CIND119 – Bank Marketing Project

20/04/2021

Contents

[Memebers 1](#_Toc69652268)

[Summary 2](#_Toc69652269)

[Tools 2](#_Toc69652270)

[Workload Distribution 2](#_Toc69652271)

[Exploratory Data Analysis 2](#_Toc69652272)

[Predictive Modeling / Classification 9](#_Toc69652273)

[Decision Tree 9](#_Toc69652274)

[Naïve Bayes 12](#_Toc69652275)

[Gradient Boosting 13](#_Toc69652276)

[Initial Results 14](#_Toc69652277)

[Feature Selection 14](#_Toc69652278)

[Re-training on selected features 15](#_Toc69652279)

[Final Results 16](#_Toc69652280)

[Conclusions and Recommendations 17](#_Toc69652281)

[Recomendations 17](#_Toc69652282)

[Appendix 18](#_Toc69652283)

[Code 18](#_Toc69652284)

## Memebers

Patrick Litte, [patrick.little@ryerson.ca](mailto:patrick.little@ryerson.ca)

Manjola Chiappetta, [m1chiappetta@ryerson.ca](mailto:m1chiappetta@ryerson.ca)

## Summary

Our client for this project is a Portuguese bank that is looking for improvements to their telemarketing strategy. This bank is marketing long-term deposit accounts such as bonds and savings accounts to a large variety of existing clients with a wide range of attributes. Our role in the project was to create a variety of predictive models which could be used to predict whether a client will subscribe to a term deposit or not, based on the bank’s past experience in marketing these products. This predictive model would enable the client to target their marketing activities more efficiently to clients with a greater likelihood of subscribing to a term deposit.

In this project we compared two different predictive models, decision tree and Naïve Bayes and compared their accuracy as well as their Area under the Receiver Operator Curve. For our extra credit, we compared these two more simple models against a more advanced Gradient Boosting model (XGBoost) to determine if a more advanced model could out preform the simpler models.

In the best versions of our models we were able to achieve a model accuracy of 89.20% with a decision tree, 88.41% with Naive Bayes, and 91.33% with XGBoost.

### Tools

This project will leverage R for the exploratory data analysis, data preprocessing, predictive modeling, and model evaluation. We will use the tidymodels framework to implement our predictive modelling, and model evaluation, and we will use the DataExplorer package to aid in exploratory data analysis.

## Workload Distribution

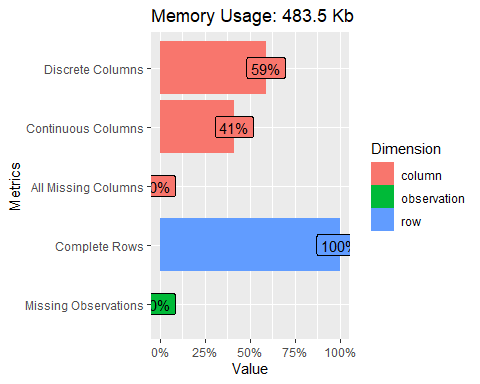
|  |  |
| --- | --- |
| Member Name | List of Tasks Preformed |
| Patrick Little | EDA , Decision Trees, XGBoost,  Feature Selection, Project Report |
| Manjola Chiappetta | EDA, Naïve Bayes, Project Report |

## Exploratory Data Analysis

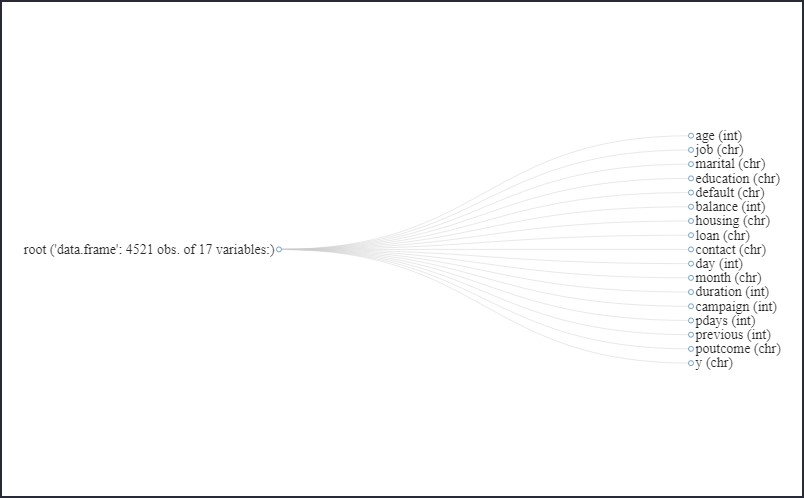
In this section we will:

* Look at the attribute types in the dataset;
* Find and handle missing values;
* Examine the pairwise association between our variables;
* Find max, min, mean and median of the attributes; and
* Determine any outlier values for the attributes under consideration.

To begin our exploratory data analysis, we can use the data explorer R package to generate an easy to consume overview of our dataset. In this plot one can observe that there are a mix of discrete and continuous variables in our dataset. Additionally, we can observe that there is no missing data observations contained within our dataset.



Looking at the individual columns of our dataset, one can observe there are 17 variables in our dataset. Age, balance, day, duration, campaign, duration, pdays, and previous are of type integer. Job, marital, education, default, housing, loan, contact, month, poutcome, and y are of type character. This can be observed in the network structure diagram below.



dataset structure network

We can examine the values of our character variables to determine if character is the appropriate data type for each of the variables. If a variable of type character contains a limited number of possible values, then converting that variable to be of type factor will allow our predictive models to interpret that data without significant additional prepossessing. If those variables contain free form text without a controlled vocabulary for that variable, then likely additional natural language processing techniques such as stemming and lemmatizaion, along with other preprocessing or feature engineering may be required before those variables would offer significant predictive value to our models.

Character Variables with all Existing Values

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Job | Marital | Education | Default | Loan | Contact | Month | pOutcome | y |
| admin. | divorced | primary | no | no | cellular | apr | failure | no |
| blue-collar | married | secondary | yes | yes | telephone | aug | other | yes |
| entrepreneur | single | tertiary |  |  | unknown | dec | success |  |
| housemaid |  | unknown |  |  |  | feb | unknown |  |
| management |  |  |  |  |  | jan |  |  |
| retired |  |  |  |  |  | jul |  |  |
| self-employed |  |  |  |  |  | jun |  |  |
| services |  |  |  |  |  | mar |  |  |
| student |  |  |  |  |  | may |  |  |
| technician |  |  |  |  |  | nov |  |  |
| unemployed |  |  |  |  |  | oct |  |  |
| unknown |  |  |  |  |  | sep |  |  |

Examining our table above of each of the variables of type character with the universe of existing values for each variables, serving as a simple data dictionary, we can determine that this is a controlled vocabulary and there is not unstructured text in these fields. Since there is a controlled vocabulary being used with these variables, we will convert these variables to be of type factor.

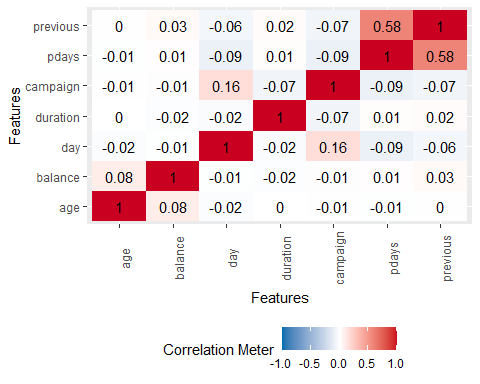
We can then examine the pairwise association between all of the variables in the dataframe. In this analysis we will use Spearman Correlation to measure the association between our numeric variables, Cramer’s V for our nominal factor data, and ANOVA to

Mixed Data Types Correlation/Association

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | assoc | type |
| previous | pdays | **0.9862904** | correlation |
| pdays | previous | **0.9862904** | correlation |
| poutcome | pdays | 0.8759717 | anova |
| pdays | poutcome | 0.8759717 | anova |
| poutcome | previous | 0.6850125 | anova |
| previous | poutcome | 0.6850125 | anova |
| month | contact | 0.5131000 | cramersV |
| contact | month | 0.5131000 | cramersV |
| job | age | 0.5061504 | anova |
| age | job | 0.5061504 | anova |
| month | housing | 0.4878000 | cramersV |
| housing | month | 0.4878000 | cramersV |
| education | job | 0.4551000 | cramersV |
| job | education | 0.4551000 | cramersV |
| month | day | 0.4490406 | anova |
| day | month | 0.4490406 | anova |

Looking at the top 8 pairs of variables in our pairwise comparison sorted by the absolute value of the association measure, we can see that the variables Previous and pDays seem to have a high measure of correlation. With a Spearman value of 0.986, we can state there is a highly monotonic relation between the variables. In this metric we are assessing if the variables move in the same direction, however this relation could have one of a number of non-linear relation types such as exponential, logistic, etc.

Observing the relations between our numeric variables with a different metric, one can use the Pearson correlation to assess the linearity of the relationships. With an R value of 0.58, the relation between the pDays and Previous variables is a moderate positive correlation. Since the R value is in a moderate range, there is still likely predictive value in including both variables in our models.

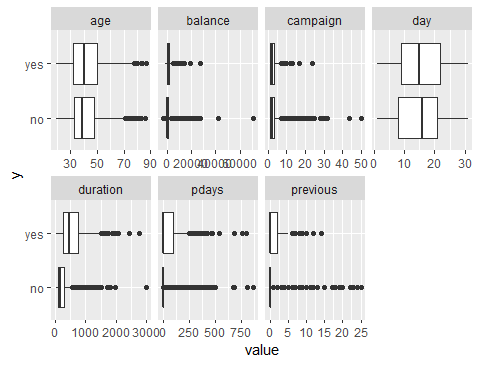


We can then look for outliers in our numeric variables. Comparing the median, mean, min, and max values for our numeric variables, we can observe that many of our numeric variables each have max or min values that vary by an order of magnitude between the IQR and the min/max value. This warrants furthur exploration into the outliers within the dataset.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| age | job | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign | pdays | previous | poutcome | y |
| Min. :19.00 | management :969 | divorced: 528 | primary : 678 | no :4445 | Min. :-3313 | no :1962 | no :3830 | cellular :2896 | Min. : 1.00 | may :1398 | Min. : 4 | Min. : 1.000 | Min. : -1.00 | Min. : 0.0000 | failure: 490 | no :4000 |
| 1st Qu.:33.00 | blue-collar:946 | married :2797 | secondary:2306 | yes: 76 | 1st Qu.: 69 | yes:2559 | yes: 691 | telephone: 301 | 1st Qu.: 9.00 | jul : 706 | 1st Qu.: 104 | 1st Qu.: 1.000 | 1st Qu.: -1.00 | 1st Qu.: 0.0000 | other : 197 | yes: 521 |
| Median :39.00 | technician :768 | single :1196 | tertiary :1350 |  | Median : 444 |  |  | unknown :1324 | Median :16.00 | aug : 633 | Median : 185 | Median : 2.000 | Median : -1.00 | Median : 0.0000 | success: 129 |  |
| Mean :41.17 | admin. :478 |  | unknown : 187 |  | Mean : 1423 |  |  |  | Mean :15.92 | jun : 531 | Mean : 264 | Mean : 2.794 | Mean : 39.77 | Mean : 0.5426 | unknown:3705 |  |
| 3rd Qu.:49.00 | services :417 |  |  |  | 3rd Qu.: 1480 |  |  |  | 3rd Qu.:21.00 | nov : 389 | 3rd Qu.: 329 | 3rd Qu.: 3.000 | 3rd Qu.: -1.00 | 3rd Qu.: 0.0000 |  |  |
| Max. :87.00 | retired :230 |  |  |  | Max. :71188 |  |  |  | Max. :31.00 | apr : 293 | Max. :3025 | Max. :50.000 | Max. :871.00 | Max. :25.0000 |  |  |
|  | (Other) :713 |  |  |  |  |  |  |  |  | (Other): 571 |  |  |  |  |  |  |

We can also use this table to provide a baseline success number for our client’s existing marketing effort. Based on the vaules of ‘y’ above, we can see that in 521 cases, customers subscribed to a term deposit product from the bank, and in 4000 cases they did not. This represents a conversion rate of 0.13025. One might deduce that a over 13% conversion rate on a marketing campaign is quite high, however using the analysis developed as part of this report, we feel we can increase our client’s conversion rate.

Looking at a box plot of our numeric variables, we can get a sense for the extent of the distribution of outliers in each variable.



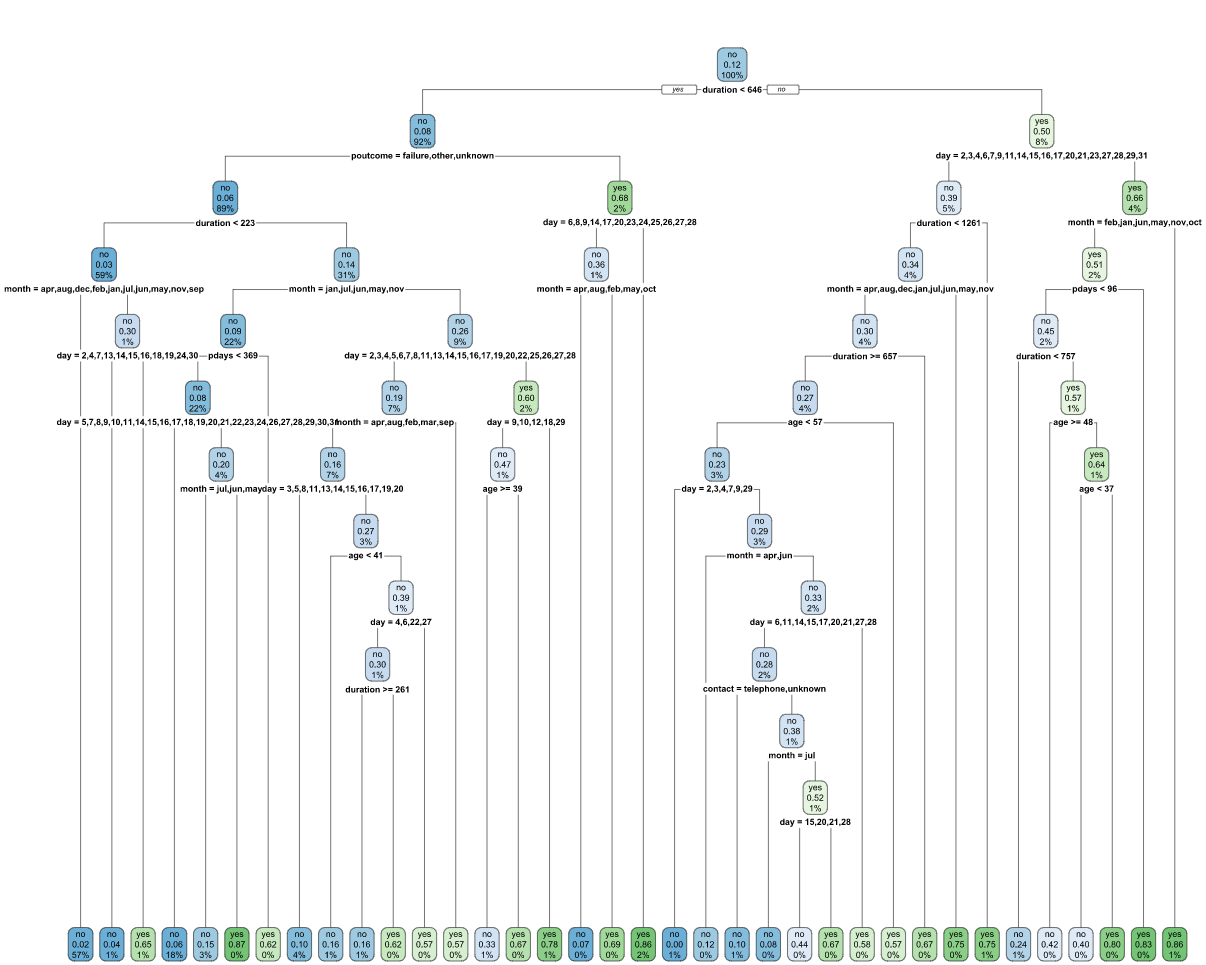
Typically, we would remove outliers from the dataset, using the standard definition of 1.5\*IQR. In the case of our dataset, using a standard outlier removal approach on all of our predictor variables results in zero rows left in our dataset for analysis. Therefore, for our modeling instead of manually removing outliers from the data, we will apply a YeoJohnson transform on our numeric variables. This should serve as an adequate method to both mitigate the effect of outliers, as well as contribute to normalizing the dataset. For our variable Day, although the data is encoded as a numeric variable, this represents more of a factor variable, therefore we will convert this variable to type factor such that it does not receive a YeoJohnson transform during our model preprocessing.

## Predictive Modeling / Classification

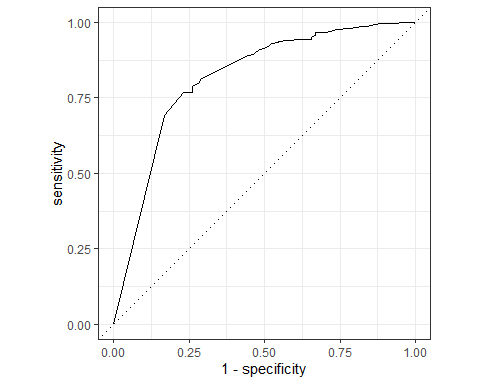
### Decision Tree

For our decision tree model, we used the rpart algorithm in the classification mode as this project is a binary classification problem. We used 10 fold cross validation to train the model. We also performed a YeoJohnson transform on our numeric variables to help reduce the effect of outliers of the dataset and contribute to normalizing our data. This model requires we set three hyperparameters: cost complexity, tree depth, and the minimum number of data points within a node for it to be split further. Instead of manually setting these hyperparameters, we tuned these hyperparameters using a latin hypercube of size 60.

We can observe the structure of our decision tree below.



We can observe the area under the receiver operator curve for this initial model below:



This model produced a confusion matrix as follows:

|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 945 | 85 |
| Yes | 55 | 45 |

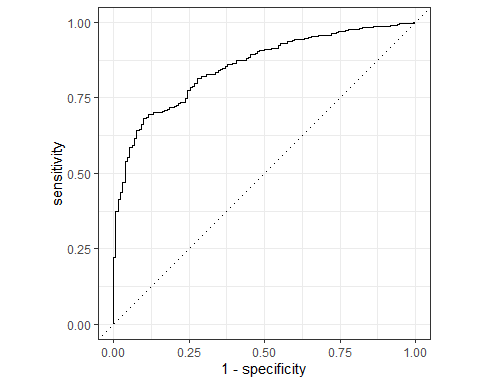
This gave us an initial model accuracy of 87.61% and AUC of 81.61%.

### Naïve Bayes

For our Naïve Bayes model, we used the Naïve Bayes algorithm implemented by the discrim package in R. There are several other Naïve Bayes implementations available within R such as KLaR, e1071, etc., however the discrim implementation of the model seems to integrate well within the tidymodels framework.

For this model there are no hyperparameters that we need to explicitly set, therefore hyperparameter tuning was not used with this model. As with our decision tree model, we used 10 fold cross validation over a simple test-train split.

Below we can observe the area under the receiver operator curve for our results with this model.



This model produced a confusion matrix as follows:

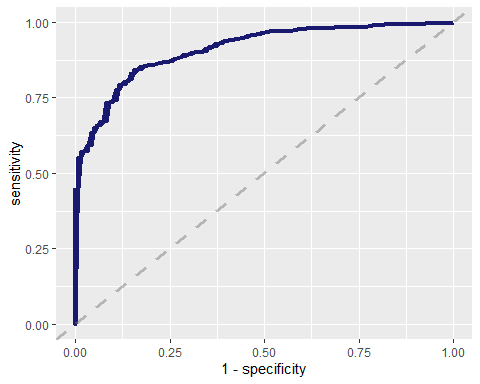
|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 957 | 92 |
| Yes | 43 | 38 |

This gave us an initial model accuracy of 88.05% and AUC of 85.33%.

### Gradient Boosting

For our Gradient Boosting model, we used the Extreme Gradient Boosting (XGBoost) algorithm implemented by the xgboost package in R. In this model there are seven hyperparameters we need to set: mtry, trees, min\_n, tree depth, learn rate, loss reduction, and sample size. For the number of trees, this is a computationally expensive hyperparameter to tune, therefore we will set this parameter manually at 2000 trees, which should be a satisfactorily large number to satisfy the universe of potential outcomes based on the size of our dataset and number of predictors. As in our decision tree model, we tuned our remaining hyperparameters with a latin hypercube of size 60. For the XGBoost model, we also dummy encoded our data to attempt achieve an optimal result.

Below we can observe the area under the receiver operator curve for our results with this model.



This model produced a confusion matrix as follows:

|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 991 | 82 |
| Yes | 18 | 39 |

This gave us an initial model accuracy of 91.33% and AUC of 91.22%.

### Initial Results

Examining the initial results of the models, we can see that we were able to achieve good results with all predictors being included in the model. With our tuned decision tree we were able to achieve a model accuracy of 87.61% and AUC of 81.61%. With our Naive Bayes model were were able to achieve an accuracy of 88.05% and AUC of 85.33%. We were able to out preform our first two models with our XGBoost model. This model achieved an accuracy of 91.33% and AUC of 91.22%.

Since the source dataset for this project is relatively small, ~4500 observations of 17 variables, we are achieving performant model training times even with 10 fold cross validation and hyperparameter tuning. The end to end model training and evaluation for our decision tree is able to complete in about two minutes running as a single threaded workflow.

Our Naive Bayes model is even more performant completing the end to end model training and evaluation workflow in about 4 seconds. Since we are not tuning hyperparameters in this model, having a fast training time is expected.

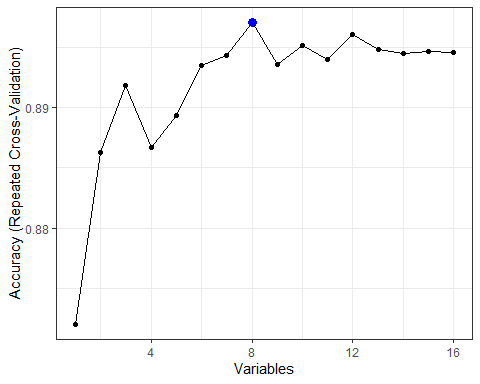
The XGBoost model is a significantly more complicated model, and therefore takes considerable more training time. In a multithreaded workflow with 4 dedicated CPU cores, the model takes about 25 minutes to train when using 2000 trees, and tuning our other hyperparameters with a latin hyper cube of size 60.

### Feature Selection

Using recursive feature selection can identify predictors that could be removed from the model, both to improve model training times, but also to potentially to reduce model overfitting. Based on a random forest algorithm, our recursive feature selection indicates that the best features to include in our model will be: Duration, Month, Day, pOutcome, pDay, age, Contact, and Previous. Additionally we will ensure we preserve our target variable ‘y’ in the dataset.

Using those features, our recursive feature selectiion algorithm is indicating the model achieved an accuracy of 89.69%, training a random forest algorithm on the training data and evaluating the results on test data, with 10 fold cross validation.

Below we can observe a graph of our model accuracy, by number of predictor variables included in the model.



Using these selected predictor variables, we will repeat the same model training and evaluation steps using only the eight predictors indicated.

### Re-training on selected features

#### Decision Tree

Using the same model training approach described in the previous section we retrained and evaluated the model using only the predictors indicated by our recursive feature selection to achieve the best result. Using only Duration, Month, Day, pOutcome, pDay, age, Contact, and Previous as predcitors in the model, we were able to achieve a confusion matrix as follows:

|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 944 | 66 |
| Yes | 56 | 64 |

This gave us a final model accuracy of 89.2% and AUC of 84.0%.

#### Naive Bayes

Again, using the same model training approach described in the previous section we retrained and evaluated the model using only the predictors indicated by our recursive feature selection to achieve the best result. With our selected features, our final Naïve Bayes model was able to achieve a confusion matrix as follows:

|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 963 | 94 |
| Yes | 37 | 36 |

This gave us a final model accuracy of 88.4% and AUC of 86.2%.

#### XGBoost

Again, using the same model training approach described in the previous section we retrained and evaluated the model using only the predictors indicated by our recursive feature selection. With our selected features, our XGBoost model achieved a confusion matrix as follows:

|  |  |  |
| --- | --- | --- |
| Prediction / Truth | No | Yes |
| No | 994 | 89 |
| Yes | 15 | 32 |

This gave us a model accuracy of 90.8% and AUC of 90.5%. A reduced model accuracy and AUC compared with our initial results.

### Final Results

Our model performance improved for both the Decision Tree and the Naive Bayes models. For the XGBoost model, limiting the number of predictors inputted into the model reduced performance.

Our model accuracy for the decision tree model increased to 89.20% with the selected features, versus 87.61% for all features selected. For the Naive Bayes model, accuracy increased to 88.4% compared with our previous result of 88.05%. With XGBoost, model accuracy decreased to 90.8% versus 91.33% with all variables feeding the model.

Additionally, by limiting the number of predictor variables we were able to train the models faster. With our subset of predictors the Decision Tree model was able to complete the end to end training and evaulation workflow in about 75 seconds, verus about two minutes with the full dataset including all variables. The Naive Bayes model also experienced a proportionally similar performance gain, completing the subset workflow in 3 seconds versus about 4 seconds with the full dataset. Our XGBoost model also had improved training times, saving about 5 minutes of training time with the smaller dataset from the 25 minutes it took to train on the full dataset.

## Conclusions and Recommendations

Based on our results we can conclude that feature selection can be an important activity to improve model performance with machine learning models such as Naive Bayes and Decision Trees. We were able to use several techniques to develop predictive models to predict if a customer would subscribe to one of client’s term deposit products with an overall accuracy of over 90%.

### Recomendations

It is recommended that our client implement a gradient boosting model such as the model developed as part of this report to pre-screen caller lists. If our client were able to restrict their outgoing calls to clients that our model predicts would be likely to subscribe to a term deposit product, they would likely drastically improve their call conversion percentage, allowing them to get a much better ROI on their marketing spend. Additionally, reducing the volume of unwanted calls to clients not interested in term deposit products may improve their overall customer satisfaction.

## Appendix

This report was created as a R markdown document. In addition to the code behind the analysis in the section below, the R markdown document defining this report is available on GitHub at <https://github.com/PatLittle/CIND119-group-project/blob/main/project.Rmd>

### Code

knitr::opts\_chunk$set(echo = TRUE)  
library(knitr)  
library(tidyverse)  
library(tidymodels)  
library(vip)  
library(rpart.plot)  
library(DataExplorer)  
library(tictoc)  
library(data.table)  
library(gtools)

```{r}

bank<-read.csv("https://raw.githubusercontent.com/PatLittle/CIND119-group-project/main/bank\_marketing/bank.csv")  
plot\_intro(bank)  
  
plot\_str(bank, type="d", print\_network = T, fontSize=25)  
  
  
job<-levels(as.factor(bank$job))  
marital<-levels(as.factor(bank$marital))  
education<-levels(as.factor(bank$education))  
default<-levels(as.factor(bank$default))  
housing<-levels(as.factor(bank$housing))  
loan<-levels(as.factor(bank$loan))  
contact<-levels(as.factor(bank$contact))  
month<-levels(as.factor(bank$month))  
poutcome<-levels(as.factor(bank$poutcome))  
y<-levels(as.factor(bank$y))  
  
na.pad <- function(x,len){  
 x[1:len]  
}  
  
makePaddedDataFrame <- function(l,...){  
 maxlen <- max(sapply(l,length))  
 data.frame(lapply(l,na.pad,len=maxlen),...)  
}  
  
a = job  
b = marital  
c = education  
d = default  
e = loan  
f = contact  
g = month  
h = poutcome  
j = y  
  
data\_dict<-makePaddedDataFrame(list("Job"=a,"Marital"=b,"Education"=c,"Default"=d,"Loan"=e,"Contact"=f,"Month"=g,"pOutcome"=h,"y"=j))  
  
data\_dict %>% na.replace("")  
options(knitr.kable.NA = '')  
kable(data\_dict, caption="Character Variables with all Existing Values")  
bank\_clean<- bank %>% mutate\_if(is.character, factor)  
  
  
  
require(rcompanion)  
  
mixed\_assoc = function(df, cor\_method="spearman", adjust\_cramersv\_bias=TRUE){  
 df\_comb = expand.grid(names(df), names(df), stringsAsFactors = F) %>% set\_names("X1", "X2")  
  
 is\_nominal = function(x) class(x) %in% c("factor", "character")  
   
 is\_numeric <- function(x) { is.integer(x) || is\_double(x)}  
  
 f = function(xName,yName) {  
 x = pull(df, xName)  
 y = pull(df, yName)  
  
 result = if(is\_nominal(x) && is\_nominal(y)){  
 cv = cramerV(as.character(x), as.character(y), bias.correct = adjust\_cramersv\_bias)  
 data.frame(xName, yName, assoc=cv, type="cramersV")  
  
 }else if(is\_numeric(x) && is\_numeric(y)){  
 correlation = cor(x, y, method=cor\_method, use="complete.obs")  
 data.frame(xName, yName, assoc=correlation, type="correlation")  
  
 }else if(is\_numeric(x) && is\_nominal(y)){  
 r\_squared = summary(lm(x ~ y))$r.squared  
 data.frame(xName, yName, assoc=sqrt(r\_squared), type="anova")  
  
 }else if(is\_nominal(x) && is\_numeric(y)){  
 r\_squared = summary(lm(y ~x))$r.squared  
 data.frame(xName, yName, assoc=sqrt(r\_squared), type="anova")  
  
 }else {  
 warning(paste("unmatched column type combination: ", class(x), class(y)))  
 }  
  
   
 result %>% mutate(complete\_obs\_pairs=sum(!is.na(x) & !is.na(y)), complete\_obs\_ratio=complete\_obs\_pairs/length(x)) %>% rename(x=xName, y=yName)  
 }  
  
  
 map2\_df(df\_comb$X1, df\_comb$X2, f)  
}  
  
cor\_data<-mixed\_assoc(bank\_clean) %>% subset(assoc<0.99999) %>% arrange(desc(abs(assoc)))  
kable(cor\_data[1:16,1:4],row.names = F, caption = "Mixed Data Types Correlation/Association")  
plot\_correlation(bank\_clean, type = "c", cor\_args = list("method"="pearson"))  
  
kable(summary(bank\_clean))  
  
  
plot\_boxplot(bank\_clean, by="y")  
  
age\_out<-boxplot(bank\_clean$age,plot=F)$out  
bal\_out<-boxplot(bank\_clean$balance,plot=F)$out  
campaign\_out<-boxplot(bank\_clean$campaign,plot=F)$out  
duration\_out<-boxplot(bank\_clean$duration,plot=F)$out  
pdays\_out<-boxplot(bank\_clean$pdays,plot=F)$out  
prev\_out<-boxplot(bank\_clean$previous,plot=F)$out  
  
bank\_clean <- bank\_clean[-which(bank\_clean$age %in% age\_out),]  
bank\_clean <- bank\_clean[-which(bank\_clean$balance %in% bal\_out),]  
bank\_clean <- bank\_clean[-which(bank\_clean$campaign %in% campaign\_out),]  
bank\_clean <- bank\_clean[-which(bank\_clean$duration %in% duration\_out),]  
bank\_clean <- bank\_clean[-which(bank\_clean$pdays %in% pdays\_out),]  
bank\_clean <- bank\_clean[-which(bank\_clean$previous %in% prev\_out),]  
  
head(bank\_clean)  
  
  
bank\_clean<- bank %>% mutate\_if(is.character, factor)  
bank\_clean$day<-as.factor(bank\_clean$day)  
  
###Decision Tree  
  
  
tic()  
  
  
set.seed(888)  
bank\_split <- initial\_split(bank\_clean, prop = 0.75,   
 strata = y)  
  
bank\_training <- bank\_split %>% training()  
bank\_test <- bank\_split %>% testing()  
bank\_folds <- vfold\_cv(bank\_training, v = 10)  
  
  
  
bank\_recipe <- recipe(y ~ ., data = bank\_training) %>%  
 step\_YeoJohnson(all\_numeric(),-all\_predictors())  
   
  
  
bank\_clean\_baked<-bank\_recipe %>%   
 prep() %>%   
 bake(new\_data = bank\_training)  
  
tree\_model <- decision\_tree(cost\_complexity = tune(),  
 tree\_depth = tune(),  
 min\_n = tune()) %>%   
 set\_engine('rpart') %>%   
 set\_mode('classification')  
  
tree\_workflow <- workflow() %>%   
 add\_model(tree\_model) %>%   
 add\_recipe(bank\_recipe)  
  
tree\_grid <- grid\_latin\_hypercube(cost\_complexity(),  
 tree\_depth(),  
 min\_n(),   
 size = 60)  
  
set.seed(888)  
  
  
tree\_tuning <- tree\_workflow %>%   
 tune\_grid(resamples = bank\_folds,  
 grid = tree\_grid)  
  
  
tree\_tuning %>% show\_best('roc\_auc')  
  
best\_tree <- tree\_tuning %>%   
 select\_best(metric = 'roc\_auc')  
  
  
final\_tree\_workflow <- tree\_workflow %>%   
 finalize\_workflow(best\_tree)  
  
  
tree\_wf\_fit <- final\_tree\_workflow %>%   
 fit(data = bank\_training)  
  
tree\_fit <- tree\_wf\_fit %>%   
 pull\_workflow\_fit()  
  
  
rpart.plot(tree\_fit$fit, roundint = F)  
  
tree\_last\_fit <- final\_tree\_workflow %>%   
 last\_fit(bank\_split)  
  
tree\_last\_fit %>% collect\_metrics()  
  
  
tree\_last\_fit %>% collect\_predictions() %>%   
 roc\_curve(truth = y, estimate = .pred\_no) %>%   
 autoplot()  
  
tree\_predictions <- tree\_last\_fit %>% collect\_predictions()  
  
conf\_mat(tree\_predictions, truth = y, estimate = .pred\_class)  
  
toc()  
  
  
tic()  
  
set.seed(888)  
nb\_split <- initial\_split(bank\_clean, prop = 0.75,   
 strata = y)  
  
nb\_training <- nb\_split %>% training()  
nb\_test <- nb\_split %>% testing()  
nb\_folds <- vfold\_cv(nb\_training, v = 10)  
  
nb\_recipe <- recipe(y ~ ., data = nb\_training) %>%  
 step\_YeoJohnson(all\_numeric(),-all\_predictors())  
   
  
  
nb\_wf <- workflow() %>%  
 add\_recipe(nb\_recipe)  
  
library(discrim)  
nb\_spec <- naive\_Bayes() %>%  
 set\_mode("classification") %>%  
 set\_engine("naivebayes")  
  
nb\_spec  
  
nb\_fit <- nb\_wf %>%  
 add\_model(nb\_spec) %>%  
 fit(data = nb\_training)  
  
nb\_wf\_final <- workflow() %>%  
 add\_recipe(nb\_recipe) %>%  
 add\_model(nb\_spec)  
  
nb\_rs <- fit\_resamples(  
 nb\_wf\_final,  
 nb\_folds,  
 control = control\_resamples(save\_pred = TRUE)  
)  
  
  
nb\_last\_fit <- nb\_wf\_final %>%   
 last\_fit(nb\_split)  
  
nb\_last\_fit %>% collect\_metrics()  
  
nb\_last\_fit %>% collect\_predictions() %>%   
 roc\_curve(truth = y, estimate = .pred\_no) %>%   
 autoplot()  
  
nb\_predictions <- nb\_last\_fit %>% collect\_predictions()  
conf\_mat(nb\_predictions, truth = y, estimate = .pred\_class)  
  
toc()  
  
tic()  
  
bank\_clean<-as.data.table(bank\_clean)  
  
set.seed(888)  
xg\_split<- initial\_split(bank\_clean)  
xg\_train<-training(xg\_split)  
xg\_test<-testing(xg\_split)  
  
set.seed(888)  
xg\_folds<-vfold\_cv(xg\_train,v=10)  
  
xgb\_spec <- boost\_tree(  
 trees = 2000,   
 tree\_depth = tune(), min\_n = tune(),   
 loss\_reduction = tune(),   
 sample\_size = tune(), mtry = tune(),   
 learn\_rate = tune()   
) %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")  
  
xgb\_spec  
  
xgb\_grid <- grid\_latin\_hypercube(  
 tree\_depth(),  
 min\_n(),  
 loss\_reduction(),  
 sample\_size = sample\_prop(),  
 finalize(mtry(), xg\_train),  
 learn\_rate(),  
 size = 60  
)  
  
xgb\_grid  
  
xgb\_recipe <- recipe(y ~ ., data = xg\_train) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes())  
   
  
  
  
xgb\_recipe %>%  
 prep() %>%  
 bake(new\_data = xg\_train)   
  
  
  
xgb\_wf <- workflow() %>%  
 add\_model(xgb\_spec) %>%  
 add\_recipe(xgb\_recipe)  
  
  
  
  
library(doParallel)  
cores<-detectCores()  
cl<- makeCluster(cores[1]-4)  
registerDoParallel(cl)  
  
  
set.seed(888)  
xgb\_res <- tune\_grid(  
 xgb\_wf,  
 resamples = xg\_folds,  
 grid = xgb\_grid,  
 control = control\_grid(save\_pred = TRUE))  
  
best\_auc <- select\_best(xgb\_res, "roc\_auc")  
  
  
  
final\_xgb <- finalize\_workflow(  
 xgb\_wf,  
 best\_auc  
)  
  
  
final\_res <- last\_fit(final\_xgb, xg\_split)  
collect\_metrics(final\_res)  
  
  
final\_res %>%  
 collect\_predictions() %>%  
 roc\_curve(y, .pred\_no) %>%  
 ggplot(aes(x = 1 - specificity, y = sensitivity)) +  
 geom\_line(size = 1.5, color = "midnightblue") +  
 geom\_abline(  
 lty = 2, alpha = 0.5,  
 color = "gray50",  
 size = 1.2  
 )  
  
  
final\_res %>%  
 collect\_predictions() %>%   
 conf\_mat(truth = y, estimate = .pred\_class)  
  
library(vip)  
final\_xgb %>%  
 fit(data = xg\_train) %>%  
 pull\_workflow\_fit() %>%  
 vip(geom = "col")  
  
toc()  
tic()  
library("caret")  
control <- rfeControl(functions = rfFuncs, # random forest  
 method = "repeatedcv", # repeated cv  
 repeats = 5, # number of repeats  
 number = 10) # number of folds  
  
# Features  
x <- bank\_clean %>%  
 select(-y) %>%  
 as.data.frame()  
  
# Target variable  
y <- bank\_clean$y  
  
# Training: 80%; Test: 20%  
set.seed(888)  
inTrain <- createDataPartition(y, p = .80, list = FALSE)[,1]  
  
x\_train <- x[ inTrain, ]  
x\_test <- x[-inTrain, ]  
  
y\_train <- y[ inTrain]  
y\_test <- y[-inTrain]  
  
result\_rfe1 <- rfe(x = x\_train,   
 y = y\_train,   
 sizes = c(1:16),  
 rfeControl = control)  
  
# Print the results  
result\_rfe1  
  
# Print the selected features  
predictors(result\_rfe1)  
  
# Print the results visually  
ggplot(data = result\_rfe1, metric = "Accuracy") + theme\_bw()  
  
toc()  
###subsetting our selected features  
bank\_selected<-bank\_clean[,c("duration","month","day","poutcome","pdays","age","contact","previous","y")]  
  
###Decision Tree  
  
  
  
tic()  
  
  
  
set.seed(888)  
bank\_split <- initial\_split(bank\_selected, prop = 0.75,   
 strata = y)  
  
bank\_training <- bank\_split %>% training()  
bank\_test <- bank\_split %>% testing()  
bank\_folds <- vfold\_cv(bank\_training, v = 10)  
  
  
  
bank\_recipe <- recipe(y ~ ., data = bank\_training) %>%  
 step\_YeoJohnson(all\_numeric(),-all\_predictors())  
   
  
  
bank\_clean\_baked<-bank\_recipe %>%   
 prep() %>%   
 bake(new\_data = bank\_training)  
  
tree\_model <- decision\_tree(cost\_complexity = tune(),  
 tree\_depth = tune(),  
 min\_n = tune()) %>%   
 set\_engine('rpart') %>%   
 set\_mode('classification')  
  
tree\_workflow <- workflow() %>%   
 add\_model(tree\_model) %>%   
 add\_recipe(bank\_recipe)  
  
tree\_grid <- grid\_latin\_hypercube(cost\_complexity(),  
 tree\_depth(),  
 min\_n(),   
 size = 60)  
  
set.seed(888)  
  
  
tree\_tuning <- tree\_workflow %>%   
 tune\_grid(resamples = bank\_folds,  
 grid = tree\_grid)  
  
  
tree\_tuning %>% show\_best('roc\_auc')  
  
best\_tree <- tree\_tuning %>%   
 select\_best(metric = 'roc\_auc')  
  
  
final\_tree\_workflow <- tree\_workflow %>%   
 finalize\_workflow(best\_tree)  
  
  
tree\_wf\_fit <- final\_tree\_workflow %>%   
 fit(data = bank\_training)  
  
tree\_fit <- tree\_wf\_fit %>%   
 pull\_workflow\_fit()  
  
  
rpart.plot(tree\_fit$fit, roundint = F)  
  
tree\_last\_fit <- final\_tree\_workflow %>%   
 last\_fit(bank\_split)  
  
tree\_last\_fit %>% collect\_metrics()  
  
  
tree\_last\_fit %>% collect\_predictions() %>%   
 roc\_curve(truth = y, estimate = .pred\_no) %>%   
 autoplot()  
  
tree\_predictions <- tree\_last\_fit %>% collect\_predictions()  
  
conf\_mat(tree\_predictions, truth = y, estimate = .pred\_class)  
  
toc()  
  
  
tic()  
  
set.seed(888)  
nb\_split <- initial\_split(bank\_selected, prop = 0.75,   
 strata = y)  
  
nb\_training <- nb\_split %>% training()  
nb\_test <- nb\_split %>% testing()  
nb\_folds <- vfold\_cv(nb\_training, v = 10)  
  
nb\_recipe <- recipe(y ~ ., data = nb\_training) %>%  
 step\_YeoJohnson(all\_numeric(),-all\_predictors())  
   
  
  
nb\_wf <- workflow() %>%  
 add\_recipe(nb\_recipe)  
  
library(discrim)  
nb\_spec <- naive\_Bayes() %>%  
 set\_mode("classification") %>%  
 set\_engine("naivebayes")  
  
  
  
nb\_fit <- nb\_wf %>%  
 add\_model(nb\_spec) %>%  
 fit(data = nb\_training)  
  
nb\_wf\_final <- workflow() %>%  
 add\_recipe(nb\_recipe) %>%  
 add\_model(nb\_spec)  
  
nb\_rs <- fit\_resamples(  
 nb\_wf\_final,  
 nb\_folds,  
 control = control\_resamples(save\_pred = TRUE)  
)  
  
  
nb\_last\_fit <- nb\_wf\_final %>%   
 last\_fit(nb\_split)  
  
nb\_last\_fit %>% collect\_metrics()  
  
nb\_last\_fit %>% collect\_predictions() %>%   
 roc\_curve(truth = y, estimate = .pred\_no) %>%   
 autoplot()  
  
nb\_predictions <- nb\_last\_fit %>% collect\_predictions()  
conf\_mat(nb\_predictions, truth = y, estimate = .pred\_class)  
  
toc()  
  
tic()  
  
bank\_selected<-as.data.table(bank\_selected)  
  
set.seed(888)  
xg\_split<- initial\_split(bank\_selected)  
xg\_train<-training(xg\_split)  
xg\_test<-testing(xg\_split)  
  
set.seed(888)  
xg\_folds<-vfold\_cv(xg\_train,v=10)  
  
xgb\_spec <- boost\_tree(  
 trees = 2000,   
 tree\_depth = tune(), min\_n = tune(),   
 loss\_reduction = tune(),   
 sample\_size = tune(), mtry = tune(),   
 learn\_rate = tune()   
) %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")  
  
  
  
xgb\_grid <- grid\_latin\_hypercube(  
 tree\_depth(),  
 min\_n(),  
 loss\_reduction(),  
 sample\_size = sample\_prop(),  
 finalize(mtry(), xg\_train),  
 learn\_rate(),  
 size = 60  
)  
  
  
  
xgb\_recipe <- recipe(y ~ ., data = xg\_train) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes())  
   
  
  
  
xgb\_recipe %>%  
 prep() %>%  
 bake(new\_data = xg\_train)   
  
  
  
xgb\_wf <- workflow() %>%  
 add\_model(xgb\_spec) %>%  
 add\_recipe(xgb\_recipe)  
  
  
  
  
library(doParallel)  
cores<-detectCores()  
cl<- makeCluster(cores[1]-4)  
registerDoParallel(cl)  
  
  
set.seed(888)  
xgb\_res <- tune\_grid(  
 xgb\_wf,  
 resamples = xg\_folds,  
 grid = xgb\_grid,  
 control = control\_grid(save\_pred = TRUE))  
  
best\_auc <- select\_best(xgb\_res, "roc\_auc")  
  
  
  
final\_xgb <- finalize\_workflow(  
 xgb\_wf,  
 best\_auc  
)  
  
  
final\_res <- last\_fit(final\_xgb, xg\_split)  
collect\_metrics(final\_res)  
  
  
final\_res %>%  
 collect\_predictions() %>%  
 roc\_curve(y, .pred\_no) %>%  
 ggplot(aes(x = 1 - specificity, y = sensitivity)) +  
 geom\_line(size = 1.5, color = "midnightblue") +  
 geom\_abline(  
 lty = 2, alpha = 0.5,  
 color = "gray50",  
 size = 1.2  
 )  
  
final\_res %>%  
 collect\_predictions() %>%   
 conf\_mat(truth = y, estimate = .pred\_class)  
  
toc()  
```

```{r ref.label = knitr::all\_labels(), echo = TRUE, eval = FALSE}

# this R markdown chunk generates a code appendix

```