Final Project Prediction

Krish Lakshmi Narayanan

- UW ID: 20873217

Summary

Preprocessing

To build my model, I preprocessed the data (removed outliers in the salary variate from the whole data set) and converted categorical variates to factors. I also converted the date variate to type date.

Transformation

- salary: I took the log of the variate salary.

New Variables

- year: year for the date of contract.
- month: month for the date of contract.
- day: day for the date of contract.
- day_of_week: day of the week for the date of contract.
- is_weekend: indicator of whether the day is a weekend.
- month_sin: sin component of the cyclical encoding of the month variate.
- month_cos: cos component of the cyclical encoding of the month variate.
- npc: Known as Net Possession Control and is calculated by takeaway - giveaway.
- pm_per_min:Plus-minus per minute and is calculated by pm/toi
- ppg: points per game and is calculated by pts/gp
- physicality: calculated as hits + 0.5*pim
- relCorsi_per_toi: calculated as relCorsi/toi

Model

The final model chosen was an XGBoost Model based on the following formula:

log(salary) ~ team + toi + gp + pos + nat + age + injure + day + is_weekend + month_sin + month_cos + npc + pm_per_min + ppg + physicality + relCorsi_per_toi

Once an initial model was built and the feature importances were looked at, we chose the features that had a greater than 0 feature importance and created a model based on those features. That was the model that we used.

Model Building

During the model building process, a lot of different models were looked at. Methods such as gbm (gradient boosting), Random Forests, Boosted Trees, and finally XGBoost (which is what the final model was based on) were looked at. We chose the best model by looking at the lowest test sample RMLSE and the training sample RMLSE to assess the fit of the model on the training data. Cross-validation was also employed to hyper tune the parameters through a series of grid searches.

1.Preprocessing

1.1 Loading data and packages

```
load("final.Rdata")

library(gbm)
library(lubridate)
library(tidyverse)
library(rpart)
library(caret)
library(randomForest)
library(missForest)
library(xgboost)
library(doParallel)
```

1.2 Data Processing

```
# Convert the 'date' column to Date format
dtrain$date <- as.Date(dtrain$date)

# Extract date features
dtrain$year <- year(dtrain$date)
dtrain$month <- month(dtrain$date)
dtrain$day <- day(dtrain$date)
dtrain$day_of_week <- as.factor(wday(dtrain$date, label = FALSE)) # Sunday = 1, Saturday = 7
dtrain$is_weekend <- as.factor(ifelse(dtrain$day_of_week %in% c(1, 7), 1, 0)) # Weekend flag

# Cyclic encoding for month (to capture periodicity)
dtrain$month_sin <- sin(2 * pi * dtrain$month / 12)
dtrain$month_cos <- cos(2 * pi * dtrain$month / 12)

# Drop the original date column (not used directly in GBM)
dtrain$date <- NULL
dtrain$month <- NULL
dtrain$day_of_week <- NULL

# Convert all character columns to factors
dtrain <- dtrain %>%
  mutate(across(where(is.character), as.factor))

# Creating new features
dtrain$npc <- dtrain$takeaway - dtrain$giveaway
dtrain$pm_per_min <- dtrain$pm/dtrain$toi
dtrain$ppg <- dtrain$pts/dtrain$gp

dtrain$takeaway <- NULL
dtrain$giveaway <- NULL
dtrain$pm <- NULL
dtrain$pts <- NULL
dtrain$goal <- NULL

# Doing the same for the testing set, we have:
```

```r
# Convert the 'date' column to Date format
dtest$date <- as.Date(dtest$date)

# Extract date features
dtest$year <- year(dtest$date)
dtest$month <- month(dtest$date)
dtest$day <- day(dtest$date)
dtest$day_of_week <- as.factor(wday(dtest$date, label = FALSE)) # Sunday = 1, Saturday = 7
dtest$is_weekend <- as.factor(ifelse(dtest$day_of_week %in% c(1, 7), 1, 0)) # Weekend flag

# Cyclic encoding for month (to capture periodicity)
dtest$month_sin <- sin(2 * pi * dtest$month / 12)
dtest$month_cos <- cos(2 * pi * dtest$month / 12)

# Drop the original date column (not used directly in GBM)
dtest$date <- NULL
dtest$month <- NULL

# Convert all character columns to factors
dtest <- dtest %>%
  mutate(across(where(is.character), as.factor))

# Creating new features
dtest$npc <- dtest$takeaway - dtest$giveaway
dtest$pm_per_min <- dtest$pm/dtest$toi
dtest$ppg <- dtest$pts/dtest$gp

dtest$takeaway <- NULL
dtest$giveaway <- NULL
dtest$pm <- NULL
dtest$pts <- NULL
dtest$goal <- NULL
dtest$day_of_week <- NULL

dtest$salary <- 1
```

Later in the process we do some further feature engineering. Here is how that looks:

```r
new_dtrain <- dtrain
new_dtrain$physicality <- new_dtrain$hits + 0.5*new_dtrain$pim
new_dtrain$relCorsi_per_toi <- new_dtrain$relCorsi/new_dtrain$toi

new_dtrain$hits <- NULL
new_dtrain$pim <- NULL
new_dtrain$relCorsi <- NULL

new_dtest <- dtest
new_dtest$physicality <- new_dtest$hits + 0.5*new_dtest$pim
new_dtest$relCorsi_per_toi <- new_dtest$relCorsi/new_dtest$toi

new_dtest$hits <- NULL
new_dtest$pim <- NULL
new_dtest$relCorsi <- NULL
```

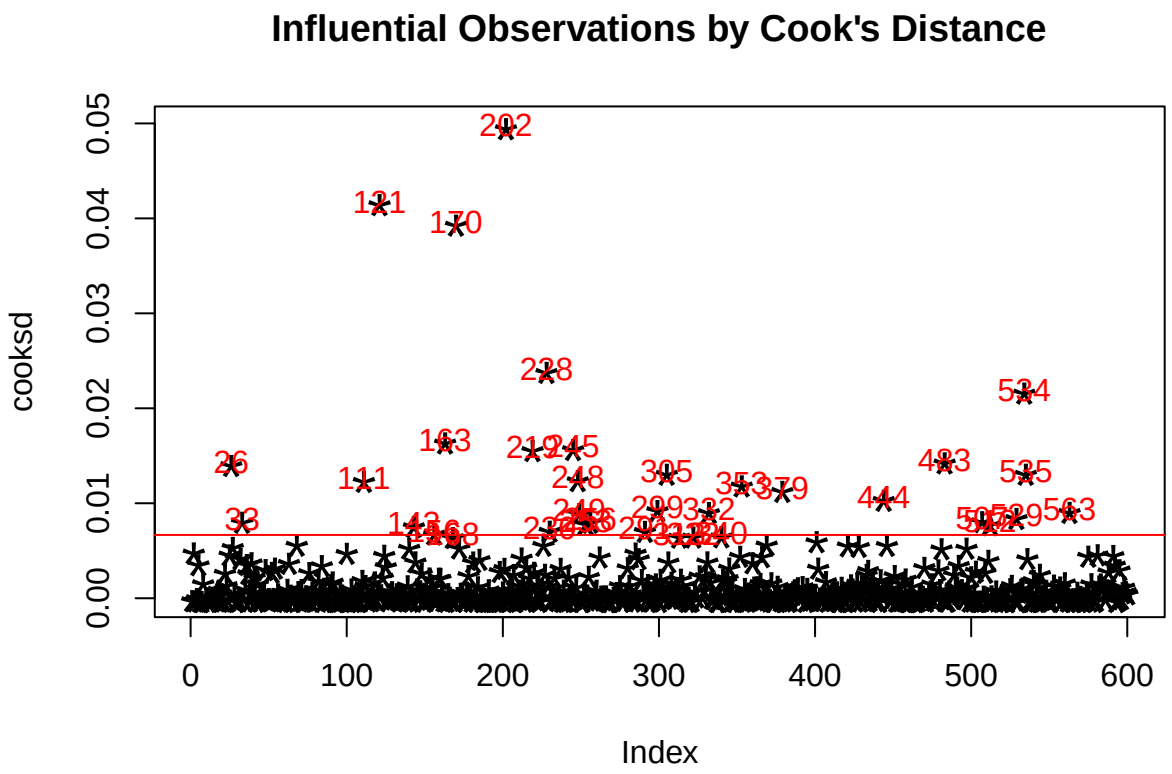
1.2 Looking at outliers

```
# Multi-collinearity checking
newer_model <- lm(salary~team+gp+injure+toi+pos+nat+age+
                    relCorsi+pim+hits+day+
                    is_weekend+month_sin+month_cos+npc+
                    pm_per_min+ppg,
                  data = dtrain)
car::vif(newer_model)
```

```
##                 GVIF Df GVIF^(1/(2*Df))
## team       17.753938 30        1.049111
## gp          4.265356  1        2.065274
## injure      1.268106  1        1.126102
## toi         4.751981  1        2.179904
## pos        26.208591 17        1.100827
## nat        11.266000 15        1.084074
## age         1.296012  1        1.138425
## relCorsi    1.202527  1        1.096598
## pim         2.521000  1        1.587766
## hits        3.163054  1        1.778498
## day         1.176518  1        1.084674
## is_weekend  1.173147  1        1.083119
## month_sin   1.231073  1        1.109537
## month_cos   1.251962  1        1.118911
## npc         1.776206  1        1.332744
## pm_per_min  1.365854  1        1.168697
## ppg         3.007695  1        1.734271
```

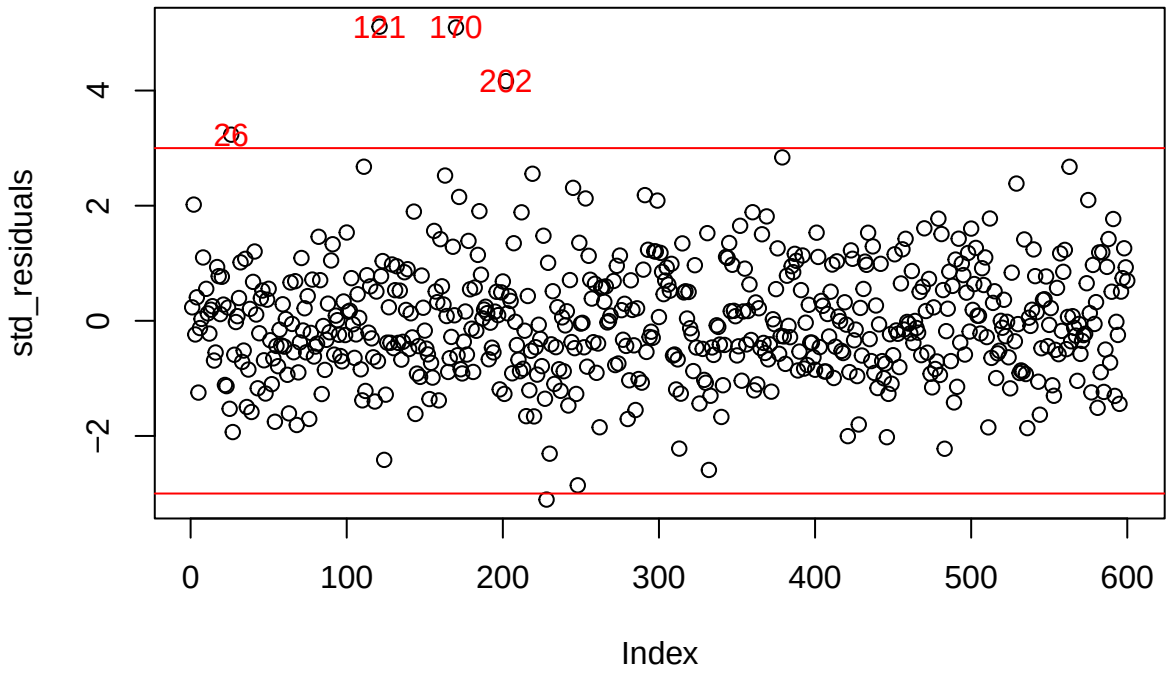
```
# Calculate Cook's Distance
cooksd <- cooks.distance(newer_model)

# Plot Cook's Distance
plot(cooksd, pch="*", cex=2, main="Influential Observations by Cook's Distance")
abline(h = 4/600, col="red") # Add cutoff line (threshold: 4/n)
text(x=1:length(cooksd), y=cooksd,
     labels=ifelse(cooksd > 4/600, rownames(dtrain), ""),
     col="red")
```




```
# Identify influential points
influential <- which(cooksd > 4/600)

# residuals
# Calculate standardized residuals
std_residuals <- rstandard(newer_model)
plot(std_residuals)
abline(h = c(-3,3), col="red") # Add cutoff line (threshold: 4/n)
text(x=1:length(std_residuals), y=std_residuals,
     labels=ifelse(std_residuals > 3, rownames(dtrain), ""),
     col="red")
```




```
# Identify outliers with standardized residuals > ±3
outliers <- which(abs(std_residuals) > 3)

# Removing outliers
dtrain <- dtrain[-outliers,]

formula_full <- "log(salary)~team+gp+injure+toi+pos+nat+age+
                    relCorsi+pim+hits+day+
                    is_weekend+month_sin+month_cos+npc+
                    pm_per_min+ppg"
```

2.Model Building

2.1 GBM Model

The outputs won't be included and the code won't be run (since it would take too long), but the following was used to create the gradient boosted model:

```
# GBM Models
Building a Gradient boosted Model

# Cross validation on shrinkage
M <- 500
k <- 5
set.seed(31853071)
# Shrinkage parameter values
alphas <- c(0.005, 0.01, 0.02, 0.03, 0.04,
            0.05, 0.06, 0.07, 0.08, 0.09,
            0.10, 0.20, 0.30, 0.40, 0.50,
            0.60, 0.70, 0.80, 0.90, 1)
n_alphas <- length(alphas)
cverror <- numeric(length = n_alphas)
Mvals <- numeric(length = n_alphas)
```

```
fit <- list(length = n_alphas)
for (i in 1:n_alphas) {
  fit[[i]] <- model.boost.shrink <- gbm(as.formula(formula_full),
                                        data=dtrain,
                                        distribution = "gaussian",
                                        shrinkage = alphas[i],
                                        n.trees = M,
                                        bag.fraction = 1,
                                        cv.folds = k,
                                        interaction.depth = 2
  )
  cverror[i] <- min(fit[[i]]$cv.error)
  Mvals[i] <- which.min(fit[[i]]$cv.error)
}
plot(alphas, cverror, type = "b",
     col = adjustcolor("firebrick", 0.7), pch=19, lwd=2,
     main = "cross-validated error", xlab = "shrinkage", ylab="cv.error")
i <- which.min(cverror)
alpha <- alphas[i]
summary(fit[[i]])


# Tree parameter values
Ms <- c(100, 200, 300, 400, 500, 600, 700, 800, 900, 1000)
n_Ms <- length(Ms)
cverror <- numeric(length = n_Ms)
fit <- list(length = n_Ms)
for (i in 1:n_Ms) {
  fit[[i]] <- model.boost.trees <- gbm(as.formula(formula_full),
                                       data=dtrain,
                                       distribution = "gaussian",
                                       shrinkage = alpha,
                                       n.trees = Ms[i],
                                       bag.fraction = 1,
                                       cv.folds = k,
                                       interaction.depth = 2
  )
  cverror[i] <- min(fit[[i]]$cv.error)
}

plot(Ms, cverror, type = "b",
     col = adjustcolor("firebrick", 0.7), pch=19, lwd=2,
     main = "cross-validated error", xlab = "Number of trees", ylab="cv.error")
i <- which.min(cverror)
M <- Ms[i]
summary(fit[[i]])

# Bag fractions parameter values
bag.fracs <- c(0.05, 0.10, 0.20, 0.30, 0.40, 0.50,
               0.60, 0.70, 0.80, 0.90, 1)

n_bag.fracs <- length(bag.fracs)
cverror <- numeric(length = n_bag.fracs)
```

```r
fit <- list(length = n_bag.fracs)
for (i in 1:n_bag.fracs) {
  fit[[i]] <- model.boost.trees <- gbm(as.formula(formula_full),
                                       data=dtrain,
                                       distribution = "gaussian",
                                       shrinkage = alpha,
                                       n.trees = M,
                                       bag.fraction = bag.fracs[i],
                                       cv.folds = k,
                                       interaction.depth = 2
  )
  cverror[i] <- min(fit[[i]]$cv.error)
}
plot(bag.fracs, cverror, type = "b",
     col = adjustcolor("firebrick", 0.7), pch=19, lwd=2,
     main = "cross-validated error", xlab = "bag fraction", ylab="cv.error")
i <- which.min(cverror)
bag.fraction <- bag.fracs[i]
summary(fit[[i]])

feature_sets <- list(
  top3 = c("toi", "ppg", "gp"),
  top5 = c("toi", "ppg", "gp", "nat", "pim"),
  top7 = c("toi", "ppg", "gp", "nat", "pim", "team", "pm_per_min"),
  top9 = c("toi", "ppg", "gp", "nat", "pim", "team", "pm_per_min", "hits", "age"),
  all =  c("team","gp","injure","toi","pos","nat","age",
           "relCorsi","pim","hits","day","day_of_week",
           "is_weekend","month_sin","month_cos","npc","pm_per_min","ppg")
)

n_features <- length(feature_sets)
cverror <- numeric(length = n_features)
fit <- list(length = n_features)
for (i in 1:n_features) {
  features <- feature_sets[[i]]
  formula <- as.formula(paste("log(salary) ~", paste(features, collapse = " + ")))
  fit[[i]] <- model.boost.trees <- gbm(formula,
                                       data=dtrain,
                                       distribution = "gaussian",
                                       shrinkage = alpha,
                                       n.trees = M,
                                       bag.fraction = bag.fraction,
                                       cv.folds = k,
                                       interaction.depth = 2
  )
  cverror[i] <- min(fit[[i]]$cv.error)
}

plot(c(3,5,7,9,18), cverror, type = "b",
     col = adjustcolor("firebrick", 0.7), pch=19, lwd=2,
     main = "cross-validated error", xlab = "Number of features", ylab="cv.error")

i <- which.min(cverror)
```

```r
# Gradient Boosted Model
gbm.final <- fit[[i]]
summary(gbm.final)
```

2.2 Random Forest Model

Again, the outputs won't be included and the code won't be run (since it would take too long), but the following was used to create the random forest model:

```r
## Random Forest Model
model.rf <- randomForest(as.formula(formula_full),
                         data = dtrain,
                         # Number of variates to select at each step
                         importance = TRUE)

# Need separate response and explanatory variate data frames
trainy <- dtrain[, "salary"]
trainx <- select(dtrain, c("team","gp","injure","toi","pos","nat","age",
                           "relCorsi","pim","hits","day","day_of_week",
                           "is_weekend","month_sin","month_cos","npc",
                           "pm_per_min","ppg"))

# 10 fold cross-validataion
model.rfcv <- rfcv(trainx = trainx, trainy = log(trainy), cv.fold = 10)
# We can plot the results
with(model.rfcv, plot(n.var, error.cv, pch = 19, type="b", col="blue"))
nfeat = as.integer(names(which.min(model.rfcv$error.cv)))

model.imp <- importance(model.rf)
model.feat <- rownames(model.imp)[1:nfeat]

rf_formula <- as.formula(paste("log(salary) ~",
                               paste(model.feat, collapse = " + ")))

model.rf.2 <- randomForest(rf_formula,
                           data = dtrain,
                           importance = TRUE)

# Need separate response and explanatory variate data frames
trainy <- dtrain[, "salary"]
trainx <- select(dtrain, all_of(model.feat))

# 10 fold cross-validataion
model.rfcv2 <- rfcv(trainx = trainx, trainy = log(trainy), cv.fold = 10)
# We can plot the results
with(model.rfcv2, plot(n.var, error.cv, pch = 19, type="b", col="blue"))

nfeat2 = as.integer(names(which.min(model.rfcv2$error.cv)))

model.imp2 <- importance(model.rf.2)
model.feat2 <- rownames(model.imp2)[1:nfeat2]

rf_formula2 <- as.formula(paste("log(salary) ~", paste(model.feat2, collapse = " + ")))
```

```
model.rf.3 <- randomForest(rf_formula2,
                           data = dtrain,
                           importance = TRUE)

# Need separate response and explanatory variate data frames
trainy <- dtrain[, "salary"]
trainx <- select(dtrain, all_of(model.feat2))

# 10 fold cross-validataion
model.rfcv3 <- rfcv(trainx = trainx, trainy = log(trainy), cv.fold = 10)
# We can plot the results
with(model.rfcv3, plot(n.var, error.cv, pch = 19, type="b", col="blue"))
```

2.3 Boosted Models

Once again, the outputs won't be included and the code won't be run (since it would take too long), but the following was used to create the Boosted model:

```
# Helper
# It will also be handy to have a function that will return the
# newdata as a list, since that is what is expected by predict
# for that argument.
# We will write a function that takes a fitted tree (or any other fit)
# THis involves a little formula manipulation ... of interest only ...
get.newdata <- function(fittedTree, test.data){
  f <- formula(fittedTree)
  as.list(test.data[,attr(terms(f), "term.labels")])
}
#
# And a similar function that will extract the response values
# This is kind of hairy, formula manipulation ... feel free to ignore ...
get.response <- function(fittedTree, test.data){
  f <- formula(fittedTree)
  terms <- terms(f)
  response.id <- attr(terms, "response")
  response <- as.list(attr(terms, "variables"))[[response.id + 1]]
  with(test.data, eval(response))
}
get.explanatory_varnames <- function(formula){
  f <- as.formula(formula)
  terms <- terms(f)
  attr(terms, "term.labels")
}
# The remaining functions are the important ones
#
getTrees <- function(data, formula, B=100, ...) {
  N <- nrow(data)
  Trees <- Map(function(i){
    getTree(formula,
            getSample(data, N),
            ...)
  },
  1:B
  )
```

```
  Trees
}
# Boosted trees
boostTree <- function(formula, data,
                      lam=0.01, M = 10,
                      control=rpart.control(), ...) {
  # Break the formula into pieces
  formula.sides <- strsplit(formula, "~")[[1]]
  response.string <- formula.sides[1]
  rhs.formula <- formula.sides[2]
  # Construct the boost formula
  bformula <- paste("resid", rhs.formula, sep=" ~ ")
  # Initialize the resid and explanatory variates
  resid <- get.response(formula, data)
  xvars <- get.newdata(formula, data)
  # Calculate the boostings
  Trees <- Map(
    function(i) {
      # update data frame with current resid
      rdata <- data.frame(resid=resid, xvars)
      # Fit the tree
      tree <- rpart(bformula, data = rdata, control=control, ...)
      # Update the residuals
      # (Note the <<- assignment to escape this closure)
      resid <<- resid - lam * predict(tree)
      # Return the tree
      tree }
    , 1:M)
  # Return the boosted function
  function(newdata){
    if (missing(newdata)) {
      predictions <- Map(function(tree) {
        # Boost piece
        lam * predict(tree)
      }, Trees)
    } else {
      predictions <- Map(function(tree){
        # New data needs to be a list
        if (is.data.frame(newdata)) {
          newdata.tree <- get.newdata(tree, newdata)
        } else {
          newdata.tree <- newdata
        }
        # Boost piece
        lam * predict(tree, newdata=newdata.tree)
      }, Trees)
    }
    # Gather the results together
    Reduce(`+`, predictions)
  }
}

boosted_model <- boostTree("salary ~pts + toi + takeaway",
```

```
                              data = dtrain, M = 1000, lam = 0.01)
```

2.4 XGBoost

We finally come to the XGBoost model

2.4.1 Base model

```
set.seed(42)
train_matrix <- model.matrix(as.formula(formula_full), data = dtrain)[,-1]  # Remove intercept
train_labels <- log(dtrain$salary)

xgb_dtrain <- xgb.DMatrix(data = train_matrix, label = train_labels)


test_matrix <- model.matrix(as.formula(formula_full), data = dtest)[,-1]

# Handle any missing features (set to 0)
missing_cols <- setdiff(colnames(train_matrix), colnames(test_matrix))
test_matrix <- cbind(test_matrix, matrix(0, nrow = nrow(test_matrix),
                                         ncol = length(missing_cols),
                                         dimnames = list(NULL, missing_cols)))

# Ensure column order matches training data
test_matrix <- test_matrix[, colnames(train_matrix)]
xgb_dtest <- xgb.DMatrix(data = test_matrix)

params <- list(
  booster = "gbtree",
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.9,
  gamma = 1
)

set.seed(42)
xgb_cv <- xgb.cv(
  params = params,
  data = xgb_dtrain,
  nrounds = 1000,
  nfold = 10,
  early_stopping_rounds = 35,
  print_every_n = 50
)
```

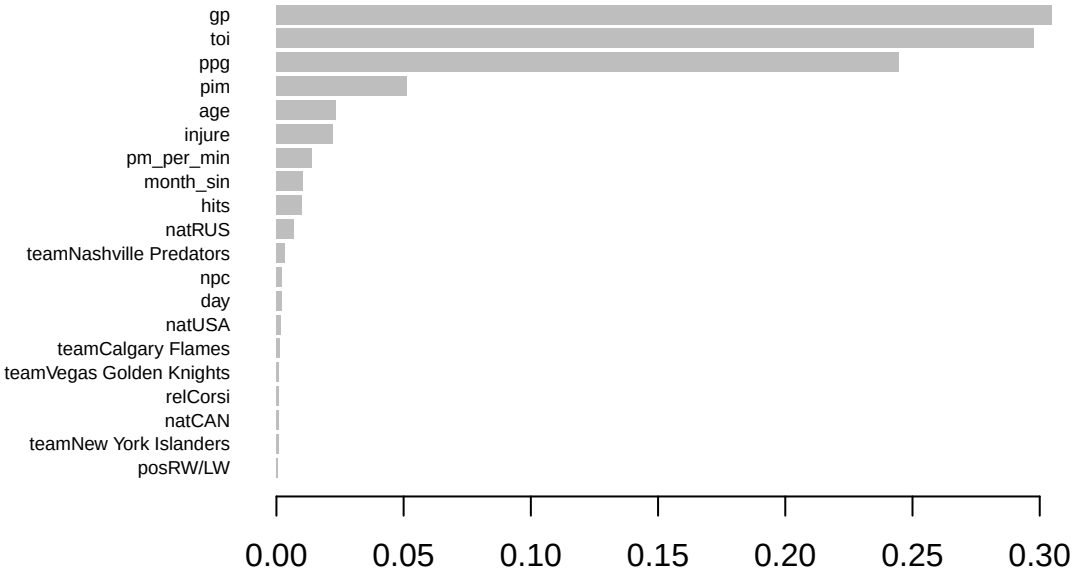
```
## [1]  train-rmse:12.160169+0.009662   test-rmse:12.159526+0.091329
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 35 rounds.
##
## [51] train-rmse:0.273642+0.004746    test-rmse:0.347639+0.042054
## [101]    train-rmse:0.248162+0.004135    test-rmse:0.334158+0.043299
## [151]    train-rmse:0.242831+0.004848    test-rmse:0.333260+0.042487
```

```
## Stopping. Best iteration:
## [117]    train-rmse:0.246387+0.003556    test-rmse:0.333057+0.042937
```

```r
# Train model
xgb_model <- xgb.train(
  params = params,
  data = xgb_dtrain,
  nrounds = xgb_cv$best_iteration
)

importance_matrix <- xgb.importance(
  feature_names = colnames(train_matrix),
  model = xgb_model
)

xgb.plot.importance(importance_matrix, top_n = 20)
```



2.4.2 Grid-search and Hyper tuning Parameters

Since the following code would take too long to run we will not be running it. However, the below is how it was run actually while building the model. We will then hard-code the values for the sake of documenting the outputs

```r
xgb_grid <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(3, 6, 9),
  eta = c(0.01, 0.1, 0.3),
  gamma = c(0, 1),
  colsample_bytree = c(0.6, 0.8, 1),
  min_child_weight = c(1, 3, 5),
  subsample = c(0.5, 0.75, 1)
)

cluster <- makeCluster(detectCores() - 2)  # Use all cores except one
registerDoParallel(cluster)

ctrl <- trainControl(
```

```
  method = "cv",
  number = 5,
  allowParallel = TRUE,
  verboseIter = TRUE
)


# Convert to caret-compatible format
train_df <- as.data.frame(train_matrix) %>%
  mutate(label = log(dtrain$salary))

set.seed(42)

cluster <- makeCluster(detectCores() - 3)  # Use all cores except 3
registerDoParallel(cluster)

xgb_tuned <- train(
  label ~ .,
  data = train_df,
  method = "xgbTree",
  trControl = ctrl,
  tuneGrid = xgb_grid,
  metric = "RMSE"
)

stopCluster(cluster)
# Best parameters
best_params <- xgb_tuned$bestTune

saveRDS(best_params, file="bestparams.RData")

best_params <- readRDS("bestparams.RData")
```

Now we hard code the values for documentation:

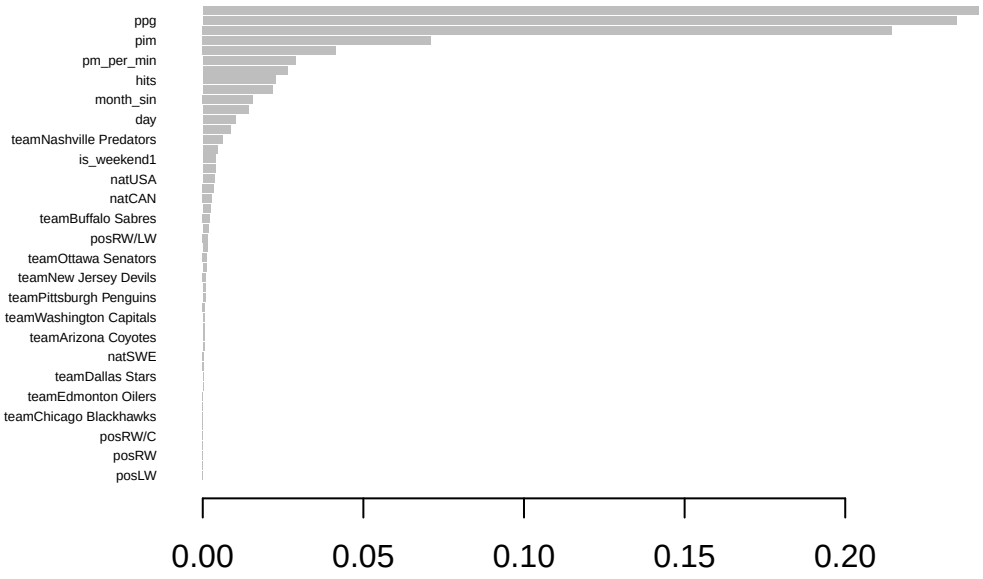
```
best_params <- data.frame(t(c(nrounds = 200,
                              max_depth = 3,
                              eta = 0.1,
                              gamma = 0,
                              colsample_bytree = 0.6,
                              min_child_weight = 3,
                              subsample = 0.75)),
                          row.names = 334)

final_params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = best_params$eta,
  max_depth = best_params$max_depth,
  gamma = best_params$gamma,
  subsample = best_params$subsample,
  colsample_bytree = best_params$colsample_bytree,
  min_child_weight = best_params$min_child_weight,
  lambda = best_params$min_child_weight
```

```
)

final_model <- xgb.train(
  params = final_params,
  data = xgb_dtrain,
  nrounds = best_params$nrounds
)

# Evaluate feature importance
importance_matrix <- xgb.importance(feature_names = colnames(train_matrix),
                                    model = final_model)
xgb.plot.importance(importance_matrix)
```



2.4.3 Feature Selection

We choose the features that have a greater than 0 importance:

```
chosen_features <- importance_matrix[importance_matrix$Importance > 0]$Feature

# Prepare data for XGBoost
xgb.dtrain <- xgb.DMatrix(data = train_matrix[, chosen_features] ,
                          label = train_labels)
xgb.dtest <- xgb.DMatrix(data = test_matrix[, chosen_features])

xgb_select_model <- xgb.train(
  params = final_params,
  data = xgb.dtrain,
  nrounds = best_params$nrounds
)
```

2.4.4 Adding additional features to XGBoost Model

We add some additional features to the above XGBoost Model and check their multi-collinearity

```
test_model <- lm(salary~., data = new_dtrain)
car::vif(test_model)
```

```
##                      GVIF Df GVIF^(1/(2*Df))
## team             19.752199 30        1.050978
## cap              51.485278  1        7.175324
## gp                4.269930  1        2.066381
## injure            1.267853  1        1.125990
## toi               4.691406  1        2.165965
## pos              27.947157 17        1.102908
## nat              11.324844 15        1.084262
## age               1.310239  1        1.144657
## year             54.080962  1        7.353976
## day               1.174529  1        1.083757
## is_weekend        1.173774  1        1.083409
## month_sin         2.304617  1        1.518096
## month_cos         2.092754  1        1.446635
## npc               1.806704  1        1.344137
## pm_per_min        1.372763  1        1.171650
## ppg               2.964617  1        1.721806
## physicality       2.374023  1        1.540787
## relCorsi_per_toi  1.156012  1        1.075180
```

```
newer_model <- lm(salary~team+gp+toi+pos+nat+age+injure+
                    day+is_weekend+month_sin+month_cos+npc+
                    pm_per_min+ppg+physicality+relCorsi_per_toi,
                  data = new_dtrain)
car::vif(newer_model)
```

```
##                      GVIF Df GVIF^(1/(2*Df))
## team             17.046683 30        1.048401
## gp                4.252851  1        2.062244
## toi               4.684354  1        2.164337
## pos              24.626138 17        1.098812
## nat              10.175860 15        1.080403
## age               1.285879  1        1.133966
## injure            1.261817  1        1.123306
## day               1.173999  1        1.083512
## is_weekend        1.173182  1        1.083135
## month_sin         1.224707  1        1.106665
## month_cos         1.248219  1        1.117237
## npc               1.773437  1        1.331704
## pm_per_min        1.339147  1        1.157215
## ppg               2.950532  1        1.717711
## physicality       2.369771  1        1.539406
## relCorsi_per_toi  1.146536  1        1.070764
```

```
new_formula_full <- "log(salary)~team+toi+gp+pos+nat+age+injure+
                    day+is_weekend+month_sin+month_cos+npc+
                    pm_per_min+ppg+physicality+relCorsi_per_toi"

new_train_matrix <- model.matrix(as.formula(new_formula_full),
                                 data = new_dtrain)[,-1]  # Remove intercept
new_train_labels <- log(new_dtrain$salary)

new_xgb_dtrain <- xgb.DMatrix(data = new_train_matrix, label = new_train_labels)
```

```
new_test_matrix <- model.matrix(as.formula(new_formula_full), data = new_dtest)[,-1]

# Handle any missing features (set to 0)
missing_cols <- setdiff(colnames(new_train_matrix), colnames(new_test_matrix))
new_test_matrix <- cbind(new_test_matrix, matrix(0, nrow = nrow(new_test_matrix),
                                        ncol = length(missing_cols),
                                        dimnames = list(NULL, missing_cols)))

# Ensure column order matches training data
new_test_matrix <- new_test_matrix[, colnames(new_train_matrix)]
new_xgb_dtest <- xgb.DMatrix(data = new_test_matrix)
```

2.4.5 Additional Feature selection

We feature select again based on the positive feature importances:

```
new_xgb_model <- xgb.train(
  params = final_params,
  data = new_xgb_dtrain,
  nrounds = best_params$nrounds
)

# Evaluate feature importance
new_importance_matrix <- xgb.importance(feature_names = colnames(new_train_matrix),
                                    model = new_xgb_model)

new_chosen_features <- new_importance_matrix[new_importance_matrix$Gain > 0]$Feature

# Prepare data for XGBoost
new_xgb.dtrain <- xgb.DMatrix(data = new_train_matrix[, new_chosen_features] ,
                          label = new_train_labels)
new_xgb.dtest <- xgb.DMatrix(data = new_test_matrix[, new_chosen_features])

new_xgb_select_model <- xgb.train(
  params = final_params,
  data = new_xgb.dtrain,
  nrounds = best_params$nrounds
)
```

2.4.6 Final Hypertuning

We hypertune the parameters once again with the new variates. Again, we won't run it in the Markdown file, but the code is given below. We will later hardcode the values.

```
final_xgb_grid <- expand.grid(
  nrounds = c(50, 100, 150, 200),
  max_depth = c(1,2,3,4),
  eta = c(0.8, 0.1, 0.15),
  gamma = c(0, 0.5),
  colsample_bytree = 1,
  min_child_weight = c(2, 3, 4),
  subsample = 1
)

cluster <- makeCluster(detectCores() - 3)  # Use all cores except one
```

```r
registerDoParallel(cluster)

final_ctrl <- trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE,
  verboseIter = TRUE
)


# Convert to caret-compatible format
train_df <- as.data.frame(train_matrix) %>%
  mutate(label = log(dtrain$salary))

set.seed(42)
new_xgb_tuned <- train(
  label ~ .,
  data = train_df,
  method = "xgbTree",
  trControl = final_ctrl,
  tuneGrid = final_xgb_grid,
  metric = "RMSE"
)

stopCluster(cluster)
#Best parameters
new_best_params <- new_xgb_tuned$bestTune

saveRDS(new_best_params, file="newbestparams.RData")

new_best_params <- readRDS("newbestparams.RData")
```

We will now hardcode the values as follows:

```r
new_best_params <- data.frame(t(c(nrounds = 200,
                                  max_depth = 3,
                                  eta = 0.1,
                                  gamma = 0,
                                  colsample_bytree = 1,
                                  min_child_weight = 4,
                                  subsample = 1)),
                              row.names = 60)

new_final_params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = new_best_params$eta,
  max_depth = new_best_params$max_depth,
  gamma = new_best_params$gamma,
  subsample = new_best_params$subsample,
  colsample_bytree = new_best_params$colsample_bytree,
  min_child_weight = new_best_params$min_child_weight
)
```

```
new_xgb_cv <- xgb.cv(
  params = new_final_params,
  data = xgb_dtrain,
  nrounds = 1000,
  nfold = 10,
  early_stopping_rounds = 100,
  print_every_n = 50
)
```

```
## [1]  train-rmse:12.159312+0.007999   test-rmse:12.159091+0.080181
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 100 rounds.
##
## [51] train-rmse:0.259648+0.004540    test-rmse:0.343959+0.036556
## [101]    train-rmse:0.195825+0.005432    test-rmse:0.326678+0.040706
## [151]    train-rmse:0.164879+0.005233    test-rmse:0.325545+0.043205
## [201]    train-rmse:0.140897+0.004355    test-rmse:0.325967+0.042846
## [251]    train-rmse:0.121527+0.003872    test-rmse:0.325921+0.043263
## Stopping. Best iteration:
## [164]    train-rmse:0.158180+0.004932    test-rmse:0.324953+0.043594
```

```
# Train final model
xgb_model_final <- xgb.train(
  params = new_final_params,
  data = new_xgb.dtrain,
  nrounds = new_best_params$nrounds
)
```

This is how we arrive at the final model, i.e., `xgb_model_final` is the model that we finally select.

The models were evaluated based on RMLSE on both the predictions training data and testing data. RMLSE for training data was calculated as:

```
sqrt(mean((log(dtrain$salary) - (predict(xgb_model_final, new_xgb.dtrain)))^2))
```

```
## [1] 1.09246
```

Whereas for the testing dataset, the RMLSE was calculated on Kaggle.

This evaluation was done for all the models that were built (not just the XGBoost models) and the best model turned out to be xgb_model_final, which was what was submitted for evaluation.

3.Evaluation

```
rm(list=ls())
# first set working directory to the data location
load("final.Rdata")
# final.Rdata contains dtrain and dtest

# load user-defined FinalModel function
source("20873217.R")
# call the function
tmp <- system.time(res <- FinalModel(dtrain, dtest))
# time used, this is the "Running_time" you report in your R source file
tmp[3]
```

```
## elapsed
##    0.33
```

```
dim(res)
```

```
## [1] 400   2
```

```
# should be 400 by 2

head(res)
```

```
##   Id    salary
## 1  1  544094.7
## 2  2  984703.2
## 3  3  612803.1
## 4  4 5677740.7
## 5  5 5718278.5
## 6  6  808645.6
```