AA1 Project: Voter turnout prediction

Benet Ramió and Pau Amargant

GCED, UPC.

Abstract

This project has applied various machine learning techniques to predict whether an eligible voter participated in the 2020 US presidential elections using a dataset of survey responses from the Current Population Survey (CPS). Throughout the project, the main challenges have been the imbalanced distribution of the target variable and the complexity of voting behaviour. To address these challenges, different preprocessing steps have been performed and integrated into pipelines. Several generative and discriminative classifiers have been fitted and compared using cross validation and metrics such as F1-score. The best performing model has been a Random Forest Classifier with a F1-macro score of 0.688.

Contents

1 Introduction

This project aims to apply the various machine learning techniques and methodologies acquired during this course to a real-world dataset. By doing so, we intend to gain a better understanding of how these algorithms perform when applied to real-world data. The project encompasses the entire cycle of a machine learning project; from raw data to making prediction with a suitable model. Therefore, it is our intention to not only apply different algorithms and methods, but also to show the difficulties encountered and how they can be resolved.

Our objective is to predict whether an eligible voter participated in the 2020 US presidential election. With this task and mind, a dataset consisting of survey responses has been used. Emphasis has been placed on detecting non-voters. A real world application of this project could be to detect individuals in which to focus a campaign to increase voter turnout. Such application would require being able to detect the non-voters while at the same time reducing the number of false positives in order to be efficient.

Our dataset has posed two significant challenges. Firstly, the distribution of the target variable is highly imbalanced, with approximately 80% of the participants voting against only 20% that did not. Imbalanced datasets often lead to biased models that prioritize the majority class, resulting in poor performance and misclassification of the minority class. Therefore, we need to employ special techniques such as oversampling, undersampling, or other methods to address this issue and ensure that our model can effectively capture patterns and make accurate predictions for both voted and non-voted individuals.

Secondly, predicting voting behaviour is inherently complex due to its nature as a social problem. Unlike precise scientific phenomena, voting decisions are influenced by a multitude of subjective factors, including personal beliefs, values, socioeconomic status, political affiliations, and social dynamics. These factors intervene in a lot of ways, making it challenging to establish clear causal relationships and formulate a deterministic model. Moreover, voting behaviour is shaped by historical, cultural, and contextual factors specific to each election cycle, which further adds to the complexity. The variability and subjectivity inherent in voting behaviour make it more difficult to develop an exact science-based predictive model.

To tackle these challenges, it is not only necessary to try different models and ensembles which can help uncover complex relationships and nonlinear dependencies within the data, but also to do an extensive task of preprocessing, carefully selecting relevant features and making the appropriate transformations in each of them.

1.1 About our dataset

The dataset for this project is sourced from *Integrated Public Use Microdata Series* (IPUMS) [1], which provides individual-level microdata samples from various surveys conducted in the United States and internationally. IPUMS ensures that the individual responses from the Census are presented in a consistent format, enabling comparison across multiple surveys editions. Furthermore, through its web interface and API, datasets can be customized in order to include specific variables and samples.

For this particular project, we utilized the Current Population Survey (CPS), a monthly survey involving around 60.000 American households. The surveys collect information on education, labour force status, demographics, and socio-economical aspects of the population.

The complete list of variables included in the population survey can be found on the appendix 8.1. It is important to note that some variables include their sample weight, which is part of the sampling design of the surveys. These weights can be used to account for the fact that the individuals in the survey may not be fully representative of the general US population. Nonetheless, in order to avoid possible leakage issues, the weights have not been used. Generally speaking, the survey included basic information such as *AGE*, *RACE*, *MARITAL STATUS* together with more specific information about the origin and place of residence. Information about the individual's education, their household income and their family structure is also included. Furthermore, the duration of residence at their current address, which might be important, is also included.

1.2 Related works

It is necessary to note that the objective of this project is related to the Adult income [2] classification problem, which is a known classification problem in data science and is commonly used in books and examples. Given the similarity between the problems, we might expect to encounter similar challenged to the ones found when working with the dataset and even though the structure of the target variable differs, it is worth examining the most successful classification methods for that problem. Preprocessing in the adult census problem usually includes normalizing and scaling the numerical variables, one-hot encoding the categorical variables and using oversampling methods in order to account for the class imbalance. [3] When it comes to the possible classification methods, random trees and logistic regression have been show to perform well.

2 Methodology

2.1 General methodology

The methodology used across this project has focused on making each step in the process and the final results easily reproducible and easily integrable into more complex machine learning processes. Our objective is to predict non-voters, and it is worth noting that the answers to voted have been coded as *YES*:1, *NO*:0.

Throughout the project, only the training split of the dataset has been used, with the Test split being used only in the final model evaluation. In order to select the best model, cross-validation has been used together with a validation set. Both the preprocessing and model fitting have been included in the cross-validation process. Cross-validation has been used to select the best hyperparameters and preprocessing pipeline for each classification model. The validation split has been fitted only one time per classification model and used to compare the performance of the different type of models.

To begin with, the data set is preprocessed in order to ensure its suitability for the models. This step, which is described in 4, has been implemented through *scikit-learn* and *imblearn* transformers joined together with a column transformer, which applies the preprocessing pipeline to each feature, taking into account that different sets of features require a different preprocessing. As explained in 4 the train-test split is made after the initial data cleaning has been performed.

To evaluate the performance of the different models and preprocessing steps, the classification models have been integrated with the preprocessing in a pipeline and grid search cross-validation has been used to search the best hyperparameters. We note that imbalanced learn implements the resampling methods in such a way that it is only used when training the model. By implementing the whole process in a single pipeline, cross-validation can be used without risking leakage, as the Scikit-learn cross-validation function is able to fit the preprocessing on only the training data in each of the splits.

2.2 Evaluation Metrics

Evaluation metrics are crucial when assessing the performance of classification models, especially in imbalanced classification problems such as ours. As explained in 1 our focus is to identify people who did not vote, which is the minority class. Therefore, standard metrics such as accuracy or error rate can be misleading and during the hyperparameter and model selection steps in 5 alternative metrics must be used.

In order to select the optimal hyperparameters for each model, the F1-score for the *NOT VOTED* class has been used. This metric, given by formula 2.2 , achieves a good balance between precision and recall.

$$F_1(\text{class} = 0) = \frac{2 \cdot \text{precission}_0 \cdot \text{recall}_0}{\text{precission}_0 + \text{recall}_0} \tag{1}$$

Given our objectives, we are interested in reducing the amount of false negatives. Nevertheless, given the imbalanced nature of our data, precision must also be taken into account, otherwise the false positive rate can quickly skyrocket.

When analysing the fitted models other metrics such as F1 Macro, average precision (AP) macro and AUC score have been also been analysed.

3 Description of the dataset

It is necessary to explore the dataset in order to understand what transformations and changes should be made and how the variables are related between them and with the target variable. In this analysis, we focus on those variables which have been used throughout the process. As explained in 4, some variables have not been used due to various issues and consequently have been left out of this analysis. It is worth noting that the data cleaning step 4.1 had been performed before plotting the data.

As has already been mentioned, our dataset is not balanced. Figure 1 shows the histogram of the target variable; 80% of the samples correspond to *YES (1)* and 20% to *NO (0)*. It is worth noting that the recorded turnout rate in the elections was of 66%. One might argue that people who did not vote are also more prone to not answering the population survey.

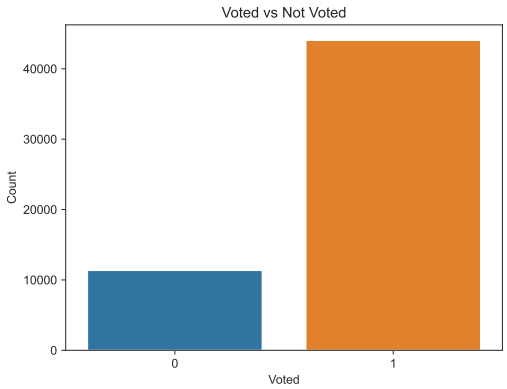


Fig. 1: Histogram of *VOTED*

We place our attention on the numerical variables. As shown in figure 9 in the appendix, we observe that most of them have their density concentrated in a few values. We believe that uncommon values might provide valuable information about voting and therefore they are not removed or imputed. Nevertheless, scaling has been used. For example, *UHRSWORKT* has a peak at 0 which corresponds to people who are not in the labour force and other at the most common hours per week (20,40 and 80). As will be explained in 4.1.2 the *YRIMMIG* variable has been transformed into a variable that accounts for the percentage of their life that a person has lived in the US.

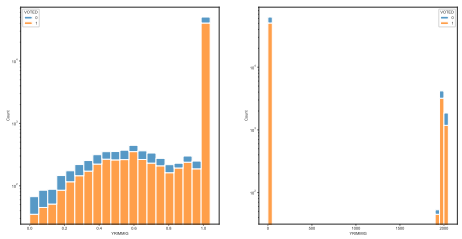


Fig. 2: Histogram of the transformed *YRIMMIG* on the left and the original variable on the right (log scale)

A bar plot of the categorical variables is shown in figure 10. We observe that some variables have few categories, while others have a great number of them. Furthermore, some categories are very uncommon.

In figure 11 the histograms of the categorical variables in which their categories have an order are shown. We note that as instructed in the lab sessions, AGE has been interpreted as a categorical variable.

In order to study the relation between the variables, figure 3 shows the correlation between the numerical variables in the dataset. We observe that, as expected, the variables which describe families are highly correlated and, while being outside the scope of this course, dimensionality reduction could be useful. The last row and column show the correlation with the target variable. We observe that there is no numerical variable with a significant correlation.

Even though they are categorical, the correlation matrix between the ordered categorical and the target variable has also been plotted in figure 3. We observe that there exist some correlation between some variables and the target.

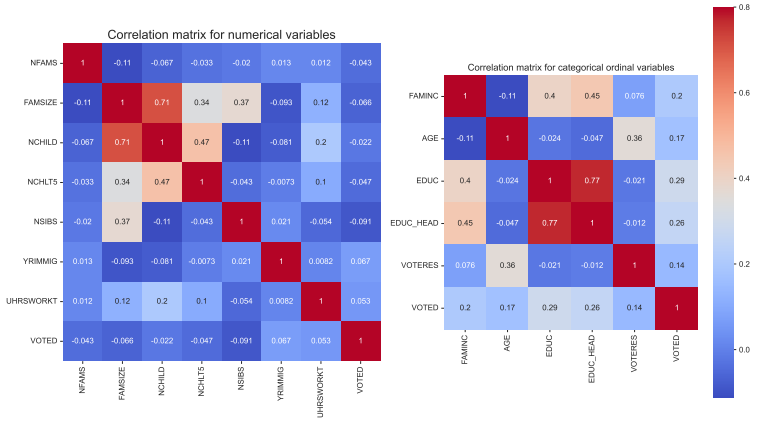


Fig. 3: Correlation matrix of the numerical variables on the left and the ordered categorical on the right

It is worth noting that the correlation matrix is not suitable for categorical variables, and alternative methods are outside the scope of this project. Nevertheless, by interpreting them as numerical, their correlation with the target has been calculated and plotted in figure 4. We observe that marital some variables have relatively high positive and negatives correlations with the target variable. A higher education level seems to be positively correlated while marital status is negatively correlated.

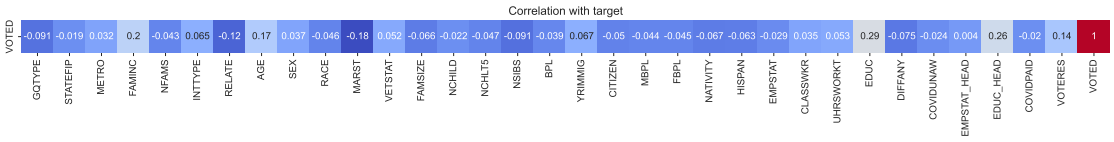


Fig. 4: Correlation with the target variable

4 Preprocessing

The preprocessing of the data has been divided into 5 steps:

1. Data cleaning
2. Numerical features scaling and transformations
3. Imputing missing values
4. Handling of the categorial variables
5. Balancing the dataset

Figure 5 shows a diagram of the different preprocessing types that can be applied to our data. In the following subsections, each of the step will be justified and described. The different possible combinations are described in 4.5 and 13.

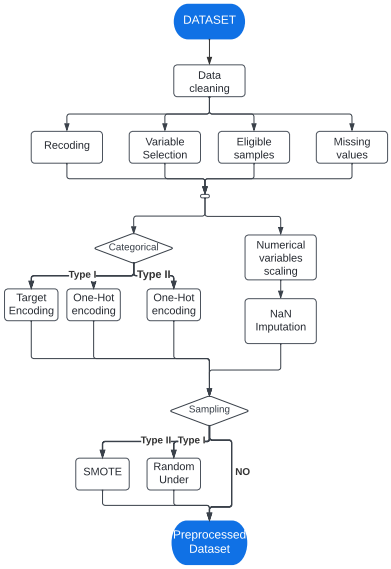


Fig. 5: Diagram of the preprocessing

4.1 Data cleaning

The initial dataset consists of 53 features and, 112037 observations. Some features have been removed due to the large amount of missing values they contain. Only the individuals who were eligible voters and answered either *YES* or *NO* the the *VOTED* question have been included. In the appendix 8.2 more details about the final features and samples selected can be found. This step has been uniformly applied to the whole dataset, including the testing split, as it does not produce leakage.

4.1.1 Feature and observations

To begin with, features have been removed in accordance to three main reasons; missing values, too many categories and possible leakage. Furthermore, the statistical weights, which are used when performing inference, have been removed.

Individuals usually take part in the monthly population survey for 4 consecutive months, with a part of the respondents being replaced at each month. Furthermore, some participants are included in the *Earner Study*, in which they answer further questions about topics of interest to the labour market during various months. The variables of the study are only answered by the individuals who also participated in the May survey, and therefore there are a lot of NaN answers. For example, the *EARNWEEK* variable contains 100451 missing answers.

Other features such as *METFIPS* (metropolitan area code) and *OCC* (occupation code) have been eliminated because they have a large amount of categories which, due to how they are coded, can not easily be grouped into larger categories. Due to the size of the dataset, it has been reasoned that these categories would only provide noise and that other categories which answer similar questions are less granulated and can be used instead.

On the other hand, some features have been eliminated because not all the individuals were eligible to answer them and there were many missing answers. Numerical features such as *ELDCH* (age of the eldest child) or *YNGCH* (age of the youngest child) could only be answered by those individuals who had a child and those who did not were coded as NIU (not in universe).

The target variable is part of the November voting supplement (a special survey) which not only includes whether an individual voted but also the reason why, the method used and the duration of residence at their current address. The reason they did or did not vote and how they voted have been removed, as they can indirectly give the answer to the target variable.

Some variables have been recoded to reduce the number of categories or change the meaning of the feature. In the cases of *BPL* (birthplace) and similars, similar countries have arbitrarily been joined into the same category. In the case of *UHRSWORKT* the values 999 (not in universe) has been interpreted as that the person does not work and set as 0 hours and the value 997 (hours vary) as a NaN in order to be imputed. In the case of the *VOTERES*, the variable contains NaN values of unknown origin, which have also imputed. Furthermore, this variable has been recoded so that the categories reflect an order.

4.1.2 Variable transformation

In the case of the *YRIMMIG* variable, which is the year a person immigrated to the US or 0 if the person was born in the US, it has been converted into a variable that accounts for the percentage of their life that a person has lived in the US.

$$PCTG = \begin{cases} 1 & \text{US native} \\ \frac{YRIMMIG}{AGE} & \text{otherwise} \end{cases} \tag{2}$$

4.2 Numerical Features scaling and imputation

The preprocessing used on the numerical features has been relatively simple. All of them have first been scaled, and after that the necessary imputations have been made. A Robust Scaler [4] has been used. This scaler removes the median and scales according to the IQR range. This scaler is less affected by extreme values than the Standard Scaler. In order to impute missing values, an iterative imputer [5] has been used. An iterative imputer takes advantage of the presence of multiple variables by using them to predict the missing values. In turns (Round-Robin fashion) the missing values of one variable are estimated using an estimator and the values are updated. This process is repeated for all the variables until the imputation converges or a fixed number of iterations is reached.

4.3 Encoding of the categorical variables

Due to the large amount of categorical features present in the dataset, how they are preprocessed has a great importance. Because of this reason, different methods have been proposed, and two different methods have been used when estimating the performance of different estimators in order to find the best estimator-encoding combination.

The first method has focused on keeping the number of variables in check. Therefore, one-hot encoding has been used only on the variables which have few categories. Meanwhile, target encoding [6] has been applied to those variables with many possible categories in order to avoid increasing the number of features to more than 700. With this method, the categories are replaced with a blend of the posterior probability of the target given that categorical value and the prior probability of the target across all the categories.

On the other hand, the second method applies one-hot encoding to all the variables, which increases the number of features to hundreds. A possible way to reduce it could be to apply multiple

correspondence analysis [7], but the available libraries in Python have proved problematic when they encounter a value which is in a category that was not present in the original dataset.

4.4 Balancing of the dataset

As has been explained in 3 the dataset is not balanced, with 80% of *YES* voted values against 20% of not voted. Most classifiers require that both classes be balanced in order to perform well. The *imblearn* package has been used to oversample and undersample the training split.

On the one hand, the majority class has been randomly undersampled using *imbalanced learn* [8]. On the other hand, through oversampling, new observations of the minority class have been generated using SMOTE[8]. The inner workings of the algorithm are outside the scope of this project but it is important to note that SMOTE does not just copy existing observation but it generates new examples that that combine characteristics of existing observations. Therefore three different options have been used; oversampling, undersampling and no resampling. The three methods have been tested when fitting models in order to find the best combination.

4.5 Preprocessing pipelines

To sum up, during the model fitting stage 6 different preprocessing pipelines have been used:

- Pipeline **AA**: One hot encoding to variables with few categories and target encoding to variables with many categories + random undersampling
- Pipeline **AB**: One hot encoding to variables with few categories and target encoding to variables with many categories + SMOTE oversampling
- Pipeline **A0**: One hot encoding to variables with few categories and target encoding to variables with many categories
- Pipeline **BA**: One hot encoding to all categorical variables + random undersampling
- Pipeline **BB**: One hot encoding to all categorical variables + SMOTE oversampling
- Pipeline **B0**: One hot encoding to all categorical variables

5 Modeling

In this section, we present the outcomes of fitting multiple models to the data. Before starting this analysis, it must be emphasized that Grid Search was used to determine the optimal hyperparameters, where applicable. Furthermore, as explained in the preprocessing stage (4), different pipeline combinations were also explored. The hyperparameters fitted can be found in appendix 8.7 while the best ones as well as the optimal preprocessing in appendix 8.8.

5.1 Generative Classifiers

We begin with Generative classifiers, which are a class of models that try to understand the underlying structure and patterns of the data, enabling them to generate new samples that possess similar characteristics. These models use the joint probability distribution of the observable and target variables in order to, through Bayes rule, obtain the posterior probability. In this project, three generative classifiers have been fitted: Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Naive Bayes.

5.1.1 Linear Discriminant Analysis (LDA)

LDA (Linear Discriminant Analysis) is a classification algorithm which assumes that the conditional probability function is a multivariate Gaussian distribution. Each class is assumed to have it's own mean μ while the covariance matrix is shared. Different assumptions about the nature of the covariance matrix can also be made. LDA aims to minimizes within-class variance and maximize between-class variance.

The table 1 presents the best mean metric values for the cross-validation tests together with the validation split, while the corresponding validation split confusion matrix is displayed in appendix figure 14. It is worth noting that performance might not be great be due to the normality and homocedasticity assumptions not being satisfied.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.506	0.798	0.652	0.720	0.392	0.876	0.713
Val	0.509	0.792	0.650	0.723	0.392	0.872	0.713

Table 1: Metrics of the LDA model

5.1.2 Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) is a classifier algorithm that extends LDA by relaxing the assumption of equal covariance matrices across classes. Unlike LDA, QDA allows each class to have its own covariance matrix, which can capture more complex relationships between features. This flexibility comes at the cost of increased model complexity, since a much higher number of parameters has to be estimated. A regularization parameter which adds continuity between each covariance matrix and the pooled covariance can be used.

The cross-validation and validation split metrics can be seen in table 2 and the confusion matrix in appendix figure 15. The moderate performance we see in QDA could be due to its assumption of separate multivariate Gaussian distributions for each class, which may not accurately represent the underlying data distribution, or because the data is not large enough to properly estimate the number of parameters.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.490	0.807	0.648	0.661	0.389	0.868	0.698
Val	0.494	0.805	0.649	0.657	0.396	0.864	0.696

Table 2: Metrics of the QDA model

5.1.3 Gaussian Naive Bayes

Naive Bayes is a probabilistic algorithm which assumes that features are conditionally independent given the class label. It calculates the posterior probability and assigns an example to the class which achieves the higher valuable. In order to account for categories not present in the training data, Naive Bayes usually utilizes Laplace Smoothing.

The cross-validation and validation metrics of the best fit can be seen in table 3, and the confusion matrix in appendix figure 16. GNB shows a moderate to low performance, probably due to the assumption of feature independence, which may not hold true for this dataset.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.496	0.816	0.656	0.651	0.410	0.869	0.701
Val	0.490	0.807	0.648	0.645	0.396	0.862	0.692

Table 3: Metrics of the Gaussian Naive Bayes model

5.2 Discriminative Classifiers

We now focus on discriminative classifiers. Unlike generative classifiers, discriminative concentrate on learning the decision boundary that separates different classes. The posterior probability density is directly estimated through various methods and used to assign points to a class. In this project, eight discriminative classifiers have been tried: K-Nearest Neighbors (KNN), Logistic Regression, Decision Tree, Random Forest, Extra Trees, Adaboost, Gradient Descent, Catboost and Linear Support Vector Machines (Linear SVM).

5.2.1 K-Nearest Neighbors (KNN)

KNN (K-Nearest Neighbors) is a classification algorithm that assigns labels to unlabeled samples based on the labels of its k nearest neighbours in the feature space. Since it is sensitive to the choice of k the optimal value has been found using a grid-search. This method can suffer from the curse of dimensionality.

In table 4, we can see mean metric values for the cross validation tests and validation split and in appendix figure 17, the confusion matrix. KNN also performed moderate to bad in our dataset, which might be for its simplicity and reliance on local patterns and the high number of features of our dataset.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.481	0.788	0.634	0.685	0.371	0.867	0.694
Val	0.490	0.791	0.640	0.681	0.382	0.864	0.695

Table 4: Metrics of the KNN model

5.2.2 Logistic Regression

Logistic regression is a widely used classification algorithm which estimates class probabilities by applying the logistic function to a linear combination of features. It is a particular case of generalized linear models. In most circumstances the model does not have a closed form solution and optimization techniques such as gradient descent have to be used. To address overfitting and improve generalization, logistic regression can include a penalty term through regularization techniques such as L1 (Lasso) and L2 (Ridge) regularization.

The metrics for the cross-validation and validation splits can be seen in table 5. The confusion matrix for the validation split is shown in appendix figure 18. This model performed poorly, indicating its limitations in capturing complex relationships and nonlinear patterns in the dataset,

given its linear assumptions. Furthermore, we can see that the metrics for CV and validation splits are pretty different, which indicates that, even though we have regularization, there is overfitting.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.432	0.784	0.608	0.589	0.341	0.850	0.651
Val	0.359	0.767	0.563	0.458	0.295	0.820	0.584

Table 5: Metrics of the Logistic Regression model

5.2.3 Decision Tree

Decision Trees creates a tree-like model of decisions by recursively partitioning the feature space based on the values of input features. At each node, the algorithm selects the most informative feature and threshold to split the data, optimizing a criterion such as Gini or Entropy for the split. Other hyperparameters, like the maximum depth of the tree, minimums samples split and leaf and maximum features, are also important to cross-validate to ensure finding the right balance between model complexity and generalization.

We can see the metrics in table 6 and the confusion matrix for the validation split in appendix figure 19. DT does not give good metrics, its simplicity may limit its ability to capture the inherent complexities in the dataset. Furthermore, DT are known to have high variance.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.486	0.797	0.641	0.676	0.382	0.868	0.697
Val	0.488	0.765	0.627	0.733	0.366	0.866	0.698

Table 6: Metrics of the Decision Tree model

5.2.4 Random Forest

Random Forests are an ensemble learning method that combine multiple decision trees to make predictions. Each tree is trained on different data, obtained by bagging, and only consideres M of the features for each node for split. Predictions are made by aggregating the outputs of individual trees. This ensemble approach helps reduce overfitting and improve generalization. Random Forest is known for its ability to handle high-dimensional data, nonlinear relationships, and outliers effectively. It is worth noting that the out of bag error could have been used, but for the sake of consistency with the other models, cross-validation has been used.

Table 7 and appendix figure 20 show the metrics and confusion table. Random Forest performed well in terms of F1 score, particularly for class 1. Its ability to capture complex patterns and relationships in the dataset contributed to its accurate predictions.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.519	0.848	0.684	0.612	0.451	0.873	0.711
Val	0.525	0.845	0.685	0.617	0.456	0.869	0.711

Table 7: Metrics of the Decision Tree model

5.2.5 Extra Trees

Extra Trees is an ensemble learning method similar to Random Forest which combines multiple decision trees to make predictions, but with a key difference: the splitting points for each feature are randomly selected instead of finding the optimal split. This extra level of randomness enhances

the ensemble's robustness to noise and outliers, making Extra Trees particularly useful for high-dimensional data or datasets with noisy features. Although it introduces increased bias compared to Random Forest, Extra Trees offers faster training and prediction times.

The metrics can be seen in table 8 for both CV and validation splits. The confusion matrix is shown in appendix figure 21. It shows a similar performance to RF, indicating its effectiveness in capturing complex relationships and patterns. Being based on decision trees, ET leveraged the inherent ability of decision trees to handle nonlinearity.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.515	0.835	0.675	0.643	0.430	0.874	0.712
Val	0.516	0.828	0.672	0.645	0.429	0.869	0.709

Table 8: Metrics of the Expansion Trees model

5.2.6 Adaboost

AdaBoost combines multiple weak classifiers to create a strong classifier by iteratively adjusting the weights of misclassified samples. The algorithm assigns higher weights to incorrectly classified samples in each iteration, making subsequent weak classifiers focus on those samples. During prediction, the weak classifiers' outputs are combined using weighted voting to make the final classification decision.

We can see the metrics in table 9 and the confusion matrix for the validation split in appendix figure 22. The model achieved a decent F1 score but may have been affected by the inherent limitations of boosting algorithms in handling imbalanced datasets, leading to slightly lower precision for class 0.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.515	0.806	0.660	0.720	0.400	0.879	0.722
Val	0.517	0.799	0.658	0.726	0.401	0.874	0.720

Table 9: Metrics of the Gradient Boosting model

5.2.7 Gradient Boosting

Gradient Boosting is an ensemble learning algorithm that combines multiple weak prediction models to create a strong predictive model. Gradient boosting models are build in a stage-wise fashion like other boosting methods, but it generalizes them by using an arbitrary differentiable loss function. Unlike AdaBoost, which focuses on adjusting sample weights, Gradient Boosting focuses on iteratively minimizing a loss function by adding weak models to the ensemble. In each iteration, the algorithm fits a weak model to the residual errors of the previous models, placing more emphasis on the samples that were poorly predicted. The weak models are added to the ensemble in a way that minimizes the loss function gradient.

Table 10 and appendix figure 23 show the metrics and confusion matrix. It demonstrates moderate to good performance, with F1 scores and precision values that were slightly lower compared to other models. This might be due to not having found the proper hyperparameters.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.513	0.799	0.656	0.735	0.394	0.879	0.723
Val	0.511	0.796	0.653	0.721	0.396	0.872	0.715

Table 10: Metrics of the Gradient Boosting model

5.3 Catboost classifier

In order to try to improve the performance metrics obtained with the base gradient boosting model, a variation of Gradient Boosting called CatBoost [9] has been used. Catboost is a gradient boosting framework developed by Yandex which has a great ability to handle categorical variables without needing to preprocess them. To do so it uses a combination of ordered boosting, random permutations and other optimization techniques. Furthermore, it has GPU support for faster training times, although it has not been used in this project. Nonetheless, performing a typical grid-search on such a model can be too time-consuming. Therefore, we have opted to use the Optuna [10] library. It is a parameter optimization framework that explores the hyperparameter space in an intelligent manner and stops training when they are not leading to a good model.

The hyperparameter search procedure has been ran with the parameters in 8.7. The best parameters, when it comes to the weighted F1 score are described in 11. The mode has been tested against the validation set, with different probability thresholds, obtaining the following results.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	ROC AUC Macro
Val	0.51	0.79	0.65	0.72	0.39	0.71

Table 11: Metrics of the CatBoost model

5.3.1 Linear Support Vector Machines

Linear Support Vector Machines is a classification algorithm that creates a hyperplane to separate different classes of data points. By transforming the input data into a higher-dimensional feature space using a linear kernel, SVM aims to find the optimal hyperplane that maximizes the margin between classes. It is a convex optimization problem which can be efficiently solved.

We can see the metrics and confusion matrix in table 12 and appendix figure 24 respectively. It shows a moderate performance, probably due to its linearity assumptions, which might not be satisfied in our dataset.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
CV	0.51	0.801	0.655	0.724	0.394	0.878	0.719
Val	0.508	0.795	0.651	0.716	0.394	0.871	0.712

Table 12: Metrics of the Linear SVM model

5.4 Model Comparison

It is now the turn to choose the best model. Table 13 shows the metrics of the validation split for all the models. Among all of them, Random Forest (RF), Adaboost (AB), and Expansion Trees (ET) showcase the best performance in terms of F1 score for class 0 and a balanced combination of recall and precision for class 0. Although the scores achieved by these models may not be exceptional due to the complexity of the dataset, they are relatively better than the ones of the other classifiers. This is because these models are good at capturing complex relationships and nonlinear patterns present in the data, which enables them to distinguish between the two classes more effectively. Notably, RF, AB, and ET are based on a simpler classifier, the Decision Tree, which likely contributes to their better fit with the dataset. Furthermore, they are models which are known for their ability of handling mixed type data.

On the other hand, Logistic Regression (LR) performs significantly worse than the other models. LR relies on linear assumptions and may struggle to capture difficult underlying patterns in situations involving complex or nonlinear relationships, which is the case with our dataset. Moreover, the solution is found through numerical optimization and with a large dataset such as ours this task be complicated.

Considering the above, the preferred model for further testing is Random Forest, which slightly outperforms Adaboost and Expansion Trees in terms of the balance between recall and precision for class 0. By selecting Random Forest, we can leverage its ability to handle complex relationships and nonlinear patterns, increasing the likelihood of achieving improved classification performance for our dataset. The optimal preprocessing is one hot encoding to variables with few categories and target encoding to variables with many categories without resampling.

Finally, it is interesting to remark that for any of the fitted models, the optimal resampling was the SMOTE oversample. A possible reason could be attributed to the presence of outliers or noisy instances within the minority class, which may lead to distorted synthetic samples when using SMOTE. Consequently, random undersampling or no resampling may be favored as they address the class imbalance while avoiding potential overfitting or inaccuracies caused by the interpolation process of SMOTE. Both methods of preprocessing are good, since they were found to be optimal for different models.

	F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
RF	0.525	0.845	0.685	0.617	0.456	0.869	0.711
AB	0.517	0.799	0.658	0.726	0.401	0.874	0.720
ET	0.516	0.828	0.672	0.645	0.429	0.869	0.709
GB	0.511	0.796	0.653	0.721	0.396	0.872	0.715
Cat	0.51	0.79	0.65	0.72	0.39	—	0.71
LDA	0.509	0.792	0.650	0.723	0.392	0.872	0.713
LSVM	0.508	0.795	0.651	0.716	0.394	0.871	0.712
QDA	0.494	0.805	0.649	0.657	0.396	0.864	0.696
GNB	0.490	0.807	0.648	0.645	0.396	0.862	0.692
KNN	0.490	0.791	0.640	0.681	0.382	0.864	0.695
DT	0.488	0.765	0.627	0.733	0.366	0.866	0.698
LR	0.359	0.767	0.563	0.458	0.295	0.820	0.584

Table 13: Metrics of the validation split for all the models fitted

6 Final model evaluation

After having compared the different models, it is now the turn to fit and evaluate the Random Forest models with the hyperparameters shown in the appendix 17.

The model will be trained on the whole training test and the test split will be used to evaluate the performance of the model.

Table 14 and Figure 6 show the metrics and confusion matrix of the model. The F1 score for individuals who did not vote is 0.524, indicating a moderate level of accuracy in correctly classifying non-voters. Conversely, for those who voted, the F1 score is 0.851, suggesting significantly better performance in identifying voters, which is expected due to the imbalanced nature of the data. To assess the overall model performance, we calculated the F1 macro score, which is 0.688, indicating a reasonable balance between the two classes. Looking at recall and precision for class 0 (not voted), the model achieved a recall of 0.628, correctly identifying 62.8% of non-voters. However, the precision for class 0 is 0.450, indicating a relatively high number of false positives, where individuals were incorrectly classified as having voted. It is worth noting that a tradeoff must be made between precision and recall. Models such as catboost can achieve a higher precision (with both precision and recall above 0.5). Nevertheless, it has been argued that is more important to identify non voters. The average precision (AP) macro, measuring the overall quality of predictions across both classes, obtained a score of 0.879, indicating a relatively good performance in ranking the predictions. Finally, the ROC AUC macro score, illustrated by the curve in Figure 7, is 0.719, demonstrating a reasonably good ability to distinguish between voted and not voted individuals given the difficulty of the problem and the imbalanced dataset.

F1 C0	F1 C1	F1 Macro	Recall C0	Precision C0	AP Macro	ROC AUC Macro
0.524	0.851	0.688	0.628	0.450	0.879	0.719

Table 14: Metrics of the test split of the best Random Forest model

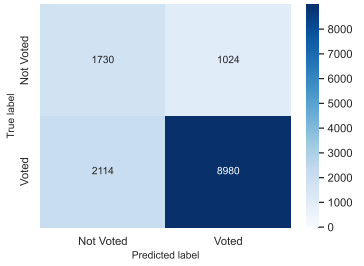


Fig. 6: Confusion matrix for the test split of the best Random forest model

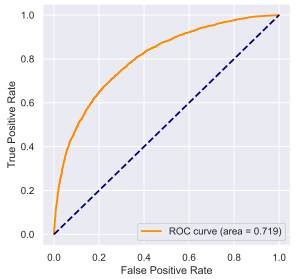


Fig. 7: ROC curve of the best Random forest model

In conclusion, the Random Forest model achieved moderate to good performance in predicting whether individuals have voted or not, considering the difficulty of our dataset. Moreover, the similar test results to the validation set (as seen in 7) suggest that our model would likely generalize well to new data. The training set consisted of 55,388 examples, and the testing set of 13,848.

One important step in evaluating the model is to perform feature selection on the training data to identify the most important variables for predicting whether someone voted or not.

However, we need to highlight that one-hot encoded features cannot be directly ranked in terms of importance since they are divided into multiple columns. Nevertheless, they collectively only contribute to approximately 15.25% of the predictive power for the target variable [1]. Figure 8 shows the importance of each feature.

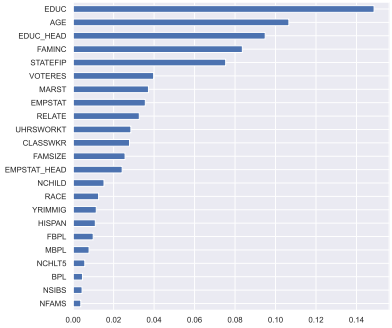


Fig. 8: Feature importance of the best Random forest model

The most crucial variable for predicting voting behavior is the level of education. In the second tier, in order of importance, we can observe the influence of age, the education level of the head of the household, family income, and state. On the other hand, the number of families in the household, siblings, and birthplace appear to have less impact on the prediction. It is important to know how these variables affect voting behavior because it helps us understand the factors that affect civic participation and electoral outcomes. Education level often correlates with higher civic engagement and political awareness, influencing voting decisions. Additionally, the education level of the head of the household can impact an individual's exposure to political discussions and values within the family. Age differences among various age groups can reflect diversity in political interest. Moreover, family income plays a role, as it can affect access to resources that may influence an individual's likelihood of voting. Furthermore, the state of residence introduces variations in political culture and policies, which in turn influence voting behavior.

By considering the interaction between these variables, we can gain a comprehensive understanding of the varied factors that support civic participation and electoral outcomes.

[1] In order to have an idea if actually there is any one-hot encoded variable which is notably important in the model, the 'permutation_importance' from 'sklearn.inspection' has been used. It is a method used to measure the importance of features in a trained machine learning model. It works by shuffling the values of each feature and evaluating the resulting decrease in model performance, allowing us to identify the features that have the greatest impact on predictions. The appendix includes Figure 12, a box plot showing the results. As we expected, it is observed that there is no one-hot encoded variable which holds considerable importance in the model.

7 Conclusions

In this project, we have applied different preprocessing techniques and models in order to predict voter turnout in the 2020 US presidential elections. Many challenges were found during various steps of the project. Nonetheless, careful treatment of the data and models allowed us to achieve acceptable results.

When first confronted with the data, different subsets of the variables and preprocessing methods were tried. After trial and error, a set of variable and different preprocessing pipelines were chosen and used when fitting the data to different machine learning models.

In the modelling phase different models were fitted and evaluated using the train split. After carefully tuning the parameters and preprocessing pipeline, Random Forest was found to be the best performing method. When evaluated with the test data a F1-macro score 0.688 was achieved.

On a personal level, this project has been both a challenge and a rewarding experience. As our first major machine learning work, it presented us with a significant learning opportunity. We gained hands-on experience in working with pipelines and deepened our understanding of different models. Despite the challenges faced, we are satisfied with the outcomes, as we have improved our skills and expanded our knowledge in the field of machine learning. This project serves as a solid foundation for our future endeavours in the field and has allowed us to see the potential of machine learning when applied to difficult real world tasks.

7.1 Possible extensions and known limitations

During this project, many compromises were made in order to be able to train a model within the scope of the *Aprenentatge Automatic 1* course. On the one hand, the preprocessing of the data was kept relatively simple in order to integrate it with scikit-learn pipelines. The ordered categorical variables could have been more thoroughly analysed and preprocessed.

On the one hand, despite the dataset size, due to its imbalanced nature and complexity of some variables more samples might have been needed in order to better train models such as gradient boosting models. A possible improvement could be achieved by aggregating data from multiple elections.

Another challenge was the training time required to train the models and perform the grid search across multiple parameters and preprocessing pipelines. Kaggle was used to train the models in the background, but this also complicated to project workflow. In order to improve the training times of some methods, the use of an accelerator such as a GPU could have been useful. Further usage of optimization libraries which can be used to intelligently search for the optimal parameters might also be interesting.

Further improvements might be achieved by the usage of more complex machine learning techniques such as deep learning.

References

[1] Ruggles, S., Flood, S., Goeken, R., Grover, J., Meyer, E., Pacas, J., Sobek, M.: IPUMS USA: Version 12.0 [dataset]. Minneapolis, MN: IPUMS. https://doi.org/10.18128/D010.V12.0 (2022)

[2] Becker, B., Kohavi, R.: Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20 (1996)

[3] Johnson, A.: Adult census income dataset using multiple machine learning models. Analytics Vidhya (2020)

[4] scikit-learn: RobustScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html. Accessed on May 28, 2023

[5] Van Buuren, S., Groothuis-Oudshoorn, K.: mice: Multivariate imputation by chained equations in r. Journal of statistical software **45**, 1–67 (2011)

[6] Micci-Barreca, D.: A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems. SIGKDD Explorations 3, 1 (2001), 27–32 (2001)

[7] Halford, M.: Prince. https://github.com/MaxHalford/prince

[8] Lemaître, G., Nogueira, F., Aridas, C.K.: Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research **18**(17), 1–5 (2017)

[9] Dorogush, A.V., Gulin, A., Gusev, G., Kazeev, N., Prokhorenkova, L.O., Vorobev, A.: Fighting biases with dynamic boosting. CoRR **abs/1706.09516** (2017) 1706.09516

[10] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. CoRR **abs/1907.10902** (2019) 1907.10902

[11] Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H.: The elements of statistical learning: data mining, inference, and prediction. Springer (2009)

[12] Arias, M.: Lecture notes from Aprenentatge Automàtic I. Universitat Politècnica de Catalunya (2023)

8 Appendices

8.1 List of variables

The variables extracted from the IPUMS CPS database are: HWTFINL, GQTYPE, HHINTYPE, STATEFIP, METRO, METFIPS, FAMINC, NFAMS, INTTYPE, WTFINL, CPSIDP, RELATE, AGE, SEX, RACE, MARST, POPSTAT, VETSTAT, FAMSIZE, NCHILD, NCHLT5, ELDCH, YNGCH, NSIBS, BPL, YRIMMIG, CITIZEN, MBPL, FBPL, NATIVITY, HISPAN, EMPSTAT, OCC, CLASSWKR, UHRSWORKT, JOBCERT, EDUC, DIFFANY, EARNWT, COMPWT, UNION, EARNWEEK, UHRSWORKORG, ELIGORG, VOWHYNOT, VOTEHOW, VOTERES, VOTED, VOSUPPWT, COVIDTELEW, COVIDUNAW, COVIDPAID, EMPSTAT_HEAD, EDUC_HEAD.

8.2 Data Cleaning

Some variables included in the raw dataset have not been included in the final project. The following list lists them together with the reason why:

- Missing values: JOBCERT,, VOTERES
- Too granular: *METFIPS, OCC, POPSTAT*
- Statistical weights and survey identifiers: *HWTFINL, COMPWT, VOSUPPWT, CPSIDP,EARNWT,HHINTYPE, WTFINL, ELIGORG*
- Many not in universe values: *UNION, ELDCH, UHRWORKORG, COVIDTELEW, YNGCH*
- Possible leakage: *VOTEHOW, VOWHYNOT*

8.3 Preprocessing types

- Basic Numerical preprocessing: The variables have been scaled. Applied to *NFAMS*, *FAMSIZE*, *NCHILD*, *NCHLT5*, *NSIBS*, *YRIMMIG*
- Numerical preprocessing and imputed: The variables are imputed and scaled. Applied to *UHRSWORKT*.
- Categorical columns one-hot encoded: *GQTYPE*, *METRO*, *INTTYPE*, *SEX*, *VETSTAT*, *CITIZEN*, *DIFFANY*, *COVIDUNAW*, *NATIVITY*, *COVIDPAID*.
- Categorical columns target encoded: *STATEFIP*, *RELATE*, *RACE*, *MARST*, *BPL*, *MBPL*, *FBPL*, *HISPAN*, *EMPSTAT*, *CLASSWKR*, *EMPSTAT_HEAD*.
- Ordered Categorical columns: they have only been recoded in order to improve interpretability and maintain the implicit order. Applied to *FAMINC*, *AGE*, *EDUC*, *EDUC_HEAD*, *VOTERES*.

8.4 Relevant plots

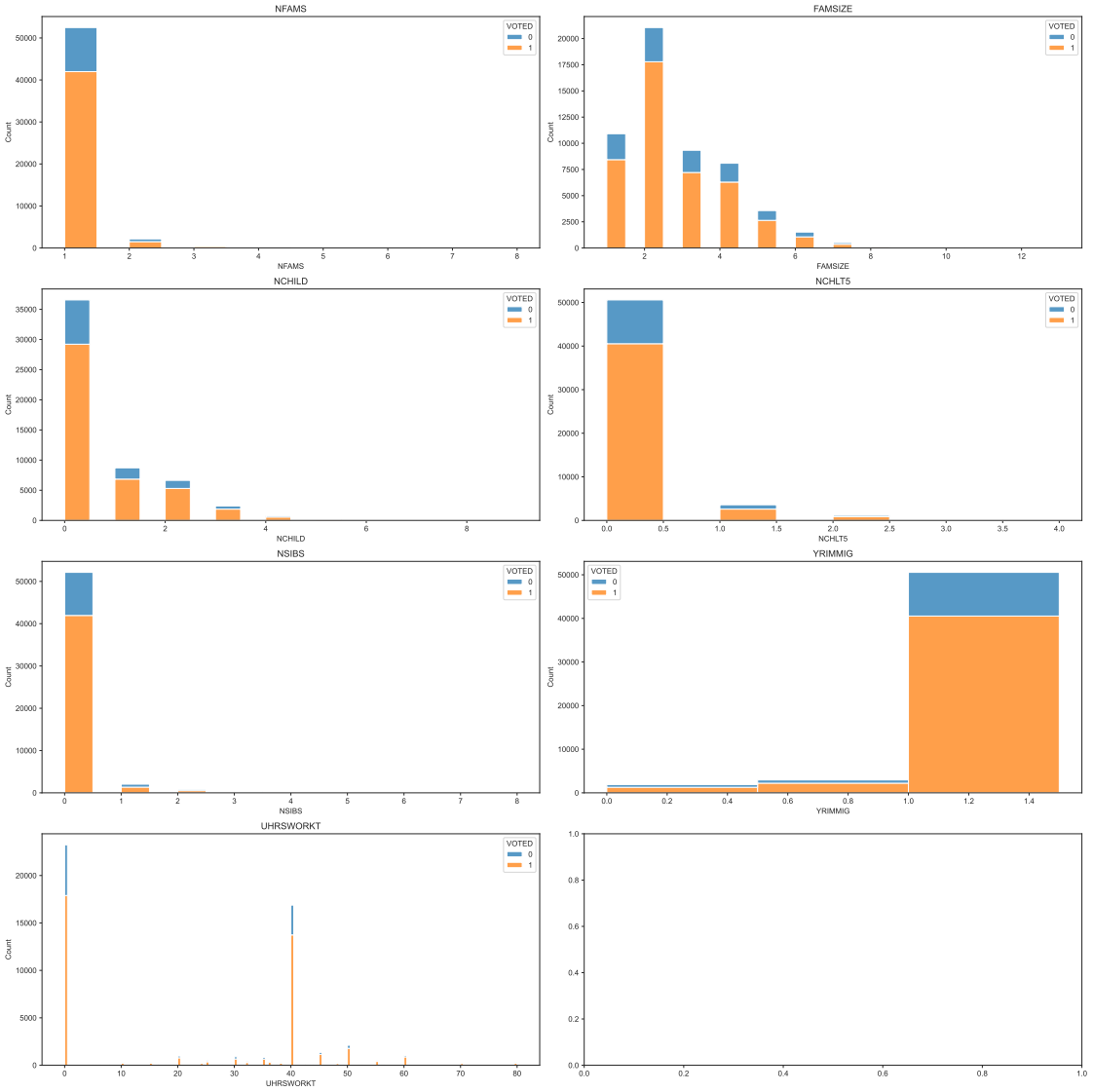


Fig. 9: Histograms of the numerical variables before preprocessing

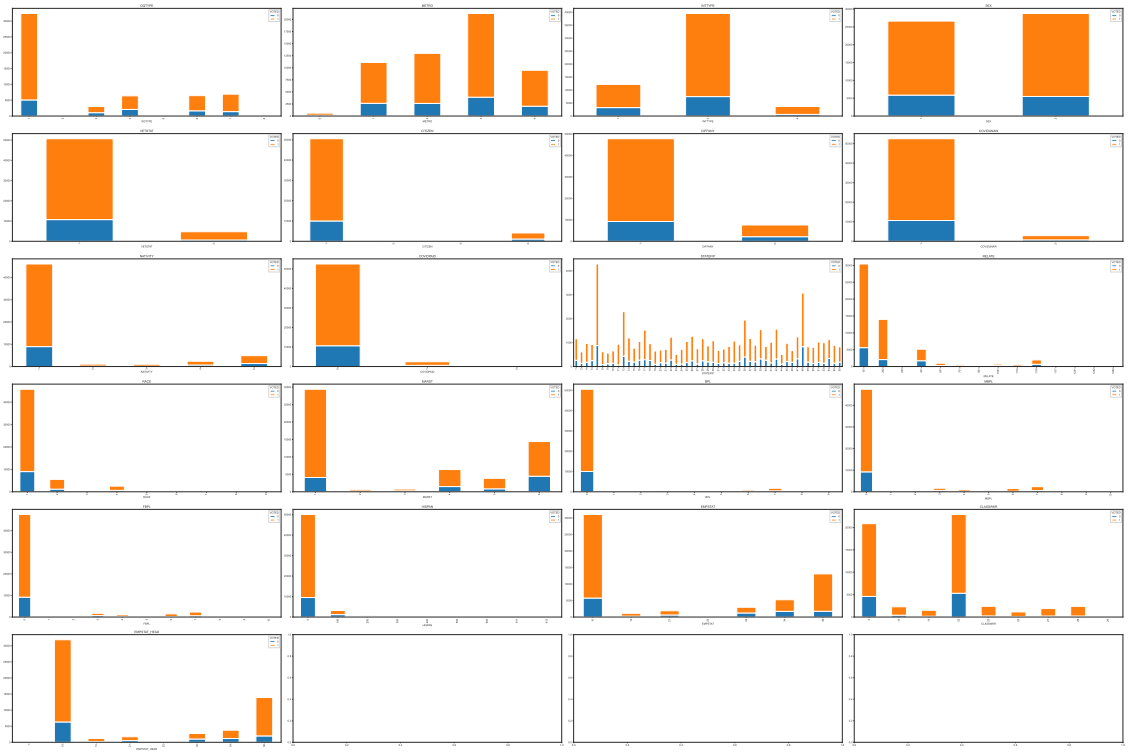


Fig. 10: Histograms of the categorical variables before preprocess

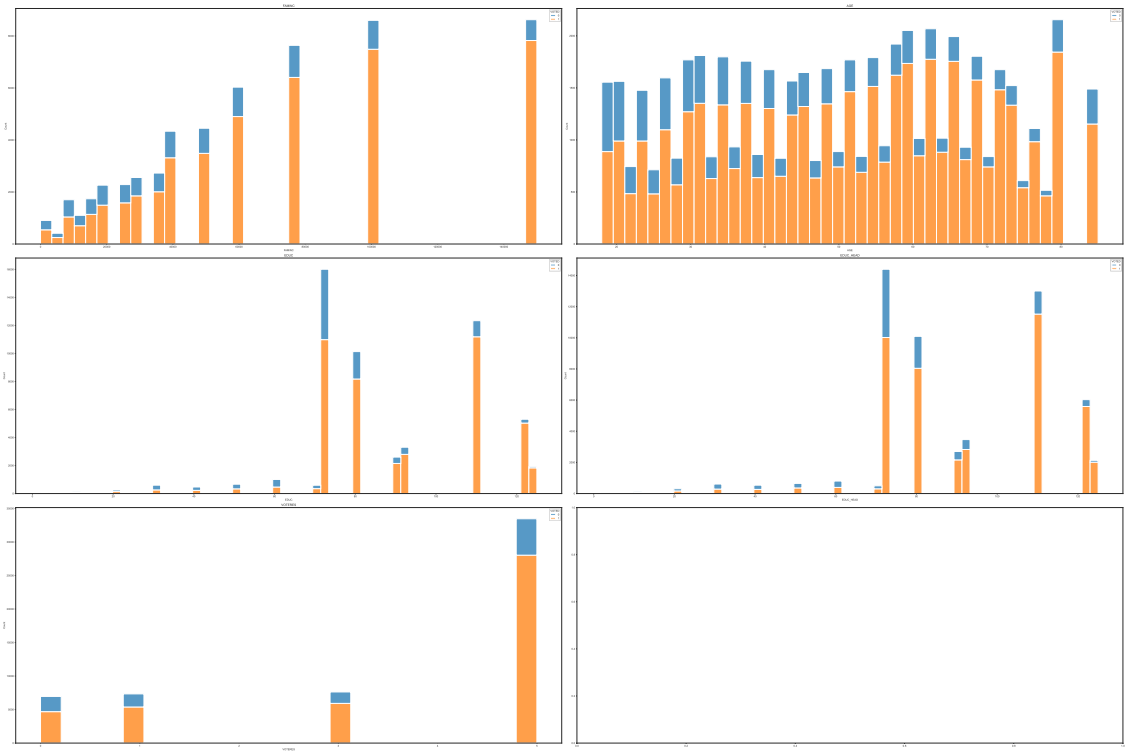


Fig. 11: Histograms of the categorical ordered variables before preprocessing

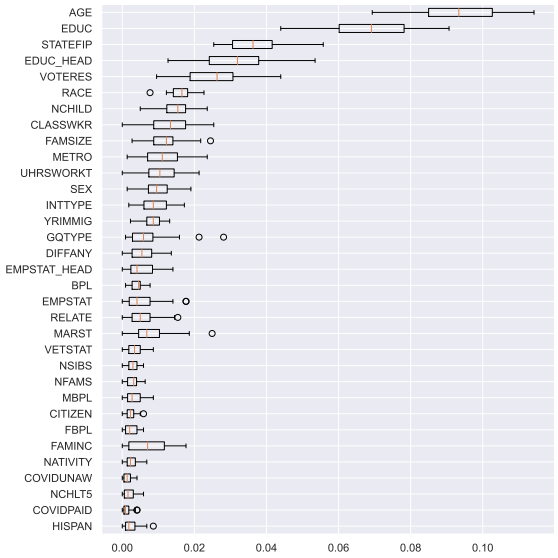
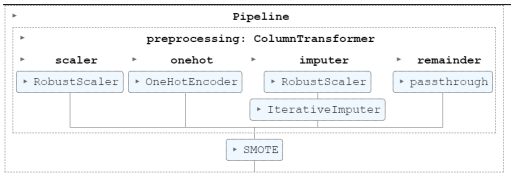
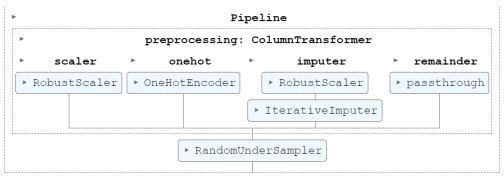


Fig. 12: Permutation Importances of the best Random forest model

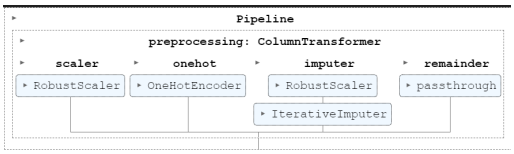
8.5 Preprocessing Diagrams



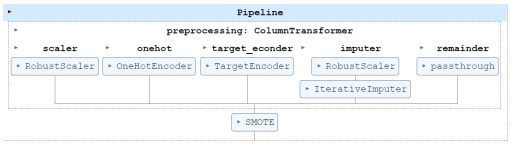
(a) Pipe OH Smote (pipe_oh_smote.png)



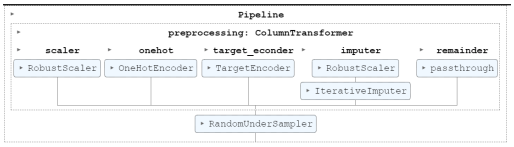
(b) Pipe OH Ran (pipe_oh_ran.png)



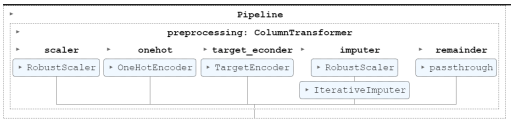
(c) Pipe OH (pipe_oh.png)



(d) Pipe OH Target Smote (pipe_oh_target_smote.png)



(e) Pipe OH Target Ran (pipe_oh_target_ran.png)



(f) Pipe OH Target (pipe_oh_target.png)

Fig. 13: Preprocessing pipelines used

8.6 Confusion matrices for the validation splits

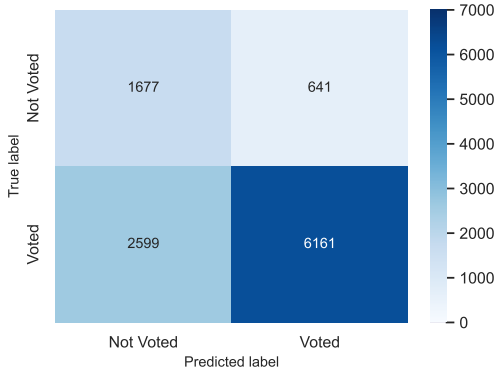


Fig. 14: Confusion matrix for the Validation split of the LDA model

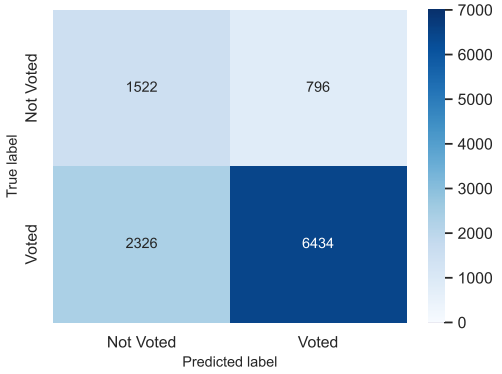


Fig. 15: Confusion matrix for the Validation split of the QDA model

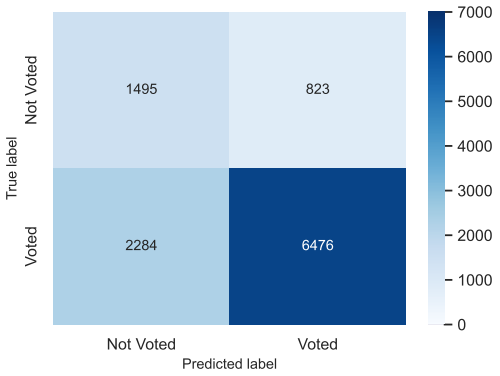


Fig. 16: Confusion matrix for the Validation split of the GNB model

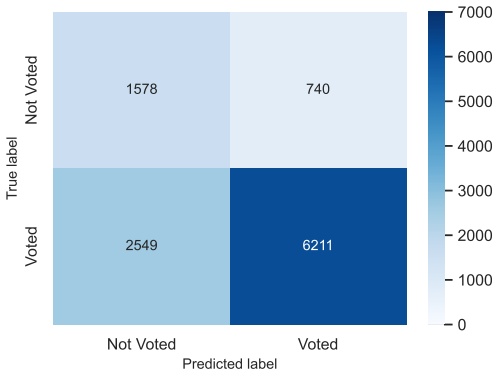


Fig. 17: Confusion matrix for the Validation split of the KNN model

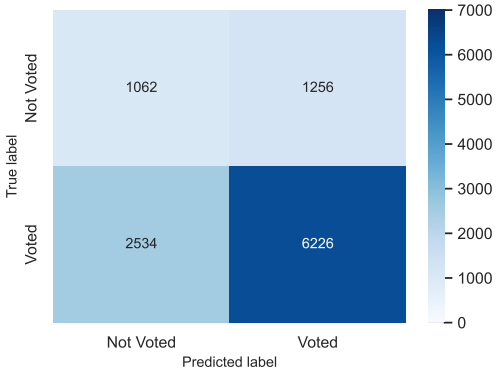


Fig. 18: Confusion matrix for the Validation split of the LR model

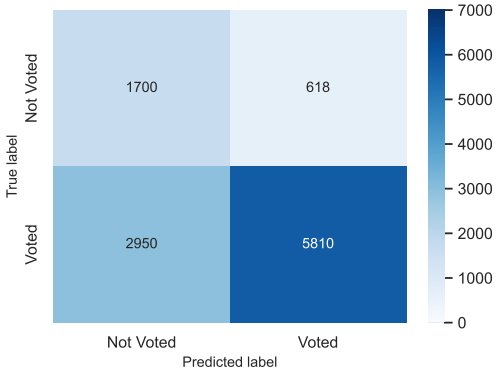


Fig. 19: Confusion matrix for the Validation split of the DT model

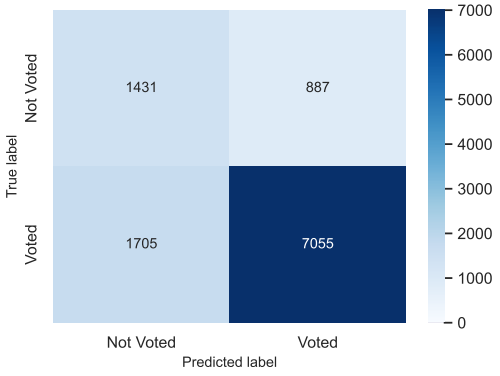


Fig. 20: Confusion matrix for the Validation split of the RF model

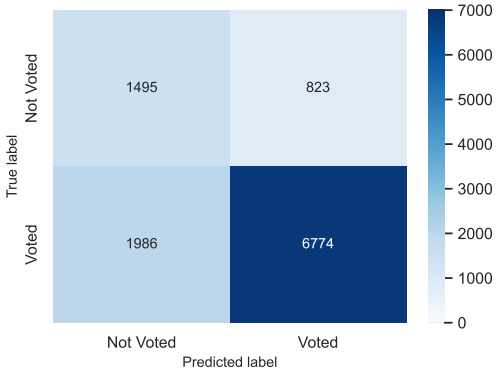


Fig. 21: Confusion matrix for the Validation split of the ET model

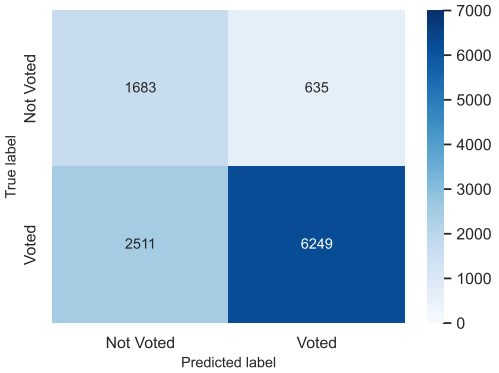


Fig. 22: Confusion matrix for the Validation split of the AB model

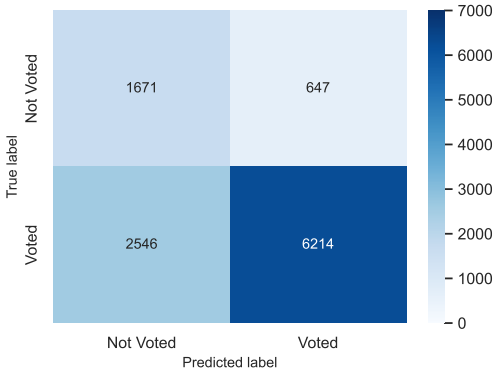


Fig. 23: Confusion matrix for the Validation split of the GB model

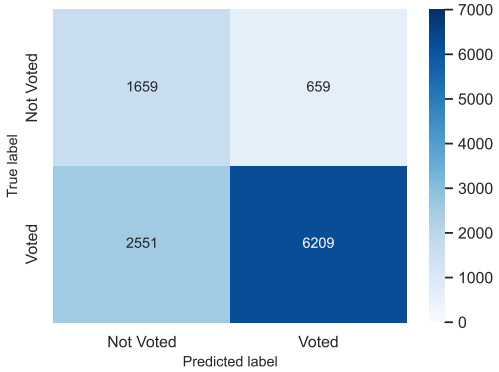


Fig. 24: Confusion matrix for the Validation split of the LSVM model

8.7 Parameters estimated

```
'reg_param': [0, 0.0001, 0.001, 0.01, 0.1, 0.5, 0.7, 1]
```

Listing 1: Gridsearch parameters for QDA

```
'n_neighbors': [i for i in range(1,50,2)]
```

Listing 2: Gridsearch parameters for KNN

```
'penalty': ['l1', 'l2'],
'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
'max_iter': [10000]
```

Listing 3: Gridsearch parameters for Logistic Regression

```
'criterion': ['gini', 'entropy'],
'max_depth': [None, 5, 10, 15, 20],
'min_samples_split': [1, 2, 3, 4, 5],
'min_samples_leaf': [1, 2, 3, 4, 5],
'max_features': ['sqrt', 'log2', None]
```

Listing 4: Gridsearch parameters for Decision Trees

```
'n_estimators': [200,None],
'max_depth': [100,None],
'min_samples_split': [4,6],
'min_samples_leaf': [4,6],
'class_weight': [None, 'balanced', 'balanced_subsample']
```

Listing 5: Gridsearch parameters for Random Forest

```
'n_estimators': [150,None],
'max_depth': [100,None],
'min_samples_split': [4,6],
'min_samples_leaf': [2,4],
'class_weight': [None, 'balanced', 'balanced_subsample']
```

Listing 6: Gridsearch parameters for Extra Trees

```
'classifier__estimator__max_depth': [2],
'classifier__estimator__min_samples_leaf': [5,7,10],
'classifier__n_estimators': [250, 275, 300],
'classifier__learning_rate': [0.1,0.15]
```

Listing 7: Gridsearch parameters for Adaptative Boosting

```
'classifier__learning_rate': [0.01, 0.01,  0.1],
'classifier__min_samples_split': [0.01, 0.1, 0.2],
'classifier__min_samples_leaf': [0.01, 0.1, 0.2],
'classifier__max_depth': [7,8,10],
'classifier__max_features': ['sqrt'],
'classifier__n_estimators': [100]
```

Listing 8: Gridsearch parameters for Gradient Descent

```
'classifier__C': [0.01, 0.05, 0.1, 0.2],
'classifier__class_weight': ['balanced'],
'classifier__max_iter': [1000, 2000, 3000, 4000, 5000],
'classifier__dual': [False],
'classifier__penalty': ['l1', 'l2']
```

Listing 9: Gridsearch parameters for Linear Support Vector Machines

```

"objective": trial.suggest_categorical("objective", ["Logloss"]),
"iterations": trial.suggest_int("iterations",300,1500),
"colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1, log=True),
"depth": trial.suggest_int("depth", 1,12),
"boosting_type": trial.suggest_categorical("boosting_type", ["Plain","Ordered"]),
"bootstrap_type": trial.suggest_categorical("bootstrap_type", [ "Bernoulli", "MVS"]),
"auto_class_weights": trial.suggest_categorical("auto_class_weights", ["Balanced"]),
```

```
"used_ram_limit": "25gb",
"eval_metric": "PRAUC",
"cat_features": cat_features,
```

Listing 10: Search parameters for Optuna catboost search procedure

8.8 Best parameters

```
Preprocessing: BB
```

Listing 11: Best parameter for LDA

```
Preprocessing: BA
'reg_param': 0.5
```

Listing 12: Best parameter for QDA

```
Preprocessing: BA
```

Listing 13: Best parameter for GNB

```
Preprocessing: BA
'n_neighbors': 43
```

Listing 14: Best parameter for KNN

```
Preprocessing: BA
'penalty': l2'
'C': 0.01
'max_iter': 10000
```

Listing 15: Best parameters for Logistic Regression

```
Preprocessing: AA
'criterion': 'gini'
'max_depth': 5
'min_samples_split': 1
'min_samples_leaf': 5
'max_features': None
```

Listing 16: Best parameters for Decision Trees

```
Preprocessing: A0
'n_estimators': 200
'max_depth': None
'min_samples_split': 6
'min_samples_leaf': 4
'class_weight': 'balanced_subsample'
```

Listing 17: Best parameters for Random Forest

```
Preprocessing: A0
'n_estimators': 150
'max_depth': None
'min_samples_split': 4
'min_samples_leaf': 4
'class_weight': 'balanced_subsample'
```

Listing 18: Best parameters for Extra Trees

```
Preprocessing: AA
'classifier__estimator__max_depth': 2
'classifier__estimator__min_samples_leaf': 10
'classifier__n_estimators': 300
'classifier__learning_rate': 0.15
```

Listing 19: Best parameters for Adaptative Boosting

```
Preprocessing: AA
'classifier__learning_rate': 0.1
'classifier__min_samples_split': 0.01
'classifier__min_samples_leaf':  0.2,
'classifier__max_depth': 10
'classifier__max_features': 'sqrt'
'classifier__n_estimators': 100
```

Listing 20: Best parameters for Gradient Descent

```
Preprocessing: B0
'classifier__C': 0.05
'classifier__class_weight': 'balanced'
'classifier__max_iter': 1000
'classifier__dual': False
'classifier__penalty': 'l1'
```

Listing 21: Best parameters for Linear Support Vector Machines