Starcraft Skill Data - Exploration

Linear Regression Model Selection

Sergei Dakov

2023-07-19

The linear regression model is used in this case a quick and basic comparison baseline. To avoid claims of an unfair comparison we will perform the same model selection steps as all other models, thus giving the LR model a "fair chance"

Data Preparation

Before we start, we load in required packages:

```
library(tidyverse) # library containing tools for streamlining and tidying data processing
library(rsample) #library for sampling and data splitting
library(naniar) #library for visualization of missing data
library(parsnip) #library for tidy model construction
library(recipes) #library to easier manipulate data for model construction
library(themis) #library for dealing with imbalances with artificial sampling
library(yardstick) #functions o calculate metrics
library(tune) #library that allows to tune multiple parameters at once
```

Next, load in some helper functions:

```
source("model_selection_skeleton.r")
```

Finally, we load in the data:

```
skillData <- read_csv("SkillCraft1_Dataset.csv") %>%
  mutate(LeagueIndex = factor(LeagueIndex)) %>%
  select(-c("Age","HoursPerWeek","TotalHours","GameID")) %>%
  mutate(LeagueIndex = as.numeric(LeagueIndex))
```

```
## Rows: 3395 Columns: 20
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): Age, HoursPerWeek, TotalHours
## dbl (17): GameID, LeagueIndex, APM, SelectByHotkeys, AssignToHotkeys, Unique...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Perform the initial split:

```
set.seed(1234)
reg_split <- initial_split(skillData,strata = "LeagueIndex")
regression_train <- training(reg_split)
regression_test <- testing(reg_split)
```

As a baseline we use a simple LR model and get a prediction, and relevant metrics for this case. Note that the linear regression model is likley to provide a non whole value thus we need to round the values

```
mod_glm <- glm(LeagueIndex~.,data=regression_train)
predicted_guess <- predict(mod_glm,regression_test) %>% round()
regression_test_round <- regression_test %>% cbind(guess = predicted_guess)
regression_test_round  %>% mutate(hit = ifelse(guess==LeagueIndex,1,0)) %>% pull(hit) -> hits
mean(hits)
```

```
## [1] 0.3976471
```

```
OnevRest(regression_test_round,truth = "LeagueIndex",guess = "guess")
```

```
## [1] 0.6083192
```

The AUC is 0.61, and accuracy of the model is 0.397.

Since there are multiple ways to round a number, we can test multiple cutoff points, and select the best performing one.

```
predicted_guess <- predict(mod_glm,regression_test)
results <- numeric()
results_means <- numeric()
for (i in seq(0,1,by=0.001)) {
predicted_cut <- cutoff(predicted_guess,i)
regression_test_cut <- regression_test %>% cbind(guess = predicted_cut)
regression_test_cut  %>% mutate(hit = ifelse(guess==LeagueIndex,1,0)) %>% pull(hit) -> hits
results<-c(results,OnevRest(regression_test_round,truth = "LeagueIndex",guess = "guess"))
results_means <- c(results_means,mean(hits))
}
max(results)
```

```
## [1] 0.6083192
```

```
(which.max(results)-1)*0.001
```

```
## [1] 0
```

```
max(results_means)
```

```
## [1] 0.4105882
```

```r
(which.max(results_means)-1)*0.001
```

```
## [1] 0.354
```

For the AUC, all cutoffs offer the same degree of separation, thus we look to accuracy as a secondary measure. In this case the best cutoff is at 0.354, with an accuracy of 0.411.

Model Selection

First, we need to re load the data, as we omitted some columns in the naive model:

```r
skillData <- read_csv("SkillCraft1_Dataset.csv") %>%
  select(-Age) %>%
  mutate(across(c("HoursPerWeek","TotalHours"),~as.numeric(.x)))
```

```
## Rows: 3395 Columns: 20
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): Age, HoursPerWeek, TotalHours
## dbl (17): GameID, LeagueIndex, APM, SelectByHotkeys, AssignToHotkeys, Unique...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `across(c("HoursPerWeek", "TotalHours"), ~as.numeric(.x))`.
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```r
glimpse(skillData)
```

```
## Rows: 3,395
## Columns: 19
## $ GameID               <dbl> 52, 55, 56, 57, 58, 60, 61, 72, 77, 81, 83, 93, 9~
## $ LeagueIndex          <dbl> 5, 5, 4, 3, 3, 2, 1, 7, 4, 4, 3, 4, 3, 3, 5, 5, 4~
## $ HoursPerWeek         <dbl> 10, 10, 10, 20, 10, 6, 8, 42, 14, 24, 16, 4, 12, ~
## $ TotalHours           <dbl> 3000, 5000, 200, 400, 500, 70, 240, 10000, 2708, ~
## $ APM                  <dbl> 143.7180, 129.2322, 69.9612, 107.6016, 122.8908, ~
## $ SelectByHotkeys      <dbl> 0.0035151591, 0.0033038124, 0.0011010906, 0.00103~
## $ AssignToHotkeys      <dbl> 2.196974e-04, 2.594617e-04, 3.355705e-04, 2.13101~
## $ UniqueHotkeys        <dbl> 7, 4, 4, 1, 2, 2, 6, 6, 2, 8, 4, 3, 1, 2, 2, 4, 1~
## $ MinimapAttacks       <dbl> 1.098487e-04, 2.940566e-04, 2.936242e-04, 5.32753~
## $ MinimapRightClicks   <dbl> 3.923169e-04, 4.324362e-04, 4.614094e-04, 5.43408~
## $ NumberOfPACs         <dbl> 0.004849036, 0.004307064, 0.002925755, 0.00378255~
## $ GapBetweenPACs       <dbl> 32.6677, 32.9194, 44.6475, 29.2203, 22.6885, 76.4~
## $ ActionLatency        <dbl> 40.8673, 42.3454, 75.3548, 53.7352, 62.0813, 98.7~
## $ ActionsInPAC         <dbl> 4.7508, 4.8434, 4.0430, 4.9155, 9.3740, 3.0965, 4~
## $ TotalMapExplored     <dbl> 28, 22, 22, 19, 15, 16, 15, 45, 29, 27, 24, 19, 1~
```

```
## $ WorkersMade          <dbl> 0.00139660, 0.00119350, 0.00074455, 0.00042620, 0~
## $ UniqueUnitsMade      <dbl> 6, 5, 6, 7, 4, 6, 5, 9, 7, 6, 7, 7, 7, 5, 7, 6, 6~
## $ ComplexUnitsMade     <dbl> 0.000000e+00, 0.000000e+00, 0.000000e+00, 0.00000~
## $ ComplexAbilitiesUsed <dbl> 0.0000e+00, 2.0757e-04, 1.8876e-04, 3.8358e-04, 1~
```

```
#New initial split
set.seed(1234)
reg_split <- initial_split(skillData,strata = "LeagueIndex")
regression_train <- training(reg_split)
regression_test <- testing(reg_split)

mod_reg <- linear_reg(engine = "glmnet",penalty = 0.01)
```

training set has 2305 points, we propose the following split; since there are relatively few missing values.

-imbalance: 800

-missing Values: 445

-interactions: 1000

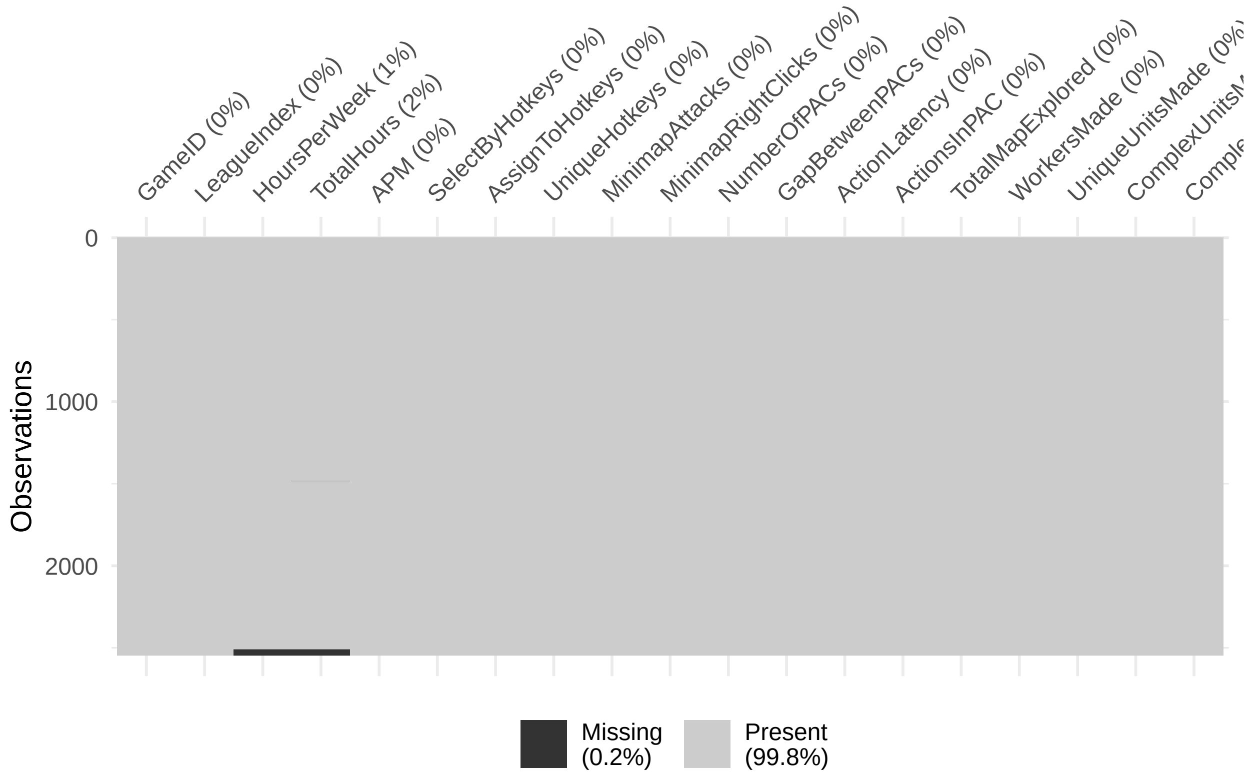
-hyper Parameters: 300

NOTE: many of the numeric variables are not normally distributed, we may apply the BoxCox transformation to normalize them, though log transform appears to be enough for some.

```
split_sizes <- c("imbalance"=800,"interactions"=1000,"tuning"=300,"missing"=445)


set.seed(1234)
miss_split <- initial_split(regression_train,strata = "LeagueIndex",prop = (split_sizes["missing"]+2)/n
train_miss <- training(miss_split)
rest_split <- testing(miss_split)

set.seed(1234)
interaction_split <- initial_split(rest_split,strata="LeagueIndex",prop = ((split_sizes["interactions"]
train_interaction <- training(interaction_split)
rest_split <- testing(interaction_split)

set.seed(1234)
imbalance_split <- initial_split(rest_split,strata="LeagueIndex",prop = ((split_sizes["imbalance"]+1)/n
train_imbalance <- training(imbalance_split)
train_tuning <- testing(imbalance_split)
```

Missing Values

```
vis_miss(regression_train)
```



The main missing values come from one category- pro players, but not entirely.

We could just assume each data point with missing data is a pro and drop the missing values or use it as an extra variable to help direct the model.

```
# substitute the mean values in place of NAs, keep the data as is
rec_mean_keep <- recipe(LeagueIndex~.,data=train_miss) %>%
  step_rm(GameID)%>%
  step_impute_mean(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# substitute the missing value using KNN, keep the data as is

rec_knn_keep <- recipe(LeagueIndex~.,data=train_miss) %>%
  step_rm(GameID)%>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# separate the rows with missing data into a separate category
regression_train_sep <- train_miss %>% drop_na()
regression_train_rest <- train_miss %>%
  anti_join(regression_train_sep) %>%
  select(LeagueIndex) %>%
  mutate(cv_split = row_number()%%4+1)
```

```
## Joining with `by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)`
```

```r
rec_separate <- recipe(LeagueIndex~.,data=regression_train_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# substitute the missing values with means, add an extra column to note which rows had missing values
rec_mean_extra <- recipe(LeagueIndex~.,data=train_miss) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_mean(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing)

# substitute missing values with KNN, add an extra column to note which rows had missing values
rec_knn_extra <- recipe(LeagueIndex~.,data=train_miss) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing)

#drop all columns with missing values

rec_drop <- recipe(LeagueIndex~.,data=train_miss) %>%
  step_rm(GameID) %>%
  step_rm(TotalHours,HoursPerWeek) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

Now we can perform cross-validation on each of the recipe and select the model with the best mean AUC:

```r
set.seed(100)
cv_splits <- vfold_cv(train_miss,v=4,strata = 'LeagueIndex')

set.seed(100)
cv_splits_res <- vfold_cv(regression_train_sep,v=4,strata = 'LeagueIndex')


lst_recs <- list("mean_keep" = rec_mean_keep,
                 "knn_keep" = rec_knn_keep,
                 "mean_extra" = rec_mean_extra,
                 "knn_extra" = rec_knn_extra,
                 "drop_drop" = rec_drop)

lst_recs_sep <- lst("drop_separate" = rec_separate)

cv_splits <- calculate_splits(cv_splits,lst_recs,mod_reg)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(guess)
##
##   # Now:
##   data %>% select(all_of(guess))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
cv_splits_res <- calculate_splits_sep(cv_splits_res,lst_recs_sep,mod_reg,regression_train_rest)

cv_splits <- cv_splits %>% full_join(cv_splits_res)
```

```
## Joining with ‘by = join_by(id)‘
```

```
cv_res_missing <- cv_splits %>% pivot_longer(cols=c(names(lst_recs),names(lst_recs_sep)),names_to = "re
  select(id,recipe,AUC) %>% separate(recipe,c("miss","row"))%>%
  group_by(miss,row) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)
```

```
## ‘summarise()‘ has grouped output by ’miss’. You can override using the
## ‘.groups‘ argument.
```

```
cv_res_missing
```

```
## # A tibble: 6 x 3
## # Groups:   miss [3]
##   miss  row        AUC
##   <chr> <chr>    <dbl>
## 1 drop  separate 0.666
## 2 knn   extra    0.629
## 3 mean  extra    0.629
## 4 drop  drop     0.612
## 5 mean  keep     0.612
## 6 knn   keep     0.609
```

Dropping the columns with missing values, as well as singling out those rows as pro players performed the best (AUC of 0.666)

Imbalance

```
regression_train %>% group_by(LeagueIndex) %>% summarise(n=n(),ratio = n()/nrow(regression_train))
```

```
## # A tibble: 8 x 3
##   LeagueIndex     n  ratio
##         <dbl> <int>  <dbl>
## 1           1   121 0.0475
## 2           2   267 0.105
## 3           3   412 0.162
## 4           4   608 0.239
## 5           5   604 0.237
## 6           6   465 0.183
## 7           7    30 0.0118
## 8           8    38 0.0149
```

The data is clearly unbalanced, possibly in accordance to the population distribution among the ranks.

#Imbalance

To deal with the imbalance of the data we have several approaches:

- keeping the data as it is (this will be used as a baseline)
- upsample (duplicate appearances of the sparse classes to increase their count)
- downsample (remove instances of the over-represented classes to bring their number down)
- SMOTE (synthetically generate new values for sparse classes by generating "in-between" values for all variables)
- SMOTE with downsampling (lower the count of over-represented classes to decrease the ammount of artificial data introduced)

```
train_imbalance_sep <- train_imbalance %>% drop_na()
train_imbalance_rest <- train_imbalance %>% anti_join(train_imbalance_sep) %>% select(LeagueIndex) %>% r
```

```
## Joining with `by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)`
```

```
rec_upsample <- recipe(LeagueIndex~.,data=train_imbalance_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_upsample(imbalance,over_ratio = 1, seed = 123) %>%
  step_rm(imbalance)

rec_downsample <- recipe(LeagueIndex~.,data=train_imbalance_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance)
```

```r
rec_smote <- recipe(LeagueIndex~.,data=train_imbalance_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio=1.5,seed =123) %>%
  step_smote(imbalance,over_ratio = 1,seed=123) %>%
  step_rm(imbalance)

rec_puresmote <- recipe(LeagueIndex~.,data=train_imbalance_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_smote(imbalance,over_ratio = 1,seed=123) %>%
  step_rm(imbalance)

rec_nothing <- recipe(LeagueIndex~.,data=train_imbalance_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

rec_extra_upsample <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_upsample(imbalance,over_ratio = 1, seed = 123) %>%
  step_rm(imbalance)

rec_extra_downsample <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance)

rec_extra_smote <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio=1.5,seed =123) %>%
  step_smote(imbalance,over_ratio = 1,seed=123,neighbors = 4) %>%
  step_rm(imbalance)
```

```
rec_extra_puresmote <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_knn(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_smote(imbalance,over_ratio = 1,seed=123,neighbors = 4) %>%
  step_rm(imbalance)

rec_extra_nothing <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)%>%
  step_mutate(hadmissing = ifelse(is.na(TotalHours),1,0)) %>%
  step_impute_mean(everything()) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()&!hadmissing)
```

Note - KNN with extra columns performed second best, so we test using it as well.

```
set.seed(100)
cv_splits_res <- vfold_cv(train_imbalance_sep,v=4,strata = 'LeagueIndex')

set.seed(100)
cv_splits <- vfold_cv(train_imbalance,v=4,strata = 'LeagueIndex')


lst_recs_sep <- list("downsample" = rec_downsample,
                     "upsample" = rec_upsample,
                     "smote" = rec_smote,
                     "pure_smote" = rec_puresmote,
                     "nothing" = rec_nothing)
lst_recs <- list("extra_upsample" = rec_extra_upsample,
                 "extra_downsample" = rec_extra_downsample,
                 "extra_smote" = rec_extra_smote,
                 "extra_pure_smote" = rec_extra_puresmote,
                 "extra_nothing"=rec_extra_nothing)

cv_splits <- calculate_splits(cv_splits,lst_recs,mod_reg)

cv_splits_res <- calculate_splits_sep(cv_splits_res,lst_recs_sep,mod_reg,train_imbalance_rest)

cv_splits <- cv_splits %>% full_join(cv_splits_res)
```

```
## Joining with `by = join_by(id)`
```

```
cv_res_imbalance <- cv_splits %>%
  pivot_longer(cols=c(names(lst_recs_sep),names(lst_recs)),names_to = "recipe",values_to = "AUC") %>%
  select(id,recipe,AUC) %>%
  group_by(recipe) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)

cv_res_imbalance
```

```
## # A tibble: 10 x 2
##    recipe              AUC
##    <chr>             <dbl>
##  1 downsample        0.670
##  2 smote             0.665
##  3 nothing           0.663
##  4 pure_smote        0.663
##  5 upsample          0.662
##  6 extra_upsample    0.635
##  7 extra_smote       0.634
##  8 extra_pure_smote 0.626
##  9 extra_nothing     0.613
## 10 extra_downsample 0.583
```

The highest result is for down-sampling with an AUC of 0.669

Interactions and Feature Engineering

First let us examine possibility of non linear relations:

```
rec_unskew <- recipe(LeagueIndex~.,data=train_interaction) %>%
  step_rm(GameID) %>%
  step_naomit(everything(),skip = FALSE) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  prep()

linearity <- (sapply(colnames(train_imbalance[-2]),Linearity_test,dat=train_interaction,y="LeagueIndex")
bind_cols("column" = colnames(train_imbalance[-2]),"significance" = linearity) %>% arrange(-abs(signifi

train_interaction_sep <- train_interaction %>% drop_na()
train_interaction_rest <- train_interaction %>% anti_join(train_interaction_sep) %>% select(LeagueIndex
```

```
## Joining with `by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)`
```

```
rec_nothing <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance)

rec_bs <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  add_role(all_of(top_nonlinear),new_role = "nonlinear") %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
```

```r
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance) %>%
  step_bs(has_role("nonlinear"))


rec_ns <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  add_role(all_of(top_nonlinear),new_role = "nonlinear") %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance) %>%
  step_ns(has_role("nonlinear"))


rec_poly <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  add_role(all_of(top_nonlinear),new_role = "nonlinear") %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio = 1,seed=123) %>%
  step_rm(imbalance) %>%
  step_poly(has_role('nonlinear'))
```

```r
set.seed(100)
cv_splits_res <- vfold_cv(train_interaction_sep,v=4,strata = 'LeagueIndex')

lst_recs_sep <- list("ns" = rec_ns,
                 "bs" = rec_bs,
                 "poly" = rec_poly,
                 "nothing" = rec_nothing)


cv_splits_res <- calculate_splits_sep(cv_splits_res,lst_recs_sep,mod_reg,train_imbalance_rest)
cv_res_linearity <- cv_splits_res %>%
  pivot_longer(cols=names(lst_recs_sep),names_to = "recipe",values_to = "AUC") %>%
  select(id,recipe,AUC) %>%
  group_by(recipe) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)

cv_res_linearity
```

```
## # A tibble: 4 x 2
##   recipe    AUC
##   <chr>   <dbl>
## 1 nothing 0.661
## 2 ns      0.656
## 3 poly    0.653
## 4 bs      0.643
```

Treating all parameters as linear lends the best results with AUC of 0.661

Interactions

There may be interactions between some (or all) of the parameters, there are multiple ways to check:

- hand picking predictors that may interact
- checking for any significant interactions between all predictors

Since we are testing a new facet of the data we can reuse old splits as well

```
train_large_sep <- rbind(train_imbalance_sep,train_interaction_sep,regression_train_sep)
train_large_rest <- rbind(train_imbalance_rest,train_interaction_rest,regression_train_rest)

rec_nothing <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_downsample(imbalance,under_ratio=1,seed =123) %>%
  step_rm(imbalance)

rec_handpicked <-recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_interact(~APM:all_numeric_predictors()) %>%
  step_nzv(all_numeric_predictors(),freq_cut = 99/1) %>%
  step_downsample(imbalance,under_ratio=1,seed =123) %>%
  step_rm(imbalance)


rec_all_interact <- recipe(LeagueIndex~.,data=train_interaction_sep) %>%
  step_rm(GameID)%>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(imbalance = factor(LeagueIndex)) %>%
  step_interact(~all_numeric_predictors():all_numeric_predictors()) %>%
  step_nzv(all_numeric_predictors(),freq_cut = 95/5) %>%
  step_downsample(imbalance,under_ratio=1,seed =123) %>%
  step_rm(imbalance)
```

```
set.seed(100)
cv_splits_res <- vfold_cv(train_interaction_sep,v=4,strata = 'LeagueIndex')


lst_recs_sep <- list("handpicked" = rec_handpicked,
                     "all" = rec_all_interact,
                     "nothing" = rec_nothing)
```

```
cv_splits_res <- calculate_splits_sep(cv_splits_res,lst_recs_sep,mod_reg,train_imbalance_rest)
cv_res_interact <- cv_splits_res %>%
  pivot_longer(cols=names(lst_recs_sep),names_to = "recipe",values_to = "AUC") %>%
  select(id,recipe,AUC) %>%
  group_by(recipe) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)

cv_res_interact
```

```
## # A tibble: 3 x 2
##   recipe       AUC
##   <chr>      <dbl>
## 1 nothing    0.661
## 2 handpicked 0.637
## 3 all        0.616
```

No interactions performed the best (AUC of 0.661)

Tuning

The hyper-parameters we can tune are:

- the frequency cutoff
- the down-sample ratio
- the model penalty

```
train_tuning_sep <- train_tuning %>% drop_na()
train_tuning_rest <- train_tuning %>%
  anti_join(train_tuning_sep) %>%
  select(LeagueIndex) %>%
  mutate(cv_split = row_number()%%4+1)
```

```
## Joining with `by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)`
```

```
set.seed(100)
cv_splits_res <- vfold_cv(train_interaction_sep,v=4,strata = 'LeagueIndex')

tuning_results <- tibble()

for( penalty in seq(0,1,by=0.1)) {
  for( freq_cut in c(99/1,95/5,97/3,90/10) ) {
    for (under_ratio in seq(0.5,1.5,by=0.1)) {
```

```
model_tuning <- linear_reg(engine = "glmnet",penalty = penalty)

    recipe_tuning <- recipe(LeagueIndex~.,data=train_tuning_sep) %>%
        step_rm(GameID)%>%
        step_YeoJohnson(all_numeric_predictors()) %>%
        step_normalize(all_numeric_predictors()) %>%
        step_mutate(imbalance = factor(LeagueIndex)) %>%
        step_interact(~APM:all_numeric_predictors()) %>%
        step_nzv(all_numeric_predictors(),freq_cut = freq_cut) %>%
        step_downsample(imbalance,under_ratio=under_ratio,seed =123) %>%
        step_rm(imbalance)
    rec_tuning <- list(recipe_tuning)
    names(rec_tuning) <- paste(penalty,freq_cut,under_ratio,sep="_")
    cv_splits_current <- calculate_splits_sep(cv_splits_res,rec_tuning,
                                             model_tuning,
                                             train_tuning_rest)
    tuning_results <- bind_rows(tuning_results,cv_splits_current)
    }
  }
}

tuning_results <- tuning_results %>% pivot_longer(cols = contains("_"),names_to = "recipe",values_to =

tuning_results %>% group_by(recipe) %>% summarize ("AUC" = mean(AUC)) %>% arrange(-AUC) %>% head(10)
```

```
## # A tibble: 10 x 2
##    recipe                      AUC
##    <chr>                     <dbl>
##  1 0.1_19_1.4                0.684
##  2 0.1_32.3333333333333_1.4 0.684
##  3 0.1_99_1.4                0.684
##  4 0.1_9_1.4                 0.684
##  5 0.2_19_1.4                0.677
##  6 0.2_32.3333333333333_1.4 0.677
##  7 0.2_99_1.4                0.677
##  8 0.2_9_1.4                 0.677
##  9 0.1_19_0.8                0.672
## 10 0.1_32.3333333333333_0.8 0.672
```

There is a tie for best performing, in all cases the best option is to use a penalty of 0.1, and and sampling ratio of 1.4, the ratio for step_nzv does not matter significantly so we will use 99/1 attempt maintain the most data in the future.

Final Prediction

```
regression_test_sep <- regression_test %>% drop_na()
regressoin_test_rest <- regression_test %>%
  anti_join(regression_test_sep) %>%
  select(LeagueIndex) %>%
  mutate("prediction"=8)
```

```
## Joining with `by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)`
```

```
regression_train_sep <- regression_train %>% drop_na()



mod_final <- linear_reg(engine = "glmnet",penalty = 0)

rec_final <- recipe(LeagueIndex~.,data=regression_train_sep) %>%
        step_rm(GameID)%>%
        step_YeoJohnson(all_numeric_predictors()) %>%
        step_normalize(all_numeric_predictors()) %>%
        step_interact(~APM:all_numeric_predictors()) %>%
        step_nzv(all_numeric_predictors(),freq_cut = 99/1) %>%
        step_mutate(imbalance = factor(LeagueIndex)) %>%
        step_downsample(imbalance,under_ratio=1.4,seed =123) %>%
        step_rm(imbalance) %>%
        prep()

train_final <- bake(rec_final,NULL)

test_final <- bake(rec_final,new_data = regression_test_sep)
fit_final <- fit(mod_final,as.numeric(LeagueIndex)~.,train_final)
predicted_final <- predict(fit_final,test_final)
predicted_final <- cutoff(predicted_final$.pred,0.354)
predicted_final <- bind_cols("LeagueIndex" = regression_test_sep$LeagueIndex,"prediction"=predicted_fina
mean(predicted_final$prediction==predicted_final$LeagueIndex)
```

```
## [1] 0.4105882
```

```
OnevRest(predicted_final,"LeagueIndex","prediction")
```

```
## [1] 0.6860088
```

The final model prediction had an One v Rest AUC of 0.69, and an accuracy of 0.411

Save the model

```
saveRDS(rec_final,"regression_model")
```