COMPARISON AND INTERPRETATION OF DIFFERENT STATISTICAL AND MACHINE LEARNING MODELS THAT PREDICT DROPOUT RATES IN PUBLIC SCHOOLS IN NEW YORK STATE

Preparer: Alexey Pankratov

Abstract

There has been ongoing interest in the US and all over the globe in the issue of secondary school dropouts because school dropouts come at a high cost to both individuals and society. Research in different countries shows that dropouts are more likely to become unemployed, stay unemployed for longer, have lower earnings, and over their life course, accumulate less wealth. This project aims to build and compare four different models predicting the dropout rates in schools in New York state in the 2019-20 education year – Regularized Logistic regression, Random Forest regression, Extreme Gradient Boosting regression, and Artificial neural network-based regression. The models are compared in terms of their predictive accuracy using mean absolute error and R^2 as performance evaluation metrics. At the same time, the project also aims to understand why models make a certain prediction in addition to assessing their prediction's accuracy. Model interpretation tool SHapley Additive exPlanations (or SHAP) is used in this project to estimate the importance of the predictors in each model and assess the similarity of the most important predictors across the models.

Introduction to the problem

Graduation from high school is considered a minimum level of education for people to successfully participate in further study and work. This is because, in most nations, secondary education serves as the foundation for entry to university and other education and training opportunities as well as preparation for entry into the labor market. Over time, it has become more and more important to decide how economic and other life benefits, such as good health and well-being, are distributed.[1]

There has been ongoing interest in the US and all over the globe in the issue of secondary school dropouts because school dropouts come at a high cost to both individuals and society. Research in different countries shows that dropouts are more likely to become unemployed, stay unemployed for longer, have lower earnings, and over their life course, accumulate less wealth (e.g., see Rumberger & Lamb, 2003; OECD, 2001; Barro, 1997; Shavit & Mueller, 1998). Dropouts also more often experience poorer physical and mental health, have higher crime rates, and less often engage in active citizenship (Owens, 2004; Rumberger, 1987). In addition to the costs for individuals, there are also social costs associated with increased welfare needs and reduced taxation revenue (Owens, 2004).[1]

In the United States, the dropout rate represents the percentage of 16- to 24-year-olds who are not enrolled in school and have not earned a high school credential (either a diploma or an equivalency credential). High school graduation rates had grown dramatically in the US from the early 1900s

when 96% of all individuals 18 years and older had not completed high school. By the 1960s, the non-completion rate was only about one-fourth of this population.[2] National Center for Education Statistics data shows that dropout rates decreased further to 8.3% in 2010 and to 5.1% percent in 2019.[3]

This project aims to build and compare four models – Regularized Logistic Regression, Random Forest regression, Extreme Gradient Boosting regression, and Artificial neural network-based regression - that predict the dropout rates in schools in New York state in the 2019-20 education year. The target variable, dropout rate, is the percentage of students of a particular gender that studied as part of a particular cohort in a particular school in New York state who, at the date of graduation of the cohort, are not enrolled in school and have not earned a high school credential. The prediction of dropout rates is based on various social and economic factors at the cohort, school, or county level described in the next section of the project.

As part of the model comparison, the project also aims to identify which of the predictors are more strongly associated with the dropout rates, using such model interpretation tools as SHapley Additive exPlanations described later in the project. It may be assumed that manipulation in predictors that have the strongest association with the target variable would change the dropout rates in the future. However, it can often be misleading, as there is a fundamental difference between correlation and causation. This project aims to make transparent the correlations picked up by predictive models. However, making correlations between the target variable and predictors transparent does not make them causal. Predictive models implicitly assume that correlation patterns will stay constant, and in order to understand what happens in case variables in the model change their behavior, different kinds of models should be built. These models are called causal models, and building them requires making assumptions and using the tools of causal analysis. Building a causal model is outside of the scope of this project.

Collection and preparation of data

The data used in the project is a combination of the datasets from different sources.

The first source is the publicly available reporting educational data provided by the New York State Education Department (NYSED). It can be found here: https://data.nysed.gov/downloads.php. The datasets of interest are - “Graduation Rate Database”, “Report Card Database” and “Student and Educator Database” for the 2019-20 education year as it is the latest available.

Graduation Rate Database contains annual graduation and dropout data for the state broken down by counties, districts, or schools. The data also includes the number of students enrolled. For each of the abovementioned aggregation levels (i.e., district, country, school), annual graduation data is included for the current four-year, five-year, and six-year cohorts. A cohort is a group of students who work through a curriculum together to achieve the same academic degree together.[4] The standard cohort length in high school in the US is four years, but there are also extended cohorts of 5 and 6 years.

Also, each aggregation level can be further disaggregated by demographic subgroups such as gender, race, or other (e.g., economically disadvantaged students or not). For example, it is possible to extract the data for white students or female students, but not for white female students. So it is

not possible to select both gender and race, as these observations are not independent of each other; observations for white students will include all white students of all genders, including females, and observations for female students will include female students of all races, including white. Therefore, it is important to set a single level of aggregation so that observations are independent. For this project, I considered aggregating dropouts by school and gender. Accordingly, dropout rates in the dataset used for the modeling are for students of a particular gender studying as part of a particular cohort in a particular school.

Report Card Database contains state, district, public school, and charter school accountability, secondary graduation rate, expenditures per pupil, and staff qualifications data. Also included are accountability and secondary graduation rate data by county and Need to Resource Capacity group. Report Card Database was originally stored in MS Access database, and the information on expenditure per pupil and inexperienced teachers was extracted in 2 respective CSV files that were merged by school id.

Student and Educator Database contains information for each school - attendance rate, number of counselors and social workers, percentage of suspensions, and percentage of pupils who are eligible for free lunch.

It should be noted that to ensure student confidentiality, NYSED does not publish data for groups with fewer than five students or data that would allow readers to easily determine the performance of a group with fewer than five students. When there were fewer than five students in a group (e.g., Hispanic), outcomes for those students were suppressed for that group and the next smallest group. Suppressed data are indicated with an "-" in the original dataset. These observations were dropped.

The second source of data is US census data for each county in New York State such as but not limited to median household income, percent of persons in poverty, percent of households with a computer/broadband internet, number of persons per household etc. This publicly available information is taken from the United States Census Bureau website. In particular, the information is taken from QuickFacts data access tool (https://www.census.gov/quickfacts/) that provides users with easy access to the basic population, business, and geography statistics for all states and counties of the USA and cities and towns with more than 5,000 people.

After extracting the NYSED and US Census data and dropping the information deemed not relevant for the project (refer to the complete list of remaining variables in Table 1), I performed the preparation of the final datasets. To do this, I first merged Graduation Rate Database and Report Card Database by School ID and then merged the resulted dataset with US Census County set by county name.

Missing values

The resulting dataset contained missing values because some schools were missing in the Report Card Database. Out of 1,275 public schools in **Graduation Rate Database,** 39 schools were missing in the Report Card Database and Student and Educator Database. These schools comprise 1.48% of all students. Considering small number of missing schools, I removed them form the population. Accordingly, the project covers all schools in New York state, which report detailed information to NYSED.

Preparation of data

After collecting the data and removing missing values, I prepared the dataset for modeling.

First of all, I performed one-hot encoding of categorical variables *cohort* and *gender*. It should be noted that the reported genders are only male and female, therefore, I kept only *gender_female* variable, where 1 values relate to females while 0 values are related to males.

Then, I split the data set into training (50%), validation (25%), and test (25%). To avoid overfitting, the models will be trained on the training dataset and then benchmarked on the validation dataset. The best model will be selected based on its validation performance metrics. Test dataset will be used to assess the performance of the final selected model. If the error on the test set that is significantly worse than the error on the validation set, this will alert that the model is not as good as you expected: the validation error is underestimated and should be reinvestigated[5].

After splitting the dataset, numerical variables of the training dataset were normalized (i.e., centered and scaled). Normalization of validation and test sets was done using only the data from the training set. Only the training set needs to be used as the validation and test sets play the role of new data that is not yet seen, so it should not be used in the training of the models. Using any information from the test or validation set before or during training introduces a potential bias in evaluating the models' performance. Normalizing the data using the training set assumes that both test and validation data have the same mean and standard deviation as the training set. Normalizing data is an important procedure that helps regularized logistic regression, and gradient descent algorithms converge.

Exploratory analysis

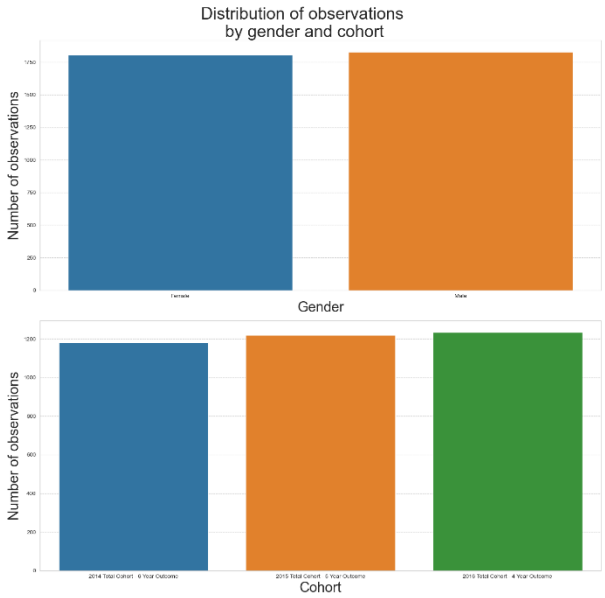
The total population used in the project consists of 7,269 observations, which represent 1,236 schools with up to 3 cohorts in each school and students of up 2 genders in each cohort. Table 1 shows the target variables and initial

Table 1. List of variables used in the project

#	Variable type	Variable name	Variable description
1	Target variable	dropout_pct	Percent of students who dropped out (Response variable).
2	Categorical predictors	cohort	Description of the cohort membership (original name of variable is MEMBERSHIP_DESC).
3		gender	Gender of student group (Male/Female). Original name of the variable is SUBGROUP_NAME.
4	School level numerical predictors	PUPIL_COUNT_TOT	Number of pupils in school
5		PER_FEDERAL_EXP	Per pupil expenditures using federal funds
6		PER_STATE_LOCAL_EXP	Per pupil expenditures using state and local funds
7		PER_TEACH_INEXP	Number of teachers with fewer than four years of experience in their positions
8		TEACH_PER_PUPIL	Number of teachers per 1 pupil in school
9		ATTENDANCE_RATE	Annual student attendance rate
10		PER_SUSPENSIONS	Percent of students suspended
11		PER_FREE_LUNCH	Percentage of enrolled students eligible for free lunch
12		COUNSELORS_PER_PUPIL	Number of counselors per 1 pupil in school
13		SOCIAL_PER_PUPIL	Number of social workers per 1 pupil in school
14	County level numerical predictors	cty_pop_under_18	Persons under 18 years, percent
15		cty_pop_over_65	Persons 65 years and over, percent
16		cty_female	Female persons, percent
17		cty_foreign_born	Foreign born persons, percent, 2016-2020
18		cty_owner_occupied	Owner-occupied housing unit rate, 2016-2020
19		cty_housing_unit_median_value	Median value of owner-occupied housing units, 2016-2020
20		cty_pers_pers_household	Persons per household, 2016-2020
21		cty_same_house	Living in same house 1 year ago, percent of persons age 1 year+, 2016-2020
22		cty_non_english	Language other than English spoken at home, percent of persons age 5 years+, 2016-2020
23		cty_with_computer	Households with a computer, percent, 2016-2020
24		cty_with_internet	Households with a broadband Internet subscription, percent, 2016-2020
25		cty_hs_graduates	High school graduate or higher, percent of persons age 25 years+, 2016-2020
26		cty_uni_degree	Bachelor's degree or higher, percent of persons age 25 years+, 2016-2020
27		cty_time_to_work	Mean travel time to work (minutes), workers age 16 years+, 2016-2020
28		cty_med_household_income	Median household income (in 2020 dollars), 2016-2020
29		cty_poverty	Persons in poverty, percent
30		cty_pop_density	Population density, pop/sqft

The training data consists of 3,634 observations, which represent 50% of the observations randomly selected from the total population.

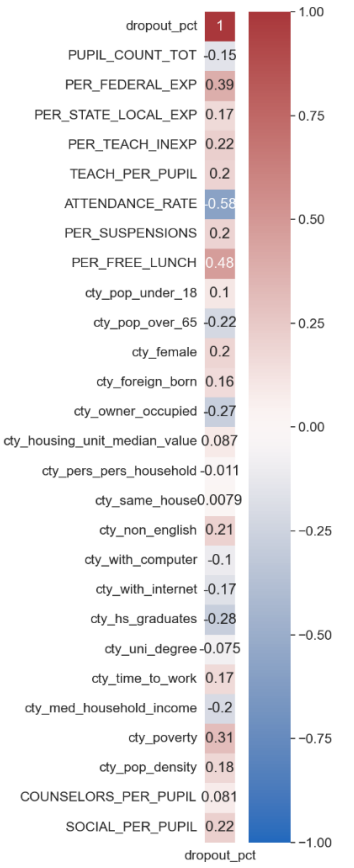
Figure 1. Number of observations of each gender (top) and cohort (bottom)



Looking at the categorical variables in Figure 1, we can see that genders are relatively equally presented in the training set. Males (1,827) are slightly more than Females (1,807), but the difference is very small. There are slightly fewer 6-year cohorts in schools (1,181) than 5-year (1,220) and 4-year (1,233). Still, the differences are also minimal, and we can assume that gender and cohorts are equally distributed in the population.

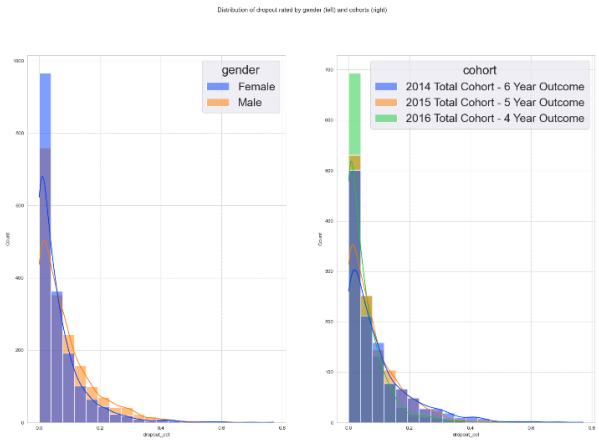
Figure 3 shows the distribution of dropout rates for each categorical variable. For all variables, we can see that the distribution of dropout rates is skewed towards zero values had has a long tail, which resembles power-law distribution. At the same time, it can also be seen that dropout rates are higher on average for males than for females, as the distribution of dropout rates for males is less skewed toward zero, and the tail is heavier. The same can be said for 6-year cohort students compared with 5-year and 4-year, and 5-year cohort students compared with 4-year cohort.

Figure 2. Correlation of dropout rate with numerical predictors



As shown in Figure 2, dropout rates do not strongly correlate with any of the continuous predictors. The strongest positive correlation is 0.48 with “PER_FREE_LUNCH” and the strongest negative correlation of -0.58 is with “ATTENDANCE_RATE”. If to look at the correlation between each pair of continuous predictor variables, 9 pairs of variables have a very strong correlation of more than 0.8 absolute in value (Table 2). The correlation matrix plot for all continuance predictors is presented the Appendix A.

Figure 3. Distribution of dropout rates by gender (left) and cohorts (right)



Strong correlation is an indication of a multicollinearity problem. Multicollinearity occurs when one or more of the independent variables in the model can be approximately determined by some of the other independent variables. When there is multicollinearity, the estimated regression coefficients of the fitted model can be highly unreliable. Consequently, any modeling strategy must check for possible multicollinearity at various steps in the variable selection process.[6]

Table 2. Continuous predictions with absolute correlation above 0.8

Variable 1	variable 2	Pearson correlation
cty_non_english	cty_foreign_born	0.97
cty_non_english	cty_owner_occupied	-0.82
cty_with_internet	cty_with_computer	0.91
cty_hs_graduates	cty_non_english	-0.82
cty_time_to_work	cty_foreign_born	0.91
cty_time_to_work	cty_non_english	0.92
cty_med_household_income	cty_with_internet	0.86
cty_pop_density	cty_owner_occupied	-0.87
cty_pop_density	cty_housing_unit_median_value	0.89

In the high-dimensional setting, the multicollinearity problem is extreme: any variable in the model can be written as a linear combination of all of the other variables in the model. Essentially, this means that we can never know exactly which variables (if any) truly are predictive of the outcome, and we can never identify the best coefficients for use in the regression.[7]

In short, multicollinearity generally will not impair the performance of the models, especially of the tree-based models that use bagging; however, it makes not possible to interpret them. This is applicable both for parameter estimates in the classical regression models such as logistic regression and feature importance calculation for complex machine learning techniques such as SHAP values discussed later in the project report.

Multicollinearity is not an issue in predictive modeling, where the goal is to use the associations between predictors and the outcome variable to generate good predictions for future outcomes. Predictive modeling aims to maximize the predictive accuracy, and variables used in a predictive model are based on association, not statistical significance or scientific meaning.

In contrast, in explanatory modeling, the main focus is in identifying variables that have a scientifically meaningful and statistically significant relationship with an outcome. The primary goal is to test the theoretical hypotheses, emphasizing both theoretically meaningful relationships and determining whether each relationship is statistically significant.

Although I do not perform hypothesis testing in the project, nevertheless I aim to shed light on what is behind the black-box models, such as artificial neural network-based or XGBoost regression. Therefore, to have interpretable models, it is important to tackle the multicollinearity problem. In this project, I reduce the effect of multicollinearity by selecting variables using regularization, as described in the following section.

Building the models

Defining loss function and performance metrics

Supervised learning techniques are used when there is a set of variables that might be denoted as inputs, which are measured or preset, and these inputs have some influence on one or more outputs, while the goal is to use the inputs to predict the values of the outputs. In the statistical literature, the inputs are often called the predictors or independent variables. In machine learning, they are also called features. The three terms above are used interchangeably in the report. The outputs are called the response, dependent, or target variables.

The responses vary in nature among the examples. In some cases, responses are quantitative, such as, for example, dropout rates among different schools. Whereas in other, the responses are qualitative and can take only a finite number of values, such as, for example, whether a particular student dropped out or not. This distinction in the responses type has led to a naming convention for the prediction tasks: regression when I predict quantitative outputs and classification when I predict qualitative outputs. In other words, classification tries to predict a class from a data item, while regression tries to predict a value.

When developing regression and classification models, the key question is to estimate parameter estimates that would provide the most accurate prediction. Parameter estimation is the process of identifying model parameters based on a given training set that are likely to yield good prediction performance. This can be viewed as an optimization process in which the space of possible parameter vectors is searched for one that optimizes an adopted performance measure.

The important difference between classification and regression is the loss function that is used to choose these parameters. The loss is the cost function used to evaluate errors, and so to train the predictor. Training a classifier involves using a loss that penalizes errors in class prediction in some way, and training a regressor means using a loss that penalizes prediction errors.[9]

The response variable in this project is dropout rates. The dropout rate represents the proportion of the students out of the total who left school before graduating. On the one hand, the target variable comes from the binomial distribution. The dropout rate is one way to show the proportion of successes (number of students who dropped out) in the total number of trials (total number of students), so finding the dropout rates can be considered a classification problem. However, on the other hand, the response in this project, dropout rates, is modeled as a continuous variable that takes values between 0 and 1. The issues associated with this are described in detail in "Conclusion and discussion section".

Different loss functions apply to various regression problems. Key loss functions include the mean squared error (MSE) and mean absolute error (MAE). The difference between these two loss functions is that MSE places much more emphasis on observations with large absolute residuals $|y_i - f(y_i)|$ during the fitting process. It is thus far less robust, and its performance severely degrades for long-tailed error distributions and especially for grossly mismeasured y-values ("outliers"). Other more robust criteria, such as MAE, perform much better in these situations. In the statistical robustness literature, a variety of regression loss criteria have been proposed that provide strong resistance (if not absolute immunity) to gross outliers while being nearly as efficient as least squares for Gaussian errors. They are often better than either for error distributions with

moderately heavy tails. One such criterion is the Huber loss criterion used for M-regression (Huber, 1964). Huber loss function combines the good properties of squared-error prediction error is close to zero, and the absolute error prediction error is large[8].

Taking into account that loss functions represent the difference between the actual observations and the observation values predicted by the model, comparing loss values will be used to determine the extent to which the model fits the data as well as whether removing some explanatory variables is possible without significantly harming the model's predictive ability.

In this report, I will use the MAE loss function for the artificial neural network-based model parameter estimation.

Another important distinction should be made between loss function and performance evaluation metrics. The loss function is used by the model to learn the relationship between input and output. The evaluation metric is used to assess how good the learned relationship is.

To make all models comparable, I selected the same metric for evaluating their performance. I chose MAE as the primary performance metric due to its advantage described above (i.e., robustness to outliers). The model with the lowest MAE will be selected.

In addition to MAE, I will also use R-squared (R^2) to compare the goodness-of-fit of the models:

$$R^2 = 1 - \frac{Residual\ sum\ of\ squares}{Total\ sum\ of\ squares} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Equation 1. Coefficient of determination (R^2)

Where y_i- observed value of the response variable,

$\hat{y}_i$ – predicted value of response variable by the model, and

$\bar{y}$ – predicted value of response variable by null model. Null in this project is simply the mean value of the response variable.

R-squared can take any value between 0 to 1. Theoretically, R-squared can even be negative in case the model predicts the target variable worse than the null model. R-squared quantifies how much of a drop in the residual sum-of-squares is accounted for by fitting the proposed model and is often referred to as the coefficient of determination. This is expressed on a helpful scale as a proportion of the total variation in the data. Values of R-squared approaching 1 indicate the model to be a good fit. Values of R-squared less than 0.5 suggest that the model gives rather a poor fit to the data.

I use two performance metrics for model comparison to check if they are aligned with each other.

Cross-validation and data split

The goal of the models built in this project is to get a regressor that works well on future data for which we might never know the true dropout rate, using a training set of examples that include observed dropout rate values. In other words, the regression model should generalize from the training data to test data. Here, it is important to make a distinction between training error and test error. The training error of a regression model is the error rate (MAE selected for this project) on examples used to train the regressor. In contrast, the test error is an error on examples not used to train the regressor. With the advanced modeling techniques that are available on the market,

models could almost perfectly fit the training data. However, such models might not have small test errors because the regression procedure is chosen to do well on the training data. This effect is often called overfitting. [9] It occurs when generalization from training data is performed poorly on the test dataset. An efficient training procedure is quite likely to find special properties of the training dataset that are not representative of the test dataset because the training dataset is not the same as the test dataset.

To avoid overfitting, I cannot only estimate MAE for the data used for training the model. Thus, I need to separate the data into training and validation sets, train the model on a training set and evaluate its performance on a validation set. A validation error is the one that matters the most, as in this case, I will get an unbiased error estimate, meaning the estimated value of error will be close to the true value of error.

I can resolve this problem with cross-validation, which involves repeatedly: splitting data into $\boldsymbol{k}$ training and validation sets uniformly and at random, training a model on the training set, evaluating it on the validation set, and then averaging the error over all splits. Each different split is usually called a fold, so the cross-validation is often called k-fold cross-validation. For example, when I choose a k of 10, I will split the data into ten parts. Then each of the ten parts will serve as validation data one time. This means that the remaining nine parts are used as training data. This procedure yields an estimate of the likely future performance of a model at the expense of substantial computation. I will then calculate the final validation error on the validation data that I split separately for this purpose. I will also use cross-validation to optimize the tuning parameters of the models, such as: α and λ for regularized logistic regression from the Equation 7, and hyperparameters for Random Forest and XGBoost models listed in Table 5 and Table 6, respectively).

As I compare different models in this project, I will also split the data into the test set. Accordingly, I use the train-validation-test split. I train models on the training data, then benchmark the models on the validation data, which is the basis for selecting the model. Then finally, I use the selected model and compute an error on the test set. A test error close to the validation error will confirm that I got an unbiased error estimate[5].

Penalized Logistic Regression

Theoretical basis

First of all, it should be noted that the response variable, dropout rate, is a proportional data, which main feature is that it is bounded by the interval [0, 1]. Proportional data is frequently studied in such areas as, for example, prevention trials.

As continuous data by its nature, the proportional data can be modeled using linear regression models.

$$\hat{Y} = X^T\beta$$

Equation 2. Prediction of linear regression

Where $\hat{Y}$ is the predicted values of the target variable, X^T- design matrix and β – vector of model parameters (coefficients).

However, modeling the proportional data using linear regression creates certain problems. Some of the statistical assumptions underlying linear regression models, such as linearity, normality, and homogeneous errors, might be violated due to these special features of proportion data. Another important issue is that linear regression predictions are not bounded so that the predicted values can be outside of the range between 0 and 1. To partially overcome some of the data issues mentioned above, one possibility is to transform the data. For example, in some fields, such as ecology, the arcsine square root transformation has long been a common practice to transform proportional data.

An alternative to the transformation methods, the generalized linear model, has become more prevalent in analyzing proportion data in clinical studies. The model naturally handles non-normality and heterogeneity issues, and the use of a link function guarantees that the fitted values will be exactly within the desired range [0, 1]. For example, logistic regression, a common generalized linear model, is suitable for a binomial outcome, where the proportion is computed as the ratio of the number of target events to the total number of trials, "n_y out of n". The fitted logistic model is often more interpretable than a transformed model, and reflects the underlying characteristics of the data better than linear regression.[10]

A GLM generalizes the normal linear model by allowing a response variable with a distribution other than normal, but a member of the exponential family of distributions and a relationship between the response and the linear component of the form $g(Y) = X^T\beta$ where $g()$ is the link function. Logistic regression is an example of GLMs, where the response variable is linked to the predictors via the logit link function.

$$logit(Y) = log(\frac{Y}{1-Y}) = X^T\beta$$

Equation 3. Logit link function

Solving for Y, the equation above can be rearranged as follows:

$$Y = \frac{e^{X^T\beta}}{1+e^{X^T\beta}} = \frac{1}{1+e^{-X^T\beta}}$$

Equation 4. Logit link function (transformed)

The function above is called the Sigmoid function, mathematically denoted as $f(x) = \frac{1}{1+e^{-x}}$. The function converts any real number into a number between 0 and 1. Logistic regression is generally used when the dependent variable is binary in nature, but as the use of a logit link function guarantees that the fitted values will be exactly within the desired range between 0 and 1, it may provide better results to the regression problem for proportional data. Logistic regression is a linear model for a transformation of the probability.

Other classical regression models such *beta* or *tobit* regression can be used to predict the proportional data. However, I use logistic regression and its variations, such as regularized logistic regression or sigmoid activation layer in the artificial neuron network (ANN) based regression in this project, as different variations of logit transformation are implemented in many well-known Python modules (keras, sklearn, xgboost) and R packages (glmnet).

Basic assumptions for logistic regression include the independence of errors, linearity in the logit for continuous variables, absence of multicollinearity, and lack of strongly influential outliers.[11]

In this project, we will use regularized regression techniques, which aim at making a **biased estimate of regression parameters**. When estimated on training data, regularized regression techniques accept larger errors in order to get better results on new test data, which is called the bias-variance trade-off. In other words, training performance is reduced by introducing bias. This bias reduces the variance in the predictions of new data. These techniques also penalize correlated variables, so the assumptions underlying logistics regression are partly circumvented by the penalized logistic regression methods, like Lasso, Ridge regression, and Elastic Net.

Moreover, all of these methods include one or several regularization parameters described above, and none of them has a definite rule for selecting the values of these parameters. The optimal value is found via a cross-validation procedure, but there are various cross-validation methods, and they can yield different results. As a result, the actual outcome of model parameters β of any of these penalized regression methods is not fully defined by the method, but can depend on the modeler's choices.[12]

Therefore, I will not make any assumptions and statements about the theoretical optimality of parameter estimates of penalized logistic regression model and other models used in this project. Instead, we will select the optimal model that performs the best performance on validation data via a cross-validation procedure.

Estimating model parameters

In a generalized linear model, we are interested in the model parameters $\beta_1, \dots, \beta_j$ that describe how the response depends on the explanatory variables. We use the observed $y_1, \dots, y_i$ to maximize the log-likelihood function (or minimize negative log-likelihood). For logistic regression, the log-likelihood function will be the following:

$$l(\beta, y) = \sum_{i=1}^{n} y_i \ln(p_i) + (y_i - 1) \ln(1 - p_i)$$

Equation 5. Log-likelihood function of logistic regression

Regularization

Generally, forward and backward stagewise regression were strategies for predictors selection in order to prevent the model from overfitting. An alternative and very powerful approach is to penalize the coefficients if they are equal to zero. The resulting model ignores the corresponding independent variables. Models built this way are often called sparse models because (one hopes) that many independent variables will have zero coefficients. So the model uses a sparse subset of the possible predictors.[9]

The **glmnet** package provides an automated way to penalize the parameter estimates. It fits a generalized linear model (GLM) so-called via penalized maximum likelihood. Concretely, it solves the following problem:

$$\text{argminmin}(-\frac{1}{N}l(\beta, y) + r(\beta))$$

Equation 6. Objective function of penalized logistic regression

Where $l(\beta, y)$ – is the log-likelihood functions described above.

$r(\beta)$ - function that introduces the parameter penalty.

N – total number of training observations.

Since this formula uses an argmin, it is a numerical search for the optimal values for betas. This means that fitting the model is simply done by iteratively searching through different values for betas and identifying which values for thetas lead to the lowest value for the rest of the objective function.

To minimize the objective function, **glmnet** package uses a "proximal Newton" algorithm. This algorithm makes repeated use of a quadratic approximation to the log-likelihood, and then weighted coordinate descent on the resulting penalized weighted least-squares problem. These constitute an outer and inner loop, also known as iteratively reweighted penalized least squares.[13]

General expression of the penalty function $r(\beta)$ for Elastic Nets is as follows:

$$r(\beta) = \lambda(\frac{(1-\alpha)}{2}\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|)$$

Equation 7. Penalty function for regularized regressions

Elastic Net regularization combines the penalties of ridge and lasso to get the best of both worlds. In the expression above, the parameter α is a "mixing" parameter that balances the R1 and R2 regularizations. If α is zero, we get L2 regularization, and if α is one, then we get L1 regularization. $\lambda > 0$ is a hyperparameter known as the regularization parameter (also known as a tuning parameter), which determines the overall strength of the penalty. The regularization path is computed for the lasso or elastic net penalty at a grid of values (on the log scale) for the regularization parameter λ.

It is known that the ridge penalty shrinks the coefficients of correlated predictors towards each other while the lasso tends to pick one of them and discard the others. The elastic net penalty mixes these two: if predictors are correlated in groups, an $\alpha = 0.5$ tends to either select or leave out the entire group of features. This is a higher level parameter, and users might pick a value upfront or experiment with a few different values.[13]

Ridge regression, while penalizing model parameter estimates, sets them very close to zero but not equal to zero, which means that it does not remove predictors from the model. As the main goal of regularization for this project is to perform variable selection, I will build only Lasso and Elastic Nets but not Ridge regression.

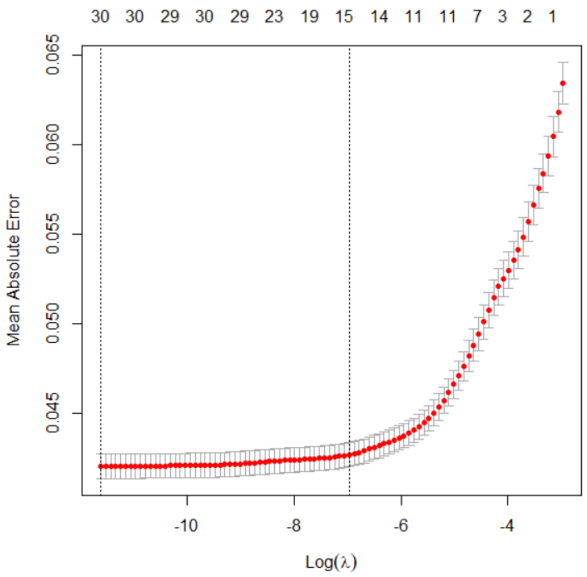
Application in the project

Both Lasso and Elastic net models are built using R package **glmnet**.

As mentioned above, only λ hyperparameter needs to be tuned in Lasso regression as another hyperparameter α is fixed at 0. The value of optimal regularization hyperparameter λ was estimated via 10-folds cross-validation that is build-in **cv.glmnet** function of **glmnet** package.

Figure 4 below shows how the MAE of the L1 regularized model changed with different values of λ hyperparameter.

Figure 4. MAE against log-λ in L1 regularized logistic regression



Mean absolute error is very low from the very beginning, which means that the model is going a good job even without regularization. However, our goal is variables selection, and we can see that for L-1 regularization, the MAE value stays almost the same until 15 variables remain in the model (values on the upper side of the plot) and only then starts to go up. This indicates that even regularization while removing almost half of the initial 31 variables from the model does not result in worse prediction quality. There are also two vertical lines in the plot. The one is at the minimum MAE corresponding to the log-λ ("log" in this report means natural log) value of -11.618. The other vertical line is within one standard error of the minimum MAE (i.e., the so-called "one-standard-error" rule) corresponding to the log-λ value of -6.966. The second line is a slightly more restricted model (since it refers to a larger value for λ). The plot shows that there is no improvement in MAE with the increasing value of the regularization parameter. However, the MAE value is almost the same in the model with 15 predictors as in the model that includes all variables. On this basis, I will select the final L1-regularized model with λ value within one standard error of the one with the minimum MAE.

Table 3. Validation performance of Lasso and Elastic Net

Metric	Lasso regression	Elastic nets
MAE	0.042	0.047
R^2	0.50	0.45

The elastic net was also built using **glmnet** package. Unlike Lasso regression, Elastic net has two tuning hyperparameters. **glmnet** package has no standard in-built function for tuning both hyperparameters α and λ, so we used the **caret** package. Caret package runs a 10-folds cross-validation of multiple models with different tuning parameters values and derives the values that minimize MAE. After performing cross-validation of the elastic nets model, I got optimal values of hyperparameter $\alpha = 0.1$ and $\lambda = 0.0446$.

After building both models with optimal hyperparameters, I calculated their performance metrics (MAE, R^2), which are presented in Table 3.

The table shows that Lasso regression performs better than Elastic net. So, I selected Lasso regression for variable selection and comparison with other more complex models analyzed later in the report. Predictors that remained in and removed from the model are presented in Table 4.

Table 4. Predictors that remained in and removed from the model

Predictors remained in the model	coefficient
PER_FREE_LUNCH	0.6234
ATTENDANCE_RATE	-0.3110
`cohort_2016 Total Cohort - 4 Year Outcome`	-0.2028
cty_foreign_born	-0.1832
gender_Female	-0.1800
cty_uni_degree	-0.1001
TEACH_PER_PUPIL	0.0928
`cohort_2014 Total Cohort - 6 Year Outcome`	0.0699
COUNSELORS_PER_PUPIL	0.0618
PER_FEDERAL_EXP	0.0611
cty_female	-0.0562
cty_with_computer	0.0553
PUPIL_COUNT_TOT	0.0444
cty_housing_unit_median_value	-0.0392
SOCIAL_PER_PUPIL	0.0085

Predictors removed from the model
`cohort_2015 Total Cohort - 5 Year Outcome`
PER_STATE_LOCAL_EXP
PER_TEACH_INEXP
PER_SUSPENSIONS
cty_pop_under_18
cty_pop_over_65
cty_owner_occupied
cty_pers_pers_household
cty_same_house
cty_non_english
cty_with_internet
cty_hs_graduates
cty_time_to_work
cty_med_household_income
cty_poverty
cty_pop_density

The correlation matrix presented in Figure 5 and the distribution of absolute correlation coefficients for pairs of variables before and after variables selection show that there are no variables with absolute Pearson correlation above 0.8 among the remaining set of variables, and only two pairs have an absolute correlation coefficient of more in the range between 0.7 and 0.8.

Figure 5. Correlation matrix of remaining continuous predictors

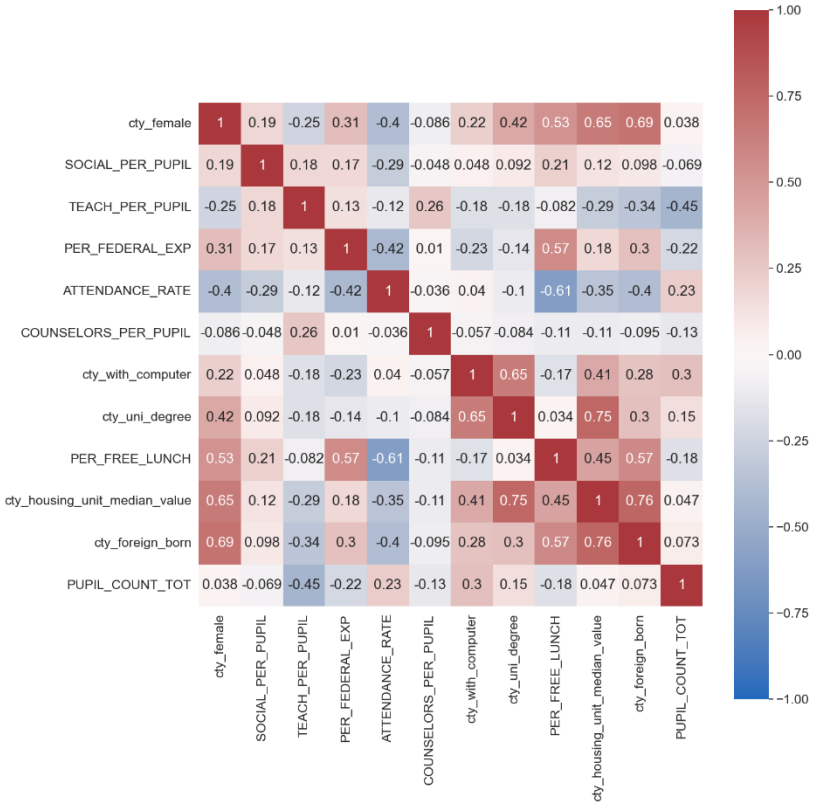
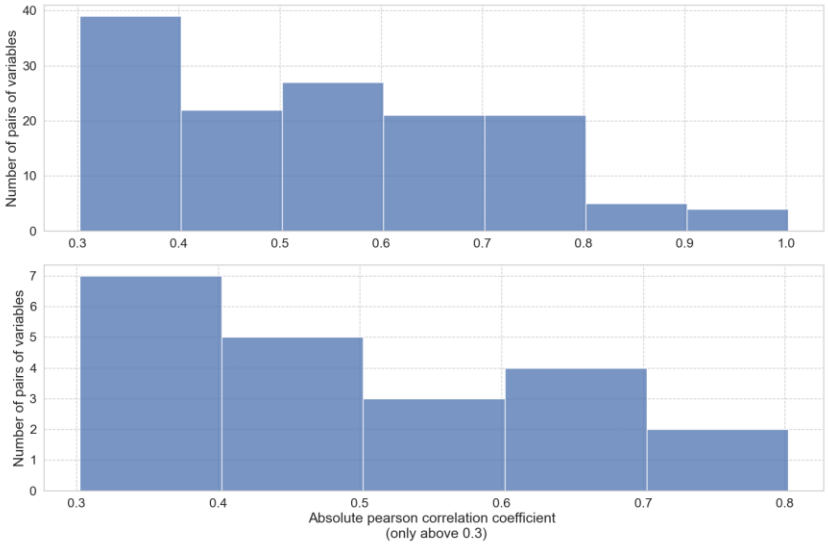


Figure 6. Distribution of absolute Pearson correlation coefficient of pairs of variables before from initial set (plot above) and after selection (plot below)



After variable selection, I build other more complex regression models – random forest, extreme gradient boosting, and artificial neural network-based models.

Random forest

Theoretical basis

Random Forest model is an easy-to-use model, and it is known to show high performance for both regression and classification problems.

The Random Forest is strongly based on the Decision Tree model. The idea behind the Decision Tree regression model can be intuitively understood as a long list of if-else statements. Each node splits the tree further into two new nodes and represents the condition. Two edges coming from the node represent the possible outcomes, i.e., whether the condition is satisfied or not, and they are leading to a further decision node or to a terminal node that shows predicted value of the response variable. Therefore, we can say that the Decision Tree is a hierarchical ordination of multiple decision splits (**called leaves**). The logical question is where those decisions come from. Unlike classification decision tree modes, where the split is based on the minimization of information entropy or Gini index, the quality of a split in regression trees is evaluated based on squared or absolute errors between the average value of the observations inside the node and its actual values. As mentioned above, I selected absolute errors as a performance assessment criterion for this project. The final predictions of the terminal node are the mean of the training subset of responses inside the node.

Random Forest adds more complexity to the Decision Tree model. As the name suggests, a Random Forest consists of a large number of Decision Trees, each of them with a slight variation.

The Random Forest uses bagging to create an ensemble of Decision Trees. Generally, a Random Forest can combine hundreds or even thousands of Decision Tree models. Additionally, when splitting a tree, random forests will only consider a random subset of possible predictors. That is, it intentionally ignores a random set of variables. Every time a new split is considered, a new random

subset of predictors is drawn. If there is a single dominant predictor in the data set, most trees will use the same predictor for the first split and ensure correlation and similarity among the trees, so using a random subset of predictors in each random decision tree ensures that the random forest consists of uncorrelated trees.

Therefore, each of these models will be fitted on random subset of training data (drawn with replacement) and random subset of predictors not to be totally equal. Then the prediction of a new instance is based on the average among prediction of the whole ensemble of these models. Where one machine learning model can sometimes be wrong, the average prediction of a large number of machine learning models is less likely to be wrong. This idea is the foundation of ensemble learning.

Below is the algorithm for building a random forest for regression problems[8]:

For b = 1 to B:

(a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
(b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
    a. Select m variables at random from the p variables.
    b. Pick the best variable/split-point among the m.
    c. Split the node into two daughter nodes.
(c) Output the ensemble of trees $\{T_b\}_1^B$.

Prediction of a new point is calculated as an average prediction of all trees in the forest:

$$\hat{f}_{rf}^{B} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Equation 8. Prediction made by Random Forest regression

Random forest model is built using **RandomForestRegressor** functions in **scikit-learn** package of Python. The model has many hyperparameters that can be tuned, such as but not limited to the maximum depth of the tree, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node, the number of features to consider when looking for the best split.

Random Forest is a computationally efficient technique that can operate quickly over large datasets. In general, more trees provide a better result. However, the improvement decreases as the number of trees increases. So, at a certain point, the benefit in prediction performance from learning more trees will be lower than the cost in computation time for learning these additional trees. Although Random Forest models have been used in many research projects and real-world applications, the associated literature provides almost no directions about how many trees should be used to compose a Random Forest. I based my decision on the work of T.M. Oshiro, P.S. Perez, and J.A. Baranauskas, who analyzed the impact of the number of trees on 29 datasets for classification problems. Their analysis showed that from 128 trees, there is no more significant difference between the forests using 256, 512, 1024, 2048, and 4096 trees. The mean and the median AUC values do not present major changes from 64 trees. Therefore, the researchers concluded that, based on the experiments, a reasonable range to is in a random forest is between 64

and 128, with no major improvements brought by a larger number of trees.[14] Therefore, I selected 100 trees, the default number of trees in the **scikit-learn** package.

Application in this project

I checked by trial and error that changes in the maximum depth of the tree (max_depth) and maximum leaf nodes (max_leaf_nodes) had the most impact on the model performance. So I selected these hyperparameters for tuning via 5-folds cross-validation.

Both the maximum depth of a tree and the maximum number of leaf nodes control the complexity of the trees. More nodes equate to deeper, more complex trees and fewer nodes result in shallower trees.

I also estimated the optimal maximum number of feature m out of total p via cross-validation. Although some literature recommends $m = \lfloor p/3 \rfloor$ or $m = \lfloor log_2 p - 1 \rfloor$, in practice, the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.[8]

I explicitly specified values for each hyperparameter. Optimal values are highlighted in **red**:

Table 5. Grid of hyperparameters tuned via cross-validation in Random Forest (optimal values are in red)

Hyperparameter name	Hyperparameter description	Values
max_feature	Number of features in each tree	[1, 3, **6**, 9, 12, 15]
max_leaf_nodes	Maximum number of leaf nodes	[500, **1000**, 1500]
max_depth	Maximal depth of trees	[10, 17, 25, **32**, 40]

Values of all other hyperparameters not mentioned above remained the default.

I performed an exhaustive search across all possible combinations for optimal values using the function GridSearchCV of the **scikit-learn** package.

Figure 7. Validation performance metrics for all combination of Random Forest model hyperparameters values in the cross-validation grid

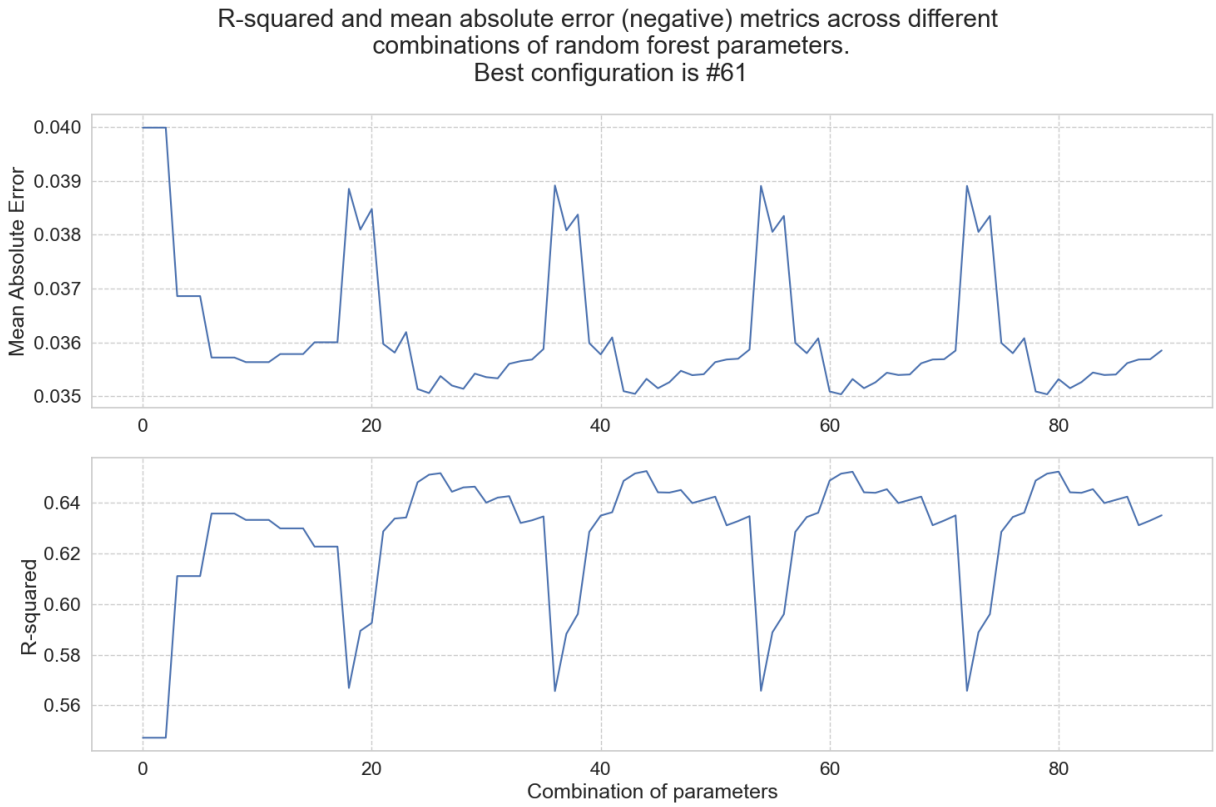
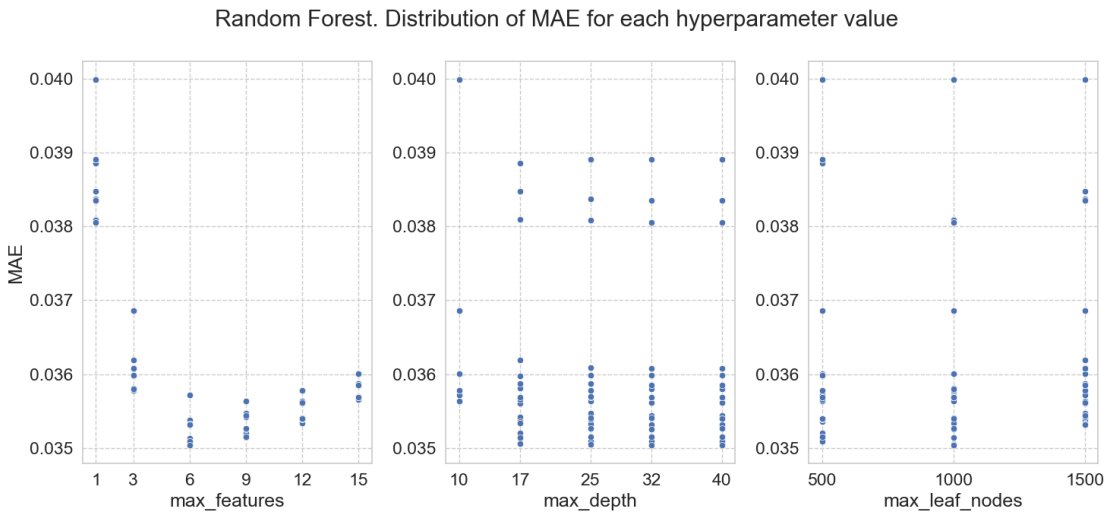


Figure 7 above shows how the MAE and R^2 change with different combinations of hyperparameters in the search grid. MAE plot almost mirrors R^2, i.e., the lowest MAE value corresponds to the highest R^2 value and vice versa. The plot also shows a pattern in which 5 "hills" are followed by 5 "valleys". It indicates that some hyperparameter in the set dominates the others. To check this, I made three scatterplots below, each showing the change in MAE of the model with the value of one hyperparameter fixed while others are changing. They are presented in Figure 8 below. The figure shows that the number of features is the most important hyperparameter as the pattern of change in MAE is pronounced the most with a change in the number of features regardless of the values of other hyperparameters. As for max depth, MAE corresponding to the lowest value of "max_depth" equal to 10 is higher on average than MAE corresponding to the larger values of "max_depth" (i.e., 17, 25, 32, and 40), making the depth of 10 too low resulting to model underfit. At the same time, there is no difference in MAE for values of "max_depth" higher than 10 that can be observable on the plot. Also, the plot does not show a big difference in the model performance for different "max_leaf_nodes" hyperparameter values.

Figure 8. Random Forest. Change in MAE of the model for each hyperparameter value fixed and all other values of other hyperparameters.



Based on the cross-validation results, the optimal value of max_depth is 32, max_leaf_nodes is 1000, and max_feature is 6.

Extreme Gradient Boosting (XGBoost)

Theoretical basis

As the Random Forest, XGBoost also uses decision tree ensembles. However, while random forest uses a bagging algorithm to create an ensemble of Decision Trees, XGBoost uses the alternative technique called gradient boosting. As for bagging, gradient boosting combines numerous small Decision Tree models to make predictions. The critical distinction with bagging is how those decision trees come to be different from each other.

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. Boosting is an iterative process. It adds more and more weak learners to the ensemble model in an intelligent way. In each step, the individual data points are weighted. Data points that are already predicted well will not be important for the learner to be added. The new weak learners will therefore focus on learning the things that are not yet understood and therefore improve the ensemble.

Currently, one of the most popular boosting algorithms is XGBoost. Instead of estimating a prediction as an average over all decision trees list as it the case in Random Forest, XGBoost prediction is a sum of prediction of all trees that were trained via additive training that can be written mathematically as in Equation 9. A detailed description of XGBoost algorithm below is taken from its official webpage https://xgboost.readthedocs.io:[15]

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F$$

Equation 9. Prediction made by XGBoost regression

where K is the number of trees (also known as additive functions), f is a function in the functional space F, and $F = \{f(x) = w_{q(x)}\}(q: R^m \rightarrow T, w \in R^T)$ is the space of regression trees.

Here q represents the structure of each tree that maps an example to the corresponding leaf index. T is the number of leaves in the tree. Each f_k corresponds to an independent tree structure q and leaf weights w. Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use w_i to represent score on i-th leaf.

To learn the set of functions used in the model, the algorithm minimizes the following regularized objective function.

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{t} \Omega(f_k)$$

$$where, \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Equation 10. General formula of objective (obj) and regularization (Ω(f)) functions in XGBoost regression

Here l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term Ω penalizes the complexity of the model (i.e., the regression tree functions).

For this project, loss function is selected as "reg:logistic", which is log loss and rounding predictions with 0.5 threshold as the response is continuous.[i]

The additional regularization term helps to smooth the final learned weights to avoid over-fitting. Hyperparameter λ controls the strength of the regularization, and γ pertains to a minimum loss reduction required to make a further partition on a leaf node of the tree.

The tree ensemble model in the equation above includes functions as parameters and cannot be optimized using traditional optimization methods in the Euclidean space. Instead, the model is trained in an additive manner, i.e., at each step, we fix what we have learned and add one new tree at a time. We write the prediction value at step t as $\hat{y}_i^{(t)}$. Then we have:

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

[i] https://stackoverflow.com/questions/53530189/the-loss-function-and-evaluation-metric-of-xgboost
https://github.com/dmlc/xgboost/issues/521

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Equation 11. Prediction by a tree at each step in XGBoost forest

It remains to ask: which tree do we want at each step? The natural thing is to add the one that optimizes our objective.

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

Equation 12. Objective function of XGBoost regression at each step

More number gradient boosting iterations (i.e., number of estimators) reduces training set errors. Increasing the number of iterations too high increases overfitting. In this case, monitoring the prediction error using a separate validation data set will help select the optimal number of gradients boosting iterations. In addition to tuning the number of estimators, it is also possible to use the depth of trees as an efficient regularization parameter. Increasing the depth of trees leads to the model that is likely going to overfit the training data.

Application in this project

To prevent overfitting and ensure that the model will show strong performance of new data, I did the following:

- Estimated optimal values of the following hyperparameters using 5-fold cross-validation. As for Random Forest, I performed an exhaustive search across all possible combinations for optimal values using the function GridSearchCV of the **scikit-learn** package.

Table 6. Grid of hyperparameters tuned via cross-validation in XGBoost (optimal values are in red)

Hyperparameter name	Hyperparameter description	Values
n_estimators	Number of gradient boosted trees. Equivalent to number of boosting rounds.	[100, 200, 300, 400, **500**]
reg_lambda	Strength of L2 regularization	[4, 6, **8**, 10, 12]
max_depth	Maximal depth of trees	[**3**, 5, 7, 9]

- Activated early stopping when training the model by setting that validation metric needs to improve at least once in every 20 rounds to continue training. Otherwise, the training stops. It is generally a good idea to select the early_stopping_rounds as a reasonable function of the total number of training epochs (10% in this case) or attempt to correspond to the period of inflection points as might be observed on plots of learning curves.[16] It is important to note that early stopping was included in the XGBoost model and artificial neural network-based model below as these models are making the solution more complex with more iterations and are more likely to overfit. As mentioned above, random forest, unlike these models, makes a prediction as the average across a "forest" of decision trees built independently of each other, so it does not require early stopping.

Figure 9 shows the change in performance metrics (MAE and R^2) with different combinations of hyperparameter values. As for the random forest, the MAE plot mirrors the R^2 plot. At the same time, the pattern of the plots is not as clear as for random forest. It means that there is more than one dominant hyperparameter in XGBoost. Figure 10 below shows the change in model performance (MAE) over model hyperparameter values. The pattern for "n_estimators" plot indicates that MAE is lower for 100 estimators, while with the increasing number of trees, it does not differ much, meaning that there are no signs of overfitting when the number of estimators grows. We can also observe the same pattern in the top left corner of "n_estimators" and "max_depth" plot, which means that the depth of 3 has high MAE values only when the number of estimators is equal to 100. When the number of estimators grows to 200 and above, MAE becomes slightly lower for max depth equal to 3 than max depth above 3. The regularization λ plot does not show any particular pattern, which means that it is the least important hyperparameter in the model.

Figure 9. Validation performance metrics for all combinations of XGBoost model hyperparameters values in the cross-validation grid

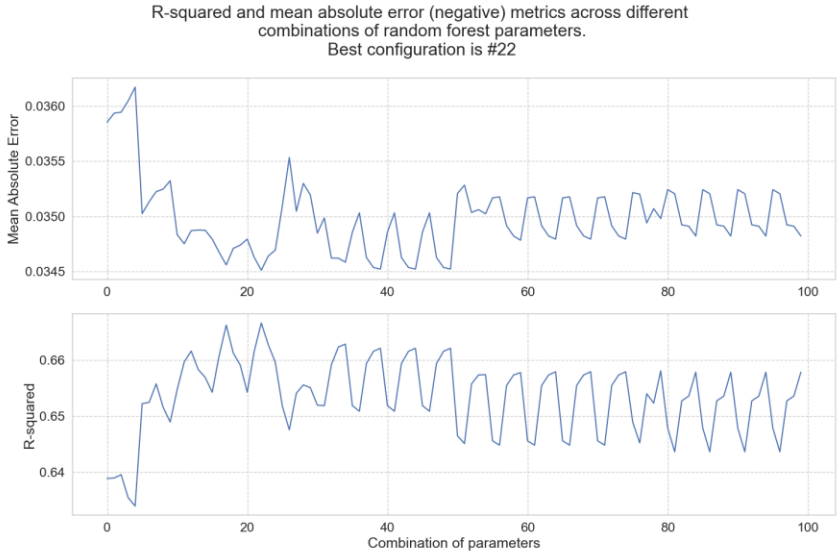
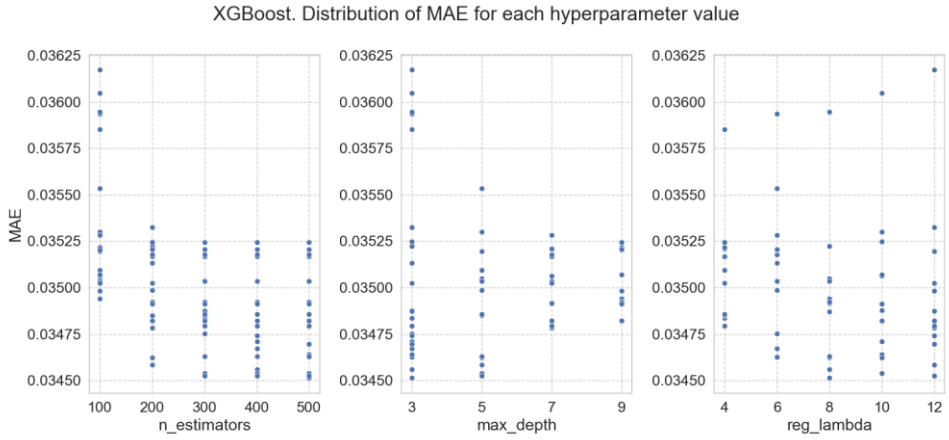


Figure 10. XGBoost. Change in MAE of the model for each hyperparameter value fixed and all other values of other hyperparameters.



The best combination is 500 trees, regularization λ of 8, and max depth of each tree of 3.

Artificial neural networks

Theoretical basis and application in this project

The final model that I will build in this project is regression based on an artificial neural network (ANN). The term artificial neural networks refers to a large class of models and learning methods. A neural network is a multi-stage regression model, typically represented by a network diagram. ANNs are comprised of node layers, and in particular of an input layer, one or more hidden layers, and an output layer. Each node in hidden layers and output layer can be considered as its own linear regression model. All nodes in an ANN are interconnected and have associated weights and thresholds. If the output value of an individual node is above the specified threshold set by a so-called activation function, then this node is activated, and the data from this note is sent to the next layer of the ANN. Otherwise, no data is passed. As a result, the output of one node becomes the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Figure 11. Architecture of a neural network (left) and a node (right)

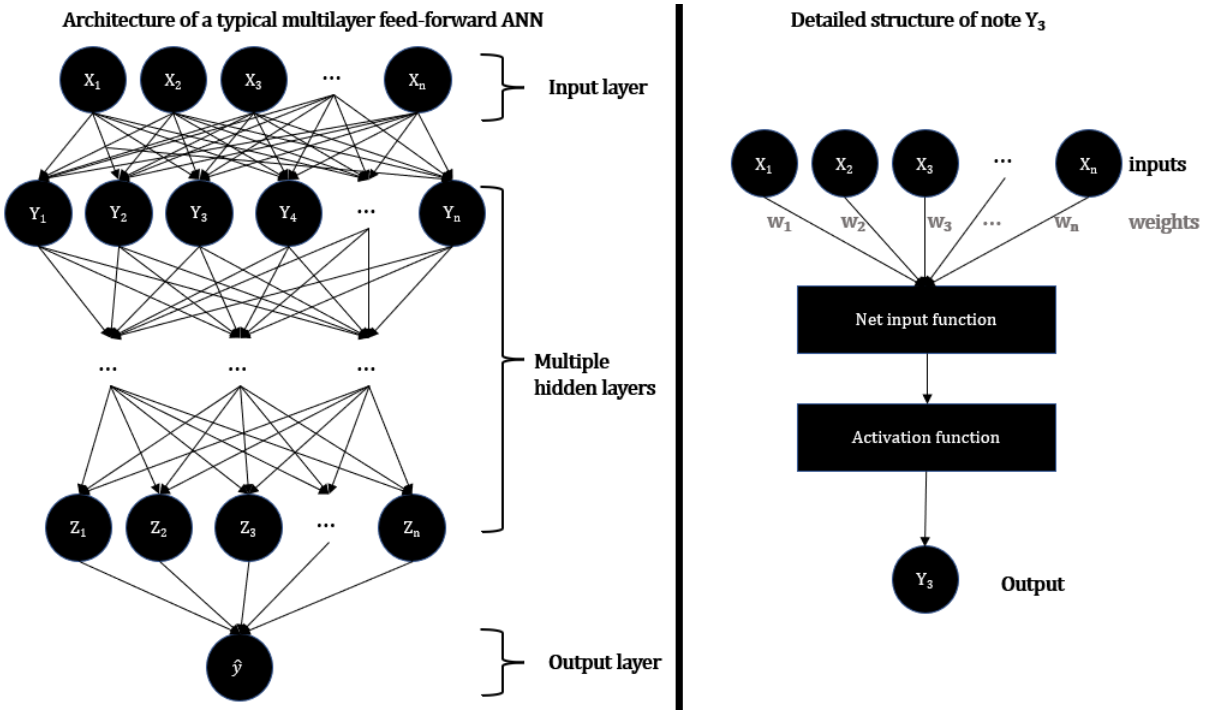


Figure 11 shows the architecture of a typical multilayer feedforward network and how a single node might look in this ANN. In addition to feedforward neural networks, there are other more complex types of ANN, such as convolutional neural networks and recurrent neural networks. Convolutional neural networks are most commonly applied to analyze visual imagery and used for unsegmented, connected handwriting or speech recognition. As the project aimed to predict dropout rates, I considered the feedforward network the most appropriate for this regression problem.

Weights of each node in ANN are estimated at the time of model fitting through the backpropagation algorithm. The backpropagation algorithm computes the gradient of the loss function with respect to the weights of the network, one layer at a time. It then iterates backward from the last layer, which is why it is called backpropagation. It is an efficient way to fit the massive number of weights needed for a neural network. Yet it is still a complex algorithm, and it takes time to compute. The backpropagation algorithm's performances depend on the choice of the optimizer.

When building the artificial neural network, the modeler needs to make a choice regarding the following:

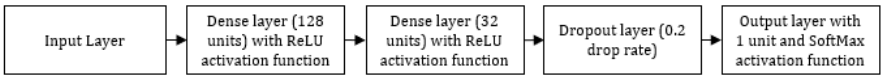
- Neural network architecture
- Loss function
- Optimizer

Neural network architecture

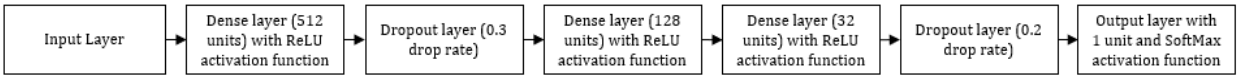
The neural network's architecture is based primarily on the modeler's choice. Generally speaking, it is better to have too many hidden units than too few. With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data; with too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used. Typically the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases.[8]

In this project, I assess three different networks:

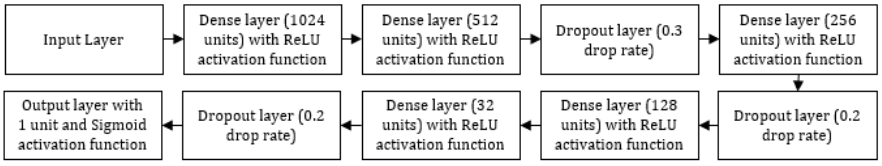
- 5-layer model:



- 7-layer model:



- 10-layer model:



All models consist of Dense and Dropout layers and differ from each other only by the number of layers. Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.[17]

$$output = activation(dot(input, kernel) + bias)$$

Equation 13. Operations performed in the Dense Layer

where,

- *input* represent the input data
- *kernel* represent the weights matrix created by the layer

- *dot* represent numpy dot product of all input and its corresponding weights
- *bias* represent a biased value used in machine learning to optimize the model
- *activation* represent the element-wise activation function.

I considered that the Dense layer would be the most appropriate for the regression purpose. While other layers, such as convolution, pooling, and recurring layers, are used in convolutional and recurrent neural networks, which purpose is different from the aim of this project as discussed above.

I used rectified linear activation function (ReLU) for all hidden dense layers and SoftMax activation for output dense layer. ReLU activation retains the maximum value of 0 and the dot product of the input layer and model weights. ReLU choice is based on the reasons described in the following paragraphs.

Other activation functions, such as sigmoid and tahn, are also commonly used in deep learning. However, the sigmoid activation function transforms a large input space into a small input space between 0 and 1. So, a large change in the input of the sigmoid function will cause a small change in the output, which makes the derivative of the sigmoid function very small. As gradients of neural networks are found using backpropagation by multiplying derivatives from the final to the initial layer using the chain rule, then multiplying derivatives of sigmoid functions in multi-layered neural networks will result in an exponentially small overall gradient of the network. So each step of gradient descent will make only a tiny change to the weights, leading to slow convergence or, in other words, the vanishing gradient problem. ReLU helps overcome the vanishing gradient problem as it does not have a small derivative. It has gradient 1 when output is above zero and 0 otherwise. Hence these derivatives will return either 0 or 1 when multiplying together in the backpropagation process. So, the update is either nothing or takes contributions entirely from the other weights and biases. Another advantage to ReLU, other than avoiding the vanishing gradients problem, is that it has a much lower run time than any sigmoid function, which uses an exponent which is computationally slow. [18]

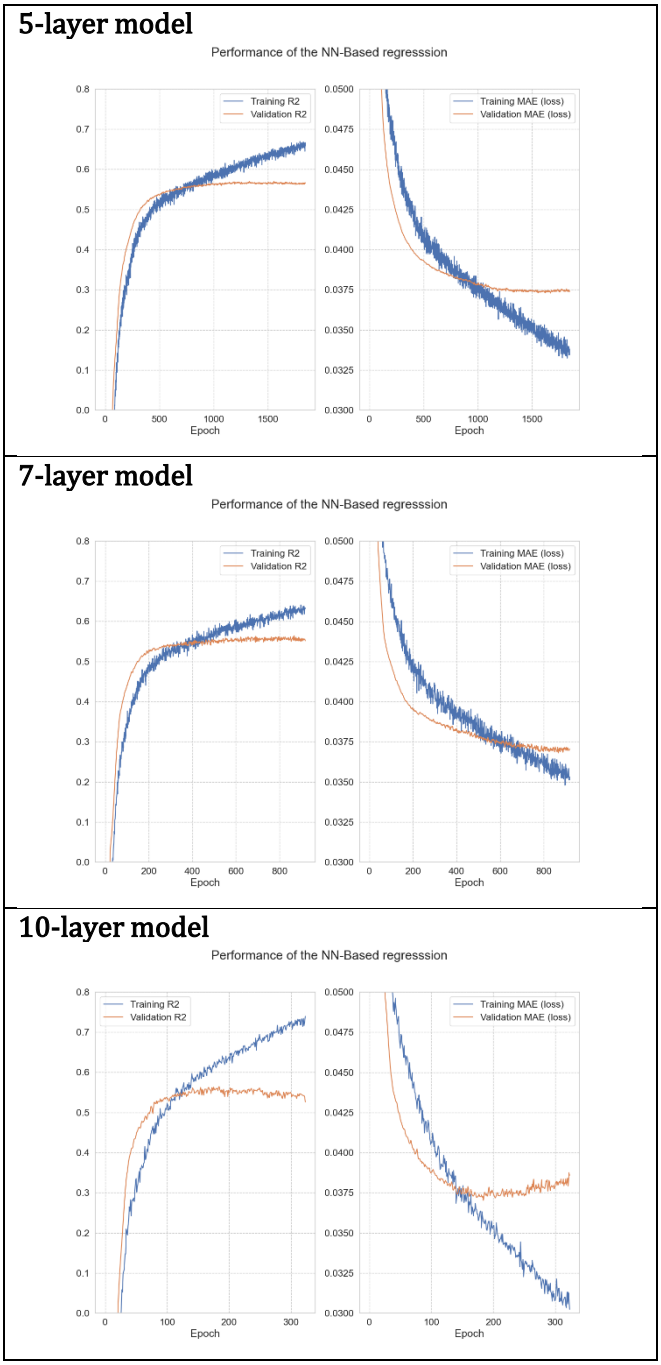
For the output layer, on the other hand, I considered that sigmoid is the most appropriate activation function as it transforms values of the input layer (X) and wights β into a single value in the range between 0 and 1 using the formula in Equation 3 or Equation 4. Therefore, the predicted value is in the same range as the target variable.

The Dropout layer randomly sets input units to 0 with a set rate at each step during training. The dropout layer is used between the Dense layers to regularize the neural network to prevent its overfitting.

Loss function

Mean absolute error was selected as a loss function in line with other models. Its choice is described previously in the project memo. In addition to the MAE loss function, I selected R^2 as a performance metric for other models.

Figure 12. Change in the validation performance metrics over the training cycle of ANN-based regression models of different architecture.



Optimizer

The critical consideration in building the neural network is which gradient descent-based optimizer to choose. There are different types of gradient descent-based optimizers, but they all aim to find the global minimum of loss function by using the gradient of this function. However, these optimizers differ in the ways that they respond to this gradient. I will first look at the most common types of gradient descent-based optimizers and then choose the most appropriate for this task:

- Batch gradient descent. This algorithm computes the instantaneous gradient of the cost function with respect to the model parameters for the entire training dataset. Considering that this algorithm uses all training data at once, it is very computationally expensive; therefore, its use is not practical. Also, as this algorithm uses the entire training dataset, it does not introduce noise in the model parameter estimates, making it more likely that I will get stuck in the local minimum of the loss function rather than its global minimum.

- Stochastic gradient descent (SGD). Stochastic gradient descent aims to overcome the drawbacks of batch gradient descent. Instead of performing computations on the whole dataset — which is redundant and inefficient — SGD only computes on a small subset of random selection of data examples, thus introducing the noise in the model parameter estimates.

- Adaptive Moment Estimation (Adam). It includes a changing learning rate and a momentum term. The optimizer keeps a running average of the recent gradients and squared gradients; effectively, it remembers the values of the gradients experienced, with the hyperparameters

controlling the length of its memory. Compared to the two Gradient Descent optimizers described above, Adam computes not instantaneous but the average gradient of the recent history. This allows Adam to introduce the momentum into gradient descent and allow parameters to change rapidly and cross local minima without getting trapped. Also, the learning rate in the Adam optimizer is divided by the square root of the average squared gradient, which ensures that each step is approximately the same size. If the second moment is large, this term will reduce the step size.

The Adam optimizer adopts better strategies to avoid the local minima. Also, Adam adapts the learning rate to the rate of the descent. There are other optimizers, such as RMSprop and Adadelta, but the way they work is similar to Adam optimizer. Adam is one of the most widely used and practical optimizers for training deep learning models so consider it to be the best overall choice.

The performance of each ANN model is presented in Figure 12. Models with 7 layers and 5 layers show almost the same performance (the lowest MAE and the largest R^2), while the model with 10 layers shows worse performance, and its plot indicates overfit. Considering that model with 5 layers is less complex than with 7 layers while showing the same performance, I selected it for comparison with other non-ANN models described above.

Comparison of models' validation performance

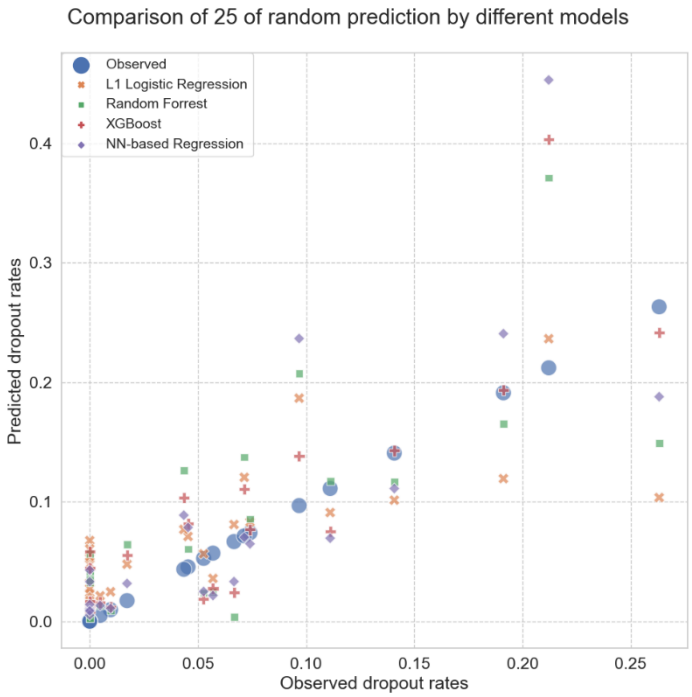
The validation performance of the best-selected model of each type is presented in Table 7. The metrics are calculated on the whole validation data set. But to estimate the variance of MAE and R^2, I took 5,000 draws of 300 random observations from the validation dataset and calculated 5,000 metrics, and calculated the standard deviation of MAE and R^2. Considering the uncertainty over the true MAE and R^2 value, XGBoost regression shows the best performance, while Random forest regression is slightly behind. In contrast, ANN-based regression and penalized logistic regression are far worse.

Table 7. Validation performance metrics of best models of each type

	For the whole validation dataset		**Bootstrap (5000 draws with 300 observations each)**			
Model	Mean absolute error	R2	Mean absolute error (mean)	R2 (mean)	Mean absolute error (std)	R2 (std)
R1 regularized logistic regression	0.042	0.50	0.042	0.50	0.002	0.06
Random forest regression	0.033	0.67	0.033	0.67	0.002	0.05
XGBoost Regression	0.033	0.70	0.032	0.69	0.002	0.05
ANN-based regression	0.037	0.57	0.037	0.56	0.002	0.06

Figure 13 shows the plot of predicted values against the observed for random 25 observations from the validation set. In my, but not all, cases, XGBoost (red cross) is closer to the observed values (blue round) than other models.

Figure 13. Comparison of predictions of 25 random observations made by selected models of each type



Feature importance

The validation performance table above shows that more complex decision tree- and neural network-based models show better performance than less complex penalized logistic regression. However, penalized logistic regression has one significant advantage over other models – it can be easily interpretable. There is a direct relationship between the predictors and the target variable. β coefficient in the logistic regression can be interpreted as how much log odds of a dropout rate increase (if β is positive) or decrease (if β is negative) when the predictor's value increases by 1. And as all predictor values are normalized, we can compare β coefficients directly with each other.

Figure 14. Predictors in L1 regularized logistic regression sorted by importance in descending order

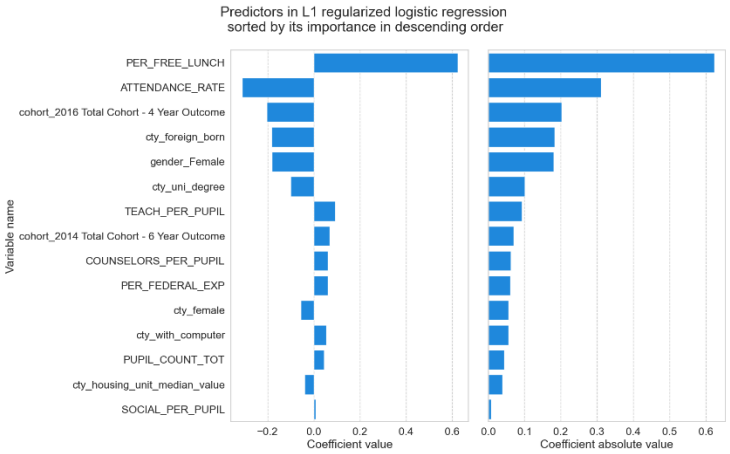


Figure 14 shows the value and absolute value of β coefficient of each predictor. We can see that PER_FREE_LUNCH and ATTENDANCE_RATE have more impact than other variables.

However, in more complex models, there is no direct relationship between predictors and target variables. But there are approaches to extract feature importance for such models.

Feature importance - Shapley Additive Explanations

Theoretical basis

The approach used in this project is called SHapley Additive exPlanations (SHAP). SHAP is a game-theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classical Shapley values from game theory and their related extensions.[19] SHAP is a model-agnostic feature extraction approach, so it can explain the output of any machine learning model. I use **shap** package in Python to calculate SHAP values for XGBoost, Random Forest, and ANN-based regression models.

Shapley values are a widely used approach from cooperative game theory that come with desirable properties. In coalitional game theory, also known as cooperative game theory, the different combinations of players are coalitions, the differences in scores are marginal contributions of each player, and the Shapley value is the average of these contributions over many simulations. For a model, the features are the players, different subsets of features are the coalitions of players, differences in predictive error are marginal contributions.[20]

The math involved in calculating Shapley values for a model involves sets and factorials and is out of the scope of this project. However, all the algorithmic details described in the papers that adapted Shapley values to machine learning.[21]

Speaking of consistency, Shapley values have several properties derived from coalitional game theory that make it ideal as a feature importance method:

- **Dummy**: If a feature i never contributes any marginal value, its Shapley value is zero $Shapley_i = 0$.
- **Substitutability**: If two given features i and j contribute equally to all their possible subsets, $Shapley_i = Shapley_j$
- **Additivity**: If a model p is an ensemble of k submodels, the contributions of a feature i in the submodels should add up.
- **Efficiency**: Likewise, all Shapley values must add up as the difference between predictions and expected values; or in other words predicted outcome of an observation is the sum of base value and Shapley values for all predictors. While base value is the mean of predicted target variable.

$$Prediction = Base\ value + \sum_{i=1}^{n} Shapley_i$$

Equation 14. Relationship between predicted outcome and SHAP values

where n is the number of features.

As described above, in theory, Shapley values (also known as SHAP values with regards to its application to machine learning) of a feature represent the average of marginal contributions of this feature to the predicted outcome across all possible permutations of features in the model. In practice, though, the computation time for Shapley values must invariably grow exponentially as features increase so that a brute-force approach would be very resource-intensive. There are several strategies to minimize computation.

The permutation method is used in this project. It works by iterating over complete permutations of the features forward and the reversed. By doing this, changing one feature at a time, we can minimize the number of model evaluations that are required and always ensure we satisfy efficiency no matter how many executions of the original model we choose to use for approximation of the feature attribution values. So the SHAP values computed, while approximate, do exactly sum up to the difference between the base value of the model and the output of the model for each explained instance.[22]

Application in this project

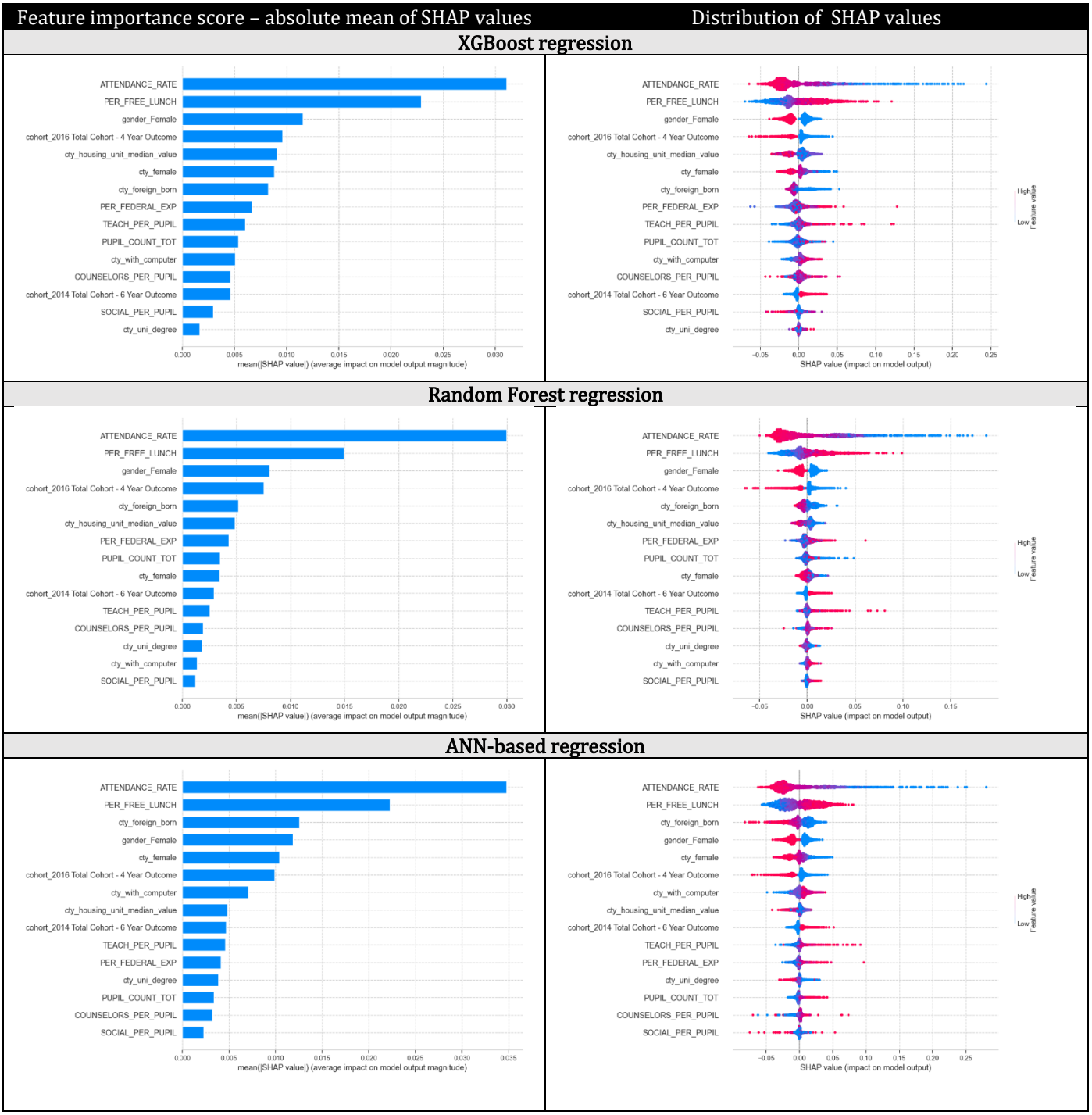
For this project, SHAP values were calculated for the validation dataset, consisting of 1,817 observations and 15 predictors. Accordingly, the SHAP values matrix will have [1817, 15] dimension as the values are calculated for each predictor in each observation. Considering the formula in the efficiency property, we can tell that the SHAP value of feature i in observation j shows how far feature i pushes the prediction in observation j away from the prediction mean across all observations.

Figure 15 shows the mean of absolute SHAP values for each model (left side of the figure), the SHAP values distribution, and its impact on the target variable (right side of the figure). The mean of absolute SHAP values can be interpreted as a feature importance score. And as we can see, the top 5 features are the same in all models, although some are in different places. Also, the least important features that are at the bottom of each of the plots on the left side are similar across the models.

The right side of the plot provides further details on how the features impact the predicted value. The violin plots show the distribution of the SHAP values for each feature across all observations. The further the SHAP values are distributed from the 0, the larger the impact. At the same time, the color of the SHAP values on the plot corresponds to the value of the prediction – red for high values and blue for low values. For example, if we take the SHAP values for ATTENDANCE_RATE that are distributed similarly in all models, we can see that most SHAP values are between -0.05 and 0 are red. At the same long tail of high SHAP values from 0 to 0.2, and these values are blue. This means that high attendance (red) tends to decrease the predicted dropout rates by up to approximately 0.05. While in a few cases, low attendance rates (blue) increase dropout by more than 0.2. PER_FREE_LUCH, on the other hand, has the opposite impact on the dropout rates predictions. All three models tend to predict higher dropout rates for schools with a high percentage of students eligible for free lunch and vice versa.

Predictive machine learning models like XGBoost or ANN models become more transparent when paired with interpretability tools like SHAP. SHAP identifies the most informative relationships between the model predictors and the outcome, which is useful for explaining what the model is doing. However, as already mentioned in the Introduction section, it is important to remember that correlation does not imply causation. SHAP makes transparent the correlations picked up by predictive ML models. But making correlations transparent does not make them causal. Finding causal links between predictors and dropout rates is outside of this project's scope.

Figure 15. SHAP results. Feature importance scores (left) and SHAP values distributions (right) for Random Forest, XGBoost and ANN-based regression



Looking at Figure 14 and Figure 15, all models have similar subsets of the most and least important features, the top five features are the same, although they are in a different order. I calculated the Spearman correlation of feature importance ranks to assess the similarity among the models more precisely. The results are presented in Table 8.

Table 8. Feature rank correlation matrix

Model	Regularized logistic regression	Random forest regression	XGBoost regression	NN regression
Regularized logistic regression	1.00	0.66	0.54	0.67
Random forest regression	0.66	1.00	0.92	0.77
XGBoost regression	0.54	0.92	1.00	0.83
NN regression	0.67	0.77	0.83	1.00

The table shows that the feature ranks in all models have a moderate-to-strong positive correlation. Feature ranks in Regularized Logistic Regression correlate the least with other models as all correlation coefficients are below 0.7. There is a stronger correlation between ANN-based regression and both tree-based models – 0.77 with Random Forest and 0.83 with XGBoost.

At the same time, the strongest feature rank correlation (0.92) is among tree-based models (XGBoost and Random Forest). Considering that these models showed the highest performance, I analyzed further feature importance of these models.

The **scikit-learn** package provides alternative feature importance metrics for both XGBoost and Random Forest. One of these metrics is based on the mean decrease in impurity (MDI). In the case of regression trees, a decrease in impurity will be equivalent to a decrease in MSE at each tree split. However, this method can be strongly biased and favor high cardinality features (typically numerical features) over low cardinality features such as binary features or categorical variables with a small number of possible categories.

Another method provided by the **scikit-learn** package is Permutation-based feature importance. Permutation-based feature importances do not exhibit such a bias. Additionally, the permutation feature importance may be computed performance metric on the model predictions and can be used to analyze any model class (not just tree-based models).[23]

Although permutation-based feature importance is considered less biased, I compared the Pearson's correlation between feature performance scores derived by the **SHAP** package and permutation-based and MDI-based scores from the **scikit-learn** package. The results for the top 5 most important SHAP features of XGBoost are presented in Table 9.

Table 9. Pearson correlation of 5 top features of XGBoost model (SHAP-based scores) with the same features of XGBoost and Random Forest derived by SHAP and MDI- and Permutation-based scores from scikit-learn package.

Model	RF feature importance (MDI)	RF feature importance (permutation)	RF feature importance (SHAP)	XGB feature importance (MDI)	XGB feature importance (permutation)	XGB feature importance (SHAP)
RF feature importance (MDI)	1.00	0.97	0.98	0.71	1.00	0.99
RF feature importance (permutation)	0.97	1.00	0.99	0.85	0.99	0.93
RF feature importance (SHAP)	0.98	0.99	1.00	0.82	0.99	0.96
XGB feature importance (MDI)	0.71	0.85	0.82	1.00	0.75	0.62
XGB feature importance (permutation)	1.00	0.99	0.99	0.75	1.00	0.98
XGB feature importance (SHAP)	0.99	0.93	0.96	0.62	0.98	1.00

As expected, there is a strong correlation between the **scikit-learn** permutation-based scores and (between 0.93 and 1). XGBoost MDI-based scores have a much weaker correlation with other

feature importance basis methods (between 0.62 and 0.85). Different results provided by the MDI-based feature importance scoring method are expected due to the reasons explained above.

Ensemble of models

Before deciding on the final model, it is also useful to check if the ensemble of the models will show a better result than a single model. In machine learning, ensemble averaging is creating multiple models and combining them to produce the desired output, as opposed to creating just one model. Frequently an ensemble of models performs better than any individual model, because the various errors of the models "average out."[24]

This project uses a weighted average ensemble of models, which is a combination of the models' predictions. The weighted average ensemble assumes that some models are better than others and should be given more weight in the overall prediction. The contribution of each model is weighted proportionally to its capability or skill.

A weighted average prediction involves first assigning a fixed weight coefficient to each ensemble member. This could be a floating-point value between 0 and 1, representing a percentage of the weight. It could also be an integer starting at 1, representing the number of votes to give each model.[25]

Table 10. Weighted average prediction of model ensembles (based on validation set)

Combination	XGB_Weight	RF_Weight	NN_Weight	GLM_Weight	MAE	R2
XGB+RF+NN+GLM	0.65	0.25	0.05	0.05	0.0321	0.705
XGB+RF+GLM	0.65	0.30	-	0.05	0.0321	0.706
XGB+RF+NN	0.65	0.30	0.05	-	0.0319	0.706
XGB+RF	0.65	0.35	-	-	0.0319	0.707
XGB+GLM	0.90	-	-	0.10	0.0327	0.701
XGB+NN+GLM	0.90	-	0.05	0.05	0.0325	0.701
XGB+NN	0.90	-	0.10	-	0.0324	0.700
XGB	1.00	-	-	-	0.0325	0.699

Considering that the XGBoost model showed the best result, I calculated the weighted average prediction with this model and all other models. The calculation was made for all possible weights with a step of 0.05 so that the sum of all weights is equal to 1. The table below shows the best predictions of different combinations of models. Considering that the standard deviation of MAE is approximately 0.002 and of R^2 is approximately 0.05 (see Table 7), the performance gains of the best combination (XGB + RF) are a 0.0006 decrease in MAE and a 0.008 increase in R^2, which is less than 1/3 of the standard deviation of both performance metrics. I considered performance gains are not substantial enough to consider more complex model and, therefore, I selected XGBoost regression model as a final choice.

Estimating the performance of the final model on the test dataset

The performance of the XGBoost regression model is presented in Table 11. Both mean absolute error and R^2 dropped compared to validation metrics; however, the drop is within one standard deviation, and I consider it acceptable.

Table 11. Performance of XGBoost on test set

Model	Mean absolute error	R2
XGBoost Regression	0.035	0.66

Conclusion and discussion

In this project, I aggregated the information on public school dropouts in New York State together with other social and economic data in public schools and counties of the state.

The aim of the project was to build 4 different models, regularized logistic regression, random forest, gradient boosting, and artificial neural network models that predict the school dropout rates and compare the performance of these models. Tree-based models and artificial neural networks built in this project are commonly known as "black-box" models that make predictions without revealing any information about their internal workings. So, another aim of this project was to use model agnostic tools that would make these models interpretable to understand what is behind the "black box". SHAP, or SHapley Additive exPlanations, was employed for this purpose in the project. After interpreting the models, I compared them not only in terms of their performance but also in terms of which variables in each model impact the prediction the most.

As presented in Table 7, tree ensemble methods, Random Forest and XGBoost outperformed ANN-based model and regularized logistic regression, while XGBoost did marginally better than Random Forest.

While tree ensemble models provide lower interpretability than logistic regression models, they can capture non-linear and more complex relationships, so I initially expected that they outperform logistic regression.

However, I did not expect Random Forest and XGBoost to show better mean absolute error and R2 than the ANN model. There may be various reasons for this:

- Generally, deep networks need a lot of data to beat more traditional machine learning algorithms, so the dataset in this project may not have enough features that ANN model could have used to beat the ensemble models.
- Deep neural networks are most useful for problems for which the features carry almost no information individually. These models are more often used for the problems with unstructured data (images, text, audio recordings), such as computer vision and natural language processing. At the same time, the problem in this project is a regression problem using a structured (tabular) input dataset. Ensemble models are far simpler models designed to work with structured data.

Looking at the contribution of variables to final predictions, I found out that all models had the same 2 most important features, "ATTENANCE_RATE" and "PER_FREE_LUNCH", and that absolute importance scores these features (β-coefficient value for regularized logistic regression and mean of SHAP values for other models) were far larger then scores of other features (see Figure 14 and Figure 15), which means that these two features contribute the most to the prediction of the dropout rate. But I need to reemphasize that high feature importance scores should not be mistaken to indicate a strong causal link between the predictors and the outcome due to the reasons already mentioned in the introduction of this project.

The strongest rank correlation was among XGBoost and Random Forest, so I compared Pearson correlation of 5 most important variables derived by different approaches to the assessment of feature importance (SHAP and scikit-learn MDI- and Permutation-based importance scores). The

correlation was almost perfect (above 0.9) for most approaches, except for MDI-based scores, as presented in Table 9.

As a final step, I assessed MAE and R2 of weighted average predictions of XGBoost with other models (see Table 10). However, the increase in performance was far below the standard deviation of performance metrics, so I considered keeping a less complex model consisting only of XGBoost.

The problem that I faced in doing this project is that some standard R and Python packages can understand that the proportional target variable is drawn from the binomial distribution, and these packages can apply the cross-entropy loss, which is used for classification problems, to train the model to predict dropout rates. An example of such packages is **glmnet** used to train regularized logistic regression described later in this report. **Glmnet** uses distribution families from the **stats** package, in which, for the binomial family, the response can be specified in one of three ways[26]:

- As a factor: "success" is interpreted as the factor not having the first level (and hence usually of having the second level).
- As a **numerical vector with values between 0 and 1**, interpreted as the proportion of successful cases (with the total number of cases given by the weights).
- As a two-column integer matrix: the first column gives the number of successes and the second the number of failures.

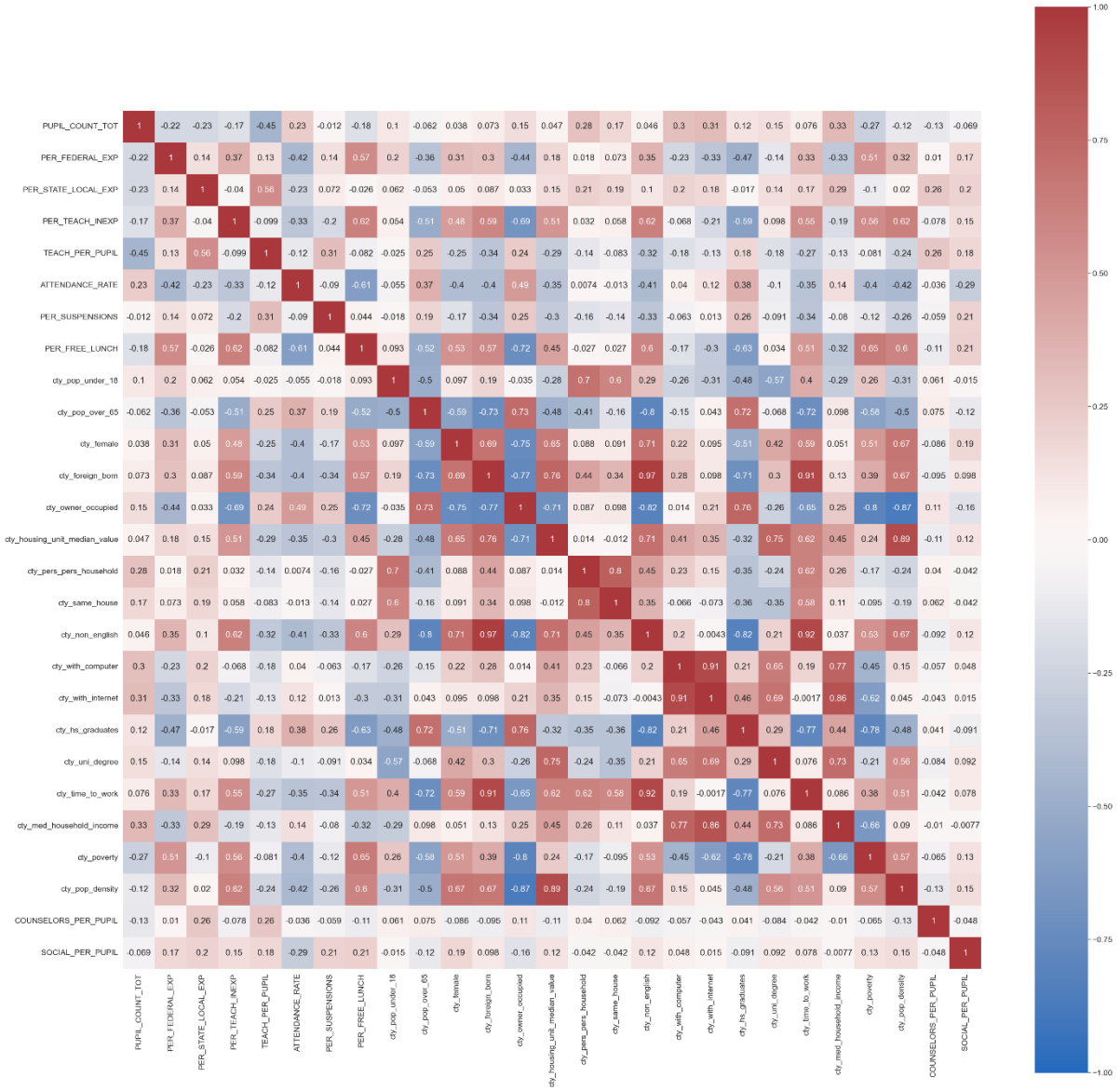
xgboost package in Python also allows using cross-entropy as an objective function for the proportional target variable.

However, other standard packages such as **Keras of TensorFlow** that was used to train artificial neural networks based models do not allow to use cross-entropy loss with a numerical response. Therefore, I built these supervised machine learning models using loss functions for regression problems. Cross-entropy loss (equal to negative log-likelihood) is described in "Penalized Logistic Regression" section of this report, and the reason for selection Mean Absolute Error as a loss for regression problems is described in "Defining loss function and performance metrics" section.

Due to limited time, I focused only on Regularized/Penalized Logistic Regression, Random Forest, XGBoost, and ANN-based regression models in this project. However, there are many other ways to model the proportional data. For example, for the type of proportional data, where the values are between 0 and 1 but are not exactly binomial, one may use an over-dispersed logistic or beta regression model, allowing more flexible variance structures[10].

As a final remark, I would like to emphasize that, as I already mentioned in the introduction section, National Center for Educational Statistics data shows that in the United States, the nationwide dropout rate decreased from 8.3 percent in 2010 to 5.1 percent in 2019. The models built in this project are based on static 2020 data and may not show the same accuracy when applied to future data.

Appendix A. Correlation matrix before variable selection



List of Sources

[1] Stephen Lamb... [et al.], editors. (2011) School Dropout and Completion: International Comparative Studies in Theory and Policy. DOI 10.1007/978-90-481-9763-7
[2] Lehr, Camilla A., et al. (2005) Graduation for All : A Practical Guide to Decreasing School Dropout, SAGE Publications.
[3] https://nces.ed.gov/fastfacts/display.asp?id=16
[4] https://en.wikipedia.org/wiki/Cohort_(educational_group)
[5] Joos Korstanje (2021) Advanced Forecasting with Python: With State-of-the-Art-Models Including LSTMs, Facebook's Prophet, and Amazon's DeepAR. ISBN: 978-1-4842-7150-6
[6] David G. Kleinbaum, Mitchel Klein (2009) Logistic Regression. A Self-Learning Text. DOI 10.1007/978-1-4419-1742-3
[7] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2021) An Introduction to Statistical Learning : with Applications in R, Third Edition, DOI 10.1007/978-1-0716-1418-1
[8] T. Hastie et al., The Elements of Statistical Learning, (2008) Second Edition, DOI: 10.1007/b94608_2
[9] D. Forsyth, Applied Machine Learning, https://doi.org/10.1007/978-3-030-18114-7 10
[10] Chen, K., Cheng, Y., Berkout, O., & Lindhiem, O. (2017). Analyzing Proportion Scores as Outcomes for Prevention Trials: a Statistical Primer. *Prevention science : the official journal of the Society for Prevention Research*, *18*(3), 312–321. https://doi.org/10.1007/s11121-016-0643-6
[11] Stoltzfus JC. Logistic regression: a brief primer. Acad Emerg Med. (2011) Oct;18(10):1099-104. DOI: 10.1111/j.1553-2712.2011.01185.x.
[12] https://stats.stackexchange.com/questions/169664/what-are-the-assumptions-of-ridge-regression-and-how-to-test-them
[13] T. Hastie, J. Qian, K. Tay, An Introduction to glmnet, https://cran.r-project.org/web/packages/glmnet/vignettes/glmnet.pdf
[14] Oshiro, T.M., Perez, P.S., Baranauskas, J.A. (2012). How Many Trees in a Random Forest?. In: Perner, P. (eds) Machine Learning and Data Mining in Pattern Recognition. MLDM 2012. Lecture Notes in Computer Science(), vol 7376. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-31537-4_13
[15] XGBoost: A Scalable Tree Boosting System Tianqi Chen, Carlos Guestrin https://xgboost.readthedocs.io/en/stable/tutorials/model.html
[16] https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/
[17] https://keras.io/api/layers/core_layers/dense/
[18] Theodoridis S., (2020) Machine Learning: A Bayesian and Optimization Perspective, Second Edition. ISBN: 978-0-12-818803-3
[19] https://shap.readthedocs.io/
[20] Masís, Serg. (2021) Interpretable Machine Learning with Python : Learn to Build Interpretable High-Performance Models with Hands-on Real-world Examples. ISBN 978-1-80020-390-7
[21] Scott Lundberg, Su-In Lee, A Unified Approach to Interpreting Model Predictions, https://doi.org/10.48550/arXiv.1705.0787
[22]https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/explainers/Permutation.html
[23] https://scikit-learn.org/stable/modules/permutation_importance.html
[24] https://en.wikipedia.org/wiki/Ensemble_averaging_(machine_learning)
[25] https://machinelearningmastery.com/weighted-average-ensemble-with-python
[26] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/family