Code

Final Project PSTAT 131

Richard Antony

2022-10-03

Introduction

This project will be about to find the best model to predict whether a debtor is defaulting their loan or not. The dataset contains 12 predictors and 1 response. In todays world, financial banking mostly generate profit from loans. If a bank give a loan to a debtor that is unable to pay back then it will be a very high credit risk. So this project objective is to minimize that credit risk by making a model with the 12 predictors to predict the response in hoping to see which charactheristic of individuals is most likely to default their loans.

In this project, I will be first doing Exploratory Data Analysis. I will look through the spread of the response and how its distributed. Then I will see how each variables correlate with one another. After doing EDA, I then will proceed to split and cross validate the data. After splitting and cross validation is done, I will then build 3 models (Logistic Regression, Tree Classifciation Model & Boosted Tree Models). Each model will have parameters that are tune to achive the best model. The metric I use for validating it to be the best model is by looking at the ROC_AUC.

Loading Packages

Hide

```r
library(ggplot2)
library(tidyverse)
library(dplyr)
library(tidyverse)
#install.packages('tidymodels')
library(tidymodels)
#install.packages('ISLR')
library(ISLR)
library(kernlab)
library(tidyverse)
library(tidymodels)
library(ISLR)
#install.packages('rpart.plot')
library(rpart.plot)
#install.packages('vip')
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
library(tidymodels)
library(ISLR)
library(tidyverse)
library(tidymodels)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(glmnet)
tidymodels_prefer()
library(tidymodels)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(tidymodels)
library(ISLR) # For the Smarket data set
library(ISLR2) # For the Bikeshare data set
library(discrim)
library(poissonreg)
library(corrr)
library(klaR) # for naive bayes
#install.packages('corrplot')
library(corrplot)
```

EDA

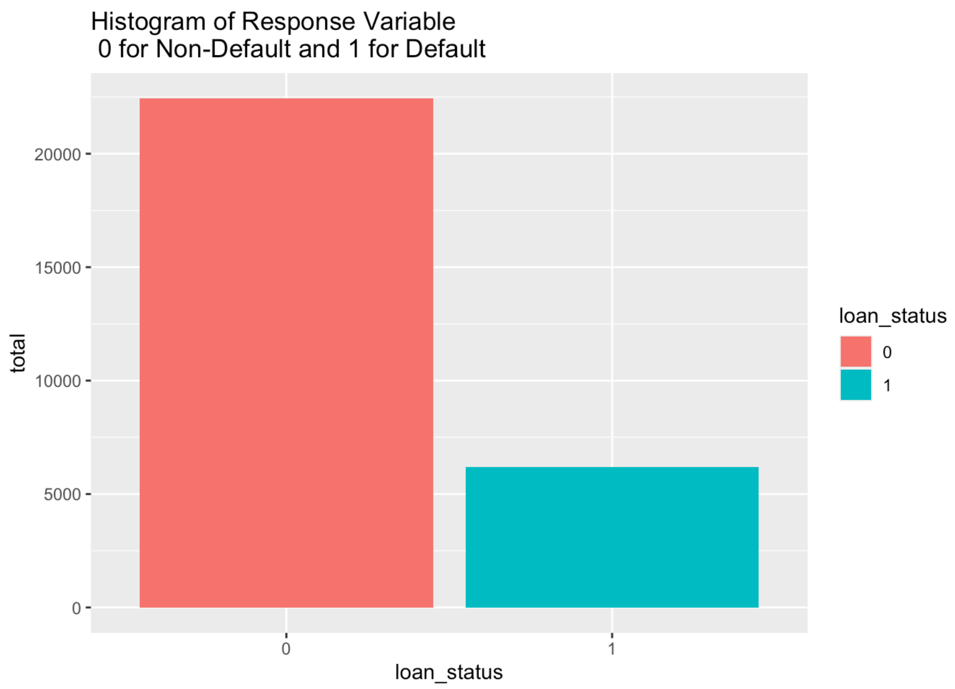
Hide

```
#Read Data

data<-read.csv('~/Desktop/PSTAT 131 /Final Project/Data
/credit_risk_dataset.csv')

# Omit values because only takes up 12% of the data whi
ch is not alot.
#32581 -28638
#3943/32581
data <-data %>% na.omit()
data<-data %>%  clean_names()

#Make variables to be factor if needed
data$person_home_ownership<-as.factor(data$person_home_
ownership)
data$loan_intent<-as.factor(data$loan_intent)
data$loan_grade<-as.factor(data$loan_grade)
data$loan_status<-as.factor(data$loan_status)
data$cb_person_default_on_file<-as.factor(data$cb_perso
n_default_on_file)

#Make double & integer variable into numeric
data$loan_int_rate <-as.numeric(data$loan_int_rate)
data$loan_percent_income<- as.numeric(data$loan_percent
_income)
data$person_age<-as.numeric(data$person_age)
data$person_income<-as.numeric(data$person_income)
data$loan_amnt<-as.numeric(data$loan_amnt)
data$cb_person_cred_hist_length<-as.numeric(data$cb_per
son_cred_hist_length)
```

Check spread of outcome variable

Hide

```
Outcome_spread<-data %>% select(loan_status) %>% group_
by(loan_status) %>% summarise(total=sum(n()))

ggplot(Outcome_spread,aes(loan_status,total,fill=loan_s
tatus)) + geom_col() + ggtitle("Histogram of Response V
ariable\n 0 for Non-Default and 1 for Default")
```

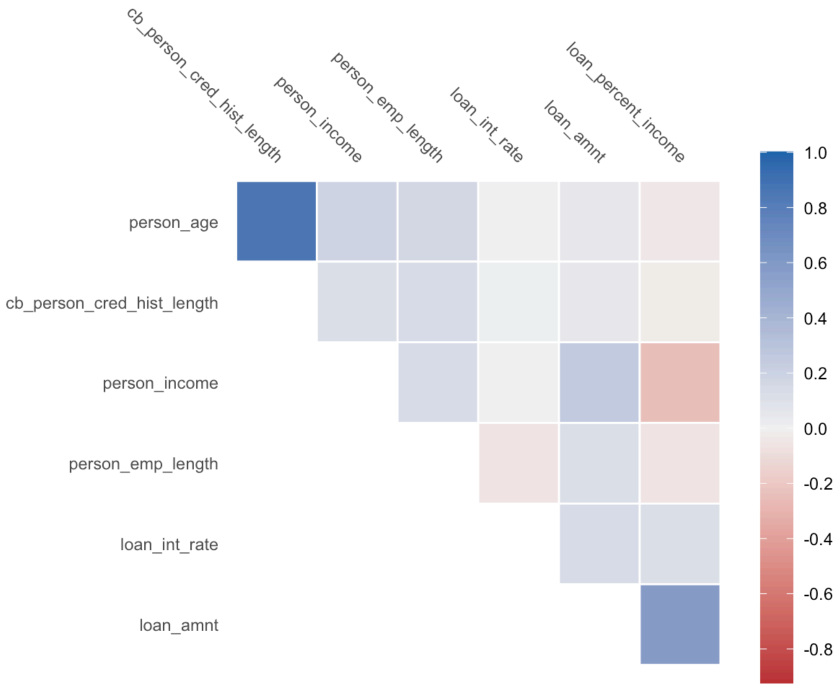


We see that in the histogram of the outcome variable most of it is spread in 0 rather than 1. The outcome data is imbalance. We see that loan_status that is '0' is 22435 while loan_status that is '1' is 6203. This means that 78% the outcome observations are '0' while 22% of the outcome observations are 22% of the total data. Most of my observations is non default and only few is default. But we need to explore the data further.

Check correlation between variables

Hide

```
corr_data<-correlate(data)

corr_data %>% autoplot(type="heatmap")
```

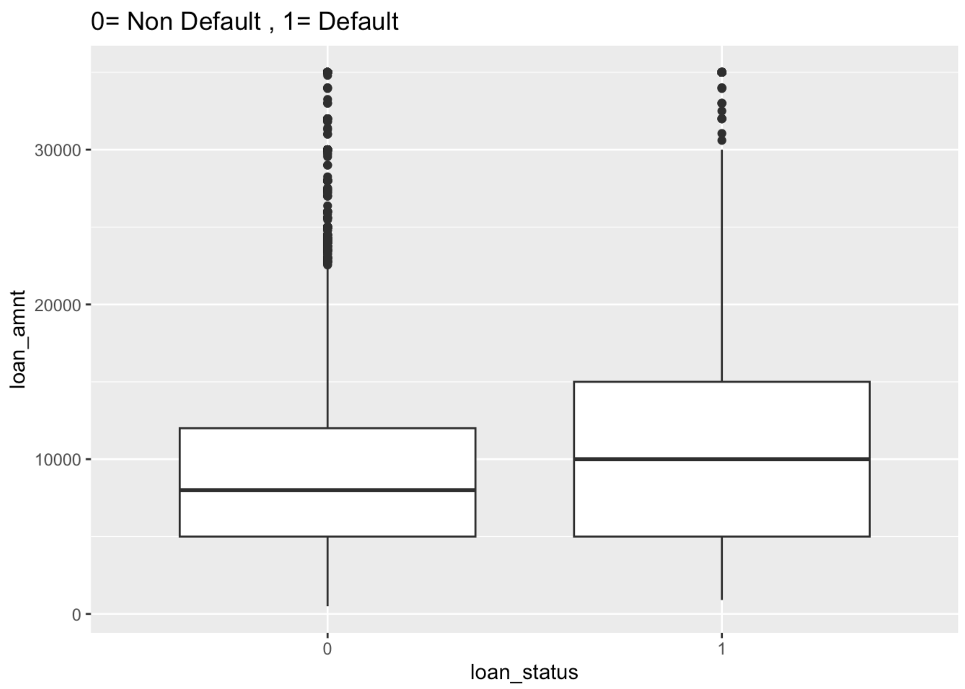


From the correlation plot with heatmap we see that some variables are highly correlated to one another. Example of this can be the variable “person_age” with “cb_person_cred_hist_length” which have a correlation between 0.8 to 1. But looking through it I dont think this has to be removed. “cb_person_cred_hist_length” is the credit history length and “person_age” is just age. Its not like the variable “cb_person_cred_hist_length” value is derived from “person_age” like the question from the homework. So I will still keep all the varaible.

Check relations between loan_status to loan_amount with boxplot

Hide

```
ggplot(data,aes(loan_status,loan_amnt)) + geom_boxplot(
) +ggtitle("0= Non Default , 1= Default")
```



‘0’ for non default . ‘1’ for default. Now in this box plot I try to see the relationship between loan amount (loan_amnt) and loan status. I see that the higher the loan amount is, the higher people would likely to default. We can observe that the median for people who defaulted (1) has a higher median than those people who dont (0). Lets explore further.

Data Splitting & Cross Validation Folding

Hide

```
set.seed(9898)
# 80 % to train 20% to test
data_split<-initial_split(data,prop = 0.8,strata = loan
_status)
data_train <-training(data_split)
data_test<- testing(data_split)
```

Data folding

Hide

```
set.seed(9898)
data_train_fold <- vfold_cv(data_train, v = 10, strata
= loan_status)
```

Receipe

Hide

```
data_rec<-recipe(loan_status~person_age+ person_income+
person_home_ownership+ person_emp_length+ loan_intent+
loan_grade+ loan_amnt+ loan_int_rate+ loan_percent_inco
me+ cb_person_default_on_file + cb_person_cred_hist_len
gth,data = data_train) %>% step_dummy(all_nominal_predi
ctors()) %>% step_normalize(all_predictors())
```

In this section, I split my data to 80% train and 20% test. This is due to the fact that the more % of the data for the train, the more observations that can be studied and thus have a higher chance to create better result for the testing data. I split it base on stratification of the target variable (loan_status). After the split was done, I then apply folding to the train data base on stratifying the target variable (loan_status) 10 times. This is to make sure that each fold will have adequate categories(0= Non Default and 1= Default). If cross validation is not done on the folds, then some folds can have the response variable imbalnce, meaning might have more 1's or 0's in the fold. Cross validation is also useful here because as we see our response variable are quite imbalance. Having Most of the response to 0 (Non Default) and less to 1 (Default). Cross validation ensures that the folds have a good amount of the 2 type of responses to be studied in the training set.

After folding and splitting, I then create a recipe for the training data. This recipe will be use heavily as it can be fit to any models depending on the set_engine and workflow.

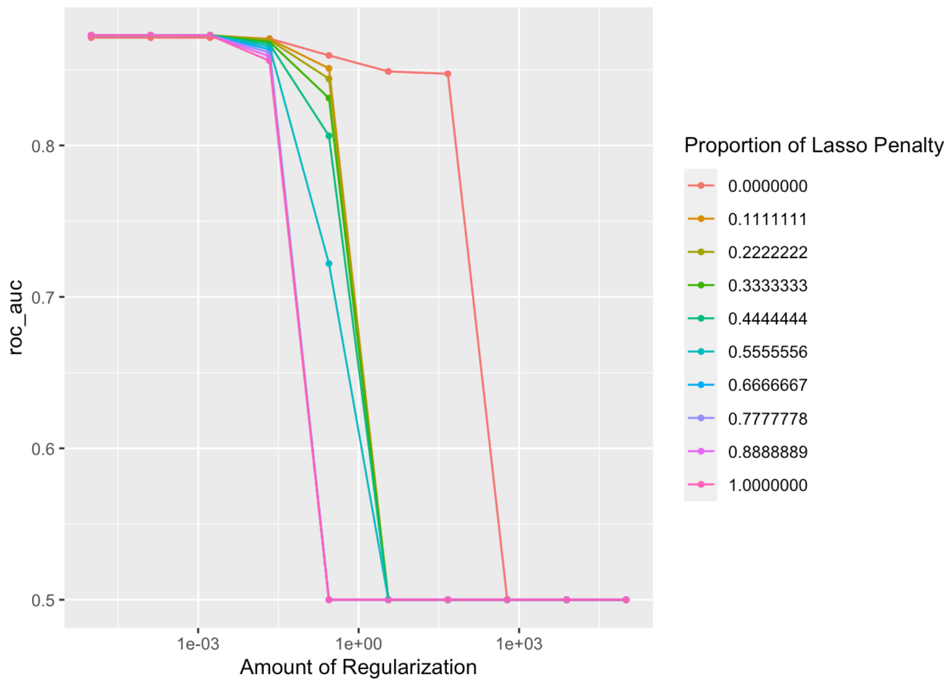
Model Creation

Logistic Regression

In this part, I tried to fit the data using logistic regression model and tune it with the parameters mixture and penalty. The penalties are both use from Lasso and Ridge regression model.

Hide

```
log_reg<-logistic_reg(penalty = tune(), mixture = tune(
)) %>%  set_engine("glmnet") %>%  set_mode("classificat
ion")

tuned_log_wkflow <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(data_rec)

degree_grid <- grid_regular(penalty(range = c(-5,5)),mi
xture(range = c(0,1)), levels = 10)

#tune_log_res <- tune_grid(object = tuned_log_wkflow, r
esamples = data_train_fold, grid = degree_grid, metrics
= metric_set(roc_auc), control = control_grid(verbose =
T))

#save(tune_log_res,file = "tune_log_res.rda")
load(file="tune_log_res.rda")

autoplot(tune_log_res)
```



Hide

```
require(knitr)
lr_metrics <- collect_metrics(tune_log_res ) %>% dplyr:
:arrange(desc(mean))

df <- data.frame(Metric = lr_metrics$.metric, Value = l
r_metrics $mean, Standard_Error = lr_metrics$std_err, P
enalty_Value = lr_metrics$penalty, Mixture_Value = lr_m
etrics$mixture )

kable(head(df))
```

Metric	Value	Standard_Error	Penalty_Value	Mixture_Value
roc_auc	0.8727775	0.0031711	0.0016681	0.1111111
roc_auc	0.8727561	0.0031856	0.0000100	0.7777778
roc_auc	0.8727561	0.0031856	0.0001292	0.7777778
roc_auc	0.8727533	0.0031875	0.0000100	0.8888889
roc_auc	0.8727533	0.0031875	0.0001292	0.8888889
roc_auc	0.8727518	0.0031890	0.0000100	1.0000000

Hide

```
best_lr_roc_auc<-df[1,2]
lr_se<-df[1,3]
```

Using Logistic Regresion and tuning the parameter mixture & penalty, I got a model that has the best ROC_AUC of 0.8727775
with standard error of 0.0031711, Penalty_Value of 0.0016681 and a 0.1111111. I will try other models from here. I will compare their ROC_AUC in which the highest is the best model.

Simple Tree Model

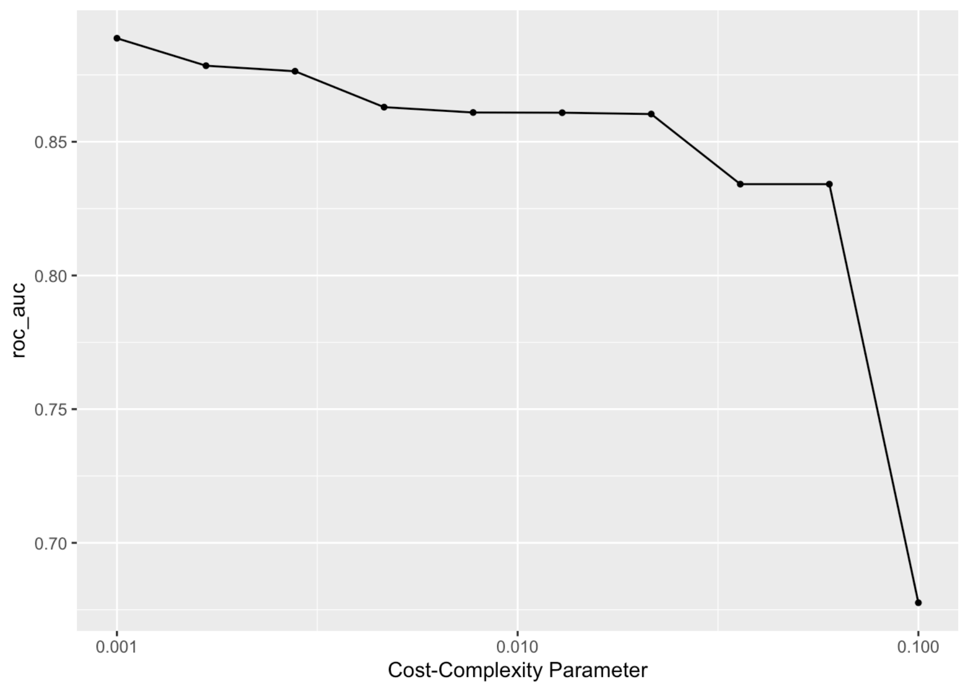
Having 11 predictors, we have a lot of ways to fit the model for the data. So the parameter we are tuning to find the best model is cost_complexity. This cost complexity parameter will determine the minimun improvement in the model needed in each node.

Hide

```
set.seed(9898)
tree_spec <- decision_tree() %>%
  set_engine('rpart') %>%
  set_mode('classification')

tree_wf <- workflow()%>%
  add_model(tree_spec %>% set_args(cost_complexity = tu
ne())) %>%
  add_formula(loan_status~person_age+ person_income+ pe
rson_home_ownership+ person_emp_length+ loan_intent+ lo
an_grade+ loan_amnt+ loan_int_rate+ loan_percent_income
+ cb_person_default_on_file + cb_person_cred_hist_lengt
h)


tree_param <- grid_regular(cost_complexity(range = c(-3
,-1)), levels = 10)

tune_tree_res <- tune_grid(tree_wf, resamples = data_tr
ain_fold, grid = tree_param, metrics = metric_set(roc_a
uc), control = control_grid(verbose = T))

#save(tune_tree_res,file = "tune_tree_res.rda")
#load(file="tune_tree_res.rda")
```

Hide

```
autoplot(tune_tree_res)
```



Hide

```
tree_metrics <- collect_metrics(tune_tree_res) %>% dply
r::arrange(desc(mean))

#Create Table
df <- data.frame(Metric = tree_metrics$.metric, Value =
tree_metrics$mean, Standard_Error = tree_metrics$std_er
r, Cost_Complexity_Value = tree_metrics$cost_complexity
)

kable(head(df))
```

Metric	Value	Standard_Error	Cost_Complexity_Value
roc_auc	0.8887092	0.0042210	0.0010000
roc_auc	0.8784456	0.0034500	0.0016681
roc_auc	0.8763692	0.0034215	0.0027826
roc_auc	0.8629446	0.0039080	0.0046416
roc_auc	0.8609464	0.0039831	0.0077426
roc_auc	0.8608701	0.0039485	0.0129155

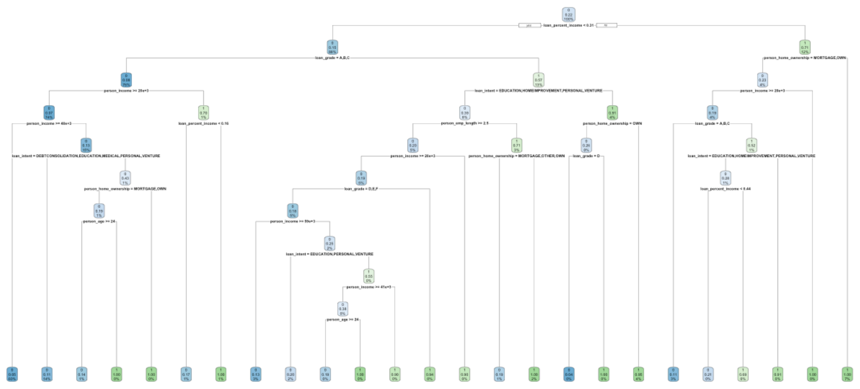
Hide

```
best_tree_roc_auc<-df[1,2]
tree_se<-df[1,3]
```

As we can see here, the simple tree model did better than the logistic regression model. We see that the simple tree model has the best roc_auc of 0.8887092 and standard error of 0.0042210 with complexity value of 0.0010000. I think the reason it did better because tree models resample the train data to each nodes and make models out of every node and compare which is the best.

Hide

```
best_tree <- select_best(tune_tree_res, metric = 'roc_a
uc')

tree_final <- finalize_workflow(tree_wf, best_tree)

tree_final_fit <- fit(tree_final, data = data_train)

tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```



Here we can see the nodes that are created by the tree model. The nodes are seleceted and created by the program.

Boosted Tree Model

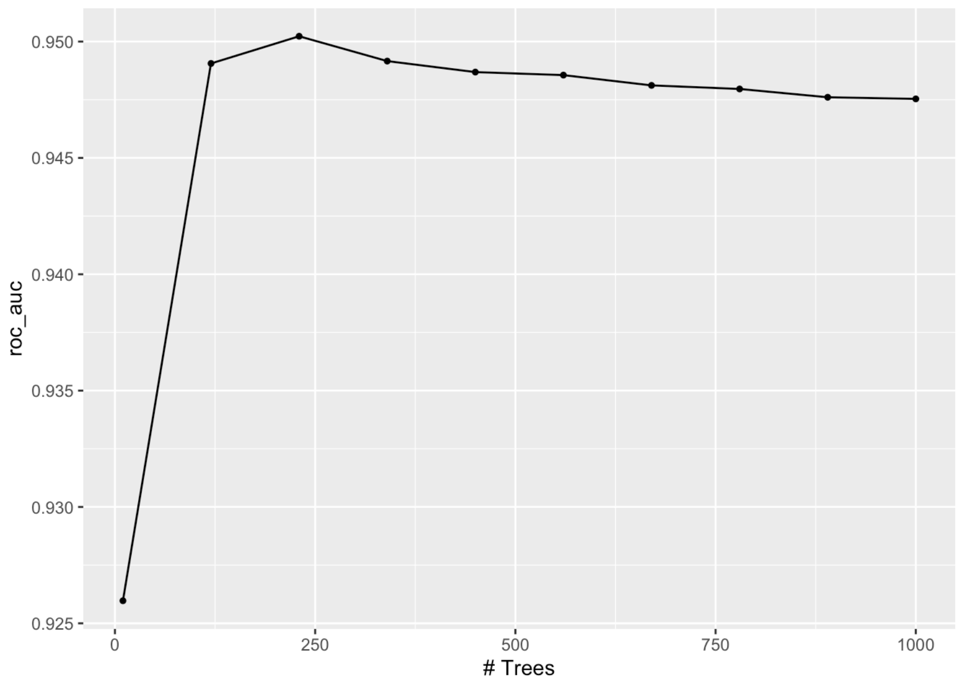
In the boosted tree model, our parameters are the trees. We are using the xgboost (Gradient Boosting) for the boosted tree models. My trees range from 10:1000 to make sure every observations is studied thouroghly. The hyperparameters here that we will be tuning is the trees ranging from 10 to 1000 so that every observations is studied thouroughly.

Hide

```
set.seed(9898)

boost_spec <- boost_tree(trees = tune()) %>%
  set_engine('xgboost') %>%
  set_mode('classification')

boost_wf <- workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(data_rec)


boost_param <- grid_regular(trees(range = c(10,1000)),
levels = 10)

tune_boost_res <- tune_grid(boost_wf, resamples = data_
train_fold, grid = boost_param, metrics = metric_set(ro
c_auc), control = control_grid(verbose = T))


#saveRDS(tune_boost_res,file="tune_boost_res.rds")
#load(file = "tune_boost_res.rds")
```

Hide

```
autoplot(tune_boost_res)
```



Hide

```
boost_metrics <- collect_metrics(tune_boost_res) %>% dp
lyr::arrange(-mean)

df <- data.frame(Metric = boost_metrics$.metric, Value
= boost_metrics$mean, Standard_Error = boost_metrics$st
d_err, trees_Value = boost_metrics$trees )

kable(head(df))
```

Metric	Value	Standard_Error	trees_Value
roc_auc	0.9502284	0.0015047	230
roc_auc	0.9491613	0.0014320	340
roc_auc	0.9490592	0.0013709	120
roc_auc	0.9486886	0.0013957	450
roc_auc	0.9485587	0.0013975	560
roc_auc	0.9481170	0.0014065	670

Hide

```
best_boost_roc_auc <- df[1,2]
boost_se <- df[1,3]
```

Using the boosted tree model we see that the best ROC_AUC is 0.9502284 with Standard Error of 0.0015047 and 230 trees. The boosted model did better than the simple tree model and logistic regression. I think it did better because it is gradient boosted and like the simple tree model it also resamples the model in each tree.

Model selection

After doing Tuned Hyperparameters for Logistic Regression, Tuned Hyperparameters for Tree Classification Model and Tuned Hyperparameters for Boosted Tree Model. We now proceed choose which model out of the 3 is the best. I will validate the model the best by their ROC_AUC

Hide

```
Models<-c("Logistic Regression","Simple Tree Model","Bo
osted Tree Model")
roc_auc<-c(best_lr_roc_auc,best_tree_roc_auc,best_boost
_roc_auc)
se<-c(lr_se,tree_se,boost_se)

df<-data.frame(Models=Models,ROC_AUC=roc_auc,Standard_E
rror =se)
df<-df %>% arrange(-roc_auc)
kable(df)
```

Models	ROC_AUC	Standard_Error
Boosted Tree Model	0.9502284	0.0015047
Simple Tree Model	0.8887092	0.0042210
Logistic Regression	0.8727775	0.0031711

As we can see from the table, Boosted Tree classifciation model did the best for this dataset as it has an ROC_AUC of 0.9502284 which is the highest and very close to 1. Not only that, this model also has the lowest standard error from the other 2 models.

This is the hyper parameters that were tuned for the best Boosted Tree Clasification Model

Hide

```
best_vals<-select_best(tune_boost_res)
kable(best_vals)
```

trees	.config
230	Preprocessor1_Model03

With 230 Trees, the Boosted Tree Model is able to produce 0.9502284 ROC_AUC. Next we proceed with this model to be fitted to the test data.

Fitting to test data and see performance

ROC Curves

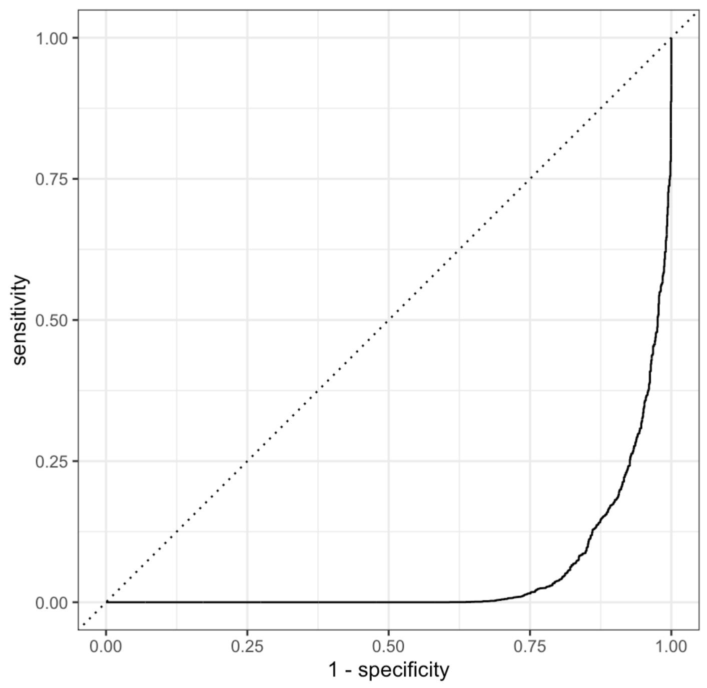
Hide

```
require(knitr)
boost_final<-finalize_workflow(boost_wf,best_vals)
boost_final_fit<-fit(boost_final,data = data_train)
boost_final_res<-augment(boost_final_fit, new_data = da
ta_test)
boost_final_res_1<- augment(boost_final_fit, new_data =
data_test) %>% roc_auc(loan_status,estimate= .pred_1)
boost_final_res_0<- augment(boost_final_fit, new_data =
data_test) %>% roc_auc(loan_status,estimate= .pred_0)

ROC_AUC=c(boost_final_res_0$.estimate,boost_final_res_1
$.estimate)
Cateogry= c('0 (Non-Default)', '1 (Default)')
df<-data.frame(ROC_AUC,Cateogry)
kable(df)
```

ROC_AUC	Cateogry
0.9496109	0 (Non-Default)
0.0503891	1 (Default)

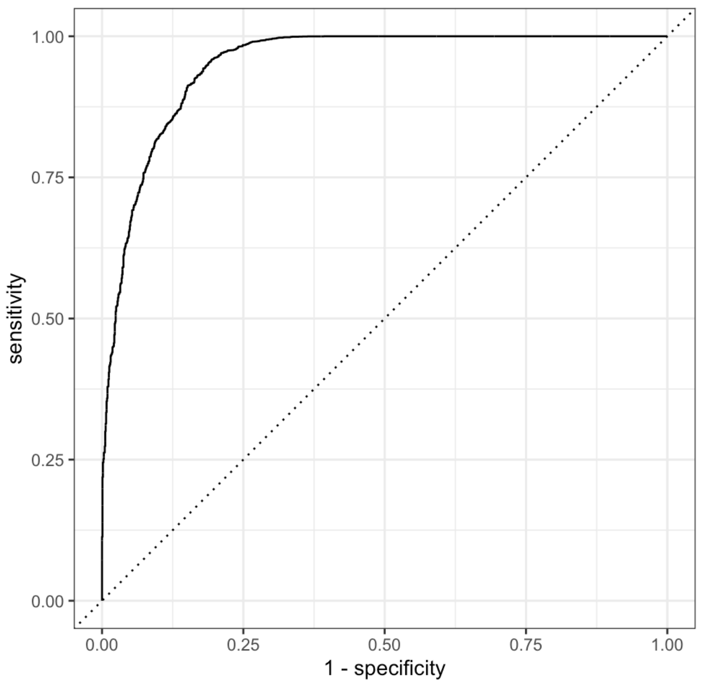
Hide

```
par(mfrow=c(1,2))
autoplot(augment(boost_final_fit, new_data = data_test)
%>% roc_curve(loan_status, estimate=.pred_1))
```



Hide

```
autoplot(augment(boost_final_fit, new_data = data_test)
%>% roc_curve(loan_status, estimate=.pred_0))
```

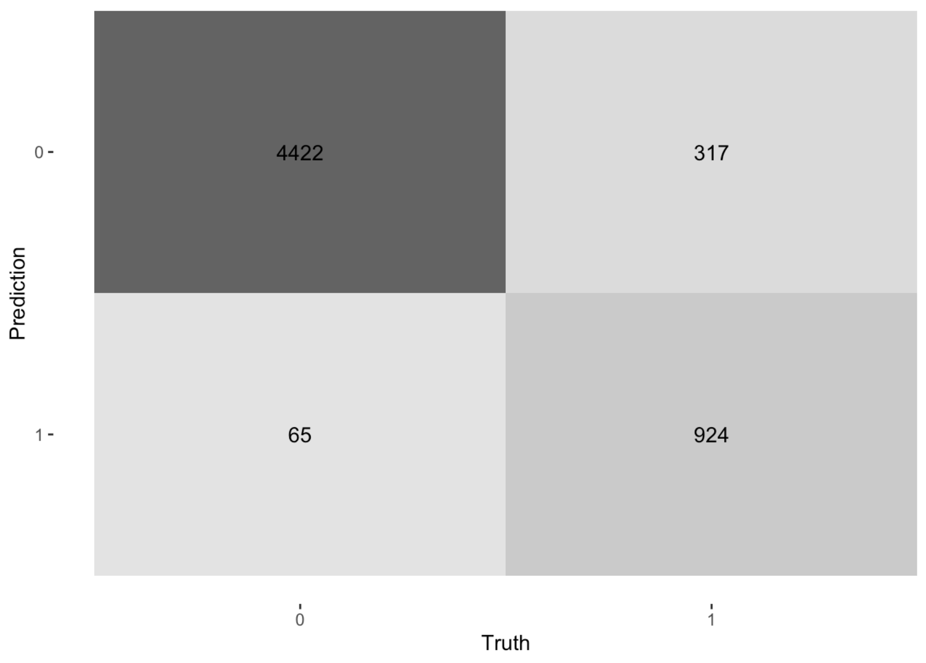


From the ROC_AUC plot, I see that if my target is to predict a person who is not going to default then I would have almost 100% chance on predicting it truely. But if I want to predict a person who is going to default then I would only have a 5% chance on predicting it correctly. This model is good at predicting person who is non default rather than default. But since I want to predict a debtor who will default, I think this model had failed.

Confusion Matrix

Hide

```
#Confusion Matrix
augment(boost_final_fit, new_data = data_test) %>%
  conf_mat(loan_status, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```



Hide

```
#accuracy, sensitivity, specificity
multi_metric <- metric_set(accuracy, sensitivity, speci
ficity)
augment(boost_final_fit, new_data = data_test) %>%
  multi_metric(truth = loan_status, estimate = .pred_cl
ass) %>% kable()
```

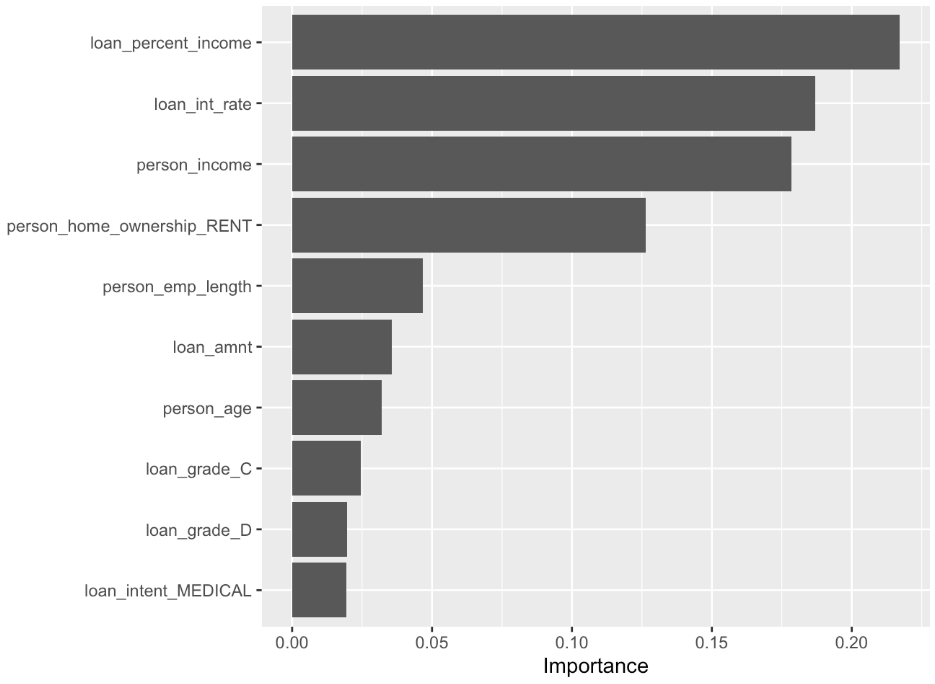
.metric	.estimator	.estimate
accuracy	binary	0.9333101
sensitivity	binary	0.9855137
specificity	binary	0.7445608

After fitting the Boosted Tree Classification Model, I see that the result that is predicted for the model towards the testing data is pretty outstanding. From the confusion matrix itself, I saw that my data predict most of the observations true. As you can see too, I obtain an accuracy of 93%, sensitivity of 98.5% and specificity of 74.4%. I think the reason behind why more of the response “0 (Non-Default)” are predicted truly more than the response “1 (Default)” is that because our data response variable consist mostly of the response “0 (Non-Default)” . If you think about it, it makes sense because a bank that has high defaults will most likely fail.

Variable Importance Plot

Hide

```
vip(boost_final_fit%>% extract_fit_parsnip())
```



From Looking at the Variable Importance Plot, I do agree loan_percent_income being the most important variable to predict loan status. The more your loan to income ratio, the more you are post a risk towards the bank.

Conclusion

This project was fun and educative for me. I tried to implement the stuff I learn in PSTAT 131 in a real world problem. Although the result is not what I wanted, I cannot say that I did everything wrongly. Maybe if I change the order factor of my response variable then this would produce a different outcome.

To conclude this project, the best model I got is a Boosted Tree Classification Model with 230 trees tuned.

References

Dataset: https://www.kaggle.com/datasets/laotse/credit-risk-dataset (https://www.kaggle.com/datasets/laotse/credit-risk-dataset)
Homeworks Labs Lecture Slides