Machine Learning I

Mystical Admissions

Predictive analysis for enchantment into a school of magic

Group 10
18th December 2023

Lourenço Passeiro - 20221838

Miguel Marques - 20221839

Peter Lekszycki - 20221840

Tomás Gonçalves - 20221894

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

ABSTRACT

Most of the aspiring students fail to secure admission in esteemed schools of magic. This study aims to create a predictive model that can assess and determine a student's admission status using a training set of 713 applications. The model discerns the attributes that distinguish successful applicants from those not suitable for enrolment. However, the process of admitting is complex, and additional factors such as the number of vacancies should be considered. These questions will be answered throughout the report.

We started by looking at and analysing our dataset, to understand it better. We got some insights from the descriptive statistics and gained a sense of the architecture of the data available to us. This initial phase facilitated a deeper comprehension of the data's structure and characteristics at our disposal.

Then, we had to filter, select, and prepare the data to be fit and generalized to be inputted into the models. From every model, we get its best version and optimize its capabilities. In the end, they were evaluated and compared to themselves, to make a choice about our best model and its predictions.

Throughout this project, we navigated the complexities of data preparation, model development, and performance evaluation, culminating in the selection of the most proficient model to forecast admission statuses.

KEYWORDS

Machine Learning; Classification; Dataset; Models; Training; Scores.

INTRODUCTION

In the mystical realm of magical education, young wizards and witches aspire to enter esteemed schools of magic – a gateway to both greatness and transformative magical potential. In this context, it is imperative to emphasize a harsh reality: a staggering two-thirds of aspiring students fail to secure admission. This remarkable statistic underscores the formidable selectivity of these institutions, highlighting the critical need for thorough scrutiny of the admissions process. This is an extremely competitive world, and the admissions process must be taken very seriously.

This study's primary goal is to create a predictive model that can assess and determine a student's admission status. To accomplish this, a training set with information regarding admission status was provided to us. Ten independent variables—five categorical and five numerical—with data on every student are included in this set.

The model, based on students' unique qualifications, experiences, and magical potential, aims to discern the attributes that distinguish successful applicants from those deemed unsuitable for enrolment. Subsequently, our model will be put to the test, using the set of tests that contain the same characteristics as the first, except the last column referring to whether the student enters the school. We will oversee this analysis.

The process of admitting is very complex, and a major doubt is whether the predictive model can carry out a faithful analysis of the problem in question. Will we have complete and sufficient information on each student and those same arrivals conclude their entry into this reputable school? Or should additional factors be considered that were not considered in this study? Such as, for example, the number of vacancies. We will try to be answer this throughout the report.

BACKGROUND

One-hot is an encoding technique used to represent categorical (nominal) variables as numerical values. On each record, the correct class will have a ‘1’ and the remaining ‘0’s. One-hot encoding has the downside of increasing dimensionality, by converting each variable into many.

RandomizedSearch is a method that implements a random search through the model parameters displayed. Unlike GridSearch which tries all possible combinations of parameters in the search space, RandomizedSearch tries a fixed number of parameter combinations. It has the advantage of being more computationally accessible, even more, if the model question is very complex and/or the parameter space is very big.

Precision and recall have inverse relationship. When we try to increase one of them, the other ends up going down. Finding the right balance between precision and recall based on the business or problem requirements is crucial and often involves selecting an appropriate threshold that optimizes the trade-off between these two metrics. By default, it is always 0.5. However, we can change it in order to find the best balance, and in our case, a better f1-score.

METHODOLOGY

INITIAL EXPLORATORY ANALYSIS

Before doing analysis, we started by taking a look at our dataset, to learn from it.

From the descriptive statistics of the datasets, we infer that:

- Both datasets follow the same tendencies and distributions;
- Presence of extreme outliers on the “Financial Background” feature;

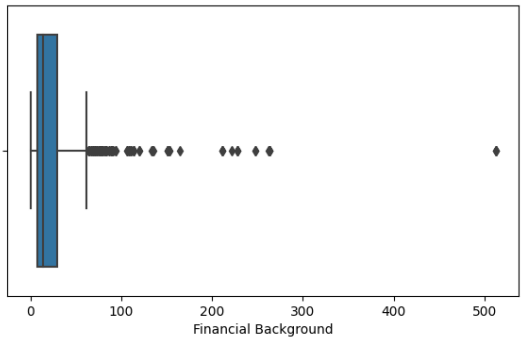

Fig. 1 - Boxplot of the variable ‘Financial Background'

- Missing values on two features:
    - ‘School Dormitory’ with over 70% of missing values on both datasets (training and testing);

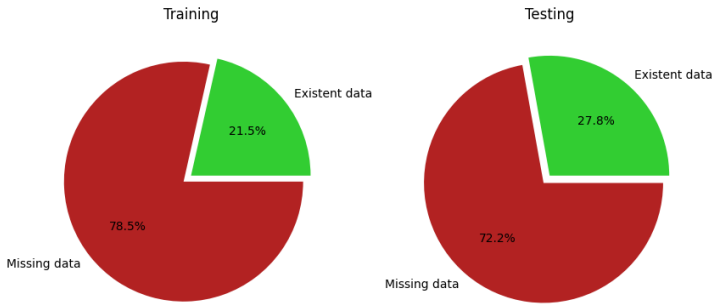

Fig. 2 – Proportion of missing values on both datasets of the variable ‘School Dormitory’

    - ‘Experience Level’ with around 20% of missing values on both datasets.

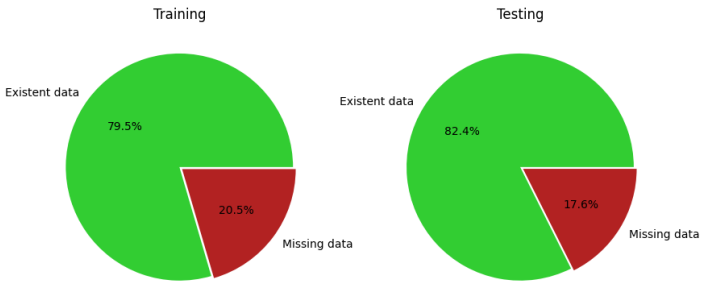

Fig. 3 – Proportion of missing values on both datasets of the variable ‘Experience Level’

- It is more probable to be rejected than to be admitted to the school;

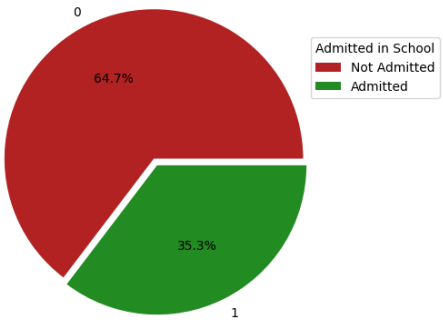

Fig. 4 – Proportion of admissions of the school

DATA PREPROCESSING

This is an essential step in the development of machine learning models, aiming to ensure the quality, consistency, and adequacy of data to the requirements of the algorithms.

Missing values treatment:

On the missing values, we took two different approaches:

- On 'School Dormitory', since a great part of the dataset is missing and its value is lost, we decided to fully eliminate that feature, because we reached the conclusion that they do not provide any useful information, given that the existent sample is very small;
- On the other hand, since 'Experience Level' only lost around 20% of the information, we decided to preserve the rest of its data, by imputing the missing values.

The chosen imputation method was a K-Dimensional Tree Classifier (KD tree). Since our dataset is very low-dimensional (only 9 features at the beginning), we concluded that this imputer would be better than for instance, Ball Tree (better on high-dimensional datasets) or the classic K-Nearest Neighbours.

Before applying the algorithm, we had to encode our categorical variables. We chose to do one-hot encoding, since all the categorical variables are nominal. One-hot encoding has the downside of increasing dimensionality, by converting each variable into many, but, since we only got 5 categorical variables, we thought this would not disturb our dataset.

Student ID	Program	Student Gender	Experience Level	Student Siblings	Student Family	Financial Background	School of Origin	Student Social Influence	Favourite Study Element
1	Sorcery School	male	22.0	1	0	7.2500	Mystic Academy	18	Fire
2	Magi Academy	female	38.0	1	0	71.2833	Eldertree Enclave	7	Fire
3	Sorcery School	female	26.0	0	0	7.9250	Mystic Academy	12	Air
5	Sorcery School	male	35.0	0	0	8.0500	Mystic Academy	12	Air
6	Sorcery School	male	NaN	0	0	8.4583	Arcan Institute	11	Earth

Fig. 5 – Training dataset, before the encoding

Student ID	Experience Level	Student Siblings	Student Family	Financial Background	Student Social Influence	Fire	Air	Earth	Sorcery School	Magi Academy	Mystic Academy	Eldertree Enclave	male
1	22.0	1	0	7.2500	18	1	0	0	1	0	1	0	1
2	38.0	1	0	71.2833	7	1	0	0	0	1	0	1	0
3	26.0	0	0	7.9250	12	0	1	0	1	0	1	0	0
5	35.0	0	0	8.0500	12	0	1	0	1	0	1	0	1
6	NaN	0	0	8.4583	11	0	0	1	1	0	0	0	1

Fig. 6 – Training dataset, after the encoding

Before applying the KNN imputator, we also need to split out the dataset into training and validation. We opted to use 75% of it for training and the rest for validation.

We defined a range of possible neighbours on 40 neighbours around the square root of our dataset's size as the K and tested with the different neighbour weighting methods (uniform and distance).

```
Weights: distance - Best number of neighbors: 9
Mean accuracy in train with 9 neighbors: 1.000000
Mean accuracy in validation with 9 neighbors: 0.726257

Weights: uniform - Best number of neighbors: 4
Mean accuracy in train with 4 neighbors: 0.784644
Mean accuracy in validation with 4 neighbors: 0.726257
```

Fig. 7 – Training and validation scores of the optimal K, for both weighting methods

- We got different Ks on the two methods. Although the validation scores were the same, we noticed a tendency to overfit using 'distance', since it got a perfect score on training. Given this, we decided to go with uniform weights.

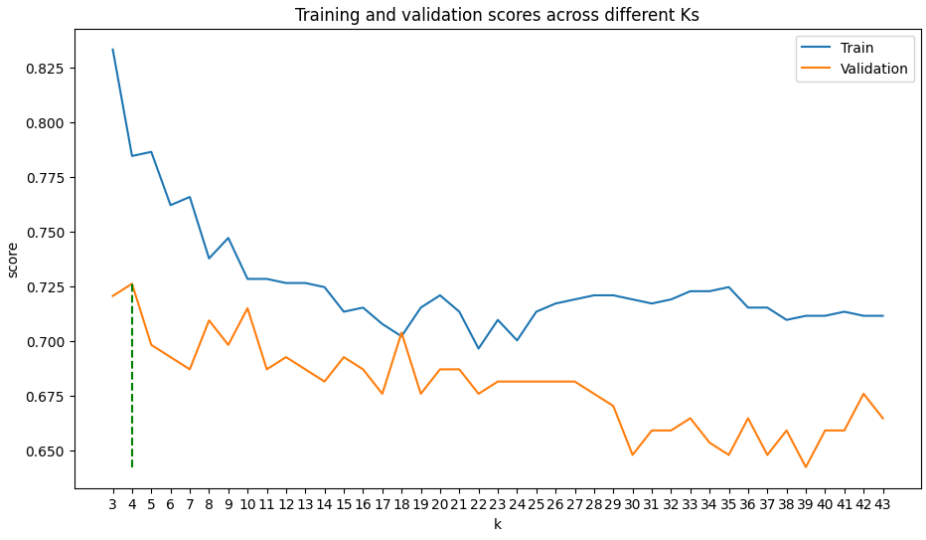

Fig. 8 – Training and validation scores for our range of Ks, on 'uniform' weighting

We imputed the missing values with 4 neighbours and uniform weighting.

<u>Feature scaling</u>

Before selecting the features we will use in the model, we have to scale them. We scaled our data on three scalers - Standard, MinMax, and Robust. Instead of choosing only one scaler a priori and using it for every model, we will use them according to each model. We chose this since this way we could maximize the performance of each model, using the best scaler for each case.

Feature selection

We started by checking the (spearman) correlation between all numerical variables and the target. There were no highly correlated variables to the target, nor between themselves.

For the numerical variables, we used two wrapper methods: RFE and Lasso. They both are going to vote either to keep or delete some features.

On RFE (Recursive Feature Elimination), we used a basic logistic regression at its base model and computed the optimal number of features – 4. Given this, it voted to delete 'Student Family'. On the other hand, Lasso voted to keep all variables.

	RFE	Lasso	What we should do?
Experience Level	Keep	Keep	Keep
Student Siblings	Keep	Keep	Keep
Student Family	Discard	Keep	Try with and without
Financial Background	Keep	Keep	Keep
Student Social Influence	Keep	Keep	Keep

Fig. 11 – Votes from each method, and our decisions

All of the features were unanimous, except "Student Family". So, we will compare our models' performance with, and without it, to decide if we should use it.

Now, for the categorical features, we used chi-square to check the importance of the categorical independent variables in the target. The result was to keep every variable except the 'Favorite Study Element'.

Student ID	Experience Level	Student Siblings	Student Family	Financial Background	Student Social Influence	Sorcery School	Magi Academy	Mystic Academy	Eldertree Enclave	male
494	3.157269	-0.520855	-0.456980	0.387600	0.611489	1.0	0.0	1.0	0.0	1.0
599	-0.244929	-0.520855	-0.456980	-0.475493	1.195369	0.0	1.0	1.0	0.0	1.0
666	0.192225	1.547076	-0.456980	0.877454	0.319549	0.0	0.0	0.0	1.0	1.0
260	1.560707	-0.520855	0.862086	-0.092218	1.341339	0.0	0.0	0.0	1.0	0.0
729	-0.339962	0.513110	-0.456980	-0.092218	-0.556272	0.0	0.0	0.0	1.0	1.0

Fig. 12 – Our final training dataset, after feature scaling (standard) and selection

MODEL SELECTION

We decided to develop the following models: Logistic Regression, Classification Trees, KNN, Naïve Bayes, Neural Networks, Bagging, Random Forests, Boosting and Stacking. On most of the models, we defined a base one and performed a parameter search (with f1-score as its scoring method) in order to retrieve the optimal parameters for each scaled dataset. We always prioritized grid searches, but when the search would be too computationally expensive (due to the model's

expensiveness/its multiple parameters/our slow computers), we opted to use random searches. This was the case with neural networks and ensemble classifiers.

After it, we computed their scores on training, and validation sets with a stratified k-fold of the training dataset with 10 splits. We repeated the same process, to a dataset without 'Student Family'.

Logistic Regression

```
'penalty' : ['l1','l2'],
'C'       : np.logspace(-3,3,7),
'solver'  : ['newton-cg', 'lbfgs', 'liblinear']}
```

Fig. 13 – Parameter space of the logistic regression GridSearch

Classification Trees

Since decision trees are not sensitive to variance in the data, feature scaling is not required here.

```
'max_depth': [4,5,6, 7, 8, 9],
'criterion': ['gini', 'entropy'],
'splitter': ['random', 'best'],
'min_samples_split': [20, 22, 24, 28],
'min_samples_leaf' : [3, 4, 5, 6, 7, 8],
'max_features': [6,7,8,9,None]}
```

Fig 14 - Parameter space of the classification tree GridSearch

K-Nearest Neighbors

```
{'n_neighbors':k,
 'metric': ['manhattan', 'euclidean', 'minkowski']}
```

Fig 15 – Parameter space of the KNN GridSearch

Naïve Bayes

This algorithm also does not require feature scaling since it is not dependent on distance.

Neural networks

We started by computing a loss curve, to see on after which number of iterations we should stop developing the model.

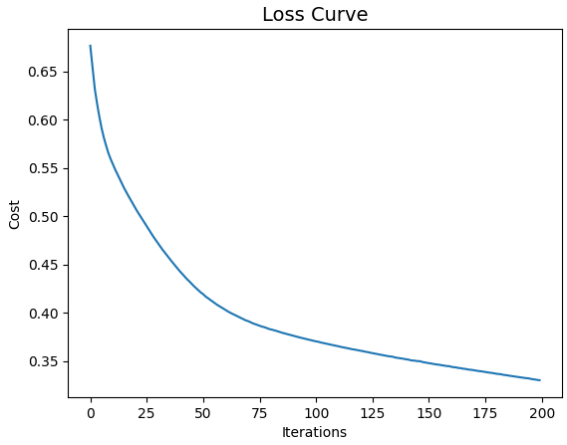

Fig 16 – Loss curve of a base neural network

After around 100/125 iterations, the increase in computational complexity stopped being lucrative. Here, as said before, we had to opted to do a random search.

```
'hidden_layer_sizes': [(80,40,20), (100,50,25), (120,60, 30)],
'max_iter': [100, 115, 130],
'solver':['lbfgs', 'sgd', 'adam'],
'learning_rate':['constant', 'invscaling', 'adaptive'],
'learning_rate_init':[0.0005,0.001, 0.002],
'activation':['identity', 'logistic', 'tanh', 'relu']}
```

Fig 17 – Parameter space of the neural network RandomizedSearch

	Scaling
Logistic Regression	Standard/Robust
Decision Tree Classifier	Indifferent
KNN Classifier	Standard
Naive Bayes	Indifferent
Neural Networks	Standard

	With/Without Student Family
Logistic Regression	Indifferent
Decision Tree Classifier	-
KNN Classifier	With
Naive Bayes	Indifferent
Neural Networks	With

Fig 18 – Individual models, their best scalers, and their preference about the variable 'Student Family'

Since every model is indifferent/prefers the Standard scaler, we use it on the ensemble classifiers. Although in class we always tested the best parameter values individually, we decided to do a randomized search, because testing parameters independently could be suboptimal, since, for instance, the optimal number of estimators can be 20 if all the other parameters were default, but the optimal combination of parameters did not have to have necessarily 20 estimators. Given this, doing some random samplings of the multiple parameters would be better.

Bagging

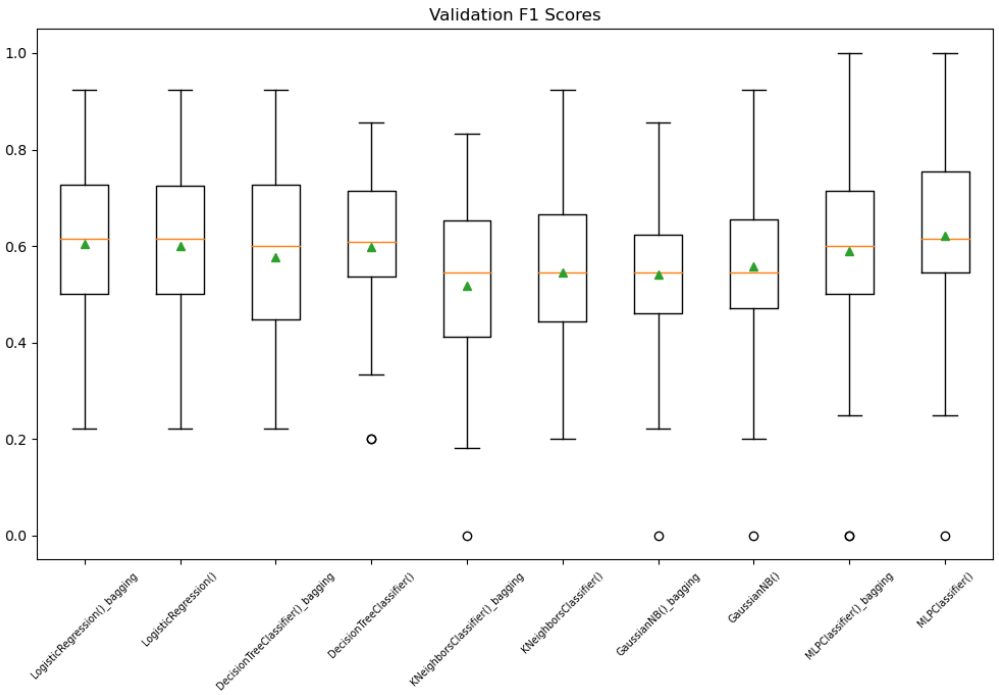

Fig 19 – Validation scores of the individual models, alone and bagging

We can see that every model got better results individually than bagging, except for logistic regression. Given this, we will explore bagging logistic regression.

```
'n_estimators':[20, 30, 50, 100, 200],
'max_features':[0.2,0.4,0.6,0.8,1.0],
'max_samples':[0.2,0.4,0.6,0.8,1.0],
'bootstrap':[True, False],
'bootstrap_features':[True, False],
'oob_score':[True, False]}
```

Fig 20 – Bagging logistic regressions RandomizedSearch parameter space

Random Forests

```
'max_depth':[3, 4, 5, 6, 7, 8, 9],
'n_estimators':[125, 150, 175, 200, 250, 300],
'max_features':[4,5,6,7,8,9,None],
'min_samples_leaf':[2, 3, 4, 5, 6, 7, 8],
'min_samples_split':[2 ,3, 4, 5, 6, 7, 8]}
```

Fig 21 – Random Forest RandomizedSearch parameter space

Boosting

Here, we explored both AdaBoost and GradientBoosting. On AdaBoost, we started to test the boosting performance of a base logistic regression and a base decision tree.

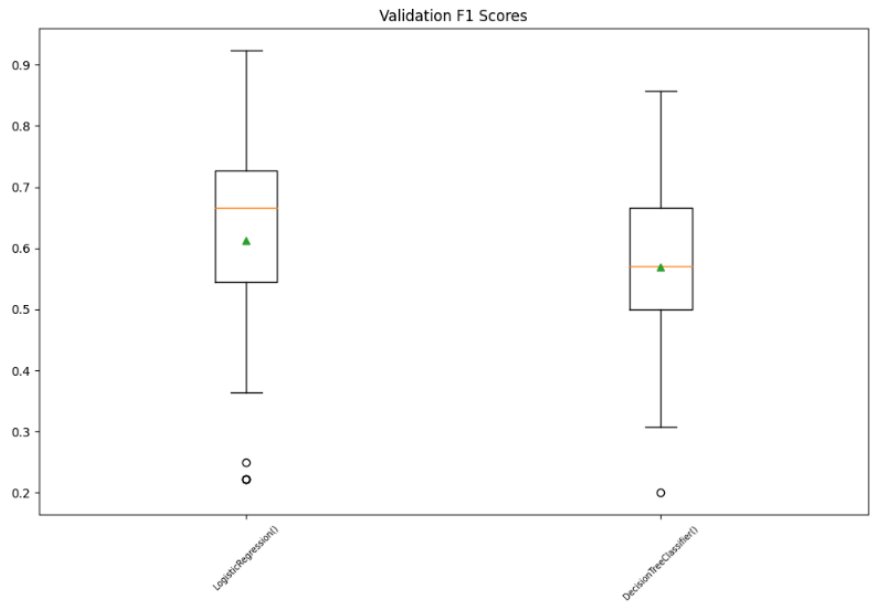

Fig. 22 – AdaBoost performances on validation with base estimators as a Logistic Regression and Decision Tree

As we can see, the logistic regression got a really better validation score.

```
'n_estimators':[2,5,10,20,30,50,100,150],
'learning_rate':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
'algorithm':["SAMME","SAMME.R"]}
```

Fig 23 – AdaBoost logistic regression GridSearch parameter space

On gradient boosting, we used a base decision tree.

```
'n_estimators':[2,5,10,20,30,50,100,150],
'learning_rate':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
'subsample':[0.2,0.4,0.6,0.8,1.0],
'max_features':[5,6,7,8,None],
'max_depth':[5, 6, 7, 8]}
```

Fig 24 – GradientBoosting RandomizedSearch parameter space

Stacking

In order to choose our base estimators, we computed and compared the f1-scores on validation data.

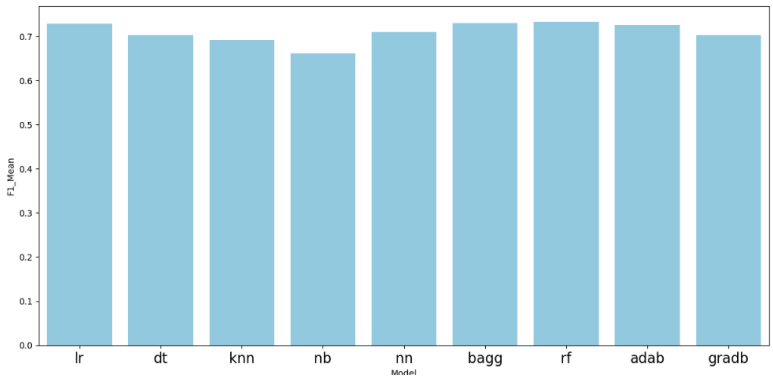

Fig 25 – Validation f1-score of our models

Based on this, the best base models are our Logistic Regression, Bagging Logistic Regression, Random Forests, and AdaBoost. We used them as the base estimators of our stacking classifier. After trying some combinations of them, we got the best results while using the four of them as base estimators. In order to find our best final classifier, we tried all of our models.

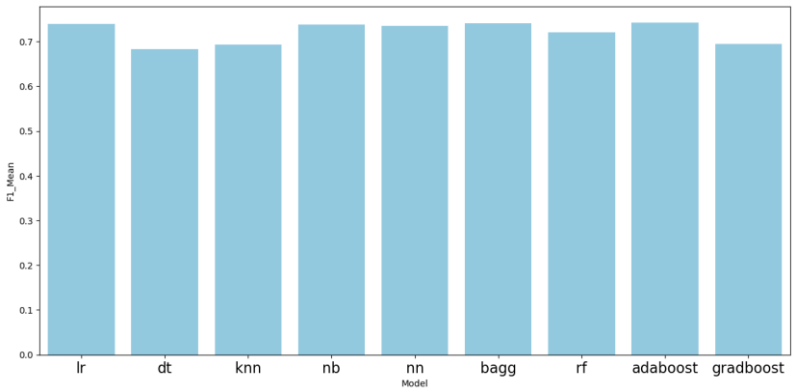

Fig 26 – Validation f1-score of our models as final estimators.

The best final estimator with the selected base estimators was the AdaBoost model.

After getting the first scores of our optimized models, we noticed a great disparity between the precision and recall scores. We found out that this could be because of our classification threshold, that by default is 0.5. To overcome this, we computed the optimal combination of the precision-recall trade-off, where the f1-score was at its highest point possible and use the present threshold. Since that was the recall that was always down, the classification threshold must decrease, to decrease the high number of false negatives we were getting.

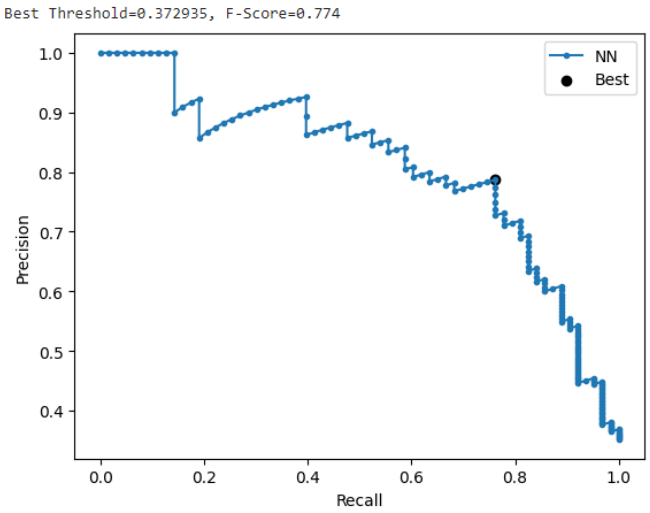

Fig 27 – Neural networks precision-recall trade-off

RESULTS

We based our model's evaluation on accuracy, precision, recall, f1-score, and kaggle score. We valued more the last two. We used a 5-fold cross-validation on the training dataset, to get more robust scores.

	Accuracy	Precision	Recall	F1	Kaggle
Logistic Regression	0.809257	0.737705	0.714286	0.725806	0.66666
Decision Tree Classifier	0.805049	0.718147	0.738095	0.727984	0.70000
KNN Classifier	0.753156	0.611765	0.825397	0.702703	0.00000
Naive Bayes	0.757363	0.618619	0.817460	0.704274	0.00000
Neural Networks	0.823282	0.769231	0.714286	0.740741	0.66666
Bagging LR	0.809257	0.745763	0.698413	0.721311	0.78260
Random Forest	0.814867	0.741935	0.730159	0.736000	0.84210
AdaBoost	0.814867	0.760870	0.694444	0.726141	0.72727
GradientBoost	0.788219	0.689139	0.730159	0.726141	0.80000
Stacking	0.816269	0.737255	0.746032	0.741617	0.85714

	Accuracy	Precision	Recall	F1	Kaggle	Total
Stacking	2.0	6.0	3.0	1.0	1.0	2.0
Random Forest	3.5	4.0	5.5	3.0	2.0	5.0
GradientBoost	8.0	8.0	5.5	5.5	3.0	8.5
Neural Networks	1.0	1.0	7.5	2.0	7.5	9.5
Decision Tree Classifier	7.0	7.0	4.0	4.0	6.0	10.0
AdaBoost	3.5	2.0	10.0	5.5	5.0	10.5
Bagging LR	5.5	3.0	9.0	8.0	4.0	12.0
Logistic Regression	5.5	5.0	7.5	7.0	7.5	14.5
Naive Bayes	9.0	9.0	2.0	9.0	9.5	18.5
KNN Classifier	10.0	10.0	1.0	10.0	9.5	19.5

Fig 28 – Final scores of our models and their ranks

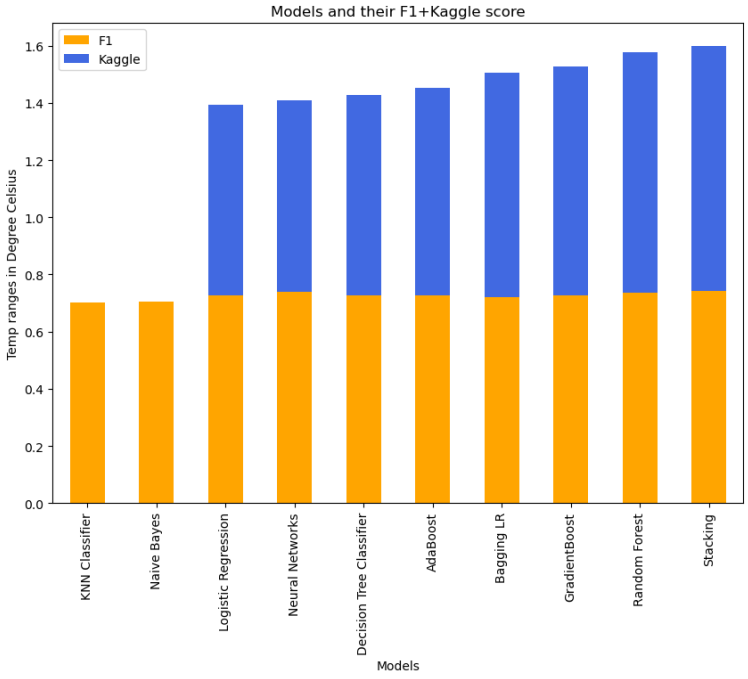

Fig 29 – Stacked bar chart of the scores

As expected, the ensemble models dominated the ranking of models. Neural networks had a great f1-score, but unfortunately, it did not translate well to the test dataset. Our best model is, by far, the Stacking Classifier. It had the best Kaggle and f1 score and also got one of the best accuracies.

Now, our best prediction on Kaggle is 0.85714. However, we got a 0.9 score there, achieved with Random Forests. Unfortunately, when trying to improve it, we happened to lose the parameters and classification threshold that got us there.

Given all this, the Stacking Classifier is our choice and we used it on the test set to get our final predictions.

DISCUSSION

With this study, we were able to get our best model (Stacking), which got the highest score on Kaggle. If we look at all the other models, none of them come close to it. The one that is closest to it is Random Forests which got a score of 0.8421. Besides these results, we can see that KNN Classifier and Naive Bayes did not have a final score on Kaggle. If we look at all their scores in the scoring methods (Accuracy, Precision, Recall, F1), we can conclude that those models were clearly the worst ones, and we did not want to waste a Kaggle submission on them. Even though our best model was not the best in all four scoring methods, when we ranked the models based on the F1-score and the Kaggle score, we were able to see that Stacking was still the best one out of the others and that Naïve Bayes was the worst one.

CONCLUSION

Finally, our thorough examination of the mystical realm of magical education admissions process gave us valuable insights. Because of the competitive nature of these prestigious schools, a strong predictive model is required to identify the characteristics that distinguish successful applicants from the rest. With this project we gained some practice on developing predictive methods, and now we have a more tangible knowledge about what problems may arise, and how to deal with them. Our investigation revealed that the Stacking Classifier model, was probably the most effective tool for predicting admission outcomes, given the characteristics of our data.

ANNEXES

https://towardsdatascience.com/preprocessing-encode-and-knn-impute-all-categorical-features-fast-b05f50b4dfaa
https://scikit-learn.org/stable/modules/model_evaluation.html
https://www.evidentlyai.com/classification-metrics/classification-threshold
https://datascience-george.medium.com/the-precision-recall-trade-off-aa295faba140
https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e
https://blog.paperspace.com/bagging-ensemble-methods/
https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/
https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

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

Universidade Nova de Lisboa