Credit Card Approval Prediction

Class B Group 4: CAI Shida, CHEN Silu, WANG Liting, ZHANG Yushuangzi

Contents

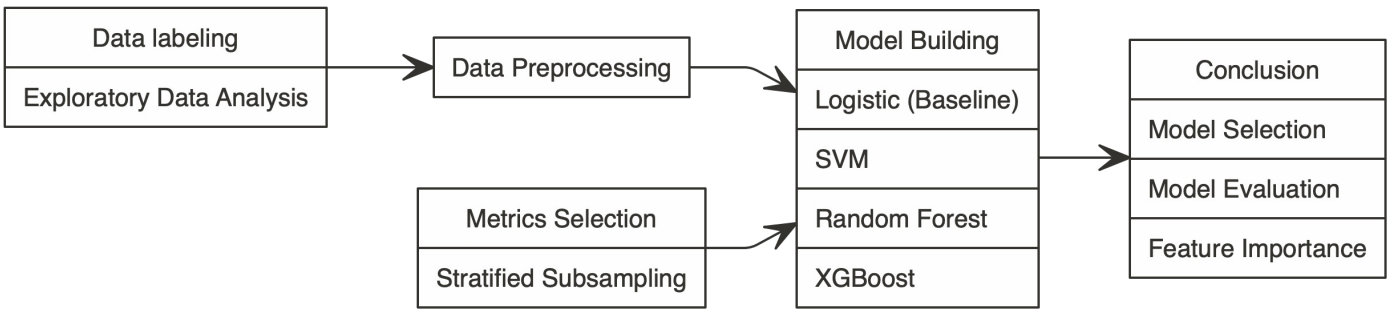
Executive Summary

A common risk control method in the financial industry is credit score card. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings, so that banks can decide whether to issue a credit card to the applicant. This project aims to apply machine learning techniques to help banks detect uncreditworthy applicants who tend to default and reject their applications so as to reduce potential loss.

Based on credit and application records, exploratory data analysis is carried out to understand the data structure and perform data preprocessing. In the modelling part, four methods including Logistic Regression, which is set as the baseline, SVM (Support Vector Machines), Random Forest, and XGBoost, are applied to solve the classification problem. By comparing the cross-validated F1-score of each optimal model after hyperparameters tuning if applied, XGBoost with the highest cv-F1 (37%) is selected and used to build the final model. When validated on the testing set, XGBoost achieves near 98% of accuracy and 35% of F1-score. Additionally, the feature importance plot suggests that banks should pay relatively more attention to the number of employed year, age and income of the potential clients, when evaluating credit card applications.

1. Introduction, Problem Description & Relevance

This project aims to implement machine learning techniques to predict if an applicant is 'good' or 'bad' client, which can help the banks to reduce credit-loss rates, increase revenue and efficiency gains. Different from other tasks, the label itself and the definition of 'good' or 'bad' is not given. We will use existing default records and external explanation to construct the response variable to facilitate our analysis. Since Logistic model is a common method for credit scoring due to its high interpretability, we will use it as our baseline model. We will also use more advanced predictive methods such as Support Vector Machines, Random Forest, and Boosting in our analysis. The framework of our project is shown in the chart below.

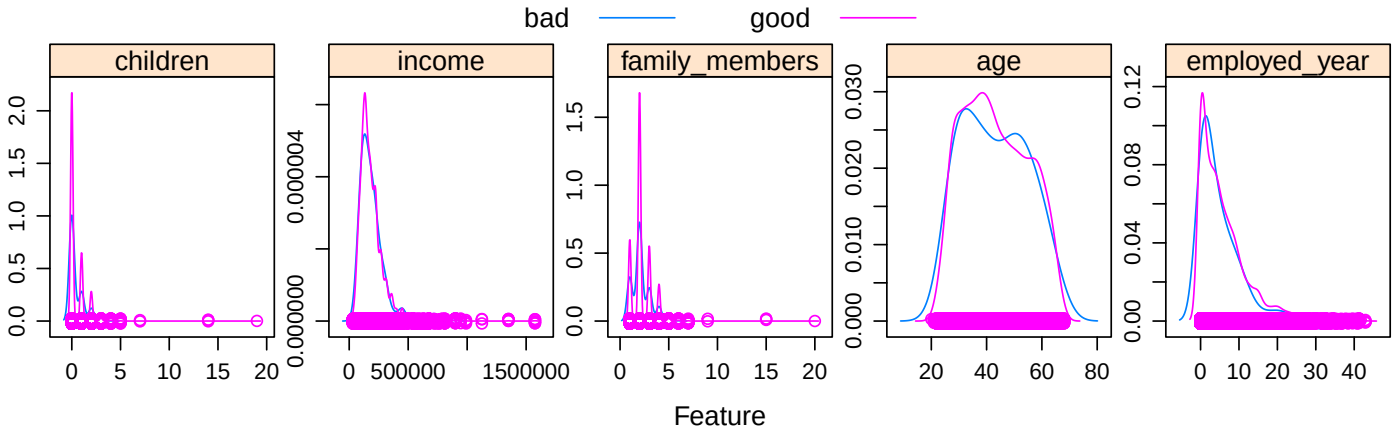


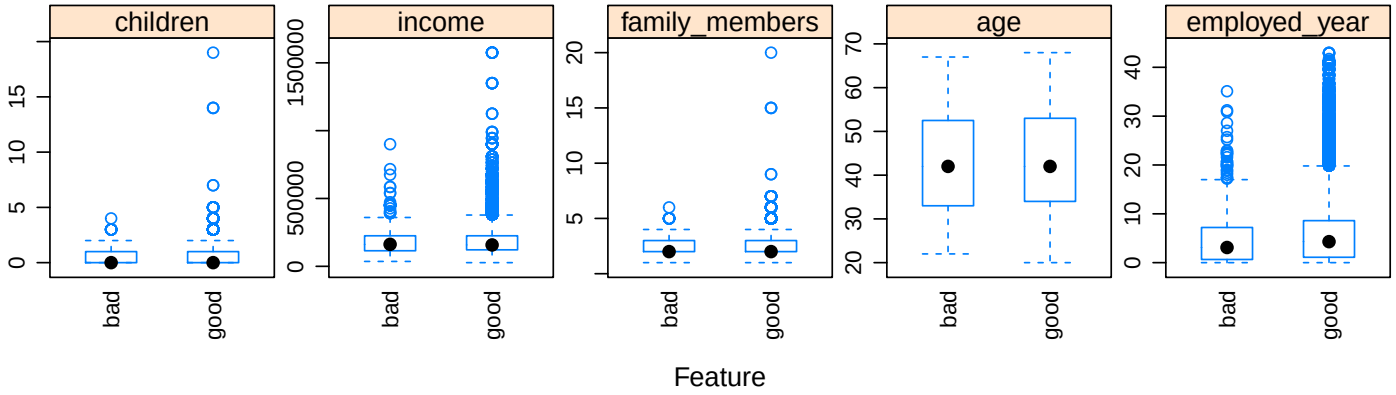
2. Data Description & Preliminary Data Analysis

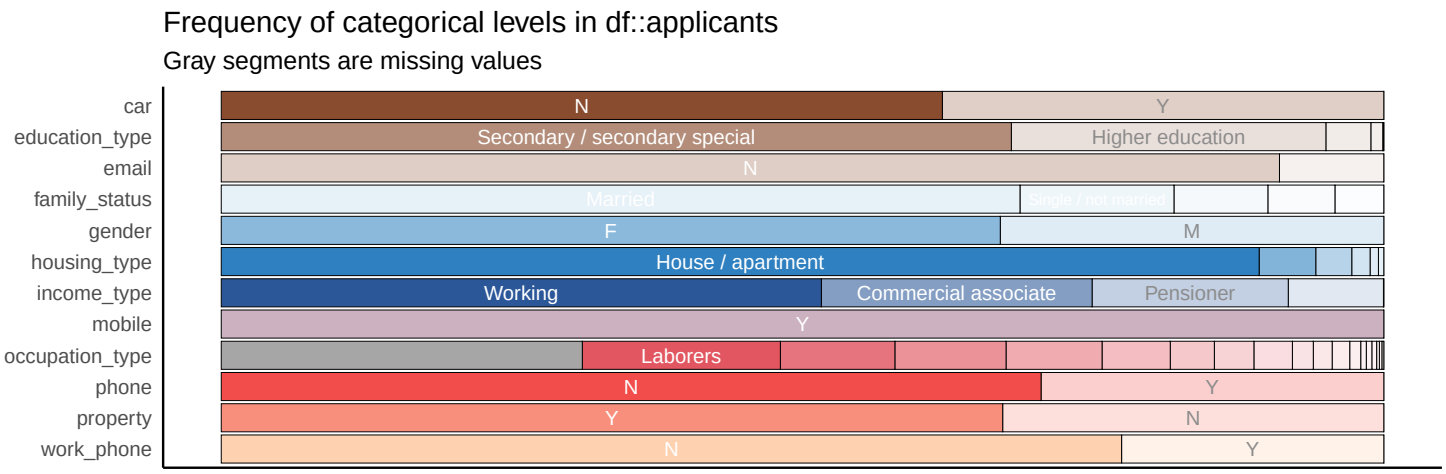
Our datasets are obtain from Kaggle. Two datasets are available for analysis, one is credit records, which contains applicants' monthly default status. The other is application records, which contains 17 features, indicating applicants' personal information.

The original dataset doesn't provide any labels indicating whether an applicant should be approved or not. so first we use the credit dataset to label each applicant. Referring to the explanation of personal credit record by Hong Kong Monetary Authority, each applicant who have a record of more than 60 days over due is labeled as bad applicant. After merging the two datasets, there're 616 bad applicants, which accounts for 1.69% of total applications.

Exploratory Data Analysis







We use histograms and boxplots to examine all 5 numeric features, and frequency plot to examine all 12 categorical features.

The following data preprocessing and feature engineering are performed with `recipes` package based on the above exploratory data analysis:

1. Use log transformation to solve data skewness of `employed_year` and `income`.
2. Remove the corresponding outliers for the two highly correlated features `children` and `family_members`.
3. Center and scale all numeric features.
4. Remove zero variance feature: `mobile`.
5. Impute missing value in `occupation_type`: for applicant who's `employed_year` is 0, impute "Unemployed", for the remaining NAs, impute "Others".
6. Dummy encoding for all factor features.

3. ML Methods, Analysis & Comparison

Subsampling Methods and Metrics Selection

First we use stratified sampling to split the data into 70% training data and 30% testing data. Since bad applicants only consist of about 1.7% of the whole dataset, the information available for identifying bad class is not enough to train the classification model. Also, using accuracy is not a good option to evaluate such classification models since the model will tend to classify nearly all applicants as good to achieve high accuracy.

To deal with imbalanced data issue, we compared several subsampling methods including undersampling, oversampling, synthetic data generation, and stratified sampling with data shrinkage.The last method is chosen to do stratified sampling using all features and shrink the data so that the ratio of good to bad is 20:1. For model evaluation metrics, we use 5-fold cross-validated F1-score for hyperparameter tuning and model selection, which imposes penalization for misclassifying the bad class or the good class.

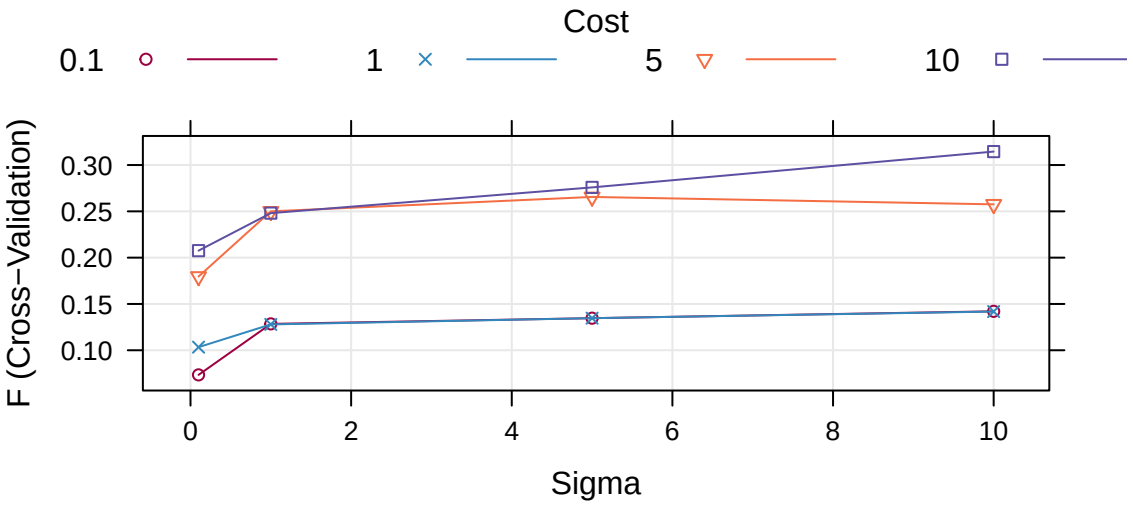
Logistic Regression

We use Logistic Regression as the baseline model. Since Logistic model has no hyperparameters, we simply fit a model using all predictors with 5-fold cross validation. The cross-validated F1-score is 6.92%. Particularly, the recall rate is extremly low, indicating Logistic model is doing very poorly to correctly identify the bad applicants.

Model	Precision	Recall	F
Logistic Regression	1	0.029	0.0692

SVM

There are two hyperparameters, gamma and cost, and many different kernels such as linear and radial in SVM model. We first fit a radial kernel SVM with these default hyperparameters: gamma = 1 and cost = 1. The cross-validated F1-score is 12.85%. Next, we continue using the radial kernel and tune both gamma and cost, which take the value of 0.1, 1, 5 and 10. The best hyperparameters are gamma = 10 and cost = 10, as shown in the hyperparameters tuning plot. After tuning hyperparameters, F1-score increased 18.6%, as shown in the table below.



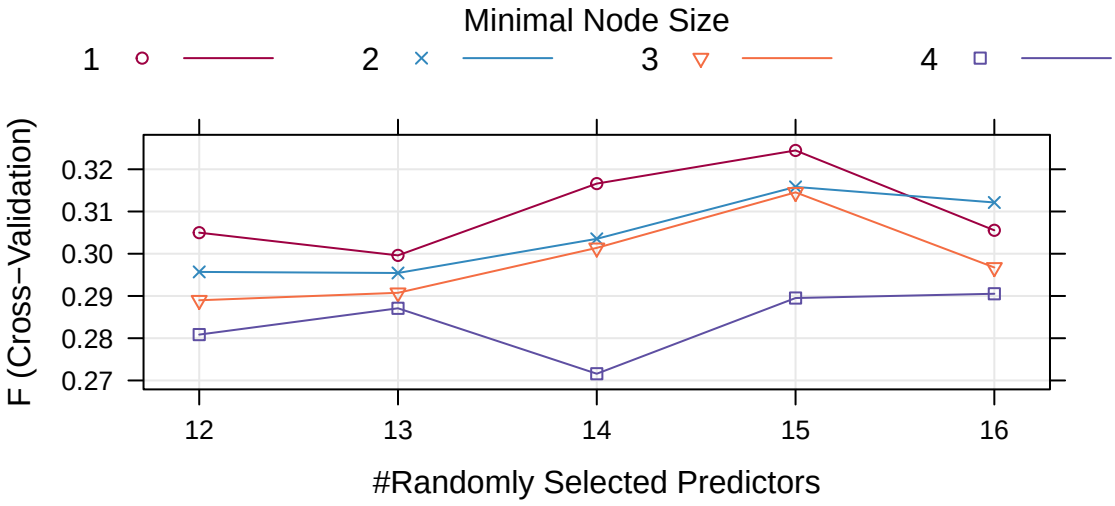
	gamma	cost	precision	recall	F
Default	1	1	0.6472	0.0716	0.1285
Tuning	10	10	0.5465	0.0899	0.3145

Random Forest

Random forest is based on applying bagging to decision trees. In addition to sampling the records, the algorithm also samples the variables, introducing randomness into both observations used in model and features used in model. The hyperparameters tuned are `mtry` (the number of randomly selected predictors) and `mini.node.size` (the minimum number of observations in one node). We apply `caret::train()` and `ranger` method for hyperparameters tuning with 5-fold cross-validation. The tuning steps are as follows:

- default model: To start with, we fit a default model with `mtry` of 4 ($\sqrt{nfeatures}$) and `mini.node.size` of 5, and get the cross-validated F1-socre of 0.0291. We use $nfeatures * 10$ as the number of trees used for the whole tuning process.
- `mtry` tuning: we first tune `mtry` ranging from 2 to 16 and from the top 10 models we find that a higher `mtry` value can achieve better F1-score. The best tune hyperparameter combination is `mtry` = 16 with `mini.node.size` = 1.
- `mini.node.size` tuning: next we fix `mtry` ranging from 8 to 16 and tune `mini.node.size` ranging from 1 to 10. From the top 10 models, model with smaller number of node size ranging from 1 to 3 tends to perform better. The best tune hyperparameter combination is `mtry` = 16 with `mini.node.size` = 2.
- final tuning: Lastly we tune both `mini.node.size` (1~4) and `mtry` (12~16). The best tune hyperparameter combination is `mtry` = 15 with `mini.node.size` = 1.

From the final tuning hyperparameters plot, we can find that with `mtry` ranging from 14 to 16 and `mini.node.size` of 1 or 2, the cross-validated F1-score will be higher. After tuning hyperparameters, cross-validated F1-score increased significantly from 2.91% to 32.45%.



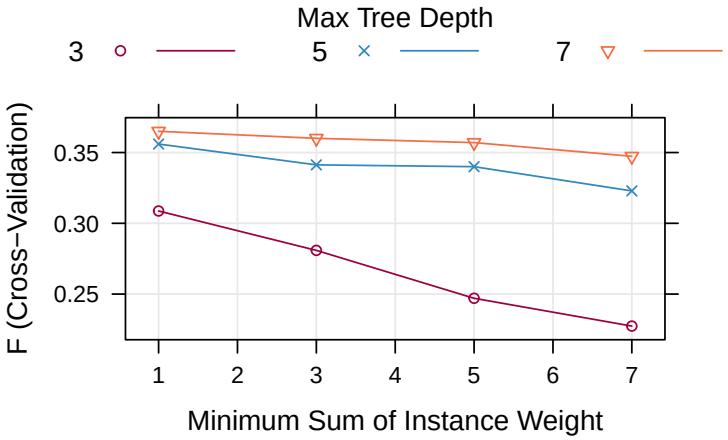
	mtry	min.node.size	Precision	Recall	F
Default	4	5	0.5417	0.0090	0.0291
Tuning	15	1	0.5327	0.2349	0.3245

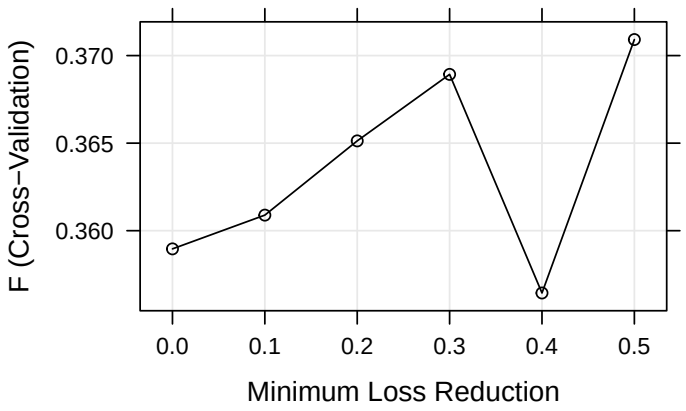
XGBoost

`caret::train()` is applied to train the XGBoost gradient boosted Tree model. Five of the available hyperparameters for XGBoost are selected using exhaustive grid search with 5-fold cross-validation. The optimal model is obtained by considering both model performance (cv-F1) and computational efficiency. The hyperparameters tuned are the number of trees, learning rate, the maximum tree depth, the minimum tree nodes and the regularization tuning hyperparameters `gamma`, other two hyperparameters `colsample_bytree` (the fraction of columns to be random samples for each tree) and `subsample` (the fraction of observations to be random samples for each tree) are both fixed at the default value of 0.8. The tuning strategy can be summarized as follows:

- default model: The hyperparameters of the default model are set to 1000, 0.1, 5, 1, and 0 for the value of the abovementioned five hyperparameters respectively.
- `eta`, `nrounds` tuning: The optimal model is selected by grid-search to determine the learning efficiency and the number of trees. The best tune hyperparameter combination is `nrounds` = 600 and `eta` = 0.3.
- `max_depth` and `min_child_weight` tuning: Next, the tree-specific parameters are adjusted with the obtained best `eta` and `nrounds` values. The best tune hyperparameter combination is `max_depth` = 7 and `min_child_weight` = 1.
- `gamma` tuning: Lastly the regularized hyperparameter `gamma` is tuned. The best `gamma` with the highest F1-score is 0.5.

The hyperparameters tuning plots also summarize the the last two tuning steps. The best hyperparameters combination and performance metrics are shown in the table below. Compared to the default model, the cross-validated F1-score increased about 4% for the model with the optimal hyperparameters.





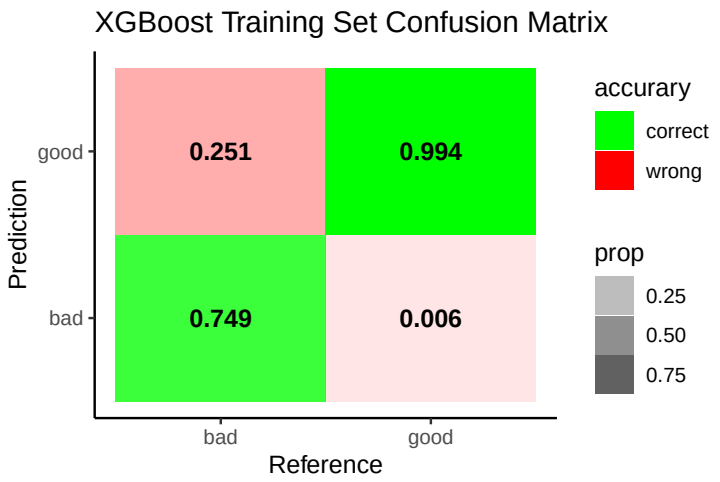
	eta	max_depth	min_child_weight	nrounds	gamma	Precision	Recall	F
Default	0.1	5	5	1000	0.0	0.54	0.24	0.33
Tuning	0.3	7	1	600	0.5	0.49	0.30	0.37

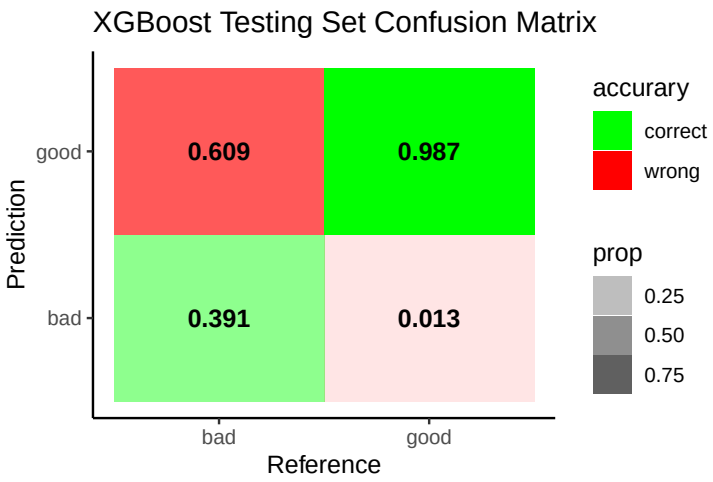
4. Conclusion

4.1 Model Selection and Evaluation

The table below summarizes the performance of the four models. Logistic regression performs 100% on precision rate but extremely low on the recall rate, indicating the model is unable to recognize the bad applicants, instead, it allocates all bad class as good class, which is not an ideal choice for the banks. Comparing the 5-fold cross-validated F1-scores of the four models, XGBoost well-balanced precision rate and recall rate so that performs the best on F1-score, hence it is selected to be the final model. Furthermore, XGBoost achieves an accuracy of 97.7% and an F1-score of 34.7% in the testing set.

Model	Precision	Recall	F
Logistic Regression	1.0000	0.0290	0.0692
SVM	0.5466	0.0899	0.3145
Random Forest	0.5327	0.2349	0.3245
XGBoost	0.4923	0.2997	0.3709

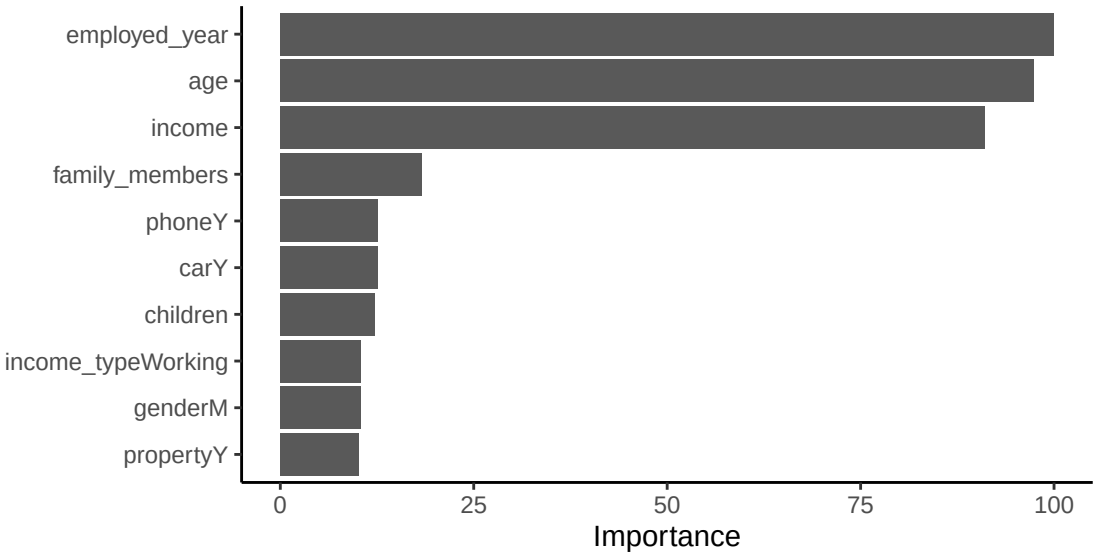




model	data	accuracy	precision	recall	F1
XGBoost	Testing Set	0.9773	0.3128	0.3905	0.3474

4.2 Business Application

From our analysis, XGBoost model achieved a classification accuracy of near 98% and performed better than baseline Logistic model with a 30% improvement on the F1-score. Based on the obtained results, the banks' risk control department can make use of it to better identify the potential uncreditworthy applicants to reduce default risk. Furthermore, the following feature importance plot suggests that when evaluating credit card applications, banks should pay more attention to identifying these three highly-ranking features: the number of employed years, age and income.



4.3 Limitation and Further Improvement

There exists some limitations for this analysis. Firstly, the dataset used in this project is relatively old with very few features that can be used for predictions nowadays. If more cutting-edging features of applicants can be obtained, the performance of the model will be enhanced and it will also be more appropriate for practical applications at the present. Besides, overall the models do not achieve attractive results of F1-score. Although the model performs well on accuracy rate, the F1-score should be considered a better option since it is more cost-saving for the bank to recognize the potential default clients and avoid approving credits to them. To deal with this problem, one of the improvements that can be made is to obtain more information or data on bad applicants' credit records. Furthermore, other subsampling methods such as smote and rose could be adopted to better structure the dataset in order to magnitude the imbalance problem.

5. References

1. Datasource: https://www.kaggle.com/rikdifos/credit-card-approval-prediction
2. Designing next-generation credit-decisioning models: https://www.mckinsey.com/business-functions/risk-and-resilience/our-insights/designing-next-generation-credit-decisioning-models
3. Flushing out the money launderers with better customer risk-rating models: https://www.mckinsey.com/business-functions/risk-and-resilience/our-insights/flushing-out-the-money-launderers-with-better-customer-risk-rating-models
4. Personal credit record: https://www.hkma.gov.hk/eng/smart-consumers/personal-credit/#personal-credit
5. Practical Guide to deal with Imbalanced Classification Problems in R: https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/
6. Peter Bruce(2020). Practical Statistics for Data Scientists(Second Edition)
7. Tom Fawcett(2016), Learning from Imbalanced Classes: https://www.svds.com/learning-imbalanced-classes/
8. Nitesh V. Chawla. "SMOTE: Synthetic Minority Over-sampling Technique", Journal of Artificial Intelligence Research 16 (2002):321–357.
9. Computing Classification Evaluation Metrics in R: https://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html
10. Dealing with Imbalanced data: https://rstudio-pubs-static.s3.amazonaws.com/607601_57a11284917f4d79933f4c4db3d41713.html
11. F1 Score vs ROC AUC vs Accuracy vs PR AUC: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc
12. Logistic Regression, Model Selection, and Cross Validation: http://www-personal.umich.edu/~gaozheng/teaching/stats414/LogisticReg/LogisticReg.html
13. Stratified Sampling On Dataset In R https://www.c-sharpcorner.com/article/how-to-perform-stratified-sampling-on-dataset-in-r/

Appendix

```
knitr::opts_chunk$set(echo = TRUE,fig.align='center', message = FALSE, warning = FALSE,eval = FALSE)

if(!require("pacman")){install.packages("pacman")}
pacman::p_load(dplyr, reshape2, data.table, scales, ggplot2, rsample, caret, recipes, vip, pROC, ROCR, xgboost

ggplot2::theme_set(ggplot2::theme_classic())
options(scipen = 999)
```

```
library(devtools)

# install and load the package from github - https://github.com/alastairrushworth/inspectdf
devtools::install_github("alastairrushworth/inspectdf")
library(inspectdf)
# https://www.littlemissdata.com/blog/inspectdf

# remote install fifer
library(remotes)
install_version("fifer", "1.0")
library(fifer)
```

```
credit <- read.csv('credit_record.csv')
application <- read.csv('application_record.csv')
credit$ID <- as.character(credit$ID)
application$ID <- as.character(application$ID)
```

```
credit[(credit$STATUS %in% c("X","C")),]$STATUS <- -1
credit$STATUS <- as.integer(credit$STATUS)
credit_record <- credit %>%
  group_by(ID) %>%
  dplyr::summarise(STATUS = max(STATUS))

# merge two datasets
# inner join
credit_df <- merge(
  x = credit_record,
  y = application,
  by = "ID"
)
perc = rep(NA, 6)
nobs = rep(NA, 6)
for (i in 0:5) {
  perc[i+1] <- nrow(credit_df[credit_df$STATUS == 5-i, ])/nrow(credit_df)
  nobs[i+1] <- nrow(credit_df[credit_df$STATUS == 5-i, ])
}
#data.frame(Status = c("past due >= 1 day", "past due >= 30 days", "over due >= 60 days", "over due >= 90 days
credit_df$LABEL <- with(credit_df, ifelse(STATUS >= 2, "bad", "good"))
```

```
# drop status after labeling
oldnames = colnames(credit_df)
newnames = c(
  'id',
  'status',
  'gender',
  'car',
  'property',
  'children',
```

```
  'income',
  'income_type',
  'education_type',
  'family_status',
  'housing_type',
  'days_birth',
  'days_employed',
  'mobile',
  'work_phone',
  'phone',
  'email',
  'occupation_type',
  'family_members',
  'label'
)
credit_df <- credit_df %>% rename_at(all_of(vars(oldnames)), ~ newnames)
```

```
# days birth: Count backwards from current day (0), -1 means yesterday
# days employed: Count backwards from current day(0). If positive, it means the person currently unemployed.
credit_df$age <- as.integer(-credit_df$days_birth / 365.25)
credit_df$employed_year <- with(credit_df, ifelse(days_employed < 0, round(-days_employed / 365.25,1), 0))
# delete these three columns
credit_df <- credit_df[,-which(names(credit_df) %in% c('id','status','days_birth','days_employed'))]
# missing value in occupation_type
credit_df[credit_df$occupation_type == "",]$occupation_type = NA
```

```
# convert dataframe to tibble, and leave out ID column which can not be used as feature
applicants <- as_tibble(credit_df)
# to make the dummy variable encoding consistent, erase original label encoding
applicants$mobile <- with(applicants, ifelse(mobile == 1, "Y", "N"))
applicants$work_phone <- with(applicants, ifelse(work_phone == 1, "Y", "N"))
applicants$phone <- with(applicants, ifelse(phone == 1, "Y", "N"))
applicants$email <- with(applicants, ifelse(email == 1, "Y", "N"))

# factor conversion
applicants[sapply(applicants, is.character)] <- lapply(applicants[sapply(applicants, is.character)], as.factor
applicants$family_members <- as.integer(applicants$family_members)
```

```
get_upper_tri<-function(cormat){
  cormat[lower.tri(cormat)] <- NA
  return(cormat)
}
# Correlation Plot
cormat <-
  applicants %>%
  select(children, income, family_members,age, employed_year) %>%
  cor()
cormat <- round(cormat, 2)
upper_tri <- get_upper_tri(cormat)
melted_cormat <- melt(upper_tri, na.rm = TRUE)
ggheatmap <-
  ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "purple", high = "blue", mid = "white", midpoint = 0, limit = c(-1,1), space = "L
  xlab("") +
  ylab("") +
  coord_fixed()

ggheatmap <-
```

```
  ggheatmap +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
  theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank(),
    legend.justification = c(1, 0),
    legend.position = c(0.5, 0.8),
    legend.direction = "horizontal")+
    guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                  title.position = "top", title.hjust = 0.5))
```

```
plot_box <- featurePlot(x = applicants[, c(4,5,15,17,18)],
            y = applicants$label,
            plot = "box",
            ## Pass in options to bwplot()
            scales = list(y = list(relation="free"),
                          x = list(rot = 90)),
            layout = c(5,1),
            auto.key = list(columns = 2))
plot_density<- featurePlot(x = applicants[, c(4,5,15,17,18)],
            y = applicants$label,
            plot = "density",
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            adjust = 1.5,
            #pch = "|",
            layout = c(5, 1),
            auto.key = list(columns = 2))
plot_frequency <- inspect_cat(applicants[,-16]) %>% show_plot(col_palette = 4)
```

```
applicants$occupation_type <- as.character(applicants$occupation_type)
applicants[is.na(applicants$occupation_type) & (applicants$employed_year == 0),]$occupation_type = 'Unemployed
applicants[is.na(applicants)] <- "Others"
# sum(is.na(applicants$occupation_type))
applicants$occupation_type <- as.factor(applicants$occupation_type)
```

```
occupation <- applicants %>% select(occupation_type)
occupation_freq <- occupation %>% count(occupation_type, sort=TRUE)
set.seed(10) # for reproducibility
p_wc <- wordcloud(words = occupation_freq$occupation_type, freq = occupation_freq$n, min.freq = 1, random.orde
```

```
ggarrange(plot_density, plot_box,plot_frequency, nrow = 3, heights = c(3,3,3))
```

```
# 1. stratified sampling with the rsample package
set.seed(123)
split <- initial_split(applicants, prop = 0.7, strata = "label")
applicants_train <- training(split)
applicants_test <- testing(split) # leave the testing data alone
#applicants_train <- subset(applicants_train, (children!=19 & children!=14 & children!=7))
```

```
library(fifer)
applicants_train_good <- applicants_train[applicants_train$label == "good",]
applicants_train_bad <- applicants_train[applicants_train$label == "bad",]
```

```r
applicants_good_sample <- fifer::stratified(applicants_train_good, c(
  'gender',
  'car',
  'property',
  'children',
  'income',
  'income_type',
  'education_type',
  'family_status',
  'housing_type',
  'age',
  'employed_year',
  'mobile',
  'work_phone',
  'phone',
  'email',
  'occupation_type',
  'family_members'),  0.38, seed = 123)

applicants_train <- rbind(applicants_good_sample, applicants_train_bad)
# remove outliers
applicants_train <- subset(applicants_train, (children!= 19 & children != 14  & children != 7))
```

```r
blueprint <- recipe(label ~ ., data = applicants_train) %>%
  step_nzv(all_predictors()) %>% # filter out zero or near-zero variance features
  step_log(income, base = 10) %>% # normalize to resolve numeric feature skewness
  step_log(employed_year, base = exp(1), offset = 1) %>% # normalize to resolve numeric feature skewness
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) # center and scale numeric features
# prepared
prepared <- prep(blueprint, training = applicants_train)
# baked
baked_train <- bake(prepared, new_data = applicants_train)
baked_test <- bake(prepared, new_data = applicants_test)
```

```r
cv <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = prSummary
)
```

```r
# using all predictors
set.seed(123)
fit_logit <- caret::train(
  label ~.,
  data = baked_train,
  method = 'glm',
  metric = "F",
  family = 'binomial',
  trControl = cv
)

logit_result <- round(fit_logit$results[,c(3,4,5)],4)
```

```r
logit_result <- data.frame(Model = "Logistic Regression", logit_result)
logit_result
```

```r
default_parameter_svm <- expand.grid(
  sigma = 1,
  C = 1)
# Train SVM model - svm_default
set.seed(123)
svm_default <- caret::train(
                  label ~.,
                  data = baked_train,
                  method = "svmRadial",
                  metric = "F",
                  trControl = cv,
                  verbose = FALSE,
                  tuneGrid = default_parameter_svm
                )
svm_default$results
```

```r
# Set tuning grid
hyper_grid_svm <- expand.grid(
  sigma = c(0.1,1,5,10),
  C = c(0.1,1,5,10))
# Tuning hyperparameters
set.seed(123)
svm_radial <- caret::train(
                  label ~.,
                  data = baked_train,
                  method = "svmRadial",
                  metric = "F",
                  trControl = cv,
                  verbose = FALSE,
                  tuneGrid = hyper_grid_svm
                )
svm_radial$bestTune
svm_radial$results %>%
  arrange(desc(F))%>%
  head(10)
svm_result <- round(arrange(svm_radial$results, desc(F))[1,c(4:6)],4)
```

```r
svm_result <- data.frame(Model = "SVM", svm_result)
```

```r
trellis.par.set(caretTheme())
hyperparameters.plot_svm <- plot(svm_radial)
```

```r
hyperparameters.plot_svm
```

```r
# summary the precision, recall, F1 of default and tuning svm model
svm_tune <- data.frame(
           gamma = c(1, 10),
           cost = c(1, 10),
           precision = c(0.6472, 0.5465),
           recall = c(0.0716, 0.0899),
           F = c(0.1285, 0.3145),
           row.names = c("Default", "Tuning"))
svm_tune
```

```r
n_features <- length(setdiff(names(baked_train), "label"))

tgrid <- expand.grid(
  .mtry = c(2, 4, 6, 8, 10), #start with 5 values from 2 to p
```

```
  .min.node.size = c(1, 5, 10), #start with 1, 5, 10
  .splitrule = "gini"
)
set.seed(123)
rf_fit_s <- caret::train(label ~ .,
                 data = baked_train,
                 trControl = cv,
                 tuneGrid = tgrid,
                 num.trees = n_features * 10,
                 metric = "F",
                 method = "ranger",
                 #importance = "permutation",
                 na.action = na.exclude)
```

```
rf_fit_s$bestTune
rf_fit_s$results %>%
  arrange(desc(F))%>%
  head(10)

rf_default <- arrange(rf_fit_s$results, desc(F))[10,]
rf_tune1 <- arrange(rf_fit_s$results, desc(F))[1,]
```

```
tgrid <- expand.grid(
  .mtry = c(4, 6, 8, 10, 12, 14, 16), #search the mtry range that covers 4 ~ 10
  .min.node.size = c(1, 5, 10), # without changing min.node.size
  .splitrule = "gini"
)
set.seed(123)
rf_fit_s1 <- caret::train(label ~ .,
                 data = baked_train,
                 trControl = cv,
                 tuneGrid = tgrid,
                 num.trees = n_features * 10,
                 metric = "F",
                 method = "ranger",
                 na.action = na.exclude)
```

```
rf_fit_s1$bestTune
rf_fit_s1$results %>%
  arrange(desc(F))

rf_tune2 <- arrange(rf_fit_s1$results, desc(F))[1,]
```

```
tgrid <- expand.grid(
  .mtry = c(8, 10, 12, 14, 16), #search the mtry range that covers 6 ~ 10
  .min.node.size = c(1, 3, 5, 7, 10), # add 3 and 7 among 1, 5 and 10
  .splitrule = "gini"
)
set.seed(123)
rf_fit_s2 <- caret::train(label ~ .,
                 data = baked_train,
                 trControl = cv,
                 tuneGrid = tgrid,
                 num.trees = n_features * 10,
                 metric = "F",
                 method = "ranger",
                 na.action = na.exclude)
```

```
rf_fit_s2$bestTune
rf_fit_s2$results %>%
  arrange(desc(F))

rf_tune3 <- arrange(rf_fit_s2$results, desc(F))[1,]
```

```
tgrid <- expand.grid(
  .mtry = c(10, 12, 14, 16), #search the mtry range that covers 8~14
  .min.node.size = c(1, 2, 3, 4, 5), # when min.node.size = 1, 3, 5, the F-score is high so put in 2 and 4
  .splitrule = "gini"
)
rf_fit_s3 <- caret::train(label ~ .,
                data = baked_train,
                trControl = cv,
                tuneGrid = tgrid,
                num.trees = n_features * 10,
                metric = "F",
                method = "ranger",
                na.action = na.exclude)
```

```
rf_fit_s3$bestTune
rf_fit_s3$results %>%
  arrange(desc(F))%>%
  head(10)

rf_tune4 <- arrange(rf_fit_s3$results, desc(F))[1,]
```

```
tgrid <- expand.grid(
  .mtry = c(12, 13, 14, 15, 16), #search the mtry range that covers 12~17
  .min.node.size = c(1, 2, 3, 4), #from 1 to 4
  .splitrule = "gini"
)
set.seed(123)
rf_fit_s4 <- caret::train(label ~ .,
                data = baked_train,
                trControl = cv,
                tuneGrid = tgrid,
                num.trees = n_features * 10,
                metric = "F",
                method = "ranger",
                importance = "permutation",
                na.action = na.exclude)
```

```
rf_fit_s4$bestTune
rf_fit_s4$results %>%
  arrange(desc(F))%>%
  head(10)

rf_tune5 <- arrange(rf_fit_s4$results, desc(F))[1,]
```

```
trellis.par.set(caretTheme())
hyperparameters.plot_rf <- plot(rf_fit_s4)
```

```
hyperparameters.plot_rf
```

```
rf_tune <- rbind(rf_default, rf_tune5)[, c(1,2,5,6,7)]
rf_tune <- rf_tune %>% mutate(across(!where(is.integer), round, digits=4))
```

```
row.names(rf_tune) <- c("Default", "Tuning")
rf_tune
```

```
rf_result <- round(rf_tune5[,c(5:7)],4)
rf_result <- data.frame(Model = "Random Forest", rf_result)
```

```
# defult model
turn_grid_xgb <- expand.grid(
  eta = 0.1,
  max_depth = 5,
  min_child_weight = 5,
  subsample = 0.8,
  colsample_bytree = 0.8,
  nrounds = 1000,
  gamma = 0)

set.seed(123)
xgb_default <- train(label~., data = baked_train,
              method = "xgbTree",
              tuneGrid = turn_grid_xgb,
              trControl = cv,
              verbose = FALSE,
              metric = "F")

xgboost_default <- arrange(xgb_default$results, desc(F))[1,c(1:3, 6:7, 9:11)]
```

```
#  Tune Tree number and learning rate
turn_grid_xgb <- expand.grid(
  eta = c(0.1, 0.3, 0.5),
  max_depth = 5,
  min_child_weight = 1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  nrounds = (1:10)*200,
  gamma = 0)

set.seed(123)
xgb_1 <- train(label~., data = baked_train,
              method = "xgbTree",
              tuneGrid = turn_grid_xgb,
              trControl = cv,
              verbose = FALSE,
              metric = "F")
```

```
# Tune min depth and max nodes
turn_grid_xgb <- expand.grid(
  eta = 0.3,
  max_depth = c(3,5,7),
  min_child_weight = c(1,3,5,7),
  subsample = 0.8,
  colsample_bytree = 0.8,
  nrounds = 600,
  gamma = 0)

set.seed(123)
xgb_2 <- train(label~., data = baked_train,
              method = "xgbTree",
              tuneGrid = turn_grid_xgb,
              trControl = cv,
```

```
              verbose = FALSE,
              metric = "F")

#xgb_2$results %>%
  #arrange(desc(F))
```

```
# gamma
turn_grid_xgb <- expand.grid(
  eta = 0.3,
  max_depth = 7,
  min_child_weight = 1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  nrounds = 600,
  gamma = (0:5)*0.1)

set.seed(123)
xgb_final <- train(label~., data = baked_train,
              method = "xgbTree",
              tuneGrid = turn_grid_xgb,
              trControl = cv,
              verbose = FALSE,
              metric = "F")

xgboost_tune <- arrange(xgb_final$results, desc(F))[1,c(1,2, 5, 7, 3, 9:11)]
```

```
trellis.par.set(caretTheme())
xgb_final_plot <- plot(xgb_final)
xgb_2_plot <- plot(xgb_2)
grid.arrange(xgb_2_plot, xgb_final_plot, ncol = 2)
```

```
xgb_tune <- rbind(xgboost_default, xgboost_tune)
row.names(xgb_tune) <- c("Default", "Tuning")
xgb_tune <- round(xgb_tune, 2)
xgb_result <- data.frame(Model = "XGBoost", round(xgboost_tune[6:8],4))
xgb_tune
```

```
# predict class on training/testing set
pred_class_train <- predict(xgb_final, baked_train)
pred_class_test <- predict(xgb_final, baked_test)

# create confusion matrix for train
cm_train <-
  confusionMatrix(
    data = pred_class_train,
    reference = baked_train$label
  )

# create confusion matrix for test
cm_test <-
  confusionMatrix(
    data = pred_class_test,
    reference = baked_test$label
  )

# create confusion matrix plot for train
table <- data.frame(cm_train$table)
plotTable <- table %>%
```

```
  mutate(accurary = ifelse(table$Prediction == table$Reference, "correct", "wrong")) %>%
  group_by(Reference) %>%
  dplyr::mutate(prop = Freq/sum(Freq))

p_cm_train <-
  ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = accurary, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = round(prop,3)), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(correct = "green", wrong = "red")) +
  ggtitle(labs(title = "XGBoost Training Set Confusion Matrix"))

# create confusion matrix plot for test
table <- data.frame(cm_test$table)
plotTable <- table %>%
  mutate(accurary = ifelse(table$Prediction == table$Reference, "correct", "wrong")) %>%
  group_by(Reference) %>%
  dplyr::mutate(prop = Freq/sum(Freq))

p_cm_test <-
  ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = accurary, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = round(prop,3)), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(correct = "green", wrong = "red")) +
  ggtitle(labs(title = "XGBoost Testing Set Confusion Matrix"))
```

```
rbind(logit_result, svm_result, rf_result, xgb_result)
```

```
grid.arrange(p_cm_train, p_cm_test, ncol = 2)
```

```
model <- "XGBoost"
data <- "Testing Set"
accuracy <- sum(pred_class_test == baked_test$label)/length(pred_class_test )
specificity <- specificity(pred_class_test , baked_test$label, positive="bad")
precision <- posPredValue(pred_class_test , baked_test$label, positive="bad")
recall <- sensitivity(pred_class_test ,baked_test$label, positive="bad")
F1 <- (2 * precision * recall) / (precision + recall)
final_result <- round(data.frame(accuracy, precision, recall, F1),4)
cbind(model, data, final_result)
```

```
# vip plot
vip(xgb_final ,num_features = 10, scale = TRUE)
```