Machine Learning I

Wizardry School Enrollment Prediction

Final Project Report

Group 04

18th December 2023

João Capitão | 20221863

Margarida Cruz | 20221929

Luís Mendes | 20221949

Dinis Gaspar | 20221869

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

ABSTRACT

Making a decision on a students' academic future is something to be taken seriously and in a fair way. On that note, this document will explain how using previous data from a wizardry school enables the creation of a machine learning algorithm that will predict, with higher probability than just using 'our gut feeling', who will enter a prestigious magic school. It is no easy task given the need to treat the historical data, check if the format received was correct and if everything inside is useful; as well as the existence of many models and respective variations. Nevertheless, it is possible to analyze and choose which algorithm/model is the best, that is, the one with more confident and accurate results when a new student tries to enter. In our case, and after an exhaustive search, it was decided that the model with less overfitting, higher F1-score and less complexity will be the one used to make the predictions for the new students.

KEYWORDS

Machine Learning, Model, Prediction, Hyperparameter, Feature, Performance.

INTRODUCTION

A wizardry school needs to find a way of choosing which new candidates will enter the school based on certain features of the person, to improve the decision-making process. The work presented here relies on the dataset provided by the school (a small one containing only 713 observations) , which represents characteristics of past candidates and whether they have been accepted or not. The aim is to create a machine learning model that will predict for new students (from another dataset) whether they are admitted to the wizardry school or not.

In the process of creating our final solution, we will encounter challenges and questions that will require more detailed analysis, such as:

- Do we need to fix any problems with our dataset?
- Is it needed to use all the information that is available in the dataset?
- Do we need to transform our dataset before using it?
- Does the dataset contain the same number of cases where the student was accepted and not accepted?
- What metric should be used to evaluate our models?
- How will the final model be chosen?

The present report is expected to significantly contribute to answer those questions and deliver a final result for our problem. It is organized as follows: section two details the methods employed, whereas section three delves deeper into the results and respective discussion. Finally, section four presents the final conclusion, with a general overview and some final remarks.

METHODOLOGY

This section presents the methods, data preprocessing techniques and machine learning algorithms used as well as the performance metrics that will be used for the models. The work was developed in *Jupyter Notebook*, using *Python* as programming language, *pandas* to treat the dataset, the *sklearn* library to build the models and *matplotlib* and *seaborn* to visualize the results.

EXPLORATORY ANALYSIS

The data exploration phase serves to enhance the comprehension of the dataset at hand. Following this line of thinking, methods such as *.head()* and *.info()* were used to gain an overview of the dataset by retrieving its initial entries as well as checking the total number of entries, the different featrues and corresponding datatypes, respectively. Also, it made use of the method *.isnull().sum()* to have the number of missing values for each feature.

In addition, to generate statistical summaries and gain insights into the dataset's key characteristics, for both numerical and categorical features, the method *.describe()* was used. To check the number of students who were admitted and rejected at the school (from the known data), *.value_counts()* was utilized.

Since the variable "StudentID" identifies each unique student and does not give any additional information in terms of predicting, after making sure there were not any duplicate values, it was appropriate to set it as the index.

Moreover, the split of the dataset into the dependent and independent features allows for a deeper look into the distribution of the variables. For visualization, bar charts were used to represent categorical data, while histograms were used for numerical features. Its analysis will determine how outliers will be handled if needed(either by defining upper and/or lower limits, or a replacement boundary).

CATEGORICAL FEATURE SELECTION

Given the absence of missing values in the categorical features of the dataset, feature selection can be confidently applied prior to encoding or imputing.

By creating bar charts for individual categorical features and incorporating the corresponding proportions of the target variable, it is possible to verify whether a variable is a relevant predictor for the target feature. Additionally, a Chi-Squared test with a significance level of 5% was used to test the independence between the features and the target.

ENCODING

To deploy the desired models effectively, it was essential to encode the categorical features in the dataset.

Since the "Gender" feature only has two different classes, it was possible to manually encode it: the value 'male' was assigned to '1' and the value 'female' was assigned to '0'. For the remaining features ("Program", "School of Origin"), Dummy Variables were created - which involved creating binary indicators for each unique category within these features. Since the Dummy approach assigns '1' to the dummy feature corresponding to the respective class and '0' to the others, the combination of exclusively '0's was leveraged to represent a particular class. This way, an unnecessary extra column for each category is avoided.

Having successfully encoded all the features, it was pertinent to check the correlation. Using the Spearman's Correlation Coefficient, features were considered highly correlated when its correlation coefficient was higher than a threshold of |0.7| (which didn't happen, as shown in the results section).

SPLITTING INTO TRAIN AND VALIDATION

To assess the model's performance on unseen data and, consequently, prevent the model from overfitting, partitioning the dataset into training and validation is a necessary step. The training set will be utilized to train the model, while the validation set will be used to estimate how well the model will generalize to new and unseen data. This split will be used at the time of evaluating the best scalar and the best parameter for imputation.

Hence, by means of the *train_test_split* function from the *sklearn* library, 80% of the observations were attributed to the training dataset and 20% were assigned to the validation dataset. It is worth noting that *shuffle=True* was used to reduce the risk of introducing biases during the split. Also, the data was stratified by the target to maintain the proportion of each class of the target feature in both the training and validation sets (the latter is quite important in the reality of an unbalanced dataset).

IMPUTING MISSING VALUES

The missing values in the "Experience Level" feature were addressed through imputation. Such choice was motivated by the desire to retain the valuable information present in the dataset, and to maintain the integrity of the feature. To achieve this, the *KNNImputer* from the *sklearn* library taking into consideration the similarities to the observation's neighbors was used.

To determine the ideal number of neighbors and whether the weight function is uniform or taking the distance into account, a loop was created. Within the loop, predictions were conducted using *Logistic Regression* for various numbers of neighbors and the different weighting criterias. The one that gave the best results for the validation dataset (defined in the previous split) was chosen - being the optimal parameters: *n_neighbors = 5, weights = 'uniform'*.

FEATURE SCALING

To ensure that all variables contribute equally to the model training process and prevent any single feature from dominating due to magnitude differences, feature scaling was performed. This ensures fair and effective learning across all features.

To achieve this, four distinct scalers were created. Specifically, *StandardScaler*, *RobustScaler*, and two instances of *MinMaxScaler* were used. In one instance of *MinMaxScaler*, a custom feature range of (-1, 1) was applied.

After fitting the scalar to the training dataset, a *Logistic Regression* was applied and generated predictions in both the training and validation datasets. Finally, the accuracy and the f1 score were assessed, among other performance metrics through the classification report; with *MinMaxScaler* with the feature range of (-1, 1) being chosen for having the best score on the validation set and minimal overfitting compared to the other models.

NUMERICAL FEATURE SELECTION

To reduce the dimensionality of the data and select only the features that seem to effectively have an impact on predicting whether a student will be accepted at the school, two different feature selection techniques were applied: Wrapper Methods and Embedded Methods. To note that, since categorical features have already been dealt with, both methods were applied to the numerical features.

In terms of Wrapper Methods, the Recursive Feature Elimination (RFE) was applied. Initially, a loop was created to find the optimal number of features to retain, based on the score achieved with that specific number of features (utilizing *RepeatedStratifiedKFold* for splitting and *Logistic Regression* as the base estimator); with the optimal performance being obtained with the 5 features. Subsequently, an instance of RFE was created having *Logistic Regression* as the estimator and incorporating the optimal number of features.

Regarding Embedded Methods, the Lasso Cross-Validated (LassoCV) algorithm was applied iteratively using cross-validation and the mean coefficients were calculated across all folds.

The results were analyzed to select whether a feature would remain for the modeling phase, and *RepeatedStratifiedKFold* was used to ensure that the decisions made were based on multiple different splits of the dataset and not just a single one.

DATASETS

The models will be tested with 3 different datasets, to see which ones generate better results. It is worth clarifying that these datasets already contain the dummy variables (with n-1 columns for each dummy variable that was created, with n = number of classes for that feature).

The datasets are:

- ***X***: has all the features that were to be kept in the feature selection phase (all numerical features and the selected categorical features) - 'Experience Level', 'Student Siblings', 'Student Family', 'Financial Background', 'Student Social Influence', 'Male', 'Program_Magi Academy', 'Program_Sorcery School', 'School of Origin_Arcan Institute', 'School of Origin_Eldertree Enclave' - ; with removal of outliers in the feature 'Financial Background' and column 'Student Siblings' encoded as 0 = no siblings, 1 = 1 sibling , 2 = 2 siblings, 3 = 3 siblings and 4 = more than 3 siblings.
- ***X_testing***: the same as **X** dataset, except for the 2 numerical features that had a low importances in LassoCV test - 'Student Family' and 'Student Social Influence'.
- ***New***: has all observations (that means no removal of outliers in the feature 'Financial Background'), the feature 'Student Siblings' encoded to 0 and 1 (0 = no siblings; 1 = at least 1 sibling) and using the same columns as ***X_testing***.

For all 3 datasets, normalization will be applied using a *MinMax* scaler ranging from -1 to 1 (the one that led to better results when trying all scalers, as stated before). Afterwards, the best parameters found previously will be used for imputation. To note that imputation is applied after scaling since it requires the computation of distances and it was intended to give the same importances to all features, to avoid having a biased imputation.

The scalled and imputed datasets will be stored in other variables called ***X_scl_imp*** (for the 'X' dataset), ***X_testing_scl_imp*** (for 'X_testing') and ***new_scl_imp*** (for 'new'). These datasets can now be used to train and compare the models (using the *GridSearchCV* method).

MODELING

In this phase, *GridSearchCV* was used to find the best hyperparameters for each type of model (Logistic Regression, Decision Tree, K-nearest-neighbors, Naive bayes, Bagging classifier, Multilayer perceptron (Neural Network), Random forest, Adaboost, Gradient boosting and Stacking).

A stratified k-fold of 7 splits was used and the ones with better F1 scores will be kept and analyzed later on. There is no special reason why one hyperparameter was chosen over another, given this is an exhaustive search trying to find the best combination of hyperparameters. The datasets to use are already scaled and imputed.

The F1 scores of the *GridSearchCV* may have some data leakage, and thus they are only used for the sake of evaluation and comparison of models. To solve that problem and to obtain trustworthy results, the function *avg_scores(method, X, y, model)* was created which receives a cross validation method (a repeated stratified k-fold of 5 repeats and 10 splits was used for our testing), the original dataset (not scaled or imputed before), the target variable of that dataset and the model to evaluate using the F1 score.

In each iteration, the function will do a split in train and validation, scale all the data learning only from the training, impute all the data learning only for the training, fit the model using only the training and calculate the F1 score for training and validation. This process is repeated until the stratified k-fold is finished and the mean F1 score for train and validation will be shown, obtaining a robust result without data leakage.

Listed below are all of the different models tested which we testes:

- **Logistic Regression:** just applied a grid search.
- **Decision tree:** starting by calculating a good value for the alpha in the decision tree to know how complex the tree should be (ccp_alpha = 0.003209919868982874), then applying a grid search.
- **K Nearest Neighbors:** started by calculating the best number of neighbors to use to achieve better results (10 neighbors), then applied a grid search.
- **Naive bayes:** since it has so few parameters, gridsearch was not used – only searched for the result of the default models of *GaussianNB()* and *MultinomialNB()*. For the latter, since it does not allow negative values, there is the need to create a second function for the average score - *avg_scores_mnb(method,X,y, model)* -, which mirrors the avg_scores function, changing only the scaling method (now using the *MinMax* scaler ranging from 0 to 1). We didn't try these models with our 'new' dataset because the results obtained with the others datasets were already bad scores.
- **Bagging classifier:** 3 bagging classifiers were created (changing the base estimator – there is a bagging of Decision Trees, a bagging of KNearestNeighbors and a bagging of Naive Bayes), in which a gridsearch was applied. We didn't test the bagging of naive bayes using the 'new' dataset because the results of F1 scores with the other datasets were very low.
- **Multilayer perceptron:** taking into account that the number of hidden neurons should be between the size of the input layer (10 or 8) and the size of the output layer (1); 2/3 the size of the input layer plus the size of the output layer (total = 6-7); and less than twice the size of the input layer (< 20); different structures of networks with between 2 and 19 hidden layer neurons were tested in total, arranged in differents layouts, and a grid search was applied.

- **Random forest:** just applied a grid search.
- **Gradient boosting:** just applied a grid search.
- **Adaboost:** just applied a grid search.
- **Stacking classifier:** built using the best models (models that achieve a F1 score in validation greater than 0.7) for ***X_testing*** and ***new*** datasets, and applied a grid search.

MAKING PREDICTIONS

Since we are going to try models with two different datasets (***X_testing*** and ***new*,** as results for the ***X*** dataset were consistently the lowest, more detailed analysis is present in the next section), we need one to prepare the test dataset to handle the 2 possibilities so to do this final predictions, so:

- We created 2 test datasets ('***test***' for '*X_testing*' and '***test_new***' for '*new*') that are preprocessed according to the dataset they are referencing,
- We will use all of our dataset to train our models
- We will scale our '*X_testing*' and '*new*' using MinMax ranging from -1:1, and imputing using the KNN imputer with the best parameters that we found (doing that individually for each train dataset, obtaining '***f_X_testing_scl_imp***' for '*X_testing*' and '***f_X_new_scl_imp***' for '*new*', and we will use those to train our models).

We need to do the same for our test dataset so we will scale and impute our test dataset with the scaler and imputer fit to the train dataset(obtaining '***test_scl_imp***' for '*X_testing'* and ***'test_scl_imp_new'*** for '*new'*, to do the prediction).

FINAL MODEL

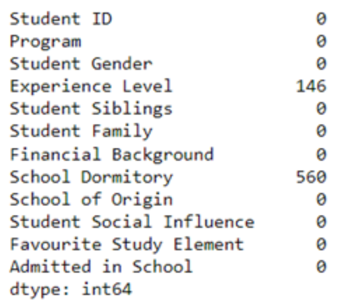
After having our best models, the selection of the final model will take into consideration the F1 scores for train and validation, as well as the complexity of the model and the score obtained on *Kaggle*.

RESULTS

EXPLORATORY ANALYSIS

In accordance with the ***Methodology*** section of this document, firstly an exploratory analysis of the dataset was performed. As shown in **Fig. 1**, the features “Experience Level” and “School Dormitory” reflect the existence of missing values. Knowing that there are a total of 713 entries, the 560 instances that correspond to missing values in the “School Dormitory” category constitute a rate of 78.5% - for this reason it was dropped.

Moreover, the countplot in **Fig. 2** indicates that the number of students who failed to enter the school (461) is substantially higher than the number of students who were accepted (252). Therefore, we are dealing with an unbalanced dataset. For that reason, in all of the analysis F1 score will be used as a metric for evaluating the results since it is the most reliable in cases like this. Apart from that, our model will be evaluated with this metric on *Kaggle*.



```
Student ID                  0
Program                     0
Student Gender              0
Experience Level          146
Student Siblings            0
Student Family              0
Financial Background        0
School Dormitory          560
School of Origin            0
Student Social Influence    0
Favourite Study Element     0
Admitted in School          0
dtype: int64
```

Figure 1 - *.isnull().sum()* output

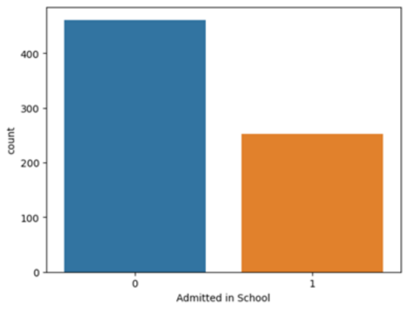


Figure 2 - countplot of the number of students admitted/rejected at the school

For a deeper look into the distribution of the features, the interpretation of **Figures 3-4** lead to the decision of handling outliers in three of the numerical variables. For “Financial Background”, an upper limit of 200 was established (and any observations with a value higher than 200 was removed), resulting in the removal of 17 observations (2.38% of the dataset). On the other hand, for features “Student Siblings” and “Student Family”, an upper limit of 4 and 3 - respectively - was defined where any value above that limit is replaced by the limit itself. This will consider someone that has 4 or more siblings as having 'many siblings', and someone that has 3 or more persons in the school as having 'many persons in the school'. This was the initial approach to handling these outliers, but as mentioned earlier during the datasets subsection other approaches will be tested.

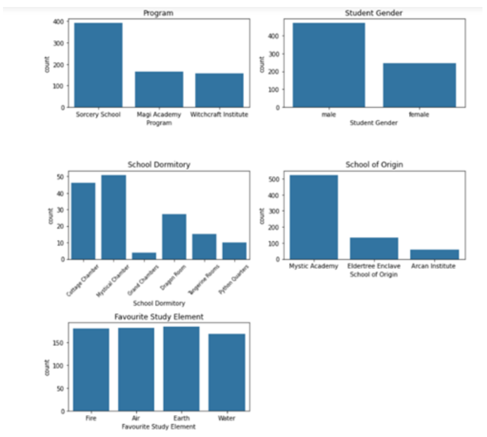


Figure 3 - Categorical Variables Distribution

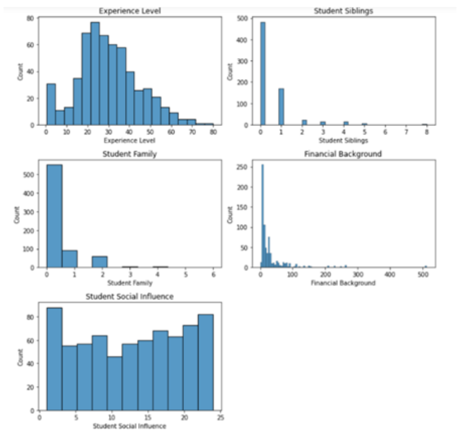


Figure 4 - Numerical Variables Distribution

CATEGORICAL FEATURE SELECTION

According to **Fig. 5**, the “Favourite Study Element” variable is not a relevant predictor for the target variable, given that the frequencies of the different classes are nearly identical and the proportions of the target variable classes within the classes (of this variable) are essentially the same. Additionally, the *Chi-Squared* test results – stated in **Fig. 6** - also identify that “Favourite Study Element” is not an important predictor. Thus, it was removed.

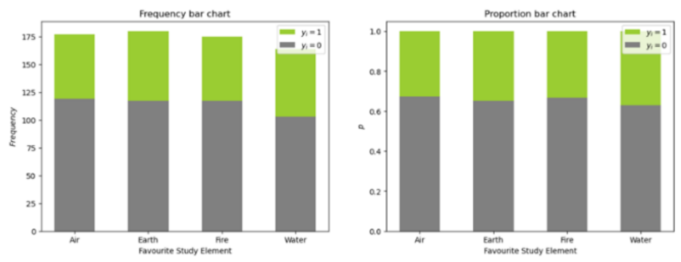


Figure 5 - “Favourite Study Element” by target feature

Predictor	Chi-Square
Program	Keep
Student Gender	Keep
School of Origin	Keep
Favourite Study Element	Discard

Figure 6 - Chi-Squared test results

ENCODING

Through the heatmap in **Fig. 7**, one verifies that the maximum correlation coefficient observed, whether positive or negative, was 0.7. Therefore, none of the variables had a high level of correlation, with only one pair hovering at the threshold commonly regarded as indicative of significant correlation.

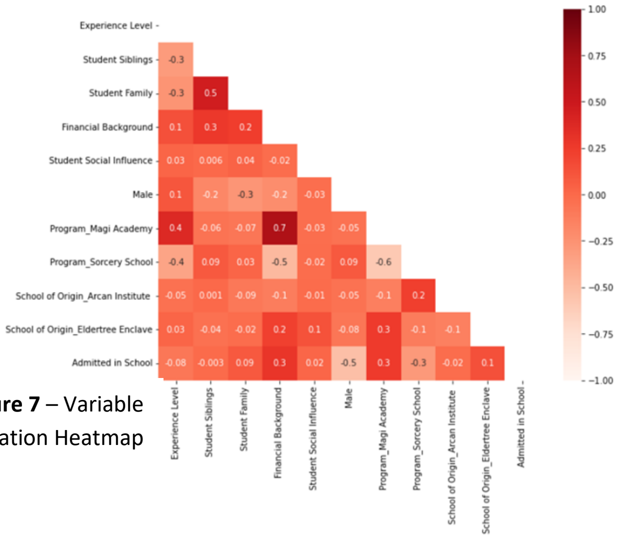


Figure 7 – Variable Correlation Heatmap

FEATURE SELECTION

As expected, the Recursive Feature Elimination recommended keeping all 5 numerical features. Regarding the Lasso Cross-Validated (LassoCV), none of the coefficients equalled 0 - which implies that all features retained some level of importance in the prediction process. From **Fig. 8** it is also possible to visualize, in greater detail, the importance of each feature based on the coefficients. The plot shows that “Student Family” and “Student Social Influence” had a quite small importance (although not null).

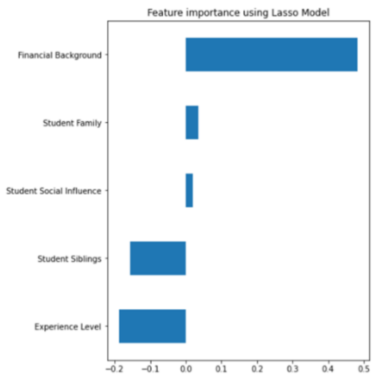


Figure 8 - Feature importance using Lasso

Predictor	RFE	Lasso	What to do? (One possible way to "solve")
Experience Level	Keep	Keep	Include in the model
Student Siblings	Keep	Keep	Include in the model
Student Social Influence	Keep	Keep ?	Try with and without
Student Family	Keep	Keep ?	Try with and without
Family_Size	Keep	Keep	Include in the model

Figure 9 – Numerical Feature Selection Results

Considering the outcomes of the tests performed for numerical feature selection, it is believed that all numerical features are useful for prediction. However, since there are two features with small importance in the LassoCV test (**Fig. 8**), to make an informed decision, it was opted to explore model performance with all features included and, separately, with these less important features removed.

MODELING

After applying the grid search, the followings results for each of our models were obtained:

	***X* DATASET**			***X_TESTING* DATASET**			***NEW* DATASET**		
MODEL	**F1 Train**	**F1 Val**	**Hyperparam.**	**F1 Train**	**F1 Val**	**Hyperparam.**	**F1 Train**	**F1 Val**	**Hyperparam.**
LogisticRegression(random_state=42)	0.7217	0.7120	**C:** 10, **class_weigh**: None, **fit_intercept:** True, **max_iter:** 80, **penalty:** l2, **random_state**: 42, **solver:** newton-cg	0.7248	0.7193	**C:** 100, **class_weigh**: None, **fit_intercept:** True, **max_iter:** 80, **penalty:** l2, **random_state**: 42, **solver:** newton-cg	0.6798	0.6612	**C:** 100, **class_weigh**:None, **fit_intercept:** False, **max_iter:** 80, **penalty:** l2, **random_state**: 42, **solver:** newton-cg
DecisionTreeClassifier()	0.8048	0.7011	**ccp_alpha:** 0.003209919868982874, **criterion**: gini, **max_depth**: None, **max_feature**: 6, **max_leaf_nodes**: None, **splitter**: best	0.8006	0.7067	**ccp_alpha:** 0.003209919868982874 , **criterion**: gini, **max_depth**: None, **max_feature**: 8, **max_leaf_nodes**: None, **splitter**: best	0.7416	0.7161	**ccp_alpha:** 0, **criterion**: entropy, **max_depth**: None, **max_feature**: 6, **max_leaf_nodes**: 8, **splitter**: best
KNeighborsClassifier()	0.9997	0.5124	**algorithm**: auto, **leaf_size**: 10, **metric**: euclidean, **n_neighbors**: 8, **weights**: distance	0.6466	0.4869	**algorithm**: ball_tree, **leaf_size**: 10, **metric**: chebyshev, **n_neighbors**: 5, **weights**: uniform	0.9750	0.5451	**algorithm**: auto, **leaf_size**: 10, **metric**: euclidean, **n_neighbors**: 8, **weights**: uniform
GaussianNB()	0.6465	0.6388	(defaults)	0.6568	0.6447	(defaults)	-	-	-
MultinomialNB()	0.5161	0.5116	(defaults)	0.5285	0.5270	(defaults)	-	-	-
BaggingClassifier(base_estimator=DecisionTreeClassifier())	0.9960	0.7291	**bootstrap**: False, **bootstrap_features**: True, **max_features**: 1.0, **max_samples**: 0.8,	0.8212	0.7272	**bootstrap**: False, **bootstrap_features**: False, **max_features**: 1.0,	0.8941	0.7523	**bootstrap**: False, **bootstrap_features**: False, **max_features**: 1.0, **max_samples**: 0.4,

			n_estimators: 50, **random_state**: 42			**max_samples**: 0.2, **n_estimators**: 100, **random_state**: 42			**n_estimators**: 150, **random_state**: 42
BaggingClassifier(base_estimator=KNeighborsClassifier())	0.6246	0.5024	**bootstrap**: False, **bootstrap_features**: False, **max_features**: 1.0, **max_samples**: 0.8, **n_estimators**: 100, **random_state**: 42	0.7963	0.5981	**bootstrap**: False, **bootstrap_features**: False, **max_features**: 0.8, **max_samples**: 1, **n_estimators**: 150, **random_state**: 42	0.7956	0.5985	**bootstrap**: False, **bootstrap_features**: False, **max_features**: 0.8, **max_samples**: 1.0, **n_estimators**: 150, **random_state**: 42
BaggingClassifier() (of naive bayes)	0.6927	0.6889	**base_estimator**: GaussianNB() **bootstrap**: False, **bootstrap_features**: False, **max_features**: 0.4, **max_samples**: 0.4, **n_estimators**: 5, **random_state**: 42	0.5404	0.5386	**base_estimator**: GaussianNB() **bootstrap**: False, **bootstrap_features**: False, **max_features**: 0.4, **max_samples**: 0.4, **n_estimators**: 5, **random_state**: 42	-	-	-
MLPClassifier()	0.7238	0.6662	**activation**: relu, **hidden_layer_sizes**: 19, **learning_rate**: constant, **learning_rate_init**: 0.001, **max_iter**: 500, **random_state**: 42, **shuffle**: True, **solver**: adam	0.7259	0.6917	**activation**: tanh, **hidden_layer_sizes**: (5,7,5), **learning_rate**: constant, **learning_rate_init**: 0.01, **max_iter**: 1000, **random_state**: 42, **shuffle**: False, **solver**: adam	0.7093	0.6804	**activation**: relu, **hidden_layer_sizes**: (5,6,5), **learning_rate**: constant, **learning_rate_init**: 0.001, **max_iter**: 500, **random_state**: 42, **shuffle**: True, **solver**: adam
RandomForestClassifier(random_state=22)	0.8891	0.7220	**ccp_alpha:** 0, **criterion**: gini, **max_depth**: 7, **max_feature**: 8, **max_leaf_nodes**: None, **n_estimators:** 50	0.8294	0.7253	**ccp_alpha:** 0.003209919868982874 , **criterion**: gini, **max_depth**: None, **max_feature**: 4,	0.8532	0.7470	**ccp_alpha:** 0, **criterion**: gini, **max_depth**: None, **max_feature**: 8, **max_leaf_nodes**: None, **n_estimators:** 100

						max_leaf_nodes: None, **n_estimators**: 10			
GradientBoostingClassifier(random_state=22)	0.8851	0.7193	**ccp_alpha**: 0.003209919868982874, **criterion**: friedman_mse, **learning_rate**: 0.5, **loss**: exponential, **max_depth**: 7, **max_features**: 8, **max_leaf_nodes**: None, **n_estimators**: 10	0.8724	0.7206	**ccp_alpha**: 0, **criterion**: squared_error **learning_rate**: 1, **loss**: exponential, **max_depth**: 7, **max_features**: 6, **max_leaf_nodes**: 12, **n_estimators**: 10	0.8751	0.7387	**ccp_alpha**: 0, **criterion**: friedman_mse **learning_rate**: 0.1, **loss**: exponential, **max_depth**: 3, **max_features**: 6, **max_leaf_nodes**: 8, **n_estimators**: 150
AdaBoostClassifier(random_state = 22)	0.8011	0.7029	**algorithm**: SAMME.R, **estimator**: None, **learning_rate**: 1, **n_estimators**: 150	0.7789	0.7148	**algorithm**: SAMME.R, **estimator**: None, **learning_rate**: 0.5, **n_estimators**: 150	0.8098	0.7232	**algorithm**: SAMME.R, **estimator**: None, **learning_rate**: 1, **n_estimators**: 150

Note: For our **'new' dataset in the random forest** model in one of the tries of the grid search we found a model (RandomForestClassifier(random_state = 22, ccp_alpha = 0.003209919868982874, criterion = 'gini', max_features = 8, n_estimators = 100)), that achieved F1 Score = 0.8523 in training and **F1 Score = 0.7532 in validation,** so instead of using the random forest that we got during the final run, we will use this one as final model of the random forest for the 'new' dataset.

Conclusions so far:

- The worst dataset is the '*X*' dataset (the one with removal of outliers in the 'financial background' column, encoded the siblings columns with 0-1-2-3) because it is the one that in general had lower F1 scores for validation and more overfitting (bigger difference between F1 score for train and validation) among the differents models that we evaluated compared to the other 2 datasets; for that reason we won't be doing more analysis with it.
- Using Naive Bayes (alone and in bagging) leads us to the worst results, that could be for the assumptions we are required to make when using this type of model (for example independence between features).
- Using KNN (alone) led us to bad results, that could be because we had data points that belong to different classes that are near each other.
- Only a few of our models managed to achieve F1 scores higher than 0.7 in validation, but the results among different models tend to be around the same numbers [0.65 ; 0.74] approximately, that could be because the classes are difficult to divide by establishing linear and non linear boundaries.

For the base estimator of the stacking the best models (models that achieve an F1-score > 0.7 in the validation) we obtained were:

- For **'X_testing'** we will use the following models as base estimator (using the hyperparameters already found): Logistic regression, Decision tree, Bagging of decision trees, Random forests, Gradient boost and Adaboost.
- For **'new'** we will use the following models as base estimator (using the hyperparameters already found): Decision tree, Bagging of decision trees, Random forests, Gradient boost and Adaboost.

Here are the results of the stacking classifiers:

	X_TESTING **DATASET**			***NEW*** **DATASET**		
MODEL	**F1 Train**	**F1 Val**	**Hyperparameters**	**F1 Train**	**F1 Val**	**Hyperparameters**
StackingClassifier() (using the best models)	0.8249	0.7336	**final_estimator**: None, **passthrough**: True, **stack_method**: predict_proba	0.8626	0.7458	**final_estimator**: AdaBoostClassifier (estimator=LogisticRegression (random_state=22), **passthrough**: False, **stack_method**: predict_proba

The Stacking of best models using the ***X_testing*** dataset was the best among all previous models found because it achieved a higher validation F1 score (the overfitting was similar to the other models). The Random Forest is still the model with best value for the F1 score among all models using the ***new*** dataset. This dataset also gives us higher F1 scores in validation compared to the models using ***X_testing***, allowing to conclude that the ***new*** dataset is better than the ***X_testing*** dataset.

MAKING PREDICTIONS

The following table presents the models results when making predictions.

	X_TESTING **DATASET**				***NEW*** **DATASET**			
MODEL	**F1 Train**	**F1 Val**	**Kaggle**	**Predictions**	**F1 Train**	**F1 Val**	**Kaggle**	**Predictions**
StackingClassifier() (using the best models)	0.8249	0.7336	0.77777	0 = 111 1 = 65	0.8626	0.7458	0.82352	0 = 112 1 = 64
RandomForest()	-	-	-	-	0.8523	0.7532	0.82352	0 = 112 1 = 64

FINAL MODEL

We selected the random forest as our predictor using the ***'new'*** dataset because it was the one that gave us the best F1 score of the validation datasets when we were testing it with the *avg_scores* function (0.7532), and because it has lower overfitting among the best models (F1 score in train =0.8523 and Validation = 0.7532).

To double check these conclusions we tested our prediction of the different best models with *Kaggle* and we achieve the same score for the best models that we found (we know that the score seen in

Kaggle is for less than 20 observations, meaning it isn't very accurate), since all scores in *Kaggle* were the same we chose the random forest for its simplicity of complexity, also because it require less time when training the model and because it is simpler to interpret the results obtained compared to the stacking models.

So our final model is: *RandomForestClassifier(random_state = 22, ccp_alpha = 0.003209919868982874, criterion = 'gini', max_features = 8, n_estimators = 100)*

And we can see the feature importances of our final model (Fig. 10), concluding that the 4 features that had more importance are: 'Gender', 'Financial Background' , 'Experience level' and 'Program Sorcery School'.

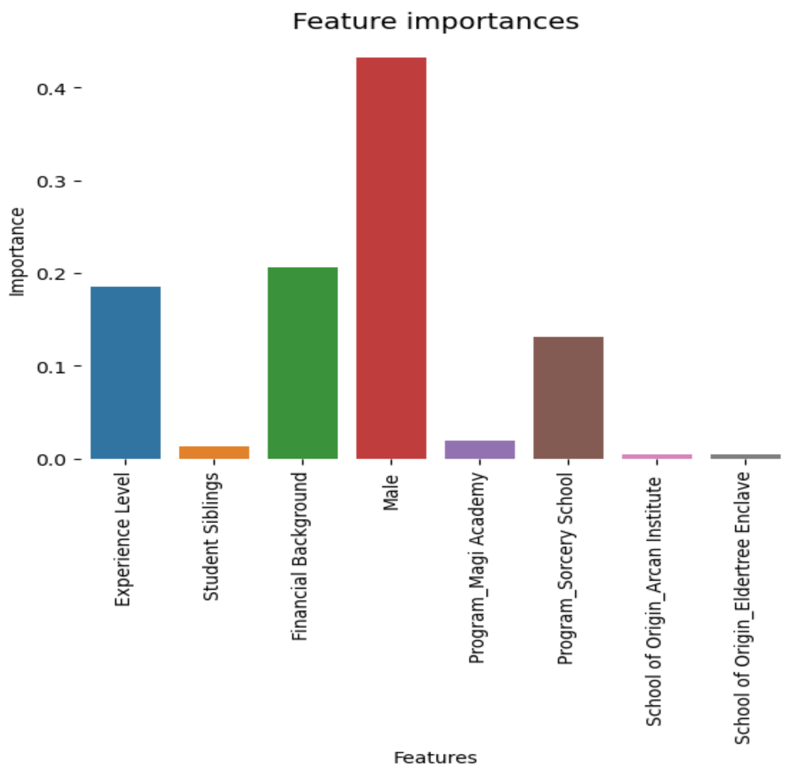


Figure 10 – Feature Importances of Final Model

CONCLUSION

In the introductory section of this report, we identified some of the challenges we would face while developing this project, we believe that through the use of a vast array of options across the preprocessing and modeling stages of this project we were able to overcome most of those challenges or, at the very least, reduce their impact on the final results.

Acknowledging that our work isn't perfect and our thought process may at times not be the best one, allowed us to critically reflect on the work done and perform changes in search of better results, while also allowing us to understand that, in some cases, it is better to perform a concrete tests even if, in theory, making a certain change seems like the wrong decision. This led us to work with an open mind and face any of the challenges presented to us with the required mindset. With this in mind, through exhaustive testing stages and reformulations of previous choices, we finish this process satisfied with the solution we have developed.

For anyone wishing to pick up this work for further development we recommend analyzing the choices we made, as with a different perspective, the decision may be different and seeking more data as the largest challenge we faced was certainly the small size of the dataset(s).

In summary, we are happy with the work we have developed, but recognize that it can always be improved and for that reason we welcome any suggestions, recommendations and criticism.