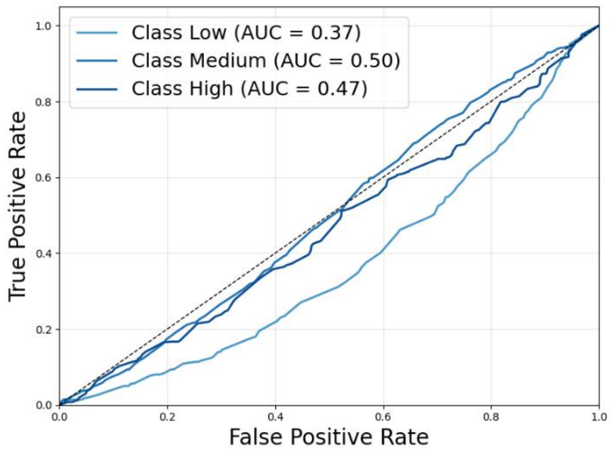


*Figure 18. One-To-Rest ROC Curve Analysis (Decision Tree)*

What is worth of mentioning in this case about the features is that the stratum feature, which already was pretty relevant in the other models (ca. 21%), now it's even more important since it reaches 32% *(Figure 19)*.

In addition to that, before building the Decision Tree the only features that affected significantly the classification output were stratum and the nature of the school. In this case, instead, there are two other factors with almost 1/5 of the weight and those are education of the father and of the mother (with the first one being slightly higher than the second one).

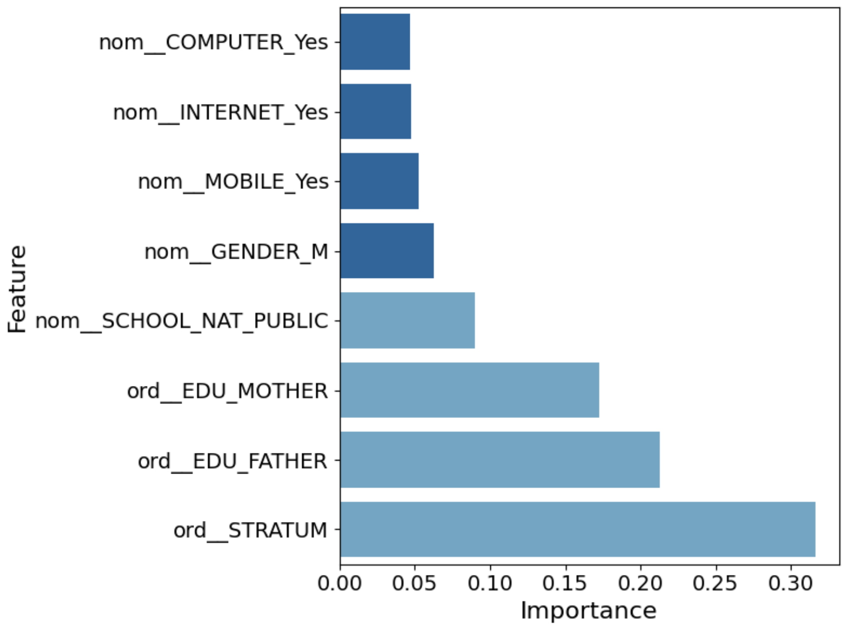


*Figure 19. Feature analysis (Decision Tree)*
With stratum being still the most influent variable, a novelty here is that the education of the parents seems to have a great impact as well.

## 5.3 Ensemble Learning Approaches

The ensemble learning technique is still part of the process of building a machine learning model that's able to predict the outcome, but this time, instead of creating a new one from scratch, we simply combine multiple pre-existing learners. This approach has been adopted by several studies to improve accuracy and produce better predictions [16]. The reason why this could work, is because several learners support each other's predictions and, at the same time, correct each other's errors, resulting in stronger and more accurate outcomes.

For the sake of completeness, we might just add a few notes on terminology. If the new model created is defined to be an *ensemble learner*, the models combined together to build it are the *base learners* [17]. In our case, our base learners are all weak, which means that they perform little better than random guessing (≈ 33%, since the target could take three values).

Now that the definition has been explained, we focus on the possible types of ensemble models. Generally, literature has widely categorized it into *parallel* and *sequential*.
The first groups includes methods that, as the name suggests, train each base learner in parallel, so independent of one another. Parallel models are further subdivided in homogeneous, that use the same base algorithm to produce all of the component base models, and heterogeneous, which differentiate to the previous ones because of their use of different algorithms instead.
The sequential methods, instead, train a new base model in order to minimize errors made by the previous model trained in the preceding step. Essentially, these learners construct base models sequentially in stages.

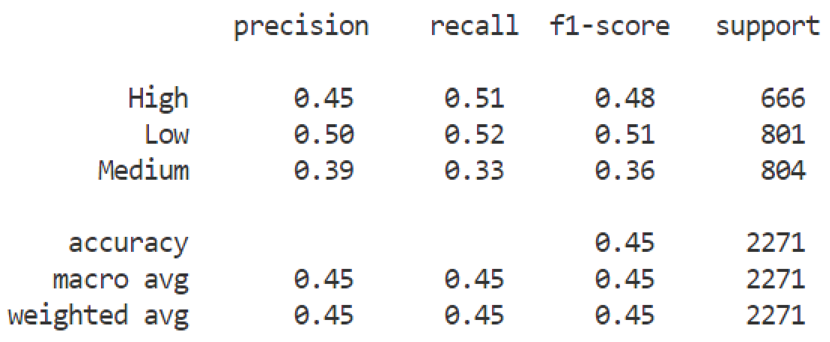
The question is now: how do ensemble methods combine base learners? There are four main techniques: bagging, boosting, stacking and voting. We will focus on each method one by one, giving a general definition first, implementing it in python and lastly analysing and comparing the results.

### 5.3.1 Bagging: Random Forest

The first option is bagging, also called bootstrap aggregating. It's a homogeneous parallel method that uses modified replicates of a given training dataset to train multiple base models with the same algorithm (this follow by definition of homogeneous given before).

This technique consists of three phases: bootstrap resampling, base learners' training, predictions' aggregation. Explaining further, the data is divided and its subset are copied in such a way to obtain new subsample dataset, with a different frequency of the instances with respect to the original configuration. Iterating this process, we have several bootstrapped samples, subsequently used to train the learners and obtain individual predictions. With the aggregation phase, we have, as the final result, a single ensemble output.

Among all the existing bagging models, the most frequently used is definitely Random Forest. This specific machine learning model is the application of the theory we've just explained, since resembles those phases using randomized decision trees as base learners. Thus, we build it with the hope to overcome our previous best results. We generate the usual classification report *(Figure 20)* and we're able to assess that the accuracy of the tuned Random Forest model reaches 45%, a great but unfortunately not outstanding result. It's still worth of mention that the combining multiple random decision trees has slightly increased, even if not significantly, the overall performance.



```
              precision    recall  f1-score   support

        High       0.45      0.51      0.48       666
         Low       0.50      0.52      0.51       801
      Medium       0.39      0.33      0.36       804

    accuracy                           0.45      2271
   macro avg       0.45      0.45      0.45      2271
weighted avg       0.45      0.45      0.45      2271
```

*Figure 20. Classification Report (Random Forest)*
This ensemble method is a combination of decision trees, which makes reasonable the results present in the report. Moreover, all the metrics are in the same range registered previously.

In addition to that, this time we register a better prediction accuracy for the two extremal groups ("Low" and "High") at the expenses of the intermediate one ("Medium") *(Figure 21 – 22)*.

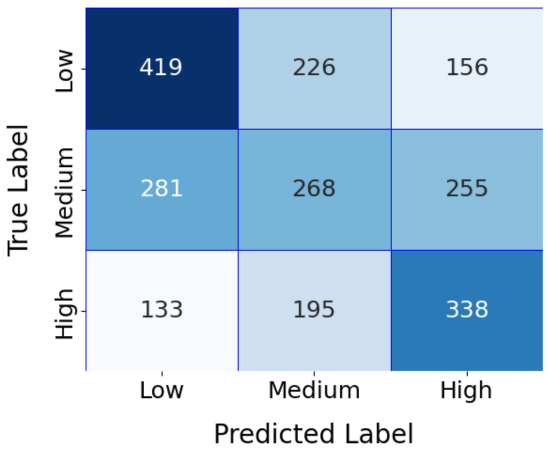


*Figure 21. Confusion Matrix (Random Forest)*
This representation resembles closely the behaviour observed by the decision tree learner.

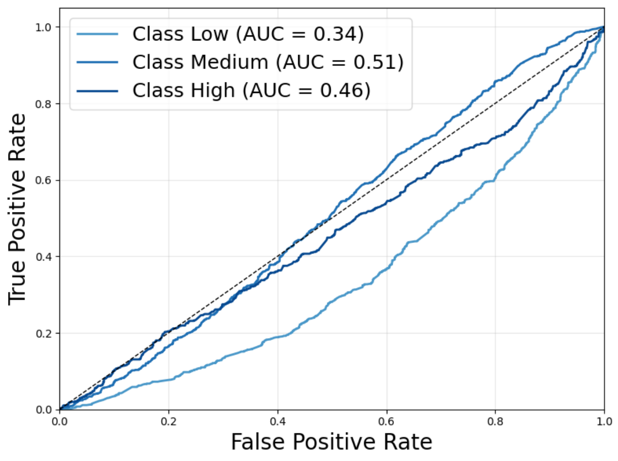


*Figure 22. One-To-Rest ROC Curve Analysis (Random Forest)*

For what concerns the most important features *(Figure 23)*, something unexpected happens. Instead of encountering a heavy impact of the stratum variable and the other ones being mostly irrelevant, here the education of the mother seems to be the most important one, with 23.79% of the prediction being based on that. The factors that closely follow it are the stratum (22.68%) and the education of the father (22.29%).

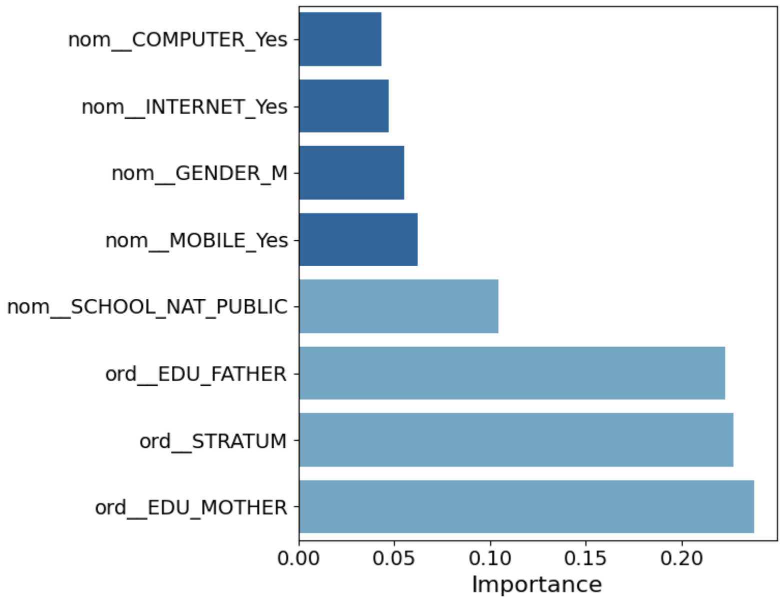
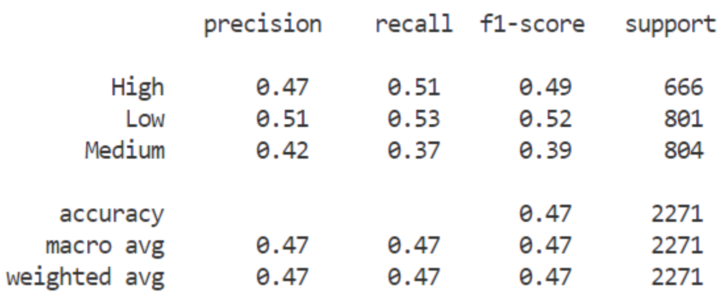


*Figure 23. Feature analysis (Random Forest)*
This model is particularly interesting, since we can see that the leading variable now is the education of the mother.

### 5.3.2 Boosting: Gradient Boosting, AdaBoost and CatBoost

Boosting is another common technique that trains a learner on some initial dataset, sampling the instances contained in it, as before, to create a new one. The difference with the previous ensemble method is the prioritization of errors and misclassifications. In specific, at each iteration a new dataset is compiled to train every time a new different model. The last step of the process is the aggregation as for the bagging. Therefore, boosting combines and weights all the learners together to produce final predictions.

There are a few popular specific boosting algorithms we're going to test. Starting with Gradient Boosting, it inverts the perspective, using residual errors to set target predictions for the next model instead of weighting misclassified samples. The performance we obtain is actually the best one so far: 47% *(Figure 24)*. Thanks to the one-to-rest ROC curve plot, supported by the confusion matrix *(Figure 25 – 26)*, we observe that this model has a particularly outstanding precision when it comes to predicting the academic performance of students with low percentile scores.



```
              precision    recall  f1-score   support

        High       0.47      0.51      0.49       666
         Low       0.51      0.53      0.52       801
      Medium       0.42      0.37      0.39       804

    accuracy                           0.47      2271
   macro avg       0.47      0.47      0.47      2271
weighted avg       0.47      0.47      0.47      2271
```

*Figure 24. Classification report (Gradient Boosting)*

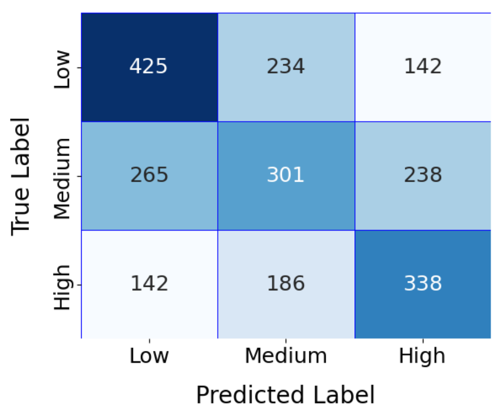


*Figure 25. Confusion Matrix (Gradient Boosting)*

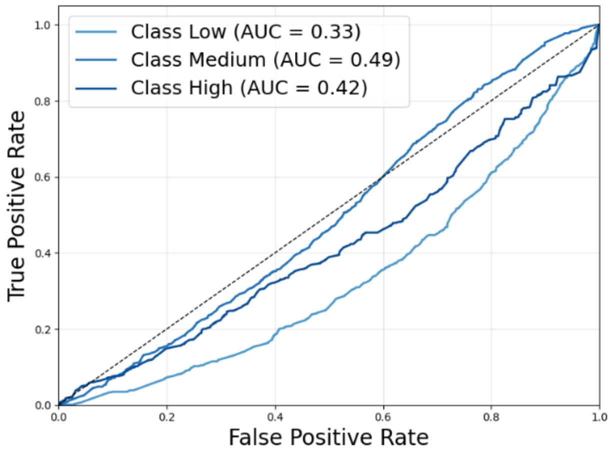


*Figure 26. One-To-Rest ROC Curve Analysis (Gradient Boosting)*

Another exceptional result is the feature importance distribution *(Figure 27)*. In fact, the stratum variable is by far the most relevant one, with a 61.98%, followed with 13.93% and 12.19% respectively by the school being public and education of the mother.

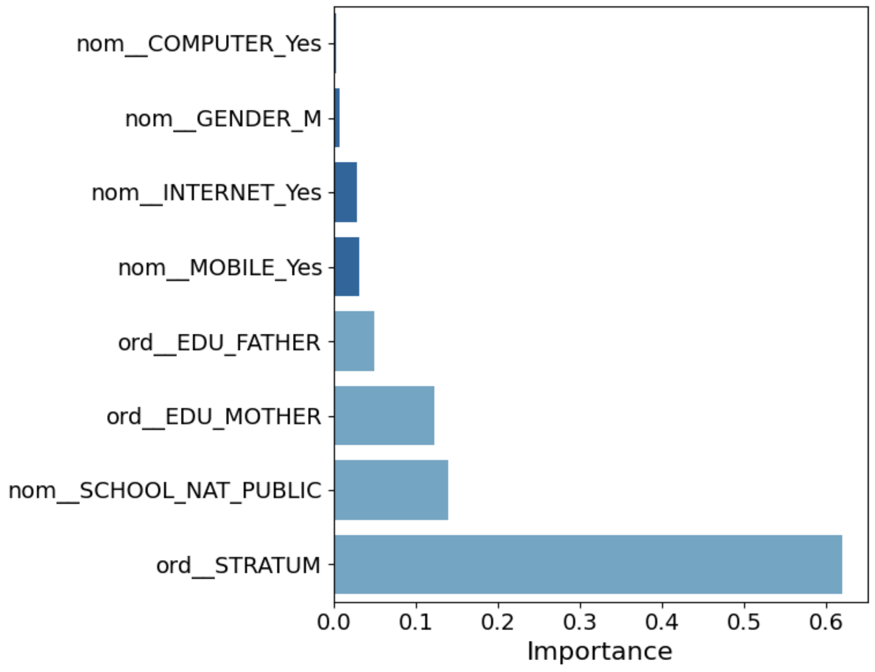
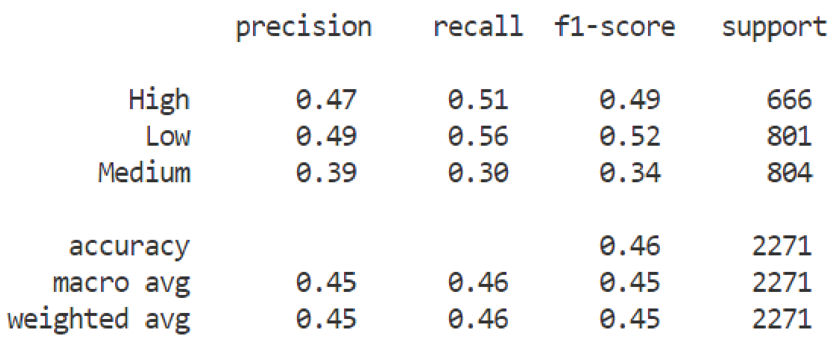


*Figure 27. Feature analysis (Gradient Boosting)*
As we can see from the feature importance graphical representation, the stratum is by far the most relevant variable with a 60% weight.

Then, we have Adaptive boosting (AdaBoost), which directly weights model errors. As previously mentioned, what this algorithm does is adding weights to the previous learner's misclassified samples, at each iteration, implying a prioritization of those by the next learner. When comparing with Gradient Boosting, we register a slightly worse accuracy (46%) *(Figure 28)*, being less precise for all the possible outputs, but still showing the same behaviour while assigning each sample to one of the three categories *(Figure 29 – 30)*.



```
              precision    recall  f1-score   support

        High       0.47      0.51      0.49       666
         Low       0.49      0.56      0.52       801
      Medium       0.39      0.30      0.34       804

    accuracy                           0.46      2271
   macro avg       0.45      0.46      0.45      2271
weighted avg       0.45      0.46      0.45      2271
```

*Figure 28. Classification report (AdaBoost)*

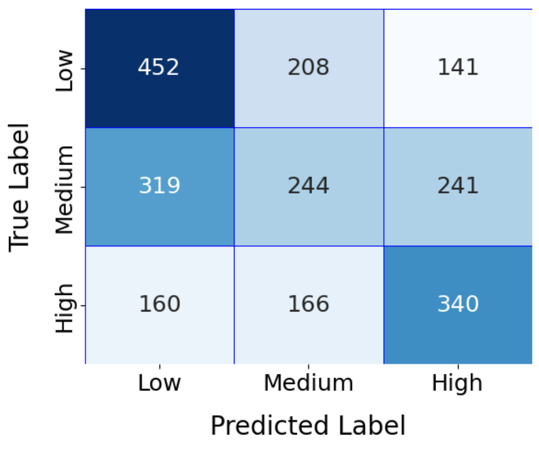


*Figure 29. Confusion Matrix (AdaBoost)*

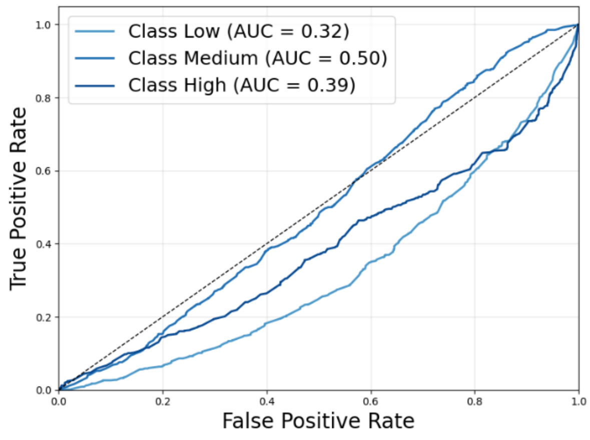


*Figure 30. One-To-Rest ROC Curve Analysis (AdaBoost)*

Here, as reported below *(Figure 31)*, the prediction is affected almost evenly by the following features: socioeconomic condition (26.12%), education of the father (21.84%), education of the mother (21.64%) and being enrolled in a public school (21.18%).

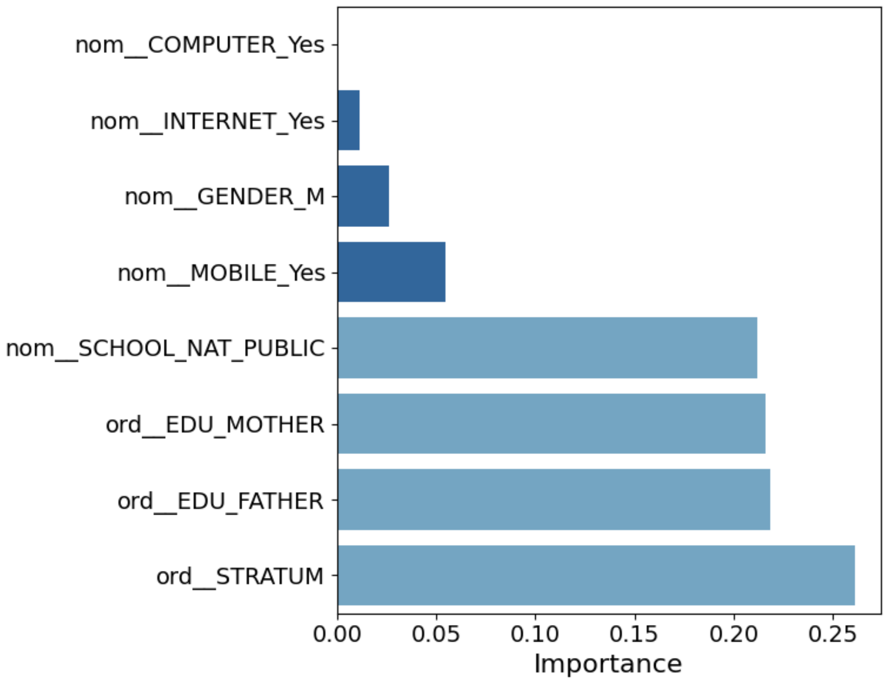
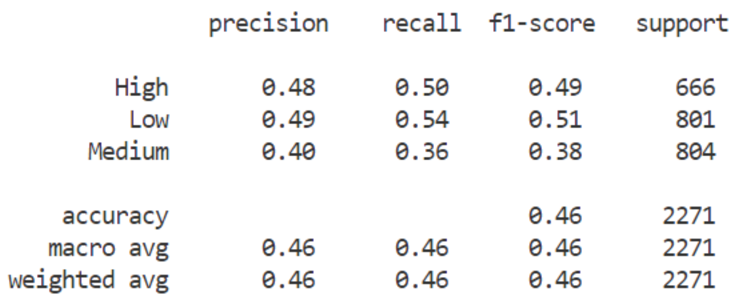


*Figure 31. Feature analysis (AdaBoost)*
While having a computer seems to be completely irrelevant for our study (not logically reasonable), stratum, education of the parents and the nature of the school have a major impact on our predictions.

Lastly, we study Category Boosting (CatBoost). This is an open-source gradient boosting library that, as the name suggests, has been designed to work with categorical variables [18]. This means that there is no need to convert these variables in numerical ones through sophisticated techniques

such as one-hot encoding, since CatBoost can easily process them natively. The working principle is the same explained before for Gradient Boosting, with a few innovations. Moreover, other than the particular ability to handle categorical features, CatBoost benefits from using an ordered boosting, which simplifies the analysis. In addition to that, the main difference with traditional gradient boosting methods, is that it doesn't build the trees leaf-wise or depth-wise, but actually structures them in order to obtain balanced trees. This ensures that all leaf nodes at a certain level share the same decision rule, leading to a faster execution and less overfitting.
The CatBoost model is not able to perform better than the previous ensemble approaches, being correct in 46% of the cases *(Figure 32 – 34)*.



```
              precision    recall  f1-score   support

        High       0.48      0.50      0.49       666
         Low       0.49      0.54      0.51       801
      Medium       0.40      0.36      0.38       804

    accuracy                           0.46      2271
   macro avg       0.46      0.46      0.46      2271
weighted avg       0.46      0.46      0.46      2271
```

*Figure 32. Classification report (CatBoost)*
High and Low class have similar precision and the model seems to be quiet good at distinguishing these two cases, while struggling with the intermediate values (Medium class).

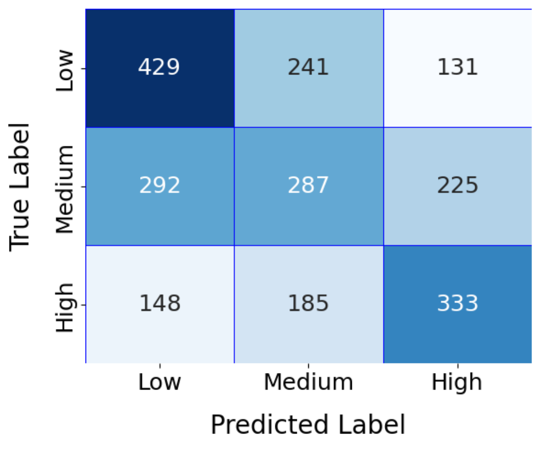


*Figure 33. Confusion Matrix (CatBoost)*

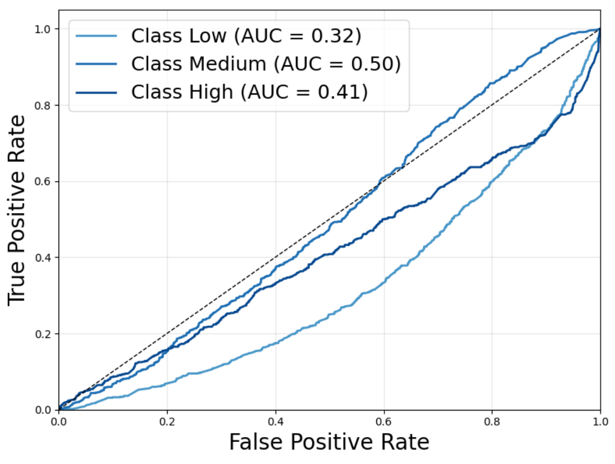


*Figure 34. One-To-Rest ROC Curve Analysis (CatBoost)*

Also the analysis of the features *(Figure 35)* doesn't provide us new information. Moreover, the most relevant variable is still the socioeconomic situation of the student with a 34.72% of weight, followed by the school being public (20.87%) and the education of the mother (17.41%).

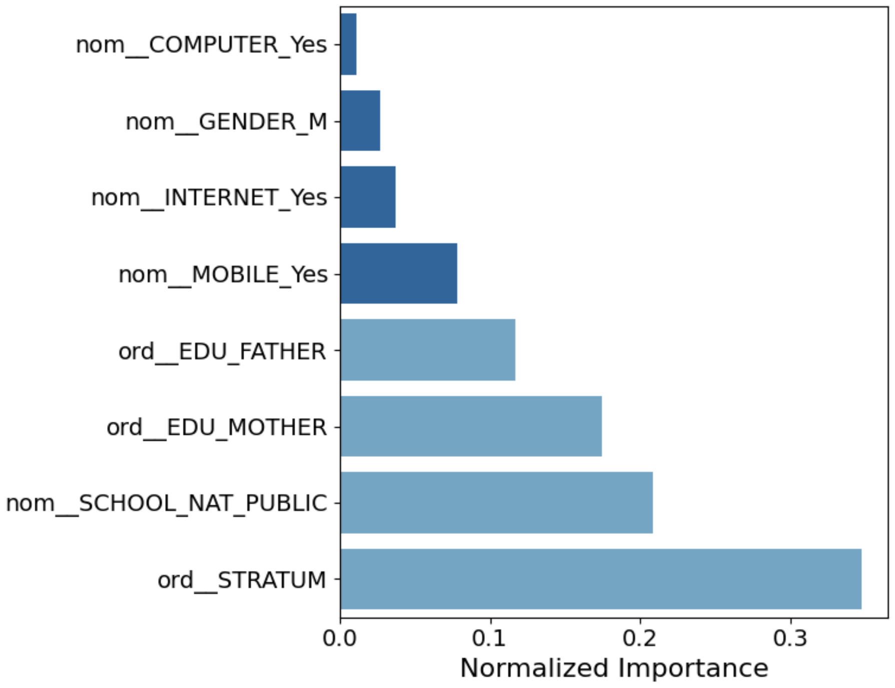


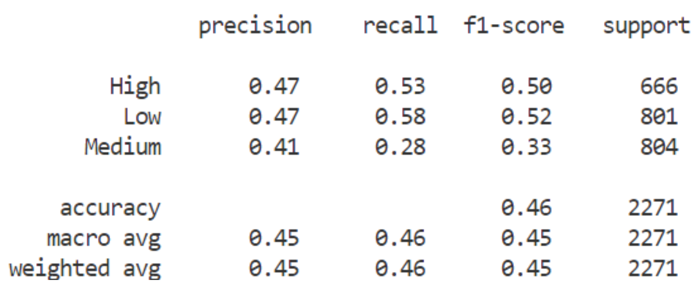
*Figure 35. Feature analysis (CatBoost)*
The nature of the school gains importance with respect to the previous models, while the top four features remain still the ones related to the socioeconomic situation, the school's type and the education of the parents.

### 5.3.3 Stacking

The stacked generalization ensemble method is a learning technique that, as such, combines multiple machine learning models through a meta-learner, which exploits the differences between the pattern followed by those models. The aim is, as always, to improve the predictions and make less mistakes possible. The difference with the previous methods is that this time a second-level learner is used in order to find the best combination of models, instead of following predefined rules.

To implement the stacking, we select three of the most accurate (but also not too complex) models: Gradient Boosting, AdaBoost and Random Forest. Each one of those was configured with 100 estimators and a fixed random state to ensure reproducibility. In addition to that, we need to choose the meta-learner, which needs to be simple, well-performing and effective, so the main eye falls on our baseline model (Logistic Regression).
After these details have been defined, we train the classifier using 5-fold cross-validation and elaborate the corresponding preprocessing pipeline.

As usual, we evaluate the model's performance on the test set. The results, which can be seen in the classification report *(Figure 36)*, show a good accuracy (46%). Also the confusion matrix *(Figure 37)* doesn't add anything new to our previous considerations, since it still shows a high precision when it comes to predicting low outcomes and a poor performance while dealing with the medium class (higher shades).



```
              precision    recall  f1-score   support

        High       0.47      0.53      0.50       666
         Low       0.47      0.58      0.52       801
      Medium       0.41      0.28      0.33       804

    accuracy                           0.46      2271
   macro avg       0.45      0.46      0.45      2271
weighted avg       0.45      0.46      0.45      2271
```

*Figure 36. Classification report (Stacking)*
The precision of the classifier on students with medium scores is not satisfying, while the overall accuracy still being acceptable.

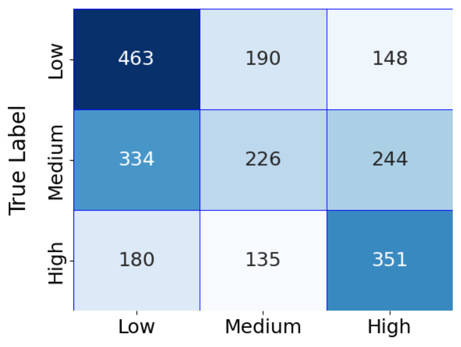


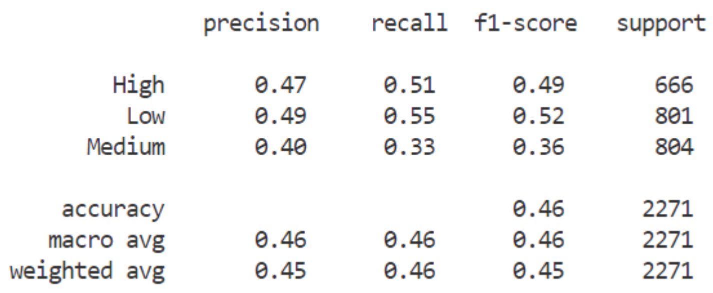
*Figure 37. Confusion Matrix (Stacking)*
The most correctly predicted samples are, as in the majority of the models, the students with low percentile scores.

### 5.3.4 Voting: Majority Vote

As the last category, we build a voting classifier. This model simply combines the predictions of the chosen base models. In a hard voting setting like this, each classifier votes for a specific class label and, in the end, the overall prediction is the one with the majority of the votes. Because of how the voting procedure is defined, this approach brings the great advantage of reducing the risk of over and underfitting, resulting in a stronger model.

To build the best learner possible, we combine the most accurate tuned models found previously in our analysis (all with 47% performance): Gradient Boosting, Support Vector Machine and AdaBoost. Then, we define the usual pipeline that describes the preprocessing steps for these classifiers first, proceed with the training phase and, lastly, evaluate the voting ensemble model on the test set.

For this last modelling attempt, we still reach the same performance as the stacking classifier and we register the same pattern in the confusion matrix representation as well *(Figure 38 – 39)*. Moreover, even though the this insight doesn't bring anything new, the results are actually surprising since the accuracy of the three chosen learners together seems to be worse than each of them separately, which is obviously not what we would expect from such an ensemble approach (that, obviously, aims to make the models stronger and not weaker as in this case).



```
              precision    recall  f1-score   support

        High       0.47      0.51      0.49       666
         Low       0.49      0.55      0.52       801
      Medium       0.40      0.33      0.36       804

    accuracy                           0.46      2271
   macro avg       0.46      0.46      0.46      2271
weighted avg       0.45      0.46      0.45      2271
```

*Figure 38. Classification report (Voting)*
The recall is high for the extreme cases and definitely lower for the medium class. The accuracy of this model aligns perfectly with earlier findings.

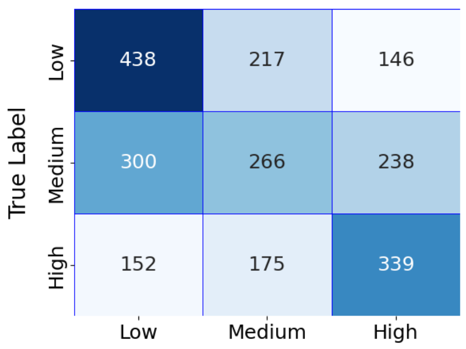


*Figure 39. Confusion Matrix (Voting)*
The majority of low scores and, after those, the high ones have been detected correctly.

# 6 Summary of Findings and Conclusions

## 6.1 Comparative Analysis of predictive models

In order to provide a clear and comprehensive comparison of the models developed throughout this study, we have summarized the performance metrics for each classifier in the table below *(Table 2)*. Specifically, it reports the precision of each learner for all the performance classes we've defined in the beginning ("Low", "Medium" and "High") and the overall accuracy achieved performing that specific classifier on the test set, after the tuning phase has happened.

Subsequently, the most accurate model for each category has been marked. Therefore, we can easily notice that Gradient Boosting is the model that predicts the output correctly for most of the cases, except when it comes to high academic performance (and, as such, high percentile score) for which CatBoost slightly prevails.

| | *Precision* | | | *Overall accuracy* |
|---|---|---|---|---|
| | *Low* | *Medium* | *High* | |
| *Logistic Regression* | 50% | 40% | 44% | 45% |
| *Support Vector Machine* | 48% | 39% | 47% | 45% |
| *K-Nearest Neighbours* | 44% | 37% | 41% | 41% |
| *Naïve Bayes* | 49% | 35% | 39% | 42% |
| *Decision Tree* | 49% | 40% | 44% | 44% |
| *Random Forest* | 50% | 39% | 45% | 45% |
| *Gradient Boosting* | **51%** | **42%** | 47% | **47%** |
| *AdaBoost* | 49% | 39% | 47% | 46% |
| *CatBoost* | 49% | 40% | **48%** | 46% |
| *Stacking* | 47% | 41% | 47% | 46% |
| *Majority Voting* | 49% | 40% | 47% | 46% |

*Table 2. Classification report (Voting)*
In this table we report the precision, for each category, and the overall accuracy of each classifier we built throughout our study. A bold line divides the simple modelling and the ensemble learners, with these being subdivided in bagging, boosting, stacking and voting.
For each column, the highest accuracy corresponding to the best model has been marked in blue.

## 6.2 Interpretation and Final Reflections

The comparative analysis of the several machine learning classifiers provides valuable insights about the factors that influence students' academic success and the feasibility of adopting predictive approaches in an educational context. Among all the models, ensemble methods seem to be the most precise in assessing the performance of the students, especially for those at risk of failing in school. This behaviour shows once again the robustness that we expect from such ensemble techniques when applied to structured educational data, because of the common feature interactions and the massive presence of non-linear patterns.

Throughout the study, thanks to the feature importance analysis, we focused on the role that socioeconomic variables play in this scenario, being able to confirm what was reported in the previous literature: educational outcomes are deeply influenced not only by individual capabilities and talent but also from the family background and the environment.

At this point, one could ask what are the practical implications of what we discovered. Basically, these results suggest that educational institutions, e.g. schools and universities, could implement systems based on machine learning in order to proactively identify students at risk of failing or underperforming in general. This would open the possibility of offering more targeted and efficient interventions.
Moreover, since a lot of the classifiers studied agreed on the fact that stratum and education of the mother are two main factors that influence the students' success, then some intervention on that side could be useful. For instance, from the economic point of view subsidies and scholarships are created to offer equal opportunities to all, while from the mother's (and, in general, family's) academic support side institutions could help providing one-to-one tutoring and mentoring programs.

In conclusion, while no model is perfect, we acknowledge that this is completely reasonable, since a student's academic performance is not only a result of its socioeconomic condition but also of its persona in general (overall, the models show a half and half relation). Therefore, this study is able to demonstrate the potential of machine learning models to support educational decision-making by providing interesting insights.
Indeed, systematically comparing them still helps us to choose which approach is the most promising to reduce gap between students. This would result in providing all individuals similar opportunities and resources, trying to minimize the impact of factors that are beyond the students' control.

## 6.3 Study Limitations and Future Research

While this study provides valuable insights into the predictive factors of academic success among university students, we should also acknowledge several limitations. We will now specifically delve into four aspects regarding different spheres: data, categorization/target variable, training and modelling.

First of all, our analysis builds its reasoning on a single dataset, which may limit the generalizability of the findings to other contexts. In particular, the data describes a specific geographic area, reflecting that specific educational system, socioeconomic structures and cultural dynamics. Obviously, those factors would change how big the impact of the single features taken into account is, but definitely not the logical reasoning that lie behind our interpretation of the final results. Other than this, also the variables included in the dataset, even though are diversified, still don't cover some aspects of the students' life, such as psychological factors and attendance records. Since it's common knowledge that these are strongly related to the academic success, it's likely that having a dataset with such additional information would make predictions more accurate.

Then, another way to improve our study could be to tackle the problem of predicting the Medium class. In fact, from the various classification reports we already observed that the precision regarding that category is significantly lower than the others, so it could be interesting to minimize the amount of misclassified students belonging to that. In order to fix it we could either simply split this class in two, linking one to the lower and the other one to the higher level) or apply more sophisticated techniques like Synthetic Minority Over-sampling Technique (SMOTE), a method that balances the class distribution in the training data helping the learning phase and improving the categorization of underrepresented sections.

Another way to try to improve accuracy, focusing on the training aspect, is adopting a k-cross validation process for the whole analysis instead of just specific models. The reason why this method could be more efficient is that it gives more reliable estimates, since in the single split case the performance heavily depends on the proportion we choose. Also, the randomizing factor results in a more stable estimate and a fairer model comparison (reducing biases), since at this point each model is tested on all parts of the data.

Lastly, shifting our attention to the model itself, one could think to either simply increase the maximum number of iterations of the models already taken into consideration or build new complex learners (e.g. Neural Networks). Both approaches would certainly bring great results, but we would require higher computational cost and more time to run the code, decreasing the general efficiency of the overall process. In fact, throughout our analysis we prioritized methods that offer a good trade-off between performance and resource consumption.

# 7 Bibliography

1. *The Knowledge Capital of Nations: Education and the Economics of Growth*. E. A. Hanushek and L. Woessmann. 2015. MIT Press;
2. *Leaving College: Rethinking the Causes and Cures of Student Attrition*. V. Tinto. 1987. The University of Chicago Press;
3. *Predicting academic success: machine learning analysis of student, parental, and school efforts*. X. Jin. 2023. DOI: 10.1007/s12564-023-09915-4;
4. *Inequality of opportunity for educational achievement in Latin America: Evidence from PISA 2006-2009*. L. F. Gamboa and F. D. Waltenberg. 2012. DOI: 10.1016/j.econedurev.2012.05.002;
5. *Inequality of opportunity in China's educational outcomes*. J. Golley and S. T. Kong. 2018. DOI: 10.1016/j.chieco.2016.07.002;
6. *Socioeconomic inequality and educational outcomes: Evidence from twenty years of TIMSS*. M. Broer, Y. Bai et al. 2019. Springer;

7. *Predicting student success: Considering social and emotional skills, growth mindset, and motivation*. N. Daley, D. Murano et al. 2025. DOI: 10.1016/j.sel.2025.100080;
8. *Predicting academic success in higher education*. E. Alyahyan and D. Düştegör. 2020. DOI: 10.1186/s41239-020-0177-7;
9. *Dataset of academic performance evolution for engineering students*. E. Delahoz-Dominguez, R. Zuluaga et al. 2020. DOI: 10.1016/j.dib.2020.105537;
10. *A comparative analysis of techniques for predicting academic performance*. N. Thai-Nghe, P. Janecek et al. 2007. DOI: 10.1109/FIE.2007.4417993;
11. *Prediction of student academic performance by an application of data mining techniques*. S. Sembiring, M. Zarlist et al. 2011. IACSIT Press;
12. *A Prediction Model for Student Academic Performance Using Machine Learning*. H. Kaur, R. Garg et al. 2022. DOI: 10.31449/inf.v47il.4297;
13. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. N. Cristianini and J. Shawe-Taylor. 2013. DOI: 10.1017/CBO9780511801389;
14. *Predicting Students Final GPA Using Decision Trees: A Case Study*. M. A. Al-Barrak and M. Al-Razgan. 2016. DOI: 10.7763/IJIET.2016.V6.745;
15. *Classification and Regression Trees*. L. Breiman, J. Friedman et al. 1984. DOI: 10.1201/9781315139470;
16. *Ensemble Methods in Machine Learning*. T. G. Dietterich. 2000. DOI: 10.1007/3-540-45014-9;
17. *What is ensemble learning?* J. Murel and E. Kavlakoglu. 2024. IBM website;
18. *Understanding CatBoost: The Gradient Boosting Algorithm for Categorical Data*. A. Kolli. 2024. Medium website.

# 8 Appendix – Python code

```
### Importing the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np

### Importing the dataset
df = pd.read_excel("data_academic_performance.xlsx")

### PROFILING
## Focus on the dataset
# Number of rows and columns
n_rows, n_columns = df.shape
print(f'Number of rows: {n_rows}, Number of columns: {n_columns}')

# Type of each column
column_types = df.dtypes
print('Type of each column:\n', column_types)

# Missing values
missing_values = df.isnull().sum()
print('Missing values:\n', missing_values)
df = df.drop(columns=['Unnamed: 9'], errors='ignore')

# Descriptive statistics
print('Descriptive statistics:\n', df.describe())

desc = df.describe()
first_cols = list(desc.columns[:4])
last_cols = list(desc.columns[-4:])
dots = ['...' for _ in range(len(desc))]
desc_subset = pd.concat([desc[first_cols], pd.DataFrame({'...': dots}, index=desc.index),
desc[last_cols]], axis=1)
fig, ax = plt.subplots(figsize=(10, 8))
ax.axis('off')
ax.axis('tight')
table = ax.table(cellText=desc_subset.round(2).values, rowLabels=desc_subset.index,
colLabels=desc_subset.columns, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(11)
table.scale(1.1, 1.1)
for key, cell in table.get_celld().items():
    cell.set_height(0.07)
for i in range(len(desc_subset)+1):
    cell = table[i, 4]
    cell.get_text().set_fontstyle('italic')
    cell.get_text().set_color('gray')
    cell.set_facecolor('#f0f0f0')
```

```
plt.tight_layout()
plt.savefig("descriptive_statistics_table_elegant.png", dpi=300)
plt.show()

# Histograms describing numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns
for column in numerical_columns:
    plt.figure(figsize = (8, 4))
    sns.histplot(df[column], kde = True, bins = 30)
    plt.title(f'Graphical representation of {column}')
    plt.show()

## Focus on the variables
# Variable selection
selected_features = ['GENDER', 'EDU_FATHER', 'EDU_MOTHER', 'STRATUM', 'INTERNET',
'COMPUTER', 'MOBILE', 'SCHOOL_NAT']
for col in ['MOBILE', 'SCHOOL_NAT', 'INTERNET', 'COMPUTER', 'GENDER']:
    print(f"Distribution of {col}:\n{df[col].value_counts()}\n")

# Defining the variables
ordinal_cols = ['STRATUM', 'EDU_FATHER', 'EDU_MOTHER']
nominal_cols = ['MOBILE', 'SCHOOL_NAT', 'INTERNET', 'COMPUTER', 'GENDER']
for col in ['EDU_FATHER', 'EDU_MOTHER']:
    df = df[~((df[col] == 0) | (df[col] == 'Not sure') | (df[col] == 'Ninguno'))]
df = df[~(df['STRATUM'] == 0)]
for col in ['EDU_FATHER', 'EDU_MOTHER', 'STRATUM']:
    df[col] = df[col].astype(str).str.strip()

# Ordinal encoding
edu_order = ['Incomplete primary', 'Complete primary', 'Incomplete Secundary',
'Incomplete technical or technological', 'Incomplete Professional Education', 'Complete
Secundary', 'Complete technique or technology', 'Complete professional education',
'Postgraduate education']

stratum_order = ['Stratum 1', 'Stratum 2', 'Stratum 3', 'Stratum 4', 'Stratum 5', 'Stratum
6']

# Creating the ordinal encoder
ordinal_encoder = OrdinalEncoder(categories=[stratum_order, edu_order, edu_order])

# Creating the nominal encoder
nominal_encoder = OneHotEncoder(drop='first', sparse_output=False)

# Combining the two encoders into a single preprocessor
preprocessor = ColumnTransformer(transformers=[('ord', ordinal_encoder, ordinal_cols),
('nom', nominal_encoder, nominal_cols)])

# Defining the target variable
df['PERFORMANCE_CLASS'] = pd.qcut(df['PERCENTILE'], q=3, labels=['Low', 'Medium',
'High'])
target = 'PERFORMANCE_CLASS'
X = df[selected_features]
y = df[target]

# Representing graphically the target variable
plt.figure(figsize=(8, 5))
```

```
ax = sns.countplot(data=df, x='PERFORMANCE_CLASS', order=['Low', 'Medium', 'High'],
palette='Blues')
plt.title('Distribution of the target variable', fontsize=14, fontweight='bold')
plt.xlabel('Performance class')
plt.ylabel('Number of observations')
plt.grid(axis='y', linestyle='--', alpha=0.7)

for p in ax.patches:
   height = p.get_height()
     ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2., height), ha='center',
va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()

# Representing graphically some of the main features
import matplotlib.gridspec as gridspec
from matplotlib.cm import get_cmap
blues = get_cmap("Blues")
fig = plt.figure(figsize=(16, 40))
gs = gridspec.GridSpec(3, 2, height_ratios=[1, 1.2, 1], hspace=0.6)
# - Gender
ax1 = fig.add_subplot(gs[0, 0])
df['GENDER'].value_counts().plot(kind='bar', ax=ax1, color=blues(0.5))
ax1.set_title('(a) Gender')
ax1.set_ylabel('Count')
ax1.set_xlabel('')
ax1.set_xticklabels(df['GENDER'].value_counts().index, rotation=0, ha='center')
# - School Nature
ax2 = fig.add_subplot(gs[0, 1])
df['SCHOOL_NAT'].value_counts().plot(kind='bar', ax=ax2, color=blues(0.6))
ax2.set_title('(b) School Nature')
ax2.set_ylabel('Count')
ax2.set_xlabel('')
ax2.set_xticklabels(df['SCHOOL_NAT'].value_counts().index, rotation=0, ha='center')
# - Mobile, Internet, Computer
ax3 = fig.add_subplot(gs[1, 0])
binary_vars = ['MOBILE', 'INTERNET', 'COMPUTER']
base_colors = [0.55, 0.65, 0.75]
xticks_labels = []
xticks_positions = []
position = 0
bar_width = 0.35
for i, var in enumerate(binary_vars):
   counts = df[var].value_counts().sort_index()
   yes_color_val = max(base_colors[i] - 0.15, 0)
   yes_color = blues(yes_color_val)
   no_color = blues(base_colors[i])
   ax3.bar(position, counts.get(1, 0), width=bar_width, color=yes_color)
   ax3.bar(position + bar_width, counts.get(0, 0), width=bar_width, color=no_color)
   xticks_positions.extend([position + bar_width/2, position + 1.5*bar_width])
   xticks_labels.extend([f'{var}_Yes', f'{var}_No'])
   position += 2
ax3.set_title('(c) Access to Devices')
ax3.set_ylabel('Count')
ax3.set_xticks(xticks_positions)
ax3.set_xticklabels(xticks_labels, rotation=45, ha='right')
```

```
for tick in ax3.get_xticklabels():
    tick.set_y(0.07)
# - Edu Mother & Edu Father
ax4 = fig.add_subplot(gs[1, 1])
edu_mother_counts = df['EDU_MOTHER'].value_counts().sort_index()
edu_father_counts = df['EDU_FATHER'].value_counts().sort_index()
x = np.arange(len(edu_mother_counts.index))
width = 0.35
ax4.bar(x - width/2, edu_mother_counts.values, width, label='Mother', color=blues(0.4))
ax4.bar(x + width/2, edu_father_counts.values, width, label='Father', color=blues(0.8))
ax4.set_xticks(x)
custom_labels = ['Compl. Secundary', 'Compl. Primary', 'Compl. Professional', 'Compl.
technical', 'Incompl. Professional', 'Incompl. Secundary', 'Incompl. Primary', 'Incompl.
technical', 'Postgraduate']
ax4.set_xticklabels(custom_labels, rotation=45, ha='right', fontsize=9)
ax4.set_title('(d) Parental Education')
ax4.set_ylabel('Count')
for tick in ax4.get_xticklabels():
    tick.set_y(0.07)
ax4.legend()
# - Stratum
ax5 = fig.add_subplot(gs[2, :])
df['STRATUM'].value_counts().sort_index().plot(kind='bar', ax=ax5, color=blues(0.6),
width=0.6)
ax5.set_title('(e) Stratum')
ax5.set_ylabel('Count')
ax5.set_xlabel('Socioeconomic Stratum')
ax5.set_xticklabels(df['STRATUM'].value_counts().sort_index().index, rotation=0,
ha='center')
plt.tight_layout(rect=[0, 0, 1, 0.95], h_pad=25)
plt.show()

## Dividing the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

### MODEL BUILDING - SUPERVISED MACHINE LEARNING MODELS
## LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression

# Pipeline
pipeline_lr = Pipeline(steps=[('preprocessor', preprocessor), ('classifier',
LogisticRegression(random_state=42, max_iter=1000))])

# GridSearch
param_grid_lr = {'classifier__C': [0.01, 0.1, 1, 10], 'classifier__penalty': ['l2'],
'classifier__solver': ['lbfgs', 'liblinear'], 'classifier__max_iter': [100, 200, 500, 1000]}
grid_search_lr = GridSearchCV(pipeline_lr, param_grid_lr, cv=5, scoring='accuracy')
grid_search_lr.fit(X_train, y_train)
y_pred_lr = grid_search_lr.predict(X_test)
print("Best Parameters (Logistic Regression):", grid_search_lr.best_params_)

# Classification report
print("Classification Report (Logistic Regression):\n", classification_report(y_test,
y_pred_lr))

# Confusion matrix
class_order = ['Low', 'Medium', 'High']
```

```
cm = confusion_matrix(y_test, y_pred_lr, labels=class_order)
print("Confusion Matrix (Logistic Regression):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
classes = ['Low', 'Medium', 'High']
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]
y_score = grid_search_lr.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
from matplotlib.cm import get_cmap
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
                label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_lr = grid_search_lr.best_estimator_.named_steps['classifier']
feature_names_lr =
grid_search_lr.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
coefficients_lr = []
for class_idx, class_label in enumerate(best_model_lr.classes_):
    coeffs = best_model_lr.coef_[class_idx]
     df_tmp = pd.DataFrame({'Feature': feature_names_lr, 'Coefficient': coeffs, 'Class':
class_label})
    coefficients_lr.append(df_tmp)
coefficients_lr = pd.concat(coefficients_lr)
ordered_classes = ['Low', 'Medium', 'High']
coefficients_lr['Class'] = pd.Categorical(coefficients_lr['Class'],
categories=ordered_classes, ordered=True)
```

```
for cls in ordered_classes:
    print(f"\nTop 10 important features for class {cls}:")
    df_class = coefficients_lr[coefficients_lr['Class'] == cls]
                                                                    df_class                              =
df_class.reindex(df_class['Coefficient'].abs().sort_values(ascending=False).index)
    print(df_class[['Feature', 'Coefficient']].head(10))

top_n = 10
fig,    axes    =    plt.subplots(nrows=1,    ncols=len(ordered_classes),    figsize=(5    *
len(ordered_classes), 6), sharey=True)
for idx, cls in enumerate(ordered_classes):
    df_class = coefficients_lr[coefficients_lr['Class'] == cls]
                                                              df_class_sorted                         =
df_class.reindex(df_class['Coefficient'].abs().sort_values(ascending=True).index)
    df_top = df_class_sorted.tail(top_n)
    colors = df_top['Coefficient'].apply(lambda x: '#67a9cf' if x > 0 else '#2166ac').tolist()
    sns.barplot(x='Coefficient', y='Feature', data=df_top, palette=colors,ax=axes[idx])
    axes[idx].axvline(x=0, color='black', linewidth=0.8)
    axes[idx].set_title(f'Class: {cls}', fontsize=15, fontweight='bold')
    axes[idx].set_xlabel('Coefficient', fontsize=16)
    axes[idx].tick_params(axis='x', labelsize=14)
    axes[idx].tick_params(axis='y', labelsize=14)
    if idx == 0:
        axes[idx].set_ylabel('Feature')
    else:
        axes[idx].set_ylabel('')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

## SUPPORT VECTOR MACHINE
from sklearn.svm import SVC

# Pipeline
pipeline_svm     =     Pipeline(steps=[('preprocessor',     preprocessor),     ('classifier',
SVC(random_state=42))])

# GridSearch
param_grid_svm = {'classifier__C': [0.01, 0.1, 1, 10], 'classifier__kernel': ['linear', 'rbf'],
'classifier__gamma': ['scale', 'auto']}
grid_search_svm     =     GridSearchCV(pipeline_svm,     param_grid_svm,     cv=5,
scoring='accuracy')
grid_search_svm.fit(X_train, y_train)
y_pred_svm = grid_search_svm.predict(X_test)
print("Best Parameters (SVM):", grid_search_svm.best_params_)

# Classification report
print("Classification Report (SVM):\n", classification_report(y_test, y_pred_svm))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_svm, labels=class_order)
print("Confusion Matrix (SVM):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm,    annot=True,    fmt='d',    cmap='Blues',    xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_svm = grid_search_svm.best_estimator_.named_steps['classifier']
if grid_search_svm.best_params_['classifier__kernel'] == 'linear':
   coeff_svm = best_model_svm.coef_[0]
                                                   feature_names_svm                              =
grid_search_svm.best_estimator_.named_steps['preprocessor'].get_feature_names_out(
)
     coefficients_svm  =  pd.DataFrame({'Feature':   feature_names_svm,   'Coefficient':
coeff_svm}).sort_values(by='Coefficient', ascending=False)
   print("Top 10 important features (SVM):\n", coefficients_svm.head(10))

   top_n = 10
   top_features = coefficients_svm.head(top_n).append(coefficients_svm.tail(top_n))
   plt.figure(figsize=(8, 6))
   colors = top_features['Coefficient'].apply(lambda x: '#67a9cf' if x > 0 else '#2166ac')
   sns.barplot(x='Coefficient', y='Feature', data=top_features, palette=colors)
   plt.axvline(x=0, color='black', linewidth=0.8)
   plt.set_xlabel('Coefficient', fontsize=16)
   plt.tick_params(axis='x', labelsize=14)
   plt.tick_params(axis='y', labelsize=14)
   plt.tight_layout()
   plt.show()
else:
   print("SVM with RBF kernel does not provide coefficients for feature importance.")

## K-NEAREST NEIGHBORS
from sklearn.neighbors import KNeighborsClassifier

# Pipeline
pipeline_knn     =     Pipeline(steps=[('preprocessor',     preprocessor),     ('classifier',
KNeighborsClassifier())])

# GridSearch
param_grid_knn   =   {'classifier__n_neighbors':   [3,   5,   7,   10],   'classifier__weights':
['uniform', 'distance'], 'classifier__metric': ['euclidean', 'manhattan']}
grid_search_knn      =      GridSearchCV(pipeline_knn,      param_grid_knn,      cv=5,
scoring='accuracy')
grid_search_knn.fit(X_train, y_train)
y_pred_knn = grid_search_knn.predict(X_test)
print("Best Parameters (KNN):", grid_search_knn.best_params_)

# Classification report
print("Classification Report (KNN):\n", classification_report(y_test, y_pred_knn))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_knn, labels=class_order)
print("Confusion Matrix (KNN):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm,    annot=True,    fmt='d',    cmap='Blues',    xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
classes = ['Low', 'Medium', 'High']
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]
y_score = grid_search_knn.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# We cannot use classical coefficients or feature importances, but we can estimate the
relevance of the features using Permutation Feature Importance (standard solution for non-
parametric models as this one).
from sklearn.inspection import permutation_importance

# Feature analysis
print("\nKNN - Feature Importance per class")
for cls in class_order:
    print(f"\nClass {cls}:")
    mask = (y_test == cls)
     result = permutation_importance(grid_search_knn, X_test[mask], y_test[mask],
n_repeats=10, random_state=42, scoring='accuracy')
    sorted_idx = result.importances_mean.argsort()[::-1]
    for i in sorted_idx:
        print(f"{X_test.columns[i]}: {result.importances_mean[i]:.4f}")

results_per_class = []
classes = np.unique(y_test)
top_n = 10
for cls in class_order:
    mask = (y_test == cls)
     result = permutation_importance(grid_search_knn, X_test[mask], y_test[mask],
n_repeats=10, random_state=42, scoring='accuracy')
    sorted_idx = result.importances_mean.argsort()[::-1][:top_n]
```

```
        df_top = pd.DataFrame({'Feature': X_test.columns[sorted_idx], 'Importance': result.importances_mean[sorted_idx], 'Class': cls})
    results_per_class.append(df_top)

df_pfi = pd.concat(results_per_class)
fig, axes = plt.subplots(nrows=1, ncols=len(class_order), figsize=(5 * len(class_order), 6), sharey=True)

for idx, cls in enumerate(class_order):
    df_class = df_pfi[df_pfi['Class'] == cls].sort_values('Importance', ascending=True)
    colors = ['#67a9cf' if val >= np.median(df_class['Importance']) else '#2166ac' for val in df_class['Importance']]
    sns.barplot(x='Importance', y='Feature', data=df_class, palette=colors, ax=axes[idx])
    axes[idx].axvline(x=0, color='black', linewidth=0.8)
    axes[idx].set_title(f'Class: {cls}', fontsize=15, fontweight='bold')
    axes[idx].set_xlabel('Importance', fontsize=16)
    axes[idx].tick_params(axis='x', labelsize=14)
    axes[idx].tick_params(axis='y', labelsize=14)
    if idx == 0:
        axes[idx].set_ylabel('Feature')
    else:
        axes[idx].set_ylabel('')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

## NAIVE BAYES
from sklearn.naive_bayes import GaussianNB

# Pipeline
pipeline_nb = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GaussianNB())
])

# GridSearch
param_grid_nb = {
    'classifier__var_smoothing': [1e-9, 1e-8, 1e-7]
}
grid_search_nb = GridSearchCV(pipeline_nb, param_grid_nb, cv=5, scoring='accuracy')
grid_search_nb.fit(X_train, y_train)
y_pred_nb = grid_search_nb.predict(X_test)
print("Best Parameters (Naive Bayes):", pipeline_nb.named_steps['classifier'].get_params())

# Classification report
print("Classification Report (Naive Bayes):\n", classification_report(y_test, y_pred_nb))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_nb, labels=class_order)
print("Confusion Matrix (Naive Bayes):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order, yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size": 18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
classes = ['Low', 'Medium', 'High']
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]
y_score = grid_search_nb.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature importance analysis is not applicable for Naive Bayes as it's a probabilistic model
(and not a linear one).

## DECISION TREE
from sklearn.tree import DecisionTreeClassifier

# Pipeline
pipeline_dt = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# GridSearch
param_grid_dt = {'classifier__max_depth': [None, 10, 20],
'classifier__min_samples_split': [2, 5], 'classifier__min_samples_leaf': [1, 2]}
grid_search_dt = GridSearchCV(pipeline_dt, param_grid_dt, cv=5, scoring='accuracy')
grid_search_dt.fit(X_train, y_train)
y_pred_dt = grid_search_dt.predict(X_test)
print("Best Parameters (Decision Tree):", grid_search_dt.best_params_)

# Classification report
print("Classification Report (Decision Tree):\n", classification_report(y_test, y_pred_dt))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_dt, labels=class_order)
print("Confusion Matrix (Decision Tree):\n", cm)
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
classes = ['Low', 'Medium', 'High']
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]
y_score = grid_search_dt.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_dt = grid_search_dt.best_estimator_.named_steps['classifier']
feature_names_dt =
grid_search_dt.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
coefficients_dt = pd.DataFrame({'Feature': feature_names_dt, 'Importance':
best_model_dt.feature_importances_}).sort_values(by='Importance',
ascending=False).head(10)
coefficients_dt = coefficients_dt.sort_values(by='Importance', ascending=True)
print("Top 10 important features (Decision Tree):\n", coefficients_dt)
median_importance = coefficients_dt['Importance'].median()
colors = ['#67a9cf' if val >= median_importance else '#2166ac' for val in
coefficients_dt['Importance']]
plt.figure(figsize=(8, 6))
sns.barplot(x='Importance', y='Feature', data=coefficients_dt, palette=colors)
plt.xlabel('Importance', fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.tick_params(axis='x', labelsize=14)
plt.tick_params(axis='y', labelsize=14)
plt.tight_layout()
plt.show()

## RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier

# Pipeline
pipeline_rf = Pipeline(steps=[('preprocessor', preprocessor), ('classifier', RandomForestClassifier(random_state=42))])

# GridSearch
param_grid_rf = {'classifier__n_estimators': [100, 200], 'classifier__max_depth': [None, 10, 20], 'classifier__min_samples_split': [2, 5], 'classifier__min_samples_leaf': [1, 2]}
grid_search_rf = GridSearchCV(pipeline_rf, param_grid_rf, cv=5, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)
y_pred_rf = grid_search_rf.predict(X_test)
print("Best Parameters (Random Forest):", grid_search_rf.best_params_)

# Classification report
print("Classification Report (Random Forest):\n", classification_report(y_test, y_pred_rf))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_rf, labels=class_order)
print("Confusion Matrix (Random Forest):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order, yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size": 18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
y_test_bin = label_binarize(y_test, classes=class_order)
n_classes = y_test_bin.shape[1]
y_score = grid_search_rf.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {class_order[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_rf = grid_search_rf.best_estimator_.named_steps['classifier']
feature_names_rf                                                              =
grid_search_rf.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
importances_rf     =     pd.DataFrame({'Feature':     feature_names_rf,     'Importance':
best_model_rf.feature_importances_}).sort_values(by='Importance', ascending=False)
print("Top 10 important features (Random Forest):\n", importances_rf.head(10))
top_rf = importances_rf.head(10).sort_values(by='Importance', ascending=True)
median_importance = top_rf['Importance'].median()
colors = ['#67a9cf' if val >= median_importance else '#2166ac'
        for val in top_rf['Importance']]
plt.figure(figsize=(8, 6))
sns.barplot(x='Importance', y='Feature', data=top_rf, palette=colors)
plt.xlabel('Importance', fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.tick_params(axis='x', labelsize=14)
plt.tick_params(axis='y', labelsize=14)
plt.tight_layout()
plt.show()

### MODEL BUILDING - ENSEMBLE LEARNING APPROACHES
## GRADIENT BOOSTING
from sklearn.ensemble import GradientBoostingClassifier

# Pipeline
pipeline_gb     =     Pipeline(steps=[('preprocessor',     preprocessor),     ('classifier',
GradientBoostingClassifier(random_state=42))])

# GridSearch
param_grid_gb     =     {'classifier__n_estimators':     [50,     100,     150,     200],
'classifier__learning_rate': [0.005, 0.01, 0.05, 0.1], 'classifier__max_depth': [2, 3, 4, 5]}
grid_search_gb = GridSearchCV(pipeline_gb, param_grid_gb, cv=5, scoring='accuracy')
grid_search_gb.fit(X_train, y_train)
y_pred_gb = grid_search_gb.predict(X_test)
print("Best Parameters (Gradient Boosting):", grid_search_gb.best_params_)

# Classification report
print("Classification   Report   (Gradient   Boosting):\n",   classification_report(y_test,
y_pred_gb))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_gb, labels=class_order)
print("Confusion Matrix (Gradient Boosting):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm,    annot=True,    fmt='d',    cmap='Blues',    xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()
```

```
# ROC Curve for One-vs-Rest classification
y_test_bin = label_binarize(y_test, classes=class_order)
n_classes = y_test_bin.shape[1]
y_score = grid_search_gb.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {class_order[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_gb = grid_search_gb.best_estimator_.named_steps['classifier']
feature_names_gb =
grid_search_gb.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
importances_gb = pd.DataFrame({'Feature': feature_names_gb, 'Importance':
best_model_gb.feature_importances_}).sort_values(by='Importance', ascending=False)
print("Top 10 important features (Gradient Boosting):\n", importances_gb.head(10))
top_gb = importances_gb.head(10).sort_values(by='Importance', ascending=True)
median_importance = top_gb['Importance'].median()
colors = ['#67a9cf' if val >= median_importance else '#2166ac' for val in
top_gb['Importance']]
plt.figure(figsize=(8, 6))
sns.barplot(x='Importance', y='Feature', data=top_gb, palette=colors)
plt.xlabel('Importance', fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.tick_params(axis='x', labelsize=14)
plt.tick_params(axis='y', labelsize=14)
plt.tight_layout()
plt.show()

## ADABOOST
from sklearn.ensemble import AdaBoostClassifier

# Pipeline
pipeline_ab = Pipeline(steps=[('preprocessor', preprocessor), ('classifier',
AdaBoostClassifier(algorithm='SAMME', random_state=42))])

# GridSearch
param_grid_ab = {'classifier__n_estimators': [50, 100, 200], 'classifier__learning_rate':
[0.01, 0.1, 1]}
grid_search_ab = GridSearchCV(pipeline_ab, param_grid_ab, cv=5, scoring='accuracy')
```

```
grid_search_ab.fit(X_train, y_train)
y_pred_ab = grid_search_ab.predict(X_test)
print("Best Parameters (AdaBoost):", grid_search_ab.best_params_)

# Classification report
print("Classification Report (AdaBoost):\n", classification_report(y_test, y_pred_ab))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_ab, labels=class_order)
print("Confusion Matrix (AdaBoost):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
y_test_bin = label_binarize(y_test, classes=class_order)
n_classes = y_test_bin.shape[1]
y_score = grid_search_ab.best_estimator_.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
             label=f'Class {class_order[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_ab = grid_search_ab.best_estimator_.named_steps['classifier']
feature_names_ab =
grid_search_ab.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
coefficients_ab = pd.DataFrame({'Feature': feature_names_ab, 'Importance':
best_model_ab.feature_importances_}).sort_values(by='Importance', ascending=False)
print("Top 10 important features (AdaBoost):\n", coefficients_ab.head(10))
top_ab = coefficients_ab.head(10).sort_values(by='Importance', ascending=True)
median_val = top_ab['Importance'].median()
colors = ['#67a9cf' if val >= median_val else '#2166ac' for val in top_ab['Importance']]
```

```
plt.figure(figsize=(8, 6))
sns.barplot(x='Importance', y='Feature', data=top_ab, palette=colors)
plt.xlabel('Importance', fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.tick_params(axis='x', labelsize=14)
plt.tick_params(axis='y', labelsize=14)
plt.tight_layout()
plt.show()

## CATBOOST
from catboost import CatBoostClassifier

# Pipeline
pipeline_cb = Pipeline(steps=[('preprocessor', preprocessor), ('classifier',
CatBoostClassifier(random_state=42, verbose=0))])

# GridSearch
param_grid_cb = {'classifier__iterations': [100, 200], 'classifier__learning_rate': [0.01,
0.1], 'classifier__depth': [5, 7, 10]}
grid_search_cb = GridSearchCV(pipeline_cb, param_grid_cb, cv=5, scoring='accuracy')
grid_search_cb.fit(X_train, y_train)
y_pred_cb = grid_search_cb.predict(X_test)
print("Best Parameters (CatBoost):", grid_search_cb.best_params_)

# Classification report
print("Classification Report (CatBoost):\n", classification_report(y_test, y_pred_cb))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_cb, labels=class_order)
print("Confusion Matrix (CatBoost):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

# ROC Curve for One-vs-Rest classification
y_test_bin = label_binarize(y_test, classes=class_order)
n_classes = y_test_bin.shape[1]
best_model_cb = grid_search_cb.best_estimator_
y_score_cb = best_model_cb.predict_proba(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_cb[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
blues = get_cmap('Blues')
colors = [blues(0.6), blues(0.75), blues(0.9)]
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2,
          label=f'Class {class_order[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.legend(loc="upper left", fontsize = 18)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Feature analysis
best_model_cb = grid_search_cb.best_estimator_.named_steps['classifier']
feature_names_cb =
grid_search_cb.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
importances_norm = best_model_cb.feature_importances_ /
best_model_cb.feature_importances_.sum()
coefficients_cb = pd.DataFrame({'Feature': feature_names_cb, 'Normalized Importance':
importances_norm}).sort_values(by='Normalized Importance', ascending=False)
print("Top 10 important features (CatBoost - Normalized):\n", coefficients_cb.head(10))
top_cb = coefficients_cb.head(10).sort_values(by='Normalized Importance',
ascending=True)
median_val = top_cb['Normalized Importance'].median()
colors = ['#67a9cf' if val >= median_val else '#2166ac' for val in top_cb['Normalized
Importance']]
plt.figure(figsize=(8, 6))
sns.barplot(x='Normalized Importance', y='Feature', data=top_cb, palette=colors)
plt.xlabel('Normalized Importance', fontsize=16)
plt.ylabel('Feature', fontsize=16)
plt.tick_params(axis='x', labelsize=14)
plt.tick_params(axis='y', labelsize=14)
plt.tight_layout()
plt.show()

## STACKING
from sklearn.ensemble import StackingClassifier

# Defining the base learners
base_learners = [('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
('gb', GradientBoostingClassifier(n_estimators=100, random_state=42)), ('ada',
AdaBoostClassifier(n_estimators=100, random_state=42))]

# Defining the meta-learner
meta_learner = LogisticRegression()

# Creating the stacking classifier (with 5-fold cross-validation)
stacked_model = StackingClassifier(estimators=base_learners,
final_estimator=meta_learner, cv=5)

# Pipeline
pipeline_stack = Pipeline(steps=[('preprocessor', preprocessor), ('classifier',
stacked_model)])
pipeline_stack.fit(X_train, y_train)
y_pred_stack = pipeline_stack.predict(X_test)

# Classification report
print("Classification Report (Stacking):\n", classification_report(y_test, y_pred_stack))
```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred_stack, labels=class_order)
print("Confusion Matrix (Stacking):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()

## VOTING - MAJORITY VOTE
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(estimators=[('gb', best_model_gb), ('svm',
best_model_svm), ('ab', best_model_ab)], voting='hard')

# Pipeline
pipeline_voting = Pipeline([('preprocessor', preprocessor), ('classifier', voting_clf)])
pipeline_voting.fit(X_train, y_train)
y_pred_voting = pipeline_voting.predict(X_test)

# Classification report
print("Classification Report (Voting):\n", classification_report(y_test, y_pred_voting))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_voting, labels=class_order)
print("Confusion Matrix (Voting):\n", cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_order,
yticklabels=class_order, cbar=False, linewidths=0.5, linecolor='blue', annot_kws={"size":
18})
plt.xlabel("Predicted Label", fontsize = 20, labelpad = 15)
plt.ylabel("True Label", fontsize = 20, labelpad = 15)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.tight_layout()
plt.show()
```