Machine Learning Report - Superman Lopez

Mihaly Gayer, Unai Lopez Fernandez, Shreyasvi Natraj

1.1 Introduction

Within the scope of the Machine Learning class 2020 at GSEM, we were provided with data regarding previous advertisement campaigns by a major tech company for the data competition called 'Online Ads Quality Evaluation'. For the objective of the given challenge, our team required to train a model that can be used to predict whether a "conversion" of the potential consumers occurs. A conversion is when a user clicks on the banner and then subscribes to the service.

We are given a 'train' dataset that consists of several thousands of previous data with known conversion outcome. Applying models to the 'train' data will enable us finding models that will predict the conversion for the 'test' data where the conversion outcome is yet unknown. **The target variable in the report is y, a categorical variable with output 0 for no conversion and 1 for conversion.**

After predicting the outcomes of the 'test' set we upload our results to the online platform, Kaggle, to see the real prediction accuracy. Therefore, a direct evaluation result is available for us based on the prediction accuracy according to the function:

$Accuracy = \frac{1}{N}\sum_{1=0}^{N} 1(y_i = \hat{y_i})$

y_i and $\hat{y_i}$ are the true and the predicted outcomes for the test set with size N. We will use this direct feedback together with our prior evaluation prediction accuracy to determine our models of choice.

In the next chapter we will describe the data by looking at each variable separately. Then, we observe the distribution and correlation structure and make conclusions how to do modifications to the parameters, a procedure called feature engineering. After that, we will identify models separately that would likely predict the conversion at a successful rate. Each of these models could be improved by fine tuning its hyperparameters. At this step, cross-validation will be implemented. For each method we would like to optimise the model by identifying the best form of input data through feature engineering and applying the most efficient hyperparameters. Consequently, we compare the fine tuned models with each other to choose one final model.

1.2 Notation introduction in modelling

The initial data sources were named accordingly:

- training.csv -> train.rds
- test.csv -> evaluation.rds

Since we only use test.csv data to evaluate and submit our results we call this evaluation set. The test set is obtained from the train set.

We used the sample.split function from the caTools R library. It is useful for splitting the data into two sets in predefined ratio while preserving relative ratios of different labels in the columns. It is particularly used during classification problem to separate the data into train and test subsets. For each final result, we used cross-validation framework instead of the train_train and test sets.

Our subset for training consists of 75 percent of the data and is named: train_train.rds

The rest of the data is used for testing the models using the 25 percent of the observations and is named as: test.rds

When we used logarithmic transformation on the data we added the notation of '_log' at the end of each data set, obtaining:

- train_log (100%)
- train_train_log (75%)
- test_log (25%)

The notation is similar when we used the normalised data adding the notation _norm in the end of each set. It is important to note that after the modelling, when evaluating the results, the data in the evaluation set (originally test.csv) has to be in the same format as the data that was used during the training of the model. Therefore our evaluation sets are transformed accordingly which gives us the names:

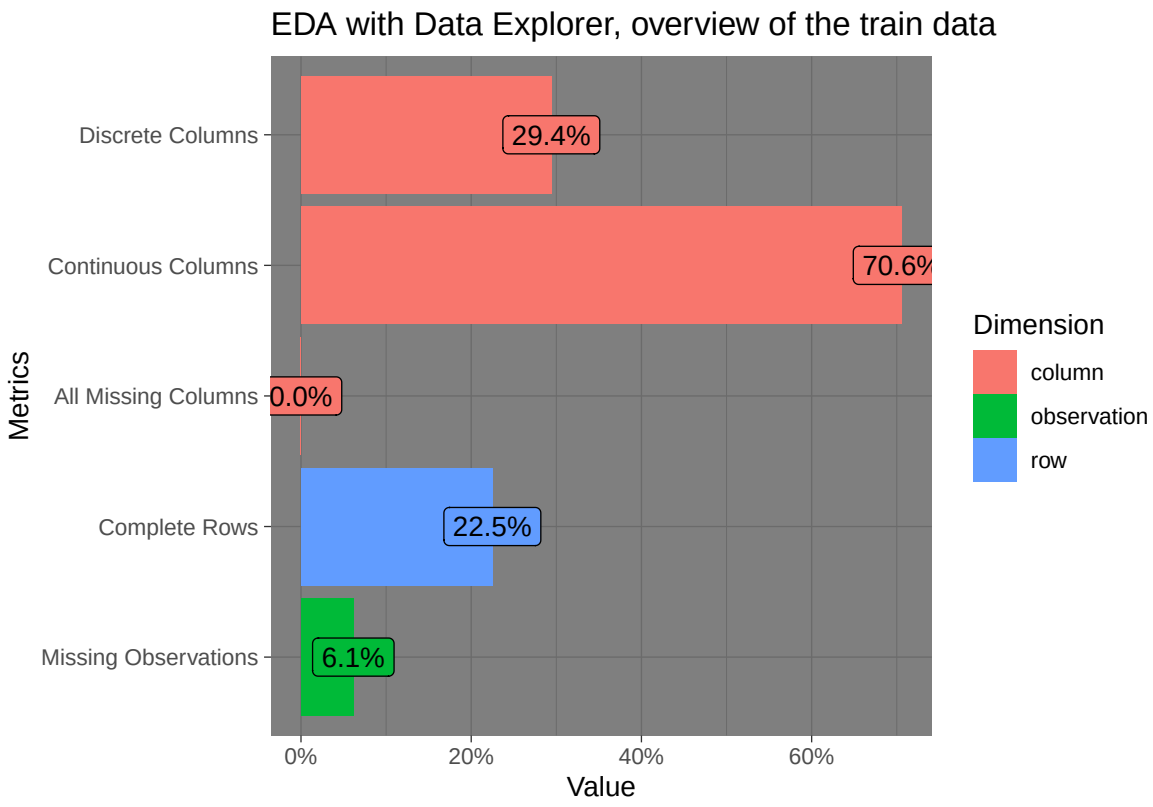
- evaluation
- evaluation_log
- evaluation_norm

In the following chapter we observe the data closely, looking at every variable in the columns, observing the number of missing values, ratios, distribution and correlation.

2. Exploratory data analysis

In the exploratory data analysis we look at the categorical and numerical variables separately in the train set. We will apply statistical models that use the x covariates to make predictions on the target variable y. The goals of carrying out an exploratory data analysis on the train set include:

- having an overview of the data such as the number of observations and number of missing information
- seeing the number of observations per category for categorical variables
- getting information about the range, the distribution and the deviation of the of the numerical variables and in case it is necessary suggest data transformation based on the result
- identifying correlations between the variables x and y, identify possible linear relationships that might have causal effects and more importance in our modelling later
- observing and check for multicollinearity between covariates
- suggesting and implement possible changes in the current form of covariate, as known as feature engineering
- last, but not least examining and check the similarities of the data between the train set and the evaluation set (previously test.csv), as when making predictions, we assume that the data of the covariates come from the same distribution for the train and evaluation data.



The train dataset is composed of 8526 rows that is equal to the number of observations, and 17 columns (including the target variable y).

The 8 numerical variables are: age,day,month,time_spent,banner_views, banner_views_old ,days_elapsed_old and x4.

We have 9 categorical variables, of those called 'outcome_old', 'job', 'marital', 'education' and 'device' have several categories and 'X1', 'X2', 'X3', 'Y' that are variables with binary outcome 0 and 1.

Firstly, we focus on the target variable. We observed that the conversion happened in 41.83 percent of the cases, and there is no missing information for the target variable. This is an important result because balanced classes is a requirement for stable prediction performance for both categories.

As a next step, we examine the numerical variables from the train set. The histogram plots (figure 2) show the values and the frequencies using barcharts. We can observe that the scale of the variables differ. Except the variables month and day, the distributions of the variables seem to be right-skewed. There tends to be several variables with very high variation throughout the data.

To have a more precise modelling result we will take the logarithm of the numerical variables, this way the distribution of the data will be more symmetric. There presence of outliers ,especially in variables time_spent, X4 and days_elapsed_old can bias predictions. Taking the logarithm will also decrease the deviation in the data. One other approach that we will follow is to normalize the numerical data. We also created the normalised version of the train and test sets and carried on with the modelling using both the logarithmic and normalised datasets.

The age variable indicates that the potential customers from the last campaign are between 18 and 94 years old. One interesting fact that stood out is that there is frequently a -1 value for variable banner_views_old. It is strange as this variable intends to show the number of days elapsed since the banner was seen in the previous ad campaign and non-positive values are not making sense. The -1 value indicates that the banner was not seen, thus -1 has an additional meaning. When log transforming the data, due to the non-positive observations the data needed to be shifted for both variables banner_views_old and time_spent.

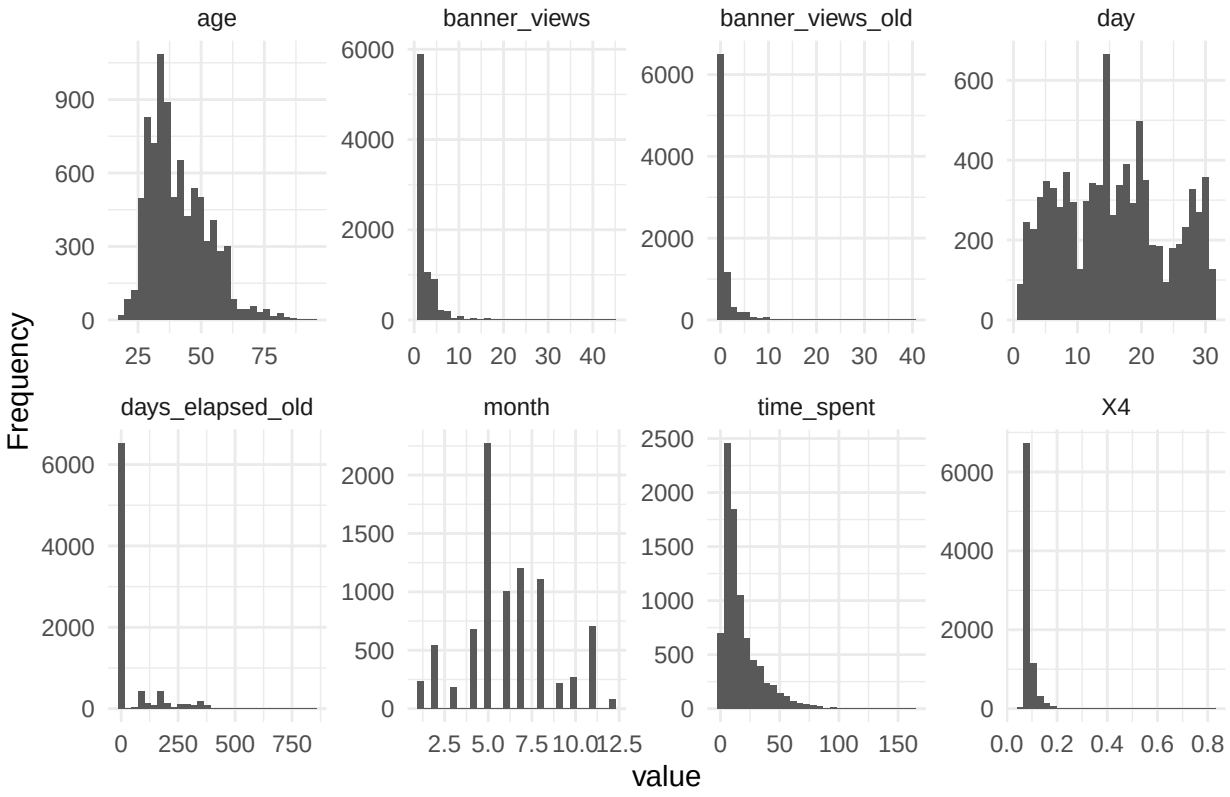


Figure 1: Histograms of numerical features

As a next step we will look at the frequencies of the different type of values for the categorical predictors and observe the number of NA values or any inconsistencies.

We found some missing values in 4 of the regressors job, education, device and outcome old. Especially high ratio of NA values occur for variable device and outcome_old.

We have 11 different job categories for the respondents and less than 1 percent of the observations have unknown job category. This should not have an effect on prediction performance. The highest proportion of job category is managers which is followed by industrial workers. Some less represented categories are unemployed, entrepreneur or housekeeper.

In terms of the highest education of the participants, more than 50 percent of the people have university education. About 31 percent have graduate school and less than 14 percent have high school education. The NA values for this category is lower than 5 percent.

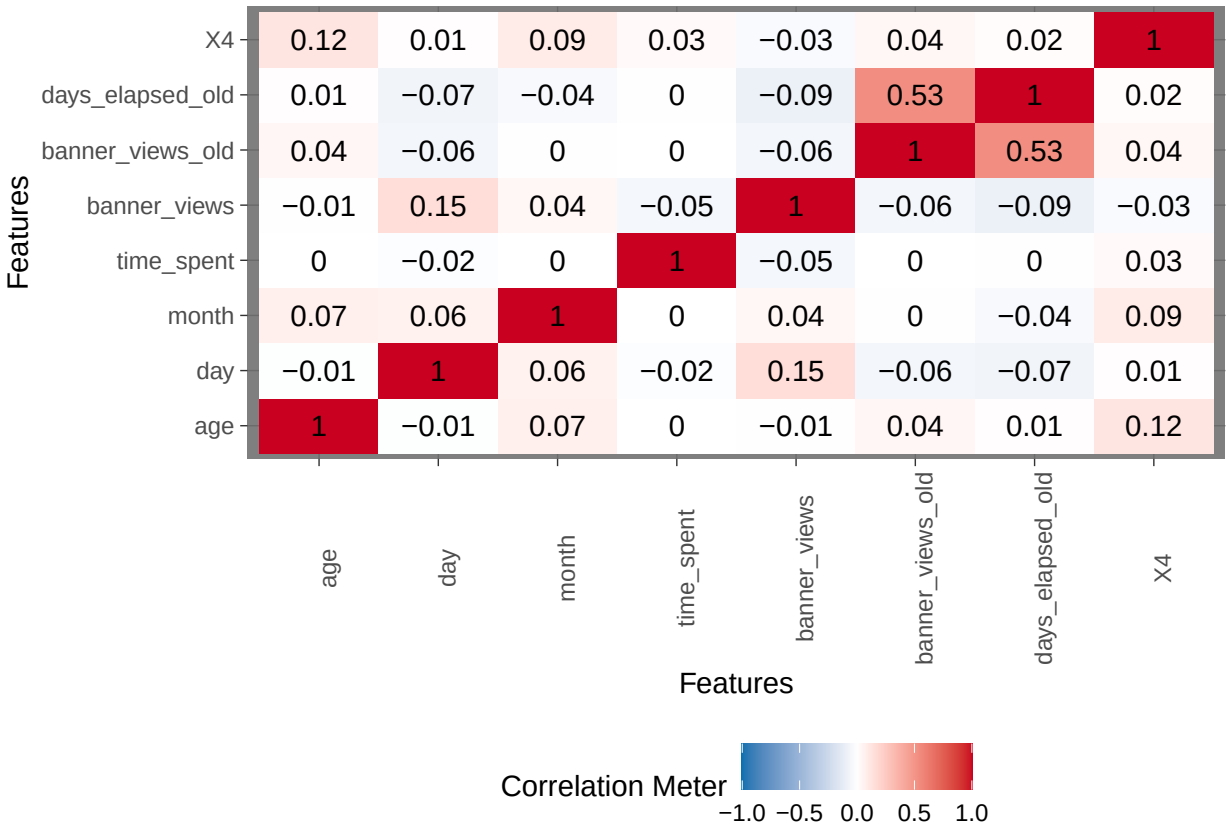
According to our data the majority of the people (more than 70 percent) sees the campaign on their smartphone, whereas less than 7 percent of the respondents have engaged on their desktop. The ratio of NA values in this category is higher (about 23 percent) compared to that of the previous two variables.

The outcome_old is an important variable as it shows information about the previous purchasing behaviour of the customers. About 10 percent of the people did not purchase the service and more than 8 percent successfully bought the product in the past. There exist an other category with less than 5 percent of responses. Out of all the responses more that 76 percent of the observations are unknown in this category, which could be an issue later in the modeling. We can see that the outcome_old NA values correspond to the bannerviews_old: -1 and outcome_old: -1 in the data.

Another categorical variable that does not have NA values is marital status. The rest of the categorical variables are X1,X2 and X3 are binary variables with no additional information and no missing values.

In the next step, we observe the Pearson correlation coefficient table and then the bivariate scatterplots of those pairs that have larger coefficients. The binary variables are also included in the table, but the categorical variables with several text outcomes are excluded.

The correlation coefficient of days elapsed old banner views old is the highest,being of 0.53. The largest linear dependence concerning the target variable is with time_spent (0.47). It allows us to conclude that on average the more time the consumers spent with the online advertisement the more likely they purchased the service, thus conversion occurred. Another significant coefficients is the one between the target variable and banner_views/banner_views old. In case we had a large correlation between two variables, we could omit one of them or create one variable merging the two. However, it is not necessary in this case. Observing the correlations of the covariates we could see that there is not multicollinearity. This is an important finding that will allows us to have more precise modelling result.



This correlation table does not include information about the variable outcome_old. We discussed that we are interested specifically in this variable as this shows the past purchasing behaviour for the same customer. Due to the large amount of missing information the correlation is looked at only for those cases when the outcome_old is not NA. Then, we transformed the data for the outcome old to numerical, where 1 represents success, -1 failure and 0 represent other category. We have obtained 0.42 as correlation coefficient for the outcome_old and the y. It is the second highest correlation, after the one of the pair time_spent and y.

The correlation plot reflects linear dependence only. Furthermore, we have looked at bivariate scatterplots to see if some non linear relation between the variables exist but we did not find a significant dependence pattern to include in the analysis (trials included transformed squared variables).

2.2 PCA transformation

Carrying out a PCA for the data transformation we found that the data cannot be efficiently explained with the first few principal components. Also, looking at the correlation plots one could conclude that it is not necessarily beneficial to do such an approach since the linear dependence is very small in the covariates. Often the high multicollinearity could motivate the PCA feature engineering but it is not the case in this situation. Although we carried out the PCA and tested a few models with the data input, we rejected to go on the path of using the principal components of the data for the final modeling.

2.3 Same distribution

The standard practice is to train the algorithm on the data used in the train set, then use the algorithm to predict the target variables. It is often over-looked to check whether the data in the train set comes from the same distribution as the data in the evaluation set (test.csv). There exist several statistical tests for

this cases, these include for example the non-parametric Kolmogorov-Smirnov test (Upton et al, 2014). The result of such test could be used to identify whether there exist enough evidence to reject the hypothesis that the data comes from the same distribution with a certain probability. The numerical variables are compared. The test carries out a comparison between the two datasets of comparison. It observes the largest possible difference of distributions and assign a p value based on the result and the sample sizes.

The p values of the test indicate the following results:

- The age of people in the old campaign significantly differs from the age of those that take part in the current campaign
- Less important finding is that the people saw the campaign on a significantly different times during the month in the old campaign compared to new campaign
- The rest of the numerical variables are concluded to be coming from the same distributions.

All in all the distribution testing compared the nature of the data of the training set and the evaluation set. We can conclude that the distribution of the two dataset is very similar, that allows us to train the machine learning algorithm on the training set and make meaningful predictions on the evaluation set with better chance.

2.4 Imputation techniques

As a next step we show how we aimed to solve the issue of having no information for certain columns. To quickly recap, the four columns of interest for the imputation, with the respective percentages of missing information per column are:

- outcome_old: 76.3%
- device: 23.41%
- education: 4.2%
- job: 0.61%

Removing the na values as NaN tends to take out a large portion of the data that might be useful for training the model. So we decided not to remove row observations that has missing information in one of the columns.

Primarily the outcome_old missing observations are aimed to be imputed. We have come up with an unconventional approach, to create a model that estimates the output outcome_old based on the known covariates of the other variables. There exist larger correlation of outcome old with the other variables including the target y. This approach uses the un-precise assumption that there is a relationship between the present information of the consumer and the past purchasing behaviour. We have built a random forest model that carried out the imputation. It resulted in having complete information for variable outcome_old, with three remaining categories: success, other, failure. Consequently we used this imputed train set to model and predict for variable y. Using this extra information from the imputed column significantly improved our prediction result.Our random forest model described later achieved a cross validated prediction accuracy of 92.7 percent on the training set. The percentage point improvement was about 5%. However, this part result only applies to our cross validated training set prediction accuracy. When it was used for predicting the final test.csv results, we faced a challenge. The evaluation set had still large amount of missing values for the outcome_old column. The same imputation procedure using modeling is not possible in this case, because the y value outcomes are unknown (contrary to the case with training set). Therefore, the precision for prediction dropped to nearly 50 percent, as the model performed poorly because it was trained on the imputed set with 3 categories in the outcome_old without the category of NA values, thus when the final evaluation was carried out the NA values were 'new' to the model.

Instead of this imputation technique we proceeded with a morel general approach for treating the missing information in the columns. We stored the 'na' responses as text and decided to deal with them as a separate category. There were cases when the category other was available, however decided to distinguish the NA-s and keep it constantly as a separate response.

3. Model building and model prediction accuracy

In this chapter introductory models are explained, then the three most optimal models are chosen. The steps of going from one model to the next includes reasons that serve as arguments for our model selection process. The model selection process was helped by the prediction accuracy of each models. However we did not solely rely on the prediction accuracies but also considered other factors such as: model description about strengths and weaknesses, experiences of others that implemented the model in similar situations (scholars and other peer reviews) and prediction stability over the two outcome categories and prediction variance. After choosing the algorithms, the sensitivity analysis of the hyperparameters are carried out with cross validation. we submitted the results to Kaggle and used the prediction accuracy feedback as a reference.

3.1 Introductory model - logistic regression

Since we are in the classification framework with two possible outcomes for the target variable y. It is convenient to apply a general logistic regression including every predictor variable in the model. The first approach then is to fit a logistic regression model using the standard R package and function glm with the parameter, family equals binomial.

- It is a classification model for 2 classes where $Y \in (0, 1)$.
- We assume that the feature vector X is such that $x_1 = 1$ (for the intercept), then $x \in R^p$ $p = 1, ..., 17$
- Logistic regression links the probability of class 1 to $x^t.\beta$ by:

$Pr(Y = 1|X = x) = sigma(x^t.\beta)$, where $sigma(t)$=$exp(t)/1 + exp(t)$

- The function $sigma$ is called the logistic function: it maps $x \in R$ to $(0, 1)$.
- The estimation of the parameter B is done by maximizing the conditional log-likelihood of the sample $(x_1, y_1), ..., (x_n, y_n)$.
- It uses the Bayes classifier based on these estimates to predict the class:
- $y_0 = 1$ if $sigma(x_0^t.B) > 1/2$

Based on the results of our exploratory data analysis, we decide to apply this model to the log transformed data to see if it improves the accuracy. We succeeded to improve by approximately 1 percent point.

test set accuracy: 0.82168 Kaggle score, 30 percent: 0.81860 Kaggle score, 70 percent: 0.82104

This model gave us an accuracy of approximately 82%. However, it uses all the parameters. As a next step we are interested to know whether there are certain variables that are not of interest when predicting y. In case there was, it is possible to omit these variables hence improve prediction. One approach to do so is to rerun the logistic regression without the variables whose coefficient were not significant in our model summary. Carrying out the model omission approach we arrived to models that were less accurate when predicting the y in the test set.

Another approach is using penalized logistic regression that imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less significant variables toward zero. This is also known as regularization. We carried out Lasso, ridge and elasticnet regularization procedures.

As a result we achieved an optimal lambda parameter quite low (almost zero), giving almost no penalty to the model. The prediction accuracy did not significantly improve. So far we used log transformed data, categorical values are treated with as dummy variables, with na values as a separate category, and several possible logistic regression models with omitting variables and penalising through regularization. Still, the prediction accuracy did not improve and surpass 82.3 percent. At this point, we decided to quit the logistic

regression framework and see whether there are other possible approaches that succeed better. Tree based method are often very accurate in classification setting with plenty of dependent variables, as a next step we are focusing on tree based method models to predict the target variable.

3.2 Tree based models - Random Forest (Bagging uncorrelated trees)

One of the most successful form of tree based model is Random Forest algorithm. Random forest uses the idea of trees bagging, averaging many noisy but approximately unbiased decision trees (the trees must have sufficient depth for being unbiased), therefore reducing the variance in the model. Since each tree generated in bagging is identically distributed (i.d. but not i.i.d.), the bias (expectation) of the average of the trees is the same as the individual trees, but improvement in accuracy can be achieved by the variance reduction.

If decision trees were uncorrelated the average variance would be decreased by a factor of 1/(number of trees). Nevertheless, this is not the case (by construction of bagging process), and therefore its variance decreased by the number of trees, and is increased by the inter-correlation. Random forest builds a large number of uncorrelated trees solving this issue.

Data Preprocessing

Since random forest isolate observations in small regions of the feature space we do not need to use the log transformed or normalized data, but our initially slightly modified train set.

Model training and parameters tuning

We use the Caret package function for random forest modeling and apply a 5 K-folds cross validation, repeated 3 times, for improving the stability of the cross validated accuracy. We found that repeated cross validation is the most precise and stable process for finding the optimal parameters and prediction accuracy.

In random forest, the principal hyperparameter is **mtry**, the number of features considered by each tree when splitting a node.The numbers of trees can be tuned but it is not a hyperparameter (an increase of its value generally improve the accuracy). Nevertheless, for computational issues, and seeing that the gain in accuracy do not compensate after a certain number of trees, we considered using 1000 trees.

Decision tree algorithm, when splitting it automatically puts aside the non-important variables as shown by Hastie et al. (2009), for that reason we include all the variables in the classification, ignoring for this model the variable selection.

First of all we will find the most optimal hyperparameter mtry using the caret package and functions expand.grid and train.

As we can observe in the above plot, the number of mtry that optimize the accuracy is peaking around 9, then stays generally on the same height. To see precisely the optimum result we use the print function and the exact value. The results of the cross validated random forest show the following:

Accuracy was used to select the optimal model using the largest value. The final value used for the model was mtry = 12.,

It results in a cross-validated prediction accuracy of 85,41%. Out of bag samples estimate of the error rate is 14,19%. The miss classification error is approximately 2 percentage point higher for the outcomes 0 and 1's.

Another interesting feature of the random forest is the variable importance. At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

We then choose for our final random forest with mtry=12 which gave the following final results for the prediction accuracies by category:

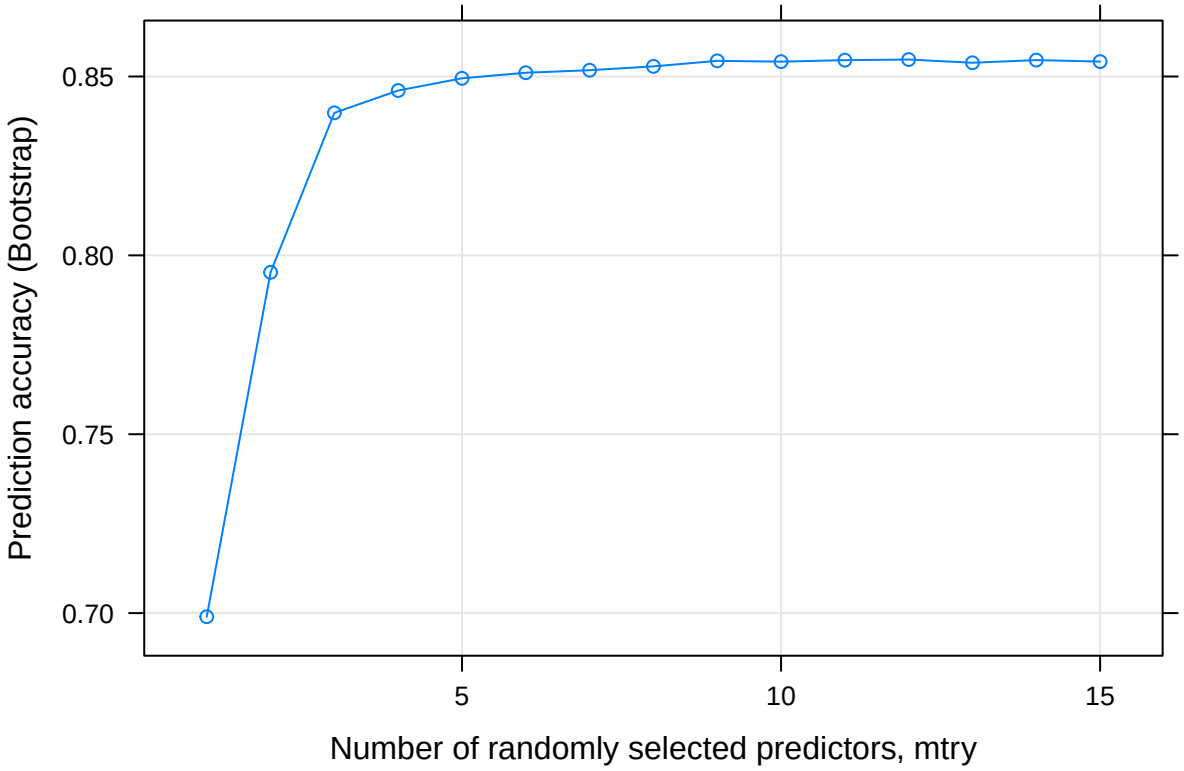


Figure 2: Optimizing the hyperparameter mtry for the Random Forest model

- Cross validated train set: 0.854734
- Kaggle public leaderboard (30 %): 0.85926
- Kaggle private leaderboard (70 %): 0.85790

It is a result that we found rather stable and a result that is to be kept for later. However, we would like to see other approaches that give a chance for improvement.

3.3 Improvement of the tree-based models

As a next step we are interested to further improve the prediction received from the so far optimal random forest model. There are several possible algorithms that we could choose from. It is worth observing in what way we would like to improve. The prediction error consists of bias, variance and irreducible error. Irreducible error is self-explanatory and impossible to avoid, but the other two forms of error are something we could focus on separately.

Priorly we used bagging that uses many unbiased models with high variance and reduces the variance by averaging. Bagging involves fitting many large trees that is then used to form one final prediction. It could be generally a good approach when one the aim is to reduce the variance of the models. We have examined our situation and came to the following conclusion: after repeated procedures of the so far best tree-based algorithm we have found that the variance of our prediction accuracy is low. Hence, as a next step we focus on finding a procedure that rather reduces the bias of the prediction. One possible approach in this setting is to use many weak, biased models with low variance and combine them to reduce the bias, a procedure called boosting. In boosting the trees are grown sequentially, using the information of the previous tree to construct the consequent tree. Therefore, the trees are slowly learning through the use of a shrinkage parameter lambda. **We have therefore decided to use the boosting approach for minimizing the bias of our prediction.**

In a general tree based boosting model the hyperparameters are:

- lambda, which is the shrinking parameter that controls the learning rate of the boosting
- d, the number of splits that control the complexity of the boosted model
- the number of trees, boosting iterations

One needs to also set the number of trees in the algorithm. Boosting is an algorithm that could easily overfit, when the number of trees are large. However, one additional advantage of the boosting process is that due to the sequential growing of trees the overfitting occurs step by step, therefore, it is easier to trace and avoid compared to other alternatives.

3.4 Boosting algorithm possibilities - C5.0

After narrowing down the number of possibilities to a tree based classification problem with boosting approach, we still have plenty of options to choose from to pick the right models. The Caret library package gives several possibilities. One model choice is C5.0 tree based boosting model. After reading researches in the field we have found a paper that came to the conclusion that boosted C5.0 classifiers perform well when stacked up against other classifiers. According to one of the article's findings "among the boosting (BST) ensembles, C5.0 t is the best, followed by MultiBoostAB LibSVM, adaboost R and other MultiBoostAB ensembles" (Fernandez-Delgado et al 2014).

the hyperparameters of the C5.0 algorithm in the caret package are:

- Model type: rule based / tree based
- Winnowing option: no winnowing/winnowing
- Number of boosting iterations

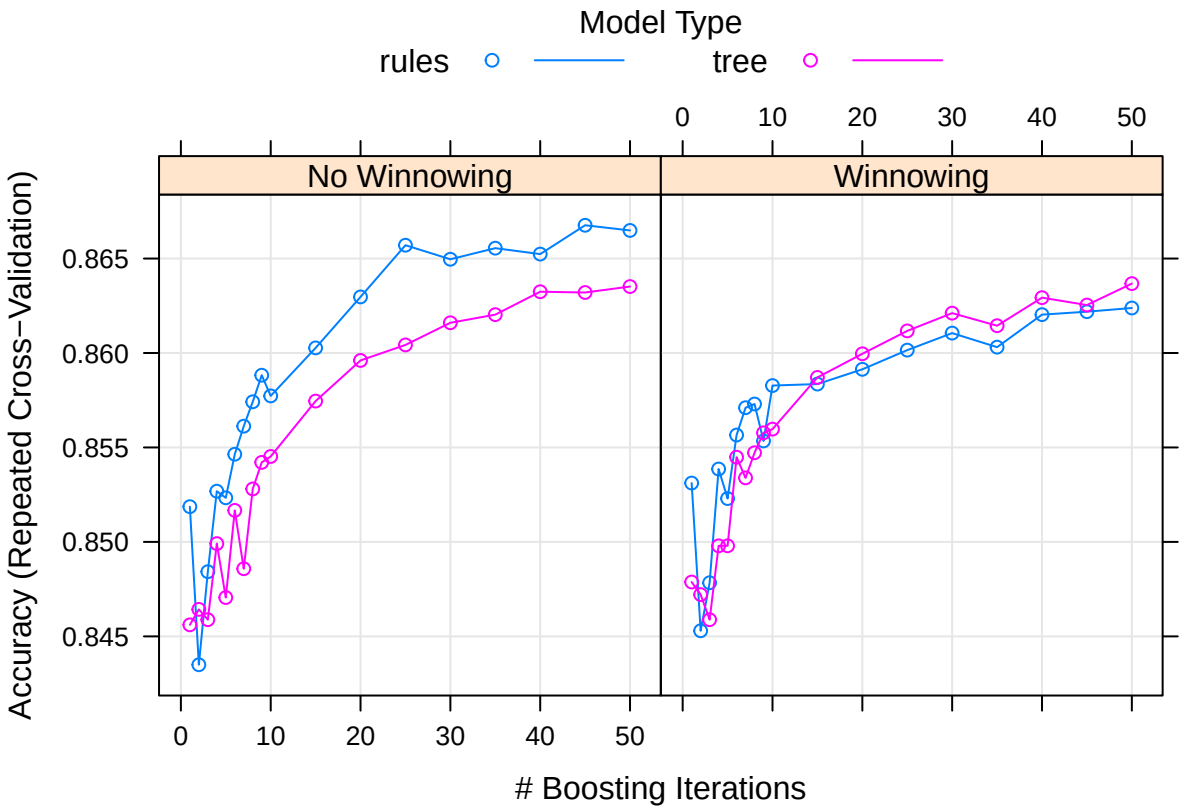
The C5.0 model can take the form of a full decision tree or a collection of rules, with the extension of boosting. An important feature of C5.0 is its ability to generate classifiers called rule sets that consist of unordered collections of if-then rules. Conditions are described for the attributes and the rules use the if-else statements to decide the prediction outcome. Rules are created in a similar fashion compared that of a decision tree, for example when creating a split. When using this model there is a possibility to use either rule based or conventional tree based model type.

The decision trees and rule sets constructed by C5.0 do not generally use all of the attributes.One of its tuning parameters is winnowing. Winnowing by the Oxford Dictionary, is the process of separating the useful part of wheat from useless part, the chaff (a term that comes from agriculture). In the C5.0 setting winnowing is pre-selecting certain variables and continue the modelling using only those that are identified as helpful. For example, in case there is an abundance and not all of the predictors are useful. Therefore winnowing means feature selection, no-winnowing means keeping all the attributes in our model.

Another powerful feature incorporated in C5.0 is adaptive boosting, based on the work of Rob Schapire and Yoav Freund (Freund et al,1999). The idea is to generate several classifiers (either decision trees or rule sets) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class. Setting the number of boosting iterations is another hyperparameter in the model.

It is not possible to choose the depth of the tree for the C.50 algorithm, as it is an algorithm that uses both rule sets and tree type. Additionally, the C5.0 algorithm's default parameter for early stopping is true. It means that the algorithm does not increase the boosting iterations if the marginal improvement in prediction is very small. This feature helps overfitting and stops if the additional tree gives poor performance.The tuneLength parameter in the caret package uses number of different values for the hyperparameters the algorithm to try different default values for the main parameter.

Using the 3 repeated cross validation procedure with 5 folds we obtained the results that help choose the optimal C5.0 hyperparameters and the corresponding prediction result.



Using the outcomes, we constructed a graph that shows for the number of boosting iteration the cross-validation prediction accuracy, for both rule based and tree based model. We have included on the same graph the results of the winnowing and non-winnowing options. From this compact graph one could see that the best choice is keeping all the predictors (No Winnowing) and using the rules-based type of the C5.0. The first local optimum of prediction accuracy is at boosting operation 25, then one could observe a slight decrease, followed by an increase of prediction performance afterwards. Although the earlystopping parameter is set to true, the algorithm has not stopped and it indicates that the performance is the best at the largest number of boosting iterations.

To avoid overfitting we overruled this result and decided to use the boosting iteration at a local optimum, when it is equal to 25, instead of using boosting iteration 45. The prediction accuracy between the two is less than 0.001 and the risk of overfitting is larger than the potential reward of using larger iteration number.

optimal hyperparameters for our model of choice for the C5.0 are:

- Rule model type
- No Winnowing
- 25 boosting iterations

The prediction performance of the optimised C5.0 could be summarized as:

cross validated accuracy: 0.8661 Kaggle private leaderboard (70 % of the data): 0.8616 Kaggle public leaderboard (30 % of the data): 0.8694

3.5 Boosting algorithm possibilities - Extreme gradient boosting for tree based models

To further improve the aim of reducing the prediction bias, we choose a separate approach within the family of tree based models with boosting extension. For the C5.0 the number of tuning parameters are fewer. In the next step we aim for a more complex, computationally expensive model with richer possibilities of fine tuning. Among the tree-based models, extreme gradient boosting is an algorithm of high performance and is designed to push the computation resources to the limits while using a more regularized model formalization to control for over-fitting according to author of the algorithm, Tianqi Chen, in the package description (Chen, 2020). http://cran.fhcrc.org/web/packages/xgboost/vignettes/xgboost.pdf

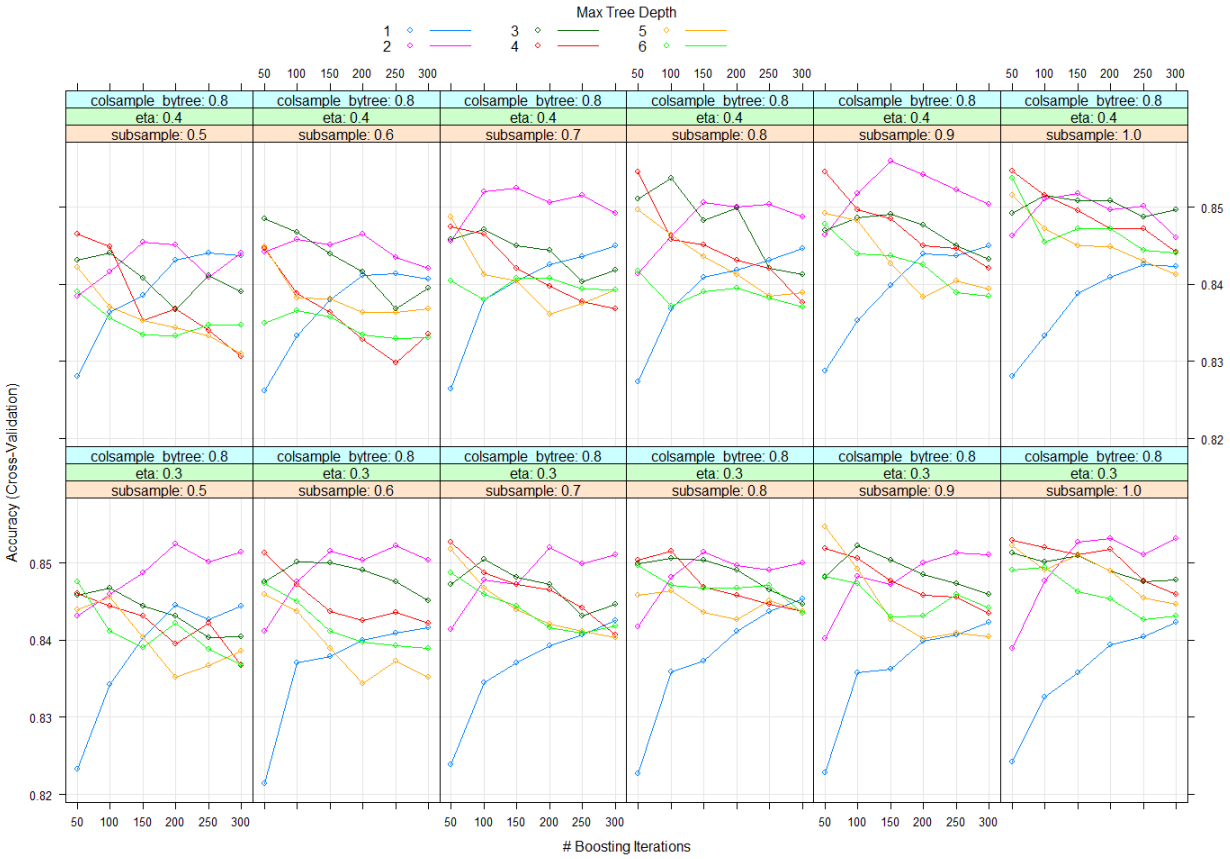
Xgboost library name stands for Extreme Gradient Boosting. It is a specific implementation of the Gradient Boosting method which uses more accurate approximations to find the best tree model (uses the second derivative). Gradient boosting methods works similarly to the general boosting algorithms but introduces a loss function and uses the gradient to minimize a loss function. In each round of training, the weak learner is built and the error rate is calculated. The gradient represents the steepness of our error function. It is used then to find the direction for improvement.Contrary to normal boosting, gradient boosting do not increment the weights of the miss classified sub samples of each iteration but tries to optimize the loss function of the previous learner by adding a new adaptive model that adds the weak learners in order to reduce the loss function. The idea is to overcome the errors in the previous learner prediction.

The function we use for the prediction is from xgboost library that is found as well in the Caret package, the name of the corresponding method is xgbTree. The hyperparameters:

- nrounds, maximum number of boosting iterations
- max_depth ,the maximum tree depth allowed for one case (number of nodes)
- eta, the shrinkage can also be interpreted as the learning rate, typical values go from 0.01 to 0.2
- gamma, the minimum amount of loss reduction in the iterations
- colsample_bytree, the fraction of columns to be randomly sampled for each tree, typical values range from 0.5 to 1.
- min_child_weight, the minimum sum of weights of all observations required in a child, where child corresponds to minimum number of instances needed to be in each node.
- sub sample, denotes the fraction of observations to be randomly sampled when creating each tree, lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting. Typical value is between 0.5 and 1 (Jain, 2016).

Similarly to other boosting approaches, the increase in nrounds and the decrease in the shrinkage parameter could likely lead to overfitting in the prediction results. Which means that the algorithm works fine for predicting its own outcomes, however when it is applied to the new data test set, the evaluation results will be less accurate. The increase of shrinkage parameter and the importance of the loss function could intervene and help avoid over-fitting. Generally the larger the maximum tree depth the larger the computational time until the convergence to a high prediction accuracy.

In order to find the optimal hyperparameters, the same cross-validation process is carried out as in the previous steps. Using 5-fold, 3 times repeated cross validation the results of this algorithm could be observed on the result graph, in the following figure.



The sensitivity analysis was run with 15 tune length to find the optimal parameters with many possibilities. The final chosen optimal hyperparameters for this model are:

- nrounds=50
- max_depth=6
- eta=0.3
- gamma=0
- colsample_bytree=0.6
- min_child_weight=1
- sub sample=0.94

Eta is 0.3 wich is larger than the typical value and this is a way of dealing with overfitting. The optimal nrounds is 50 and we can generally observe in the plots that when this value increases overffiting appears and accuracy drops. The ideal number of nodes is 6, models with less nodes perform worse while larger values of this will overfit. Along these parameters the optimal prediction performance is obtained with a larger subsample ratio, 0.94.

4. The final prediction results

Instead of submitting the evaluation results of one algorithm such as extreme gradient boosting method separately, we came up with the idea of creating an ensemble of the models. The result is not an ensemble model but an ensemble of the prediction outcomes.

The final prediction result uses the information obtained by the three most optimal models:

- Random forest
- C5.0
- Extreme gradient boosting model

Each of these models are tree based models that have been optimized by cross validation and intuitive decisions to receive the hyperparameters associated with the best possible prediction performance.

Random forest model aggregates the results of hundreds of uncorrelated trees. Our aim was to (1) respond to the variance error with this model. Through bagging this could be achieved with our random forest model of choice. (2) Then, to respond to reducing the prediction bias with a boosting algorithm. For the boosting algorithm we choose two different algorithms.

After running the optimized models, the prediction outcomes are stored separately.The aggregation of the prediction result is carried out to obtain the final vector of ones and zeros for the prediction of the target variable. The question arose, what is the best way to aggregate the prediction results? If for one observation the prediction outcomes are different for one model compared to that of the other model, how to decide whether the final answer of y should be zero or one?

There are different ways of aggregating the outcomes. We have chosen to use the aggregate majority vote option. Each model in the ensemble votes for an outcome zero or one. For instance, if two models predict 1 and one model predict 0 as an outcome, 1 is chosen.

4.1 Prediction result ensemble of Random Forest and C5.0 algorithm

Primarily we used this strategy for two models, the C5.0 and Random Forest algorithms. However, the issue arises when the number of models are even, because there could be even number of votes for the outcomes. To make a decision in such cases, we used the by-class cross-validated prediction accuracy. If one model had a larger precision of predicting one category compared to that of the other model then the decision was made accordingly and the final prediction outcome was the one of the more accurate model. Using the C5.0 model and the Random Forest model, this approach gave us the following results:

Random forest and C5.0 rule type, no-winnowing boosted model with an aggregated prediction approach, using majority vote option with decision criterion of by-category prediction accuracy:

- Kaggle private score (70% of the data): 0.86159
- Kaggle public score (30% of the data): 0.87099

4.2 Prediction result ensemble of Random Forest model, C5.0 and Extreme gradient boosting algorithm

For the final submission we modified our strategy including another boosting method, the extreme gradient boosted model. If the number of models are odd, the majority vote strategy works without any additional decision criterion on the by-category prediction classes and the final prediction result is always chosen according to the category with higher number of votes.

Random forest model, Extreme gradient boosted model and C5.0 rule type, no-winnowing boosted model with an aggregated prediction approach, using majority vote option as decision criterion:

- Kaggle private leaderboard (70% of the data): 0.85991
- Kaggle public leaderboard (30% of the data): 0.87177

The final prediction performances achieved by our most accurate 4 Kaggle submission are illustrated in the table below:

Model	Private Leaderboard prediction accuracy	Public Leaderboard prediction accuracy
Random Forest	0.8579	0.85926
C5.0	0.86159	0.86942
Ensemble 1	0.86159	0.87099
Ensemble 2	0.85991	0.87177

5. Conclusions and summary

In this report we had the opportunity of predicting the purchasing behaviour of consumers. We were given a dataset with known outcome from a previous advertisement campaign. It contained information on the consumers that helped us train machine learning algorithms for estimating the target variable y, the conversion. We first explored the dataset, identified categorical and numerical variables, observed the number of missing information per variable and draw special attention to these columns. The highest ratio of missing information concerns the variable outcome_old. A random forest imputation technique was carried out to fill in the missing information in this column, however, this did not lead to better prediction performance as the same approach could not be applied in the test set that is to be evaluated. The na values were then treated as a separate category. The numerical variables were log transformed to reduce the large variance and treat the outliers. Due to the binary nature of the target variable y, we started with an introductory logistic regression that included the all the variables for prediction. The obtained prediction accuracy was about 82 percent. Then, we followed by using ridge and lasso regularization methods to penalize for the less significant predictors, we did not achieve significant improvement. We concluded that it is beneficial to keep each attributes and use another approach, therefore explored the possibilities of using tree-based models. We then identified the 3 best possible tree based models.

As the prediction error generally comes from the variance error and the prediction bias (apart from the irreducible error), we chose separate models that excel in responding to these challenges. We used bagging and boosting algorithms. We first used the knowledge of the course then and carried out research, reading articles to find additional algorithms that could be useful in such cases. Upon finding the right algorithm, we optimised the hyperparameters by 3 times repeated cross validation of k-folds=5.

Our first model of choice was a random forest model with 12 number of randomly selected predictors. It achieved a prediction accuracy over 85.5 percent. Then, we proceeded to improve performance with two boosted tree based algorithms, the C5.0 rule type no-winnowing model with 25 boosted iterations and an extreme gradient boosted algorithm.

Instead of evaluating these models separately, we created an ensemble of the three final models. The ensemble was comparing the outcome of the models for each observation one by one. We submitted two versions to Kaggle, in version one we used the information from the random forest model and the C5.0. When the models voted for different outcome, the by-category prediction was compared for the two models and the decision was made according to the more reliable vote of the two models.

Our final submission was an ensemble of the three models, where the predicted outcomes of each model were compared for each observation. The majority vote strategy was applied to decide the final prediction estimate. Arriving to an over 86 percent prediction accuracy. This solution includes the strength of an optimized bagging algorithm, a less complex fine tuned boosted algorithm and a high performance more complex boosting algorithm for reducing prediction bias. In the future the final result could be subject to further improvements with ideas such as fine tuning the weight system for the decision making at the aggregation of the three models with neural networks.

References

Kolmogorov-Smirnov test:

- A Dictionary of Statistics, Graham Upton and Ian Cook, Oxford University Press, 2014

C.5 model selection and hyperparameters:

- Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?, Journal of Machine Learning Research 15 (2014) 3133-3181, Manuel Fernandez-Delgado, Eva Cernadas, Senen Barro, Dinani Amorim http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf

shorturl.at/htCFM

- https://cran.r-project.org/web/packages/C50/C50.pdf Max Kuhn [aut, cre], Steve Weston [ctb], Mark Culp [ctb], Nathan Coulter [ctb], Ross Quinlan [aut] (Author of imported C code), RuleQuest Research [cph] (Copyright holder of imported C code), Rulequest Research Pty Ltd. [cph] (Copyright holder of imported C code)

Extreme gradient boosting explanations and codes:

- model parameter explanations

Complete Guide to Parameter Tuning in XGBoost with codes in Python, 2016, AARSHAY JAIN

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

- https://www.shirin-glander.de/2018/11/ml_basics_gbm/ author: Tianqi Chen

*The Elements of Statistical Learning, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie (2001) Springer