HomeCreditModeling

Alexia Wells

2024-10-29

Read Me

If you would like to run the contents of this document on your own, please remove eval=FALSE in the modeling code chunks. They are set to false because these models take a long time computationally. Thank you.

Prepare the Data

Loading Libraries and Data

```
# Load libraries
library(tidyverse)
library(dplyr)
library(DataExplorer)
library(ggplot2)
library(caret)
library(skimr)
library(tidymodels)
library(patchwork)
library(missForest)
library(kableExtra)
library(ROSE)
library(car)
library(ggcorrplot)
library(discrim) # Library for naive
library(bonsai) # Boosted trees + bart
library(lightgbm) # Boosted trees + bart
library(stacks)
library(data.table)
library(dbarts)
library(rpart)

# Read input data
train <- vroom::vroom("FivePercBalancedTrain.csv")
test <- vroom::vroom("ImputedTest.csv")
previous_app_file <- vroom::vroom('previous_application.csv')

# Factor the variables
train <- train |>
  mutate(across(c(TARGET, NAME_CONTRACT_TYPE, CODE_GENDER, FLAG_OWN_CAR,
                  FLAG_OWN_REALTY, NAME_TYPE_SUITE,NAME_INCOME_TYPE,
                  NAME_EDUCATION_TYPE,NAME_FAMILY_STATUS,NAME_HOUSING_TYPE,
                  FLAG_EMP_PHONE, FLAG_WORK_PHONE, FLAG_PHONE,FLAG_EMAIL,
                  OCCUPATION_TYPE, CNT_FAM_MEMBERS,
                  REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY,
                  WEEKDAY_APPR_PROCESS_START, HOUR_APPR_PROCESS_START,
                  REG_REGION_NOT_WORK_REGION,REG_CITY_NOT_LIVE_CITY,
                  REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY,
                  ORGANIZATION_TYPE,
                  WALLSMATERIAL_MODE, OBS_30_CNT_SOCIAL_CIRCLE,
                  OBS_60_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE,
                  DEF_60_CNT_SOCIAL_CIRCLE, FLAG_DOCUMENT_3, FLAG_DOCUMENT_6,
                  FLAG_DOCUMENT_8, OWN_CAR_AGE_BIN, HOUSING_INFO), as.factor))

# Do the same for test
test <- test |>
  mutate(across(c(NAME_CONTRACT_TYPE, CODE_GENDER, FLAG_OWN_CAR,
                  FLAG_OWN_REALTY, NAME_TYPE_SUITE,NAME_INCOME_TYPE,
                  NAME_EDUCATION_TYPE,NAME_FAMILY_STATUS,NAME_HOUSING_TYPE,
                  FLAG_EMP_PHONE, FLAG_WORK_PHONE, FLAG_PHONE,FLAG_EMAIL,
```

```
                  OCCUPATION_TYPE, CNT_FAM_MEMBERS,
                  REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY,
                  WEEKDAY_APPR_PROCESS_START, HOUR_APPR_PROCESS_START,
                  REG_REGION_NOT_WORK_REGION,REG_CITY_NOT_LIVE_CITY,
                  REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY, ORGANIZATION_TYPE,
                  WALLSMATERIAL_MODE, OBS_30_CNT_SOCIAL_CIRCLE,
                  OBS_60_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE,
                  DEF_60_CNT_SOCIAL_CIRCLE, FLAG_DOCUMENT_3, FLAG_DOCUMENT_6,
                  FLAG_DOCUMENT_8, OWN_CAR_AGE_BIN, HOUSING_INFO),
                as.factor))

# Forgot to make
train2 <- train |>
  dplyr::select(-CNT_CHILDREN_BIN)
```

Balancing the Data

```
# Loading balanced train
balanced_train <- vroom::vroom("BalancedTrain.csv")

balanced_train <- balanced_train |>
  mutate(across(c(TARGET, NAME_CONTRACT_TYPE, CODE_GENDER, FLAG_OWN_CAR,
                  FLAG_OWN_REALTY, NAME_TYPE_SUITE,NAME_INCOME_TYPE,
                  NAME_EDUCATION_TYPE,NAME_FAMILY_STATUS,NAME_HOUSING_TYPE,
                  FLAG_EMP_PHONE, FLAG_WORK_PHONE, FLAG_PHONE,FLAG_EMAIL,
                  OCCUPATION_TYPE, CNT_FAM_MEMBERS,
                  REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY,
                  WEEKDAY_APPR_PROCESS_START, HOUR_APPR_PROCESS_START,
                  REG_REGION_NOT_WORK_REGION,REG_CITY_NOT_LIVE_CITY,
                  REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY, ORGANIZATION_TYPE,
                  WALLSMATERIAL_MODE, OBS_30_CNT_SOCIAL_CIRCLE,
                  OBS_60_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE,
                  DEF_60_CNT_SOCIAL_CIRCLE, FLAG_DOCUMENT_3, FLAG_DOCUMENT_6,
                  FLAG_DOCUMENT_8, OWN_CAR_AGE_BIN, HOUSING_INFO, CNT_CHILDREN_BIN),
                as.factor)) |>
  dplyr::select(-CNT_CHILDREN_BIN)
```

Create a control grid so that models can be stacked if desired

```
untunedModel <- control_stack_grid() #If tuning over a grid
```

Feature Engineering

```
set.seed(123)

# Certain features do not work with naive bayes, lets remove them
naive_data <- train2 |>
  dplyr::select(-c(NAME_CONTRACT_TYPE, CODE_GENDER, FLAG_OWN_CAR, FLAG_OWN_REALTY,
                   NAME_TYPE_SUITE, NAME_INCOME_TYPE, NAME_EDUCATION_TYPE, NAME_FAMILY_S
TATUS,
                   NAME_HOUSING_TYPE, FLAG_EMP_PHONE, FLAG_WORK_PHONE, FLAG_PHONE,
                   FLAG_EMAIL, OCCUPATION_TYPE, CNT_FAM_MEMBERS, REGION_RATING_CLIENT,
                   REGION_RATING_CLIENT_W_CITY, WEEKDAY_APPR_PROCESS_START,
                   HOUR_APPR_PROCESS_START,REG_REGION_NOT_WORK_REGION,
                   REG_CITY_NOT_LIVE_CITY, REG_CITY_NOT_WORK_CITY,LIVE_CITY_NOT_WORK_CIT
Y,
                   ORGANIZATION_TYPE, WALLSMATERIAL_MODE, OBS_30_CNT_SOCIAL_CIRCLE,
                   DEF_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCLE,
                   DEF_60_CNT_SOCIAL_CIRCLE, FLAG_DOCUMENT_3, FLAG_DOCUMENT_6,
                   FLAG_DOCUMENT_8,  OWN_CAR_AGE_BIN, HOUSING_INFO))

# Drop rows with missing 'CNT_PAYMENT' + convert to numeric
previous_app_file <- previous_app_file |>
  na.omit(CNT_PAYMENT) |>
  mutate(CNT_PAYMENT = as.numeric(CNT_PAYMENT))

# Adding all FE to train and test
train2 <- train2 %>%
  mutate(screwratio1 = (AMT_CREDIT - AMT_GOODS_PRICE) / AMT_GOODS_PRICE,
         screwratio2 = (AMT_CREDIT - AMT_GOODS_PRICE) / AMT_CREDIT,
         saint_CNT = AMT_CREDIT / AMT_ANNUITY,
         angel_CNT = AMT_GOODS_PRICE / AMT_ANNUITY,
         simple_diff = AMT_CREDIT - AMT_GOODS_PRICE,
         credit_to_annuity_ratio = AMT_CREDIT / AMT_ANNUITY,
         credit_to_goods_ratio = AMT_CREDIT / AMT_GOODS_PRICE,
         credit_to_income_ratio = AMT_CREDIT / AMT_INCOME_TOTAL,
         income_credit_percentage = AMT_INCOME_TOTAL / AMT_CREDIT,
         income_per_person = AMT_INCOME_TOTAL / as.numeric(CNT_FAM_MEMBERS),
         payment_rate = AMT_ANNUITY / AMT_CREDIT,
         phone_to_birth_ratio = DAYS_LAST_PHONE_CHANGE / DAYS_BIRTH)

test <- test %>%
  mutate(screwratio1 = (AMT_CREDIT - AMT_GOODS_PRICE) / AMT_GOODS_PRICE,
         screwratio2 = (AMT_CREDIT - AMT_GOODS_PRICE) / AMT_CREDIT,
         saint_CNT = AMT_CREDIT / AMT_ANNUITY,
         angel_CNT = AMT_GOODS_PRICE / AMT_ANNUITY,
         simple_diff = AMT_CREDIT - AMT_GOODS_PRICE,
         credit_to_annuity_ratio = AMT_CREDIT / AMT_ANNUITY,
         credit_to_goods_ratio = AMT_CREDIT / AMT_GOODS_PRICE,
         credit_to_income_ratio = AMT_CREDIT / AMT_INCOME_TOTAL,
         income_credit_percentage = AMT_INCOME_TOTAL / AMT_CREDIT,
         income_per_person = AMT_INCOME_TOTAL / as.numeric(CNT_FAM_MEMBERS),
         payment_rate = AMT_ANNUITY / AMT_CREDIT,
```

```
         phone_to_birth_ratio = DAYS_LAST_PHONE_CHANGE / DAYS_BIRTH)

# No nas, which is good
any(is.na(train2))
```

```
## [1] FALSE
```

```
any(is.na(test))
```

```
## [1] FALSE
```

In Sample Model Performance

Create Train and Validation Set

```
set.seed(123)  # for reproducibility
data_split <- initial_split(train2, prop = 0.7, strata = TARGET)  # Stratify if your tar
get is imbalanced
train_set <- training(data_split)
test_set <- testing(data_split)
```

Check Cross Validation for Best Performing, Individual Models

I believe the differences I am seeing between the in sample and out of sample methods are negligible. Overall, I would argue the results have been relatively similar. Below, I will include the roc_auc and accuracy for the in sample vs. the kaggle result.

Please note that my best performing model was actually an ensemble method of random forest, bart, logistic regression, and extra trees. This model gave me a kaggle score of 0.70311.

BART

In sample, we get roc_auc of 0.693 and an accuracy of 0.647. For comparison, the kaggle result was 0.70113.

```
my_recipe <- recipe(TARGET ~ ., data = train_set) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%  # One-hot encode nominal var
iables
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Make sure it is taking dbarts from tidymodels
tidymodels_prefer()

bart_model <- bart(trees=tune()) %>% # BART figures out depth and learn_rate
set_engine("dbarts") %>% # might need to install
set_mode("classification")

# Create workflow for the BART model
bart_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(bart_model) # Use the model structure for BART, but fit it directly later

# Create 10-fold cross-validation
cv_folds <- vfold_cv(train_set, v = 10, strata = TARGET)
grid_of_tuning_params <- grid_regular(trees(), levels = 5)

# Fit with cross-validation
cv_results <- bart_wf %>%
  tune_grid(
    resamples = cv_folds,  # Cross-validation folds
    grid = grid_of_tuning_params, # Grid of tuning parameters
    metrics = metric_set(roc_auc, accuracy), # Relevant metrics
    control = control_grid(save_pred = TRUE) # Use control_grid for tuning
  )

# Select Best Parameter
best_params <- select_best(cv_results, metric = "accuracy")

# Finalize workflow
final_wf <- bart_wf %>%
  finalize_workflow(best_params)

final_fit <- final_wf %>%
  last_fit(data_split)

# Collect metrics to evaluate test performance
final_metrics <- collect_metrics(final_fit)
final_metrics[,1:3]
```

Logistic Regression

In sample, we get roc_auc of 0.691 and an accuracy of 0.634. For comparison, the kaggle result was 0.67896.

```
# Recipe for logistic model
my_recipe <- recipe(TARGET ~ ., data = train_set) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%  # One-hot encode nominal var
iables
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Logistic regression model
log_model <- logistic_reg(mixture=tune(), penalty=tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

log_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(log_model)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(mixture(), penalty(), levels = 5) ## L^2 total tun
ing possibilities

# Fit with cross-validation
cv_results <- log_wf %>%
  tune_grid(
    resamples = cv_folds,  # Cross-validation folds
    grid = grid_of_tuning_params, # Grid of tuning parameters
    metrics = metric_set(roc_auc, accuracy), # Relevant metrics
    control = control_grid(save_pred = TRUE) # Use control_grid for tuning
  )

# Select Best Parameter
best_params <- select_best(cv_results, metric = "accuracy")

# Finalize workflow
final_wf <- log_wf %>%
  finalize_workflow(best_params)

final_fit <- final_wf %>%
  last_fit(data_split)

# Collect metrics to evaluate test performance
final_metrics <- collect_metrics(final_fit)
final_metrics[,1:3]
```

Random Forest

In sample, we get roc_auc of 0.692 and an accuracy of 0.647. For comparison, the kaggle result was 0.69877.

```
# Recipe
my_recipe <- recipe(TARGET ~ ., data = train_set) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

rf_model <- rand_forest(min_n=tune(), trees=tune()) %>%
  set_engine("ranger") %>% # What R function to use7
  set_mode("classification")

rf_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(rf_model)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(trees(), min_n(), levels = 5)


# Fit with cross-validation
cv_results <- rf_wf %>%
  tune_grid(
    resamples = cv_folds,  # Cross-validation folds
    grid = grid_of_tuning_params, # Grid of tuning parameters
    metrics = metric_set(roc_auc, accuracy), # Relevant metrics
    control = control_grid(save_pred = TRUE) # Use control_grid for tuning
  )

# Select Best Parameter
best_params <- select_best(cv_results, metric = "accuracy")

# Finalize workflow
final_wf <- rf_wf %>%
  finalize_workflow(best_params)

final_fit <- final_wf %>%
  last_fit(data_split)

# Collect metrics to evaluate test performance
final_metrics <- collect_metrics(final_fit)
final_metrics[,1:3]
```

Extra Trees

In sample, we get roc_auc of 0.692 and an accuracy of 0.650. For comparison, the kaggle result was 0.69339.

```
extra_trees_model <- rand_forest(trees = 1000, min_n = tune(), mode = "classification")
%>%
  set_engine("ranger", splitrule = "extratrees") %>%
  set_mode("classification")

extra_trees_wf <- workflow() %>%
  add_model(extra_trees_model) %>%
  add_recipe(my_recipe)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(min_n(), levels = 5)

# Fit with cross-validation
cv_results <- extra_trees_wf %>%
  tune_grid(
    resamples = cv_folds,  # Cross-validation folds
    grid = grid_of_tuning_params, # Grid of tuning parameters
    metrics = metric_set(roc_auc, accuracy), # Relevant metrics
    control = control_grid(save_pred = TRUE) # Use control_grid for tuning
  )

# Select Best Parameter
best_params <- select_best(cv_results, metric = "accuracy")

# Finalize workflow
final_wf <- extra_trees_wf %>%
  finalize_workflow(best_params)

final_fit <- final_wf %>%
  last_fit(data_split)

# Collect metrics to evaluate test performance
final_metrics <- collect_metrics(final_fit)
final_metrics
```

Out of Sample Model Performance / Kaggle Approach

Naive Bayes

The best result for this model was 0.60757 with auc instead of roc_auc.

```
#conflicted::conflicts_prefer(yardstick::accuracy)

# Naive Bayes Recipe
my_recipe <- recipe(TARGET ~ . , data=naive_data) %>%
  step_novel(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Create model
nb_model <- naive_Bayes(Laplace=tune(), smoothness=tune()) %>%
set_mode("classification") %>%
set_engine("naivebayes")

# Create workflow
nb_wf <- workflow() %>%
add_recipe(my_recipe) %>%
add_model(nb_model)

## Tune smoothness and Laplace

# Grid of values to tune over
grid_of_tuning_params <- grid_regular(Laplace(), smoothness(), levels = 5)

# Split data for CV (5-10 groups)
folds <- vfold_cv(naive_data, v=5, repeats = 1)

# Run the CV
nb_CV_results <- nb_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

# Find best tuning parameters
bestTune <- nb_CV_results %>%
  select_best(metric = "roc_auc")

# Finalize the workflow and fit it
final_wf <- nb_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=naive_data)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)
```

```
# Write out file
vroom::vroom_write(x=kaggle_submission, file="./NaiveBayesPreds.csv", delim=",")
```

LightGBM Boosted trees

The best result for this model was 0.68676.

```
# Final recipe
my_recipe <- recipe(TARGET ~ . , data=train2) %>%
  step_novel(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Boosted Trees
boost_model <- boost_tree(tree_depth=tune(), trees=tune(), learn_rate=tune()) %>%
set_engine("lightgbm") %>% #or "xgboost" but lightgbm is faster
set_mode("classification")

# Create workflow for the boosted tree model
boost_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(boost_model)

# Split data for CV (5-10 groups), redfining seperate from the naive above
folds <- vfold_cv(train2, v=5, repeats = 1)

# Grid of values to tune over
grid_of_tuning_params <- grid_regular(tree_depth(), trees(), learn_rate(), levels = 5)

# Run the CV
boost_CV_results <- boost_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

# Find best tuning parameters
bestTune <- boost_CV_results %>%
  select_best(metric = "roc_auc")

# Finalize the workflow and fit it
final_wf <- boost_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=train2)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")
predictions
any(is.na(predictions))

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./BoostedPreds.csv", delim=",")
```

BART (Bayesian Additive Regression Trees)

The best result for this model was 0.70113. This was the best performing individual model.

```
my_recipe <- recipe(TARGET ~ ., data = train2) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%  # One-hot encode nominal var
iables
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Make sure it is taking dbarts from tidymodels
tidymodels_prefer()

bart_model <- bart(trees=tune()) %>% # BART figures out depth and learn_rate
set_engine("dbarts") %>% # might need to install
set_mode("classification")

# Create workflow for the BART model
bart_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(bart_model) # Use the model structure for BART, but fit it directly later

# Grid of values to tune over
grid_of_tuning_params <- grid_regular(trees(), levels = 5)

# Run the CV
bart_CV_results <- bart_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

# Find best tuning parameters
bestTune <- bart_CV_results %>%
  select_best(metric = "roc_auc")

# Finalize the workflow and fit it
final_wf <- bart_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=train2)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")
predictions

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./BartPreds.csv", delim=",")
```

Logistic Regression

The best result for this model was 0.67896.

```
# Recipe for logistic model
my_recipe <- recipe(TARGET ~ ., data = train2) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%  # One-hot encode nominal var
iables
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Logistic regression model
log_model <- logistic_reg(mixture=tune(), penalty=tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

log_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(log_model)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(mixture(), penalty(), levels = 5) ## L^2 total tun
ing possibilities

## Run the CV
log_CV_results <- log_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

## Find Best Tuning Parameters
log_bestTune <- log_CV_results %>%
  select_best(metric = "roc_auc")

## Finalize the Workflow & fit it
final_wf <-log_wf %>%
  finalize_workflow(log_bestTune) %>%
  fit(data=train2)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")
predictions
any(is.na(predictions))

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./LogisticPreds.csv", delim=",")
```

Random Forest

The best result for this model was 0.69877.

```
# Recipe
my_recipe <- recipe(TARGET ~ ., data = train2) %>%
  step_unknown(all_nominal_predictors()) |>
  step_novel(all_nominal_predictors()) %>%    # Handle unseen levels in test data
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

rf_model <- rand_forest(min_n=tune(), trees=tune()) %>%
  set_engine("ranger") %>% # What R function to use7
  set_mode("classification")

rf_wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(rf_model)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(trees(), min_n(), levels = 5)

## Run the CV
rf_CV_results <- rf_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

## Find Best Tuning Parameters
bestTune <- rf_CV_results %>%
  select_best(metric = "roc_auc")

## Finalize the Workflow & fit it
final_wf <-rf_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=train2)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./RandomForestPreds.csv", delim=",")
```

Extra Trees

The best result for this model was 0.69339.

```
extra_trees_model <- rand_forest(trees = 1000, min_n = tune(), mode = "classification")
%>%
  set_engine("ranger", splitrule = "extratrees") %>%
  set_mode("classification")

extra_trees_wf <- workflow() %>%
  add_model(extra_trees_model) %>%
  add_recipe(my_recipe)

## Grid of values to tune over
grid_of_tuning_params <- grid_regular(min_n(), levels = 5)

## Run the CV
extra_trees_CV_results <- extra_trees_wf %>%
  tune_grid(resamples=folds,
            grid=grid_of_tuning_params,
            metrics=metric_set(accuracy, roc_auc),
            control = untunedModel)

## Find Best Tuning Parameters
bestTune <- extra_trees_CV_results %>%
  select_best(metric = "roc_auc")

## Finalize the Workflow & fit it
final_wf <-extra_trees_wf %>%
  finalize_workflow(bestTune) %>%
  fit(data=train2)

## Make predictions
predictions <- predict(final_wf, new_data=test, type="prob")

# Prepare data for kaggle submission
kaggle_submission <- predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./ExtraTreesPreds.csv", delim=",")
```

Stacked Predictions

Results for All Ensemble Methods

- LightGBM + BART + Logistic Regression: 0.69169
- Random Forest, Logistic Regression, and BART: 0.70050

- LightGBM + Random Forest + Extra Trees + Logistic Regression: 0.69765
- Random Forest + BART + Logistic Regression + Extra Trees: 0.70311 best stacked so far

Code for All Ensemble Methods

```
# Specify with models to include, switched to roc_auc because I got a warning about that

# 3rd Place approach from Kaggle - LightGBM, Random Forest, Extra Trees, logistic regres
sion
# Not as good as the final stacked model I went with
my_stack <- stacks() %>%
  add_candidates(boost_CV_results) %>%
  add_candidates(rf_CV_results) %>%
  add_candidates(extra_trees_CV_results) %>%
  add_candidates(log_CV_results)

# Original Method, not as good as others below
my_stack <- stacks() %>%
  add_candidates(boost_CV_results) %>%
  add_candidates(bart_CV_results) %>%
  add_candidates(log_CV_results)

# Second best performing
my_stack <- stacks() %>%
  add_candidates(rf_CV_results) %>%
  add_candidates(bart_CV_results) %>%
  add_candidates(log_CV_results)

# Best performing
my_stack <- stacks() %>%
  add_candidates(rf_CV_results) %>%
  add_candidates(bart_CV_results) %>%
  add_candidates(log_CV_results) %>%
  add_candidates(extra_trees_CV_results)

## Fit the stacked model
stack_mod <- my_stack %>%
  blend_predictions() %>% # LASSO penalized regression meta-learner
  fit_members() ## Fit the members to the dataset

## Use the stacked data to get a prediction
stacked_predictions <- stack_mod %>%
  predict(new_data=test, type = "prob")

# Prepare data for kaggle submission
kaggle_submission <- stacked_predictions %>%
  dplyr::bind_cols(., test) %>%
  dplyr::select(SK_ID_CURR, .pred_1) %>%
  rename(TARGET=.pred_1)

# Write out file
vroom::vroom_write(x=kaggle_submission, file="./StackedPreds.csv", delim=",")
```

Results

Cleaning Data Approach

For cleaning the data, I used the previous work that was done in my EDA, which I will summarize. To start off, I removed any predictors with near-zero variance or zero variance. From there, there were only 86 columns left. This was nice because it makes the data slightly more approachable. Next, I imputed the data. I found that missForest took too long to run, instead for numeric variables, I decided to impute using the mean. For categorical/factor variables I used mode. This was a good solution because it was much faster. Since I imputed the train data, the best practice is to do the same to the test. I had to make sure that all the levels in the data line up too.

Balance Data Approach

As for balancing the data, this was also done in my EDA. Considering that the data was highly imbalanced my approach was to undersample using the ROSE library. For reference, downsampling randomly selects and removes observations from the majority class until it is the same size as the minority class. This was useful considering the majority class had far more observations. In the future, it would be interesting to experiment with upsampling and see if that makes any improvements to model performance.

Once the data was balanced, I created several datasets for my teammates to use. There was an imputed test.csv, an imputed and balanced train.csv, and lastly a data frame that contained only 5% of the imputed and balanced train.csv. The purpose of the 5% data frame was to help speed up model building by making the data more digestible.

Feature Engineering Approach

Columns Previously Created by Me

What made the models drastically more successful was adding feature engineering. There were two variables that I feature engineered during my EDA process. These were OWN_CAR_AGE_BIN and HOUSING_INFO, both binning variables.

The original OWN_CAR_AGE variable represented the age of a client's car, a numerical variable. This column has 200912 missing values. There was informative missingness which was why I transformed this variable to a factor. The completed variable had 4 categories - Missing, Low, Medium, and High. It was difficult to decided what ages to consider low, medium, and high. As of 2024, S&P Global states that most vehicles are 12.6 years old on average. Google suggested that a "sweet spot" for cars is between 2-5 years old. I also read that cars in the 15-20 year age tend to be nearing the end of their services. For that reason, Low is from 0-5, Medium from 6-15, and High from 16-100.

Now lets address the HOUSING_INFO variable. There were many columns representing information that had to do with housing, about 25. I wanted to filter through them and decide whether applicants had all the columns filled, some, or few. Few included 0-2, some had greater than 2, and all meant each column was filled with a value. My code was written to avoid considering NAs as a "value."

New Columns

The variables I created for feature engineering were good, but I wanted to add more to push my models to the next level. For that purpose, I started looking into what leaderboard kagglers suggested.

To start off, our professor suggested we looked into financial feature engineering because he noticed that was helping accuracy. I found this discussion board https://www.kaggle.com/competitions/home-credit-default-risk/discussion/64600 (https://www.kaggle.com/competitions/home-credit-default-risk/discussion/64600) which mentioned simple features that helped make models top 50. This kernel was actually written in python, so I transformed the most important sections, into R code. Namely, I combined previous_application.csv and created these columns: screwratio1, screwratio2, saint_CNT, angel_CNT, and simple_diff. The creator of this kernel was James Davis and he got an accuracy of 0.79 on Kaggle.

Finally, there were a few other "simple" feature engineering columns that were created only using the regular application_train data. These columns were credit_to_annuity_ratio, credit_to_goods_ratio, credit_to_income_ratio, income_credit_percentage, income_per_person, payment_rate, and phone_to_birth_ratio. Unfortunately, I did not keep track of whose leaderboard solution this came from, so I cannot give credit at the moment.

Best Performing Models

Below is more information about three of my best performing models. For reference, I used accuracy and roc_auc to compare model performance. For the most part, they returned fairly similar results. On average, the roc_auc was typically higher than the accuracy.

Also, it is important to note that the varying models required unique recipes. This is because there were instances where some models required certain transformations. For example, only numeric data or performed best with normalized data.

Ensemble Method/Stacked Predictions

I got the idea of an ensemble method approach because I saw several leaderboard submissions that got their best results that way. While I tried several combinations of the models I created, the best performing was Random Forest, BART, Logistic Regression, and Extra Trees with an accuracy of 0.70311. What is interesting about the stacked model is that it includes a lasso penalized regression meta learner for building the predictions. That aspect is what helped increase the overall accuracy.

BART

BART was my first individual model to hit the 0.70 threshold with 0.70113. I was very excited when this happened! It was the only model to get to 0.70.

Random Forest

Random forest performed well the whole time, but the feature engineering got it to 0.69877.

Future

Now that the modeling process is over, I want to focus more on interpreting the data. Additionally, I am curious to know if using the full dataset vs. only 5% to train the models would drastically increase our results on Kaggle, for instance. That would be interesting to look into, but it would take a very long time.