Obtaining Pitching+ and Stuff+ for TrackMan and Statcast Data

Riku Komatani

2023-02-03

This Rmd file contains functions that generates Pitching+ and Stuff+; quantities that evaluate how effective a pitch is based on pitch flight metrics. We used Statcast data from 2019 and 2021, and Illini TrackMan Baseball Data from 2022 ~ 2023 in the following codes.

Importing Libraries

```
#Libraries needed
library(tidyverse)
library(mgcv)
library(broom)
library(ggplot2)
library(ggpubr)
library(Lahman)
library(ranger)
library(caret)
library(retrosheet)
library(stringr)
library(vctrs)
library(rsample)
library(gbm)
library(ROCR)
library(data.table)
library(illinibaseball)
library(DBI)
library(RMySQL)
```

Steps 1 ~ 5: Creating models for different pitch types using 2019 MLB Statcast data.

Step 1: Getting 2019 MLB Statcast data ready.

```
#Import data: Use any season after 2015 from Statcast
sc_2019 <- read_csv("C:/Users/12244/STAT430/statcast/2019.csv")

#Filter data to only inplay or swing and miss
sc_2019_1 <- sc_2019 %>%
  filter(description %in% c("hit_into_play", "swinging_strike", "swinging_strike_blocked"))


#Assign 1 for inplay and 0 for swing and miss
sc_2019_2 <- sc_2019_1 %>%
  mutate(play = case_when(description == "hit_into_play" ~ 1,
                          description %in% c("swinging_strike", "swinging_strike_blocked") ~ 0,
                          TRUE ~ 0))


#Adjust horizontal data to account for R and L pitchers
sc_2019_3 <- sc_2019_2 %>%
  mutate(release_pos_x_adj = ifelse(p_throws == "R", release_pos_x, - release_pos_x),
         pfx_x_adj = ifelse(p_throws == "R", pfx_x, - pfx_x),
         spin_axis_adj = ifelse(p_throws == "R", spin_axis, 360 - spin_axis)) %>%
  filter(!is.na(release_speed),
         !is.na(release_pos_z),
         !is.na(release_pos_x_adj),
         !is.na(release_extension),
         !is.na(pfx_x_adj),
         !is.na(pfx_z),
         !is.na(release_spin_rate),
         !is.na(spin_axis_adj)) %>%
  mutate(play = as.factor(play))
```

Step 2: Adding velocity and movement differences from pitcher's fastest pitch type for offspeed pitches onto 2019 MLB Statcast data.

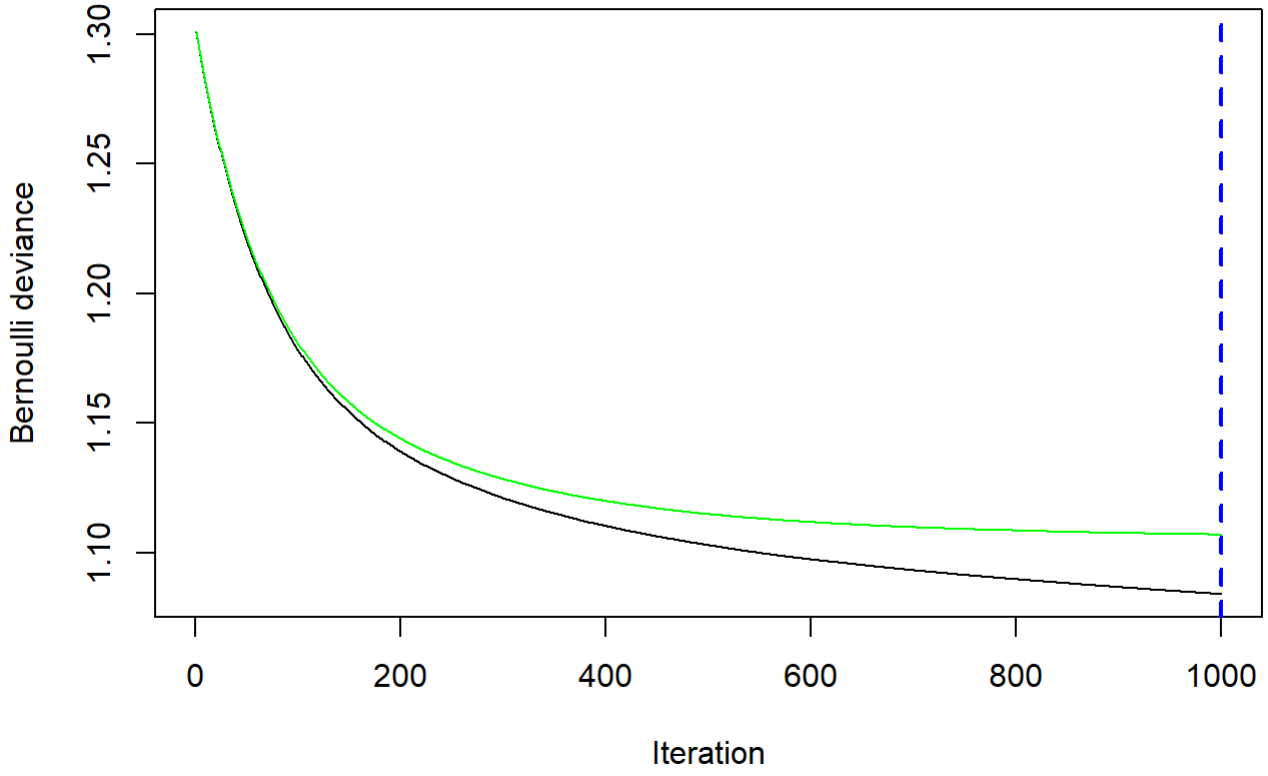
```
#Obtain velocity and movement difference from pitcher's fastest pitch type for non fastest pitch
types by using data table
dt_19 <- setDT(sc_2019_3)

dt1_19 <- data.table(pitcher = 0, fast_velo = 0, fast_pfx_x_adj = 0, fast_pfx_z = 0, fastest_pit
ch_type = " ")

for (p in unique(dt_19[["pitcher"]])) {
  data_p <- dt_19[pitcher == p]
  if ("FF" %chin% unique(data_p)[["pitch_type"]]) {
    data_FF <- data_p[pitch_type == "FF"]
    dt1_19 <- rbindlist(list(dt1_19, list(p, mean(data_FF[["release_speed"]]),
                                          mean(data_FF[["pfx_x_adj"]]), mean(data_FF[["pfx_
z"]]), "FF")))

  } else if ("SI" %chin% unique(data_p)[["pitch_type"]]) {
    data_SI <- data_p[pitch_type == "SI"]
    dt1_19 <- rbindlist(list(dt1_19, list(p, mean(data_SI[["release_speed"]]),
                                          mean(data_SI[["pfx_x_adj"]]), mean(data_SI[["pfx_
z"]]), "SI")))

  } else if ("FC" %chin% unique(data_p)[["pitch_type"]]) {
    data_FC <- data_p[pitch_type == "FC"]
    dt1_19 <- rbindlist(list(dt1_19, list(p, mean(data_FC[["release_speed"]]),
                                          mean(data_FC[["pfx_x_adj"]]), mean(data_FC[["pfx_
z"]]), "FC")))
  }
}

dt1_19 <- dt1_19[-1,]

#Data frame with pitcher and his fastest pitch info
data_pitcher_fastest <- as.data.frame(dt1_19)

dt2_19 <- merge(dt_19, dt1_19, all.x = TRUE, by = "pitcher")

dt3_19 <- dt2_19[, ':='(release_speed_diff = fifelse(pitch_type != fastest_pitch_type, release_s
peed - fast_velo, -Inf),
                        xmov_diff_adj = fifelse(pitch_type != fastest_pitch_type, pfx_x_adj - fa
st_pfx_x_adj, -Inf),
                        zmov_diff = fifelse(pitch_type != fastest_pitch_type, pfx_z - fast_pfx_
z, -Inf))]

sc_2019_4 <- as.data.frame(dt3_19)
```

Step 3: Separating 2019 MLB Statcast data to different pitch types.

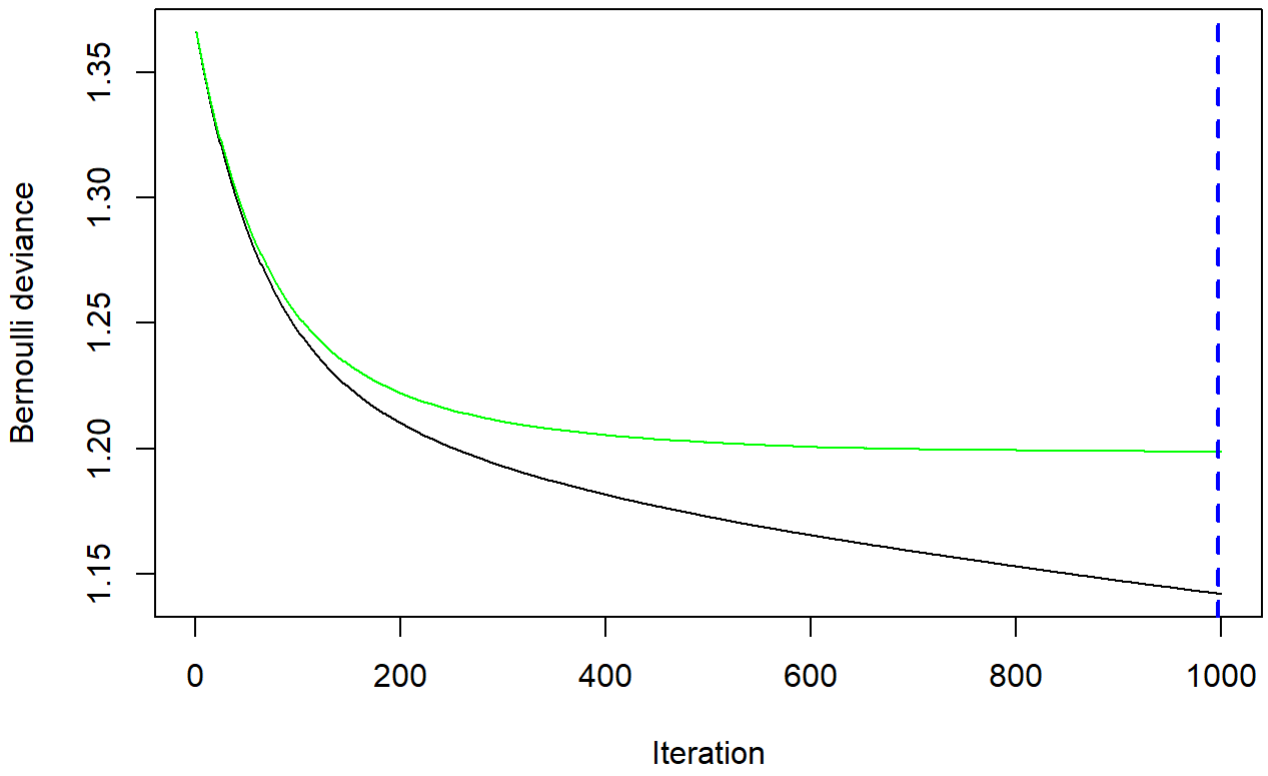
```
#Assign data to different pitch types
sc_2019_FF <- sc_2019_4 %>%
  filter(pitch_type == "FF")

sc_2019_CH <- sc_2019_4 %>%
  filter(pitch_type == "CH")

sc_2019_FC <- sc_2019_4 %>%
  filter(pitch_type == "FC")

sc_2019_CU <- sc_2019_4 %>%
  filter(pitch_type == "CU")

sc_2019_SI <- sc_2019_4 %>%
  filter(pitch_type == "SI")

sc_2019_SL <- sc_2019_4 %>%
  filter(pitch_type == "SL")
```

Step 4: Creating Models for Pitching+ (probability of contact based on pitch flight metrics with location) for each pitch types using 2019 MLB Statcast data. (This might take a while to run ~ 5min).

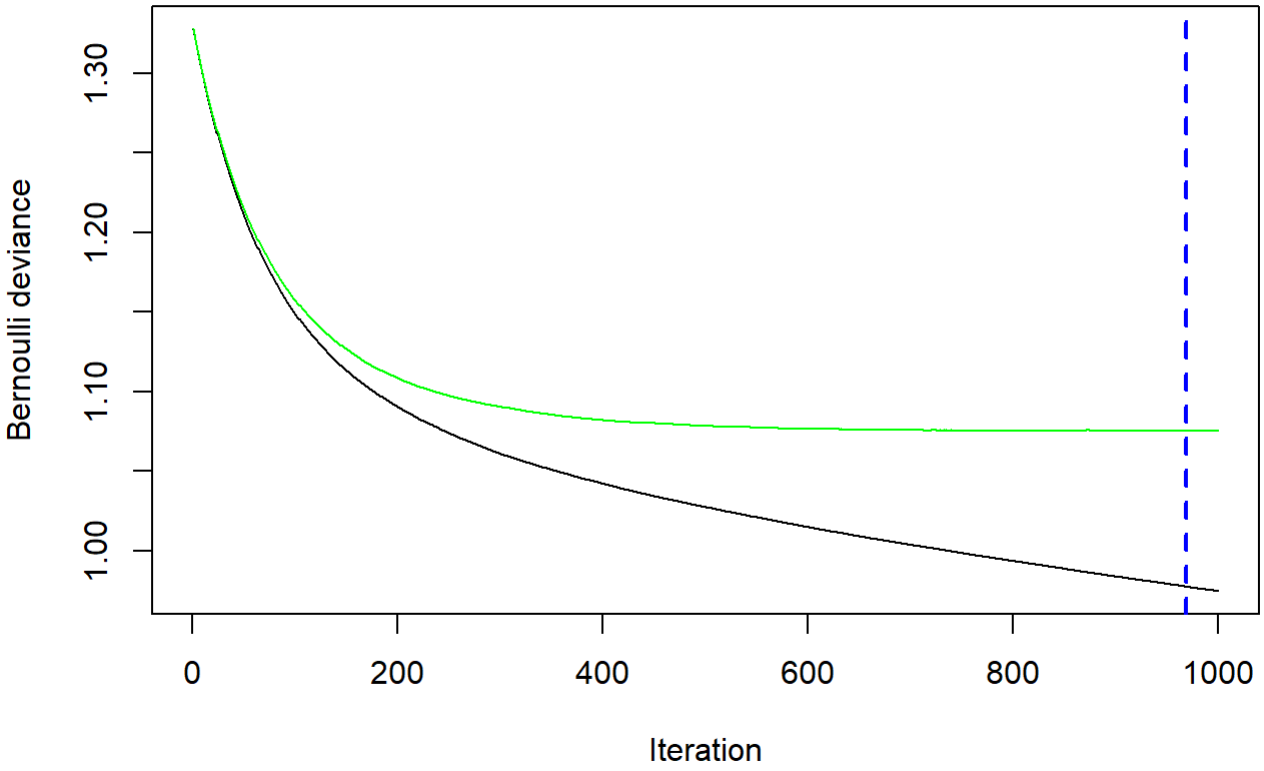
```
###With Location: Swing probability based on location and pitch quality

Pitch_columns <- c("play", "release_speed", "release_pos_z", "release_pos_x_adj",
                   "release_extension", "pfx_x_adj", "pfx_z", "release_spin_rate", "spin_axis_ad
j",
                   "plate_x", "plate_z",
                   "release_speed_diff", "xmov_diff_adj", "zmov_diff")

###############FF
#Select only useful columns
df_19_FF <- sc_2019_FF %>%
  select(Pitch_columns[-c(12, 13, 14)])

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_FF$play <- as.logical(as.integer(df_19_FF$play)-1)

##Binary model
SimpleGBMModel_FF <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_FF,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_FF <- gbm.perf(SimpleGBMModel_FF, method = "cv")
```



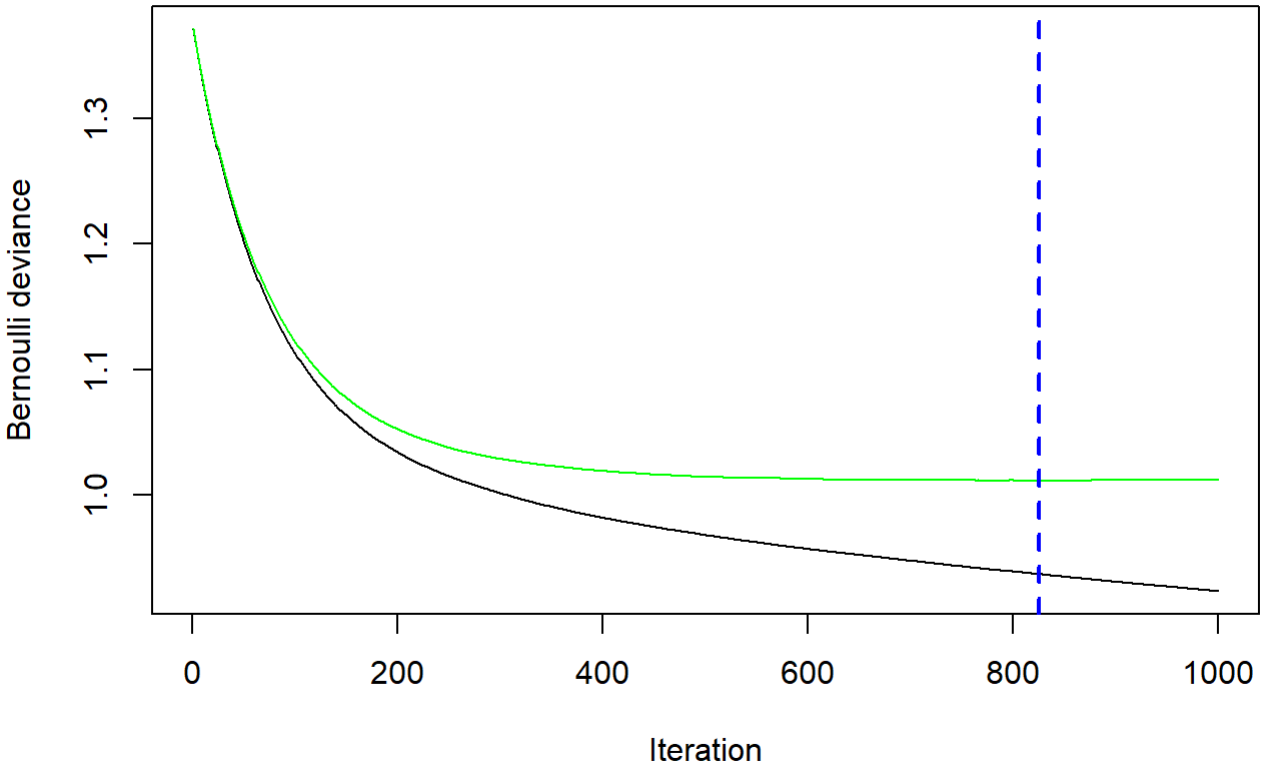

```
##############CH
#Select only useful columns
df_19_CH <- sc_2019_CH %>%
  select(all_of(Pitch_columns))

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_CH$play <- as.logical(as.integer(df_19_CH$play)-1)

##Binary model
SimpleGBMModel_CH <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_CH,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_CH <- gbm.perf(SimpleGBMModel_CH, method = "cv")
```



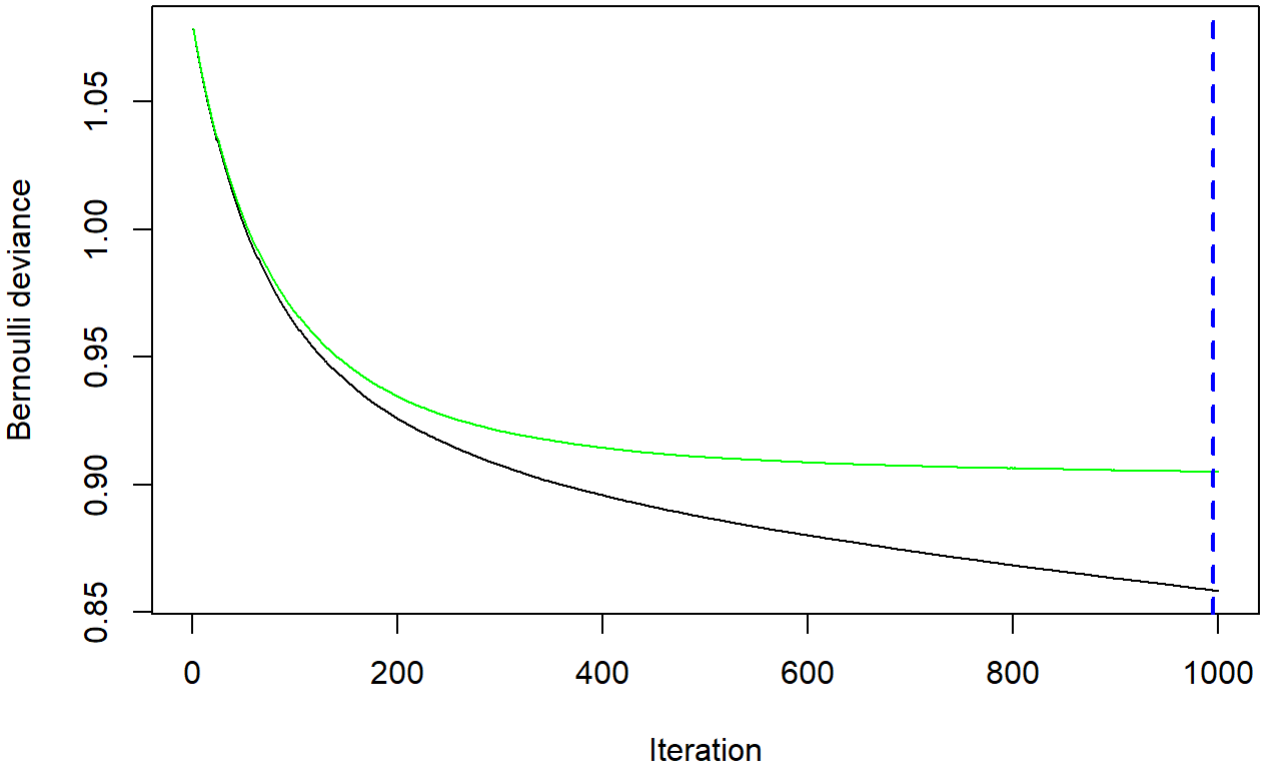

```
##########################FC
#Select only useful columns
df_19_FC <- sc_2019_FC %>%
  select(all_of(Pitch_columns))

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_FC$play <- as.logical(as.integer(df_19_FC$play)-1)

##Binary model
SimpleGBMModel_FC <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_FC,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_FC <- gbm.perf(SimpleGBMModel_FC, method = "cv")
```



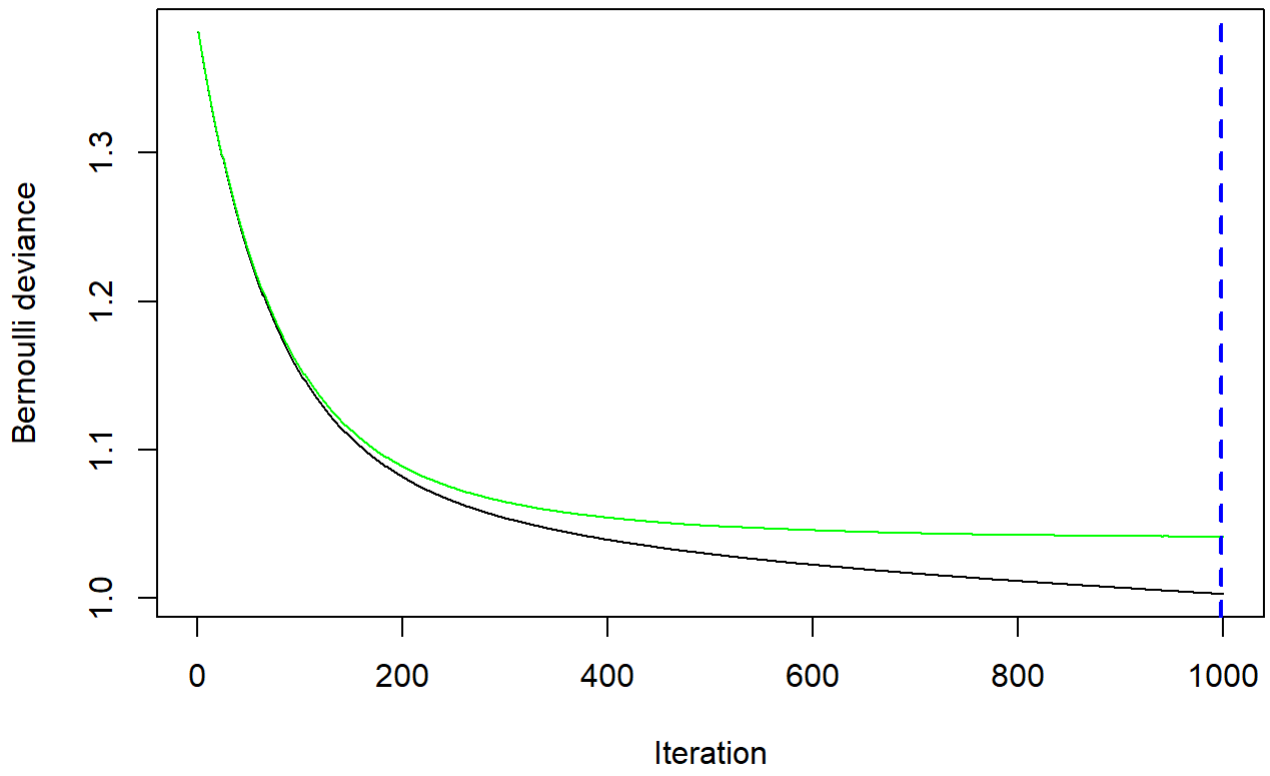

```
##########################CU
#Select only useful columns
df_19_CU <- sc_2019_CU %>%
  select(all_of(Pitch_columns))

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_CU$play <- as.logical(as.integer(df_19_CU$play)-1)

##Binary model
SimpleGBMModel_CU <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_CU,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_CU <- gbm.perf(SimpleGBMModel_CU, method = "cv")
```




```
##########################SI
#Select only useful columns
df_19_SI <- sc_2019_SI %>%
  select(all_of(Pitch_columns))

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_SI$play <- as.logical(as.integer(df_19_SI$play)-1)

##Binary model
SimpleGBMModel_SI <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_SI,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_SI <- gbm.perf(SimpleGBMModel_SI, method = "cv")
```

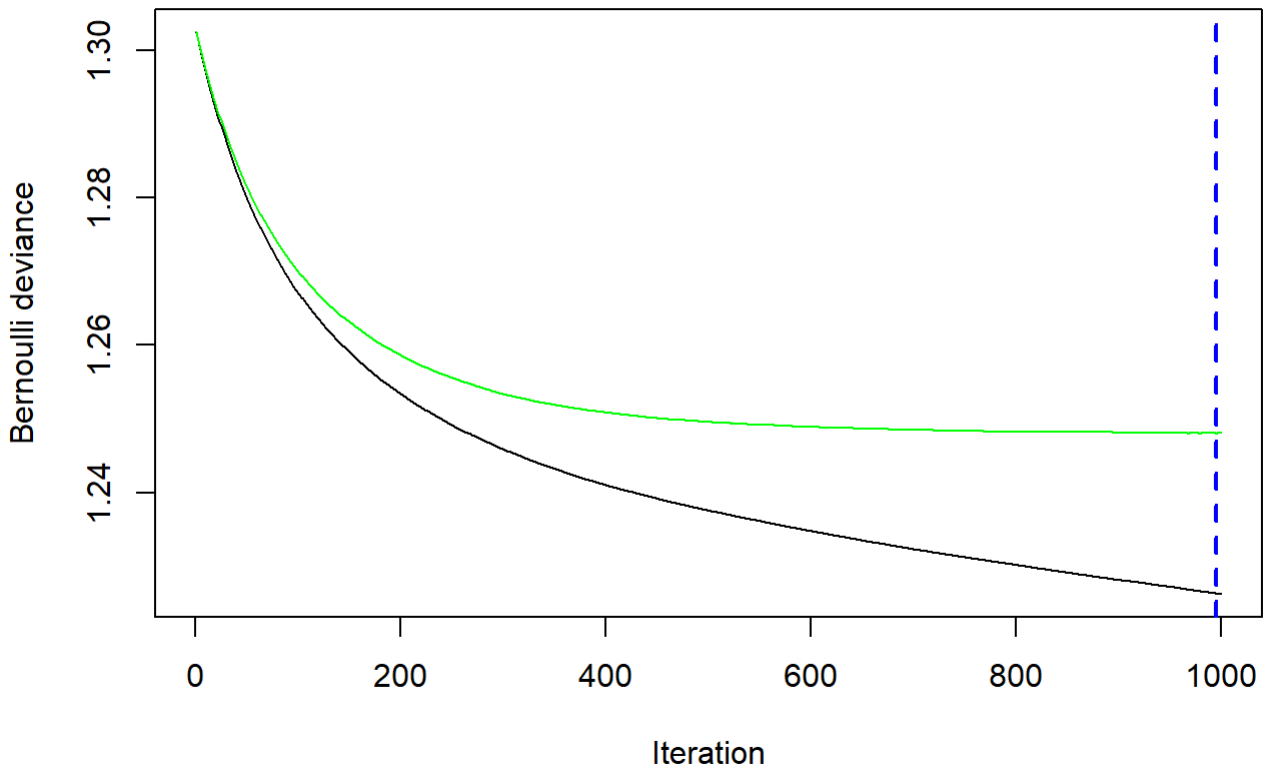



```
##########################SL
#Select only useful columns
df_19_SL <- sc_2019_SL %>%
  select(all_of(Pitch_columns))

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_SL$play <- as.logical(as.integer(df_19_SL$play)-1)

##Binary model
SimpleGBMModel_SL <- gbm(formula = play ~ . ,
                         distribution = "bernoulli",
                         data = df_19_SL,
                         n.trees = 1000,
                         #maximum depth of each tree
                         interaction.depth = 10,
                         #learning rate
                         shrinkage = 0.008,
                         cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_SL <- gbm.perf(SimpleGBMModel_SL, method = "cv")
```

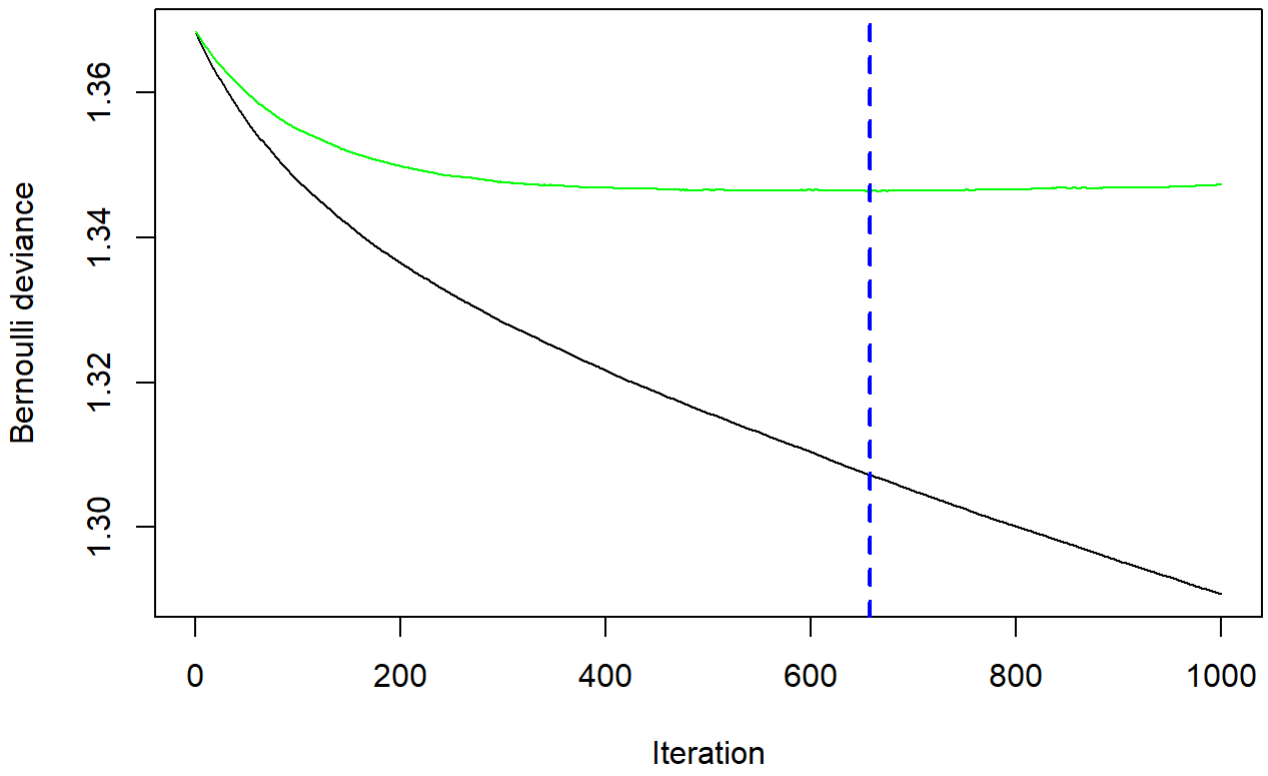


Step 5: Creating Models for Stuff+ (probability of contact based on pitch flight metrics without location) for each pitch types using 2019 MLB Statcast data. (This might take a while to run ~ 5min).

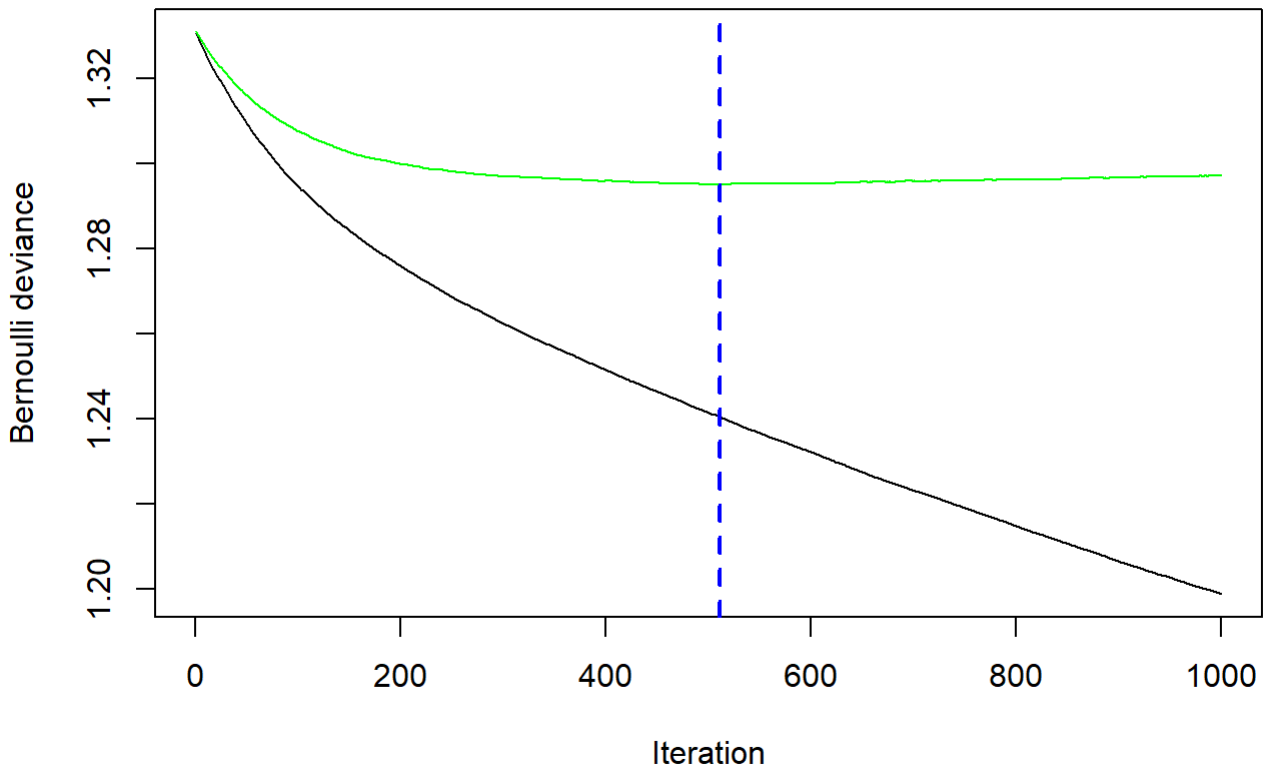
```
#################Without location: Swing probability only based on pitch quality

Stuff_columns <- c("play", "release_speed", "release_pos_z", "release_pos_x_adj",
                   "release_extension", "pfx_x_adj", "pfx_z", "release_spin_rate", "spin_axis_ad
j",
                   "release_speed_diff", "xmov_diff_adj", "zmov_diff")
########################FF
#Select only useful columns
df_19_FF_2 <- sc_2019_FF %>%
  select(Stuff_columns[-c(10, 11, 12)])

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_FF_2$play <- as.logical(as.integer(df_19_FF_2$play)-1)

##Binary model
SimpleGBMModel_FF_2 <- gbm(formula = play ~ . ,
                        distribution = "bernoulli",
                        data = df_19_FF_2,
                        n.trees = 1000,
                        #maximum depth of each tree
                        interaction.depth = 10,
                        #learning rate
                        shrinkage = 0.008,
                        cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_FF_2 <- gbm.perf(SimpleGBMModel_FF_2, method = "cv")
```



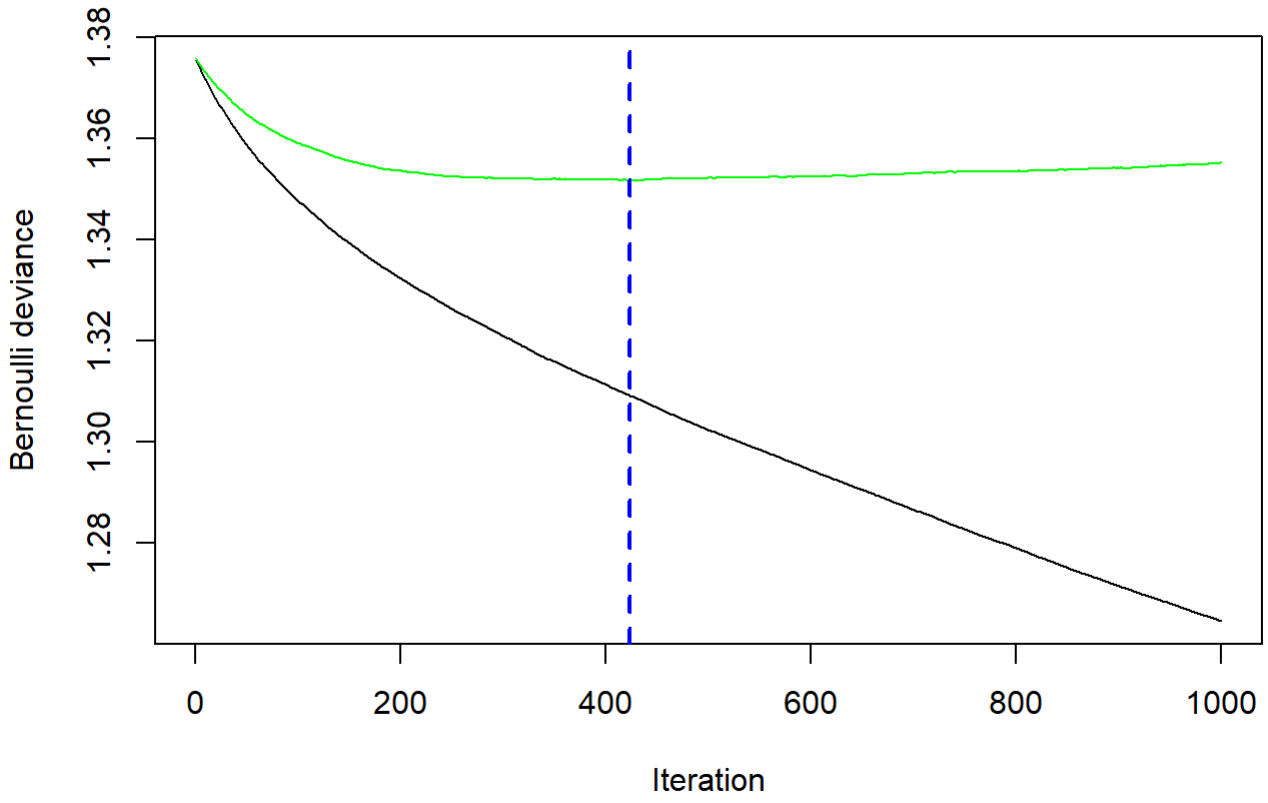

```
####################CH
#Select only useful columns
df_19_CH_2 <- sc_2019_CH %>%
  select(Stuff_columns)

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_CH_2$play <- as.logical(as.integer(df_19_CH_2$play)-1)

##Binary model
SimpleGBMModel_CH_2 <- gbm(formula = play ~ . ,
                           distribution = "bernoulli",
                           data = df_19_CH_2,
                           n.trees = 1000,
                           #maximum depth of each tree
                           interaction.depth = 10,
                           #learning rate
                           shrinkage = 0.008,
                           cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_CH_2 <- gbm.perf(SimpleGBMModel_CH_2, method = "cv")
```



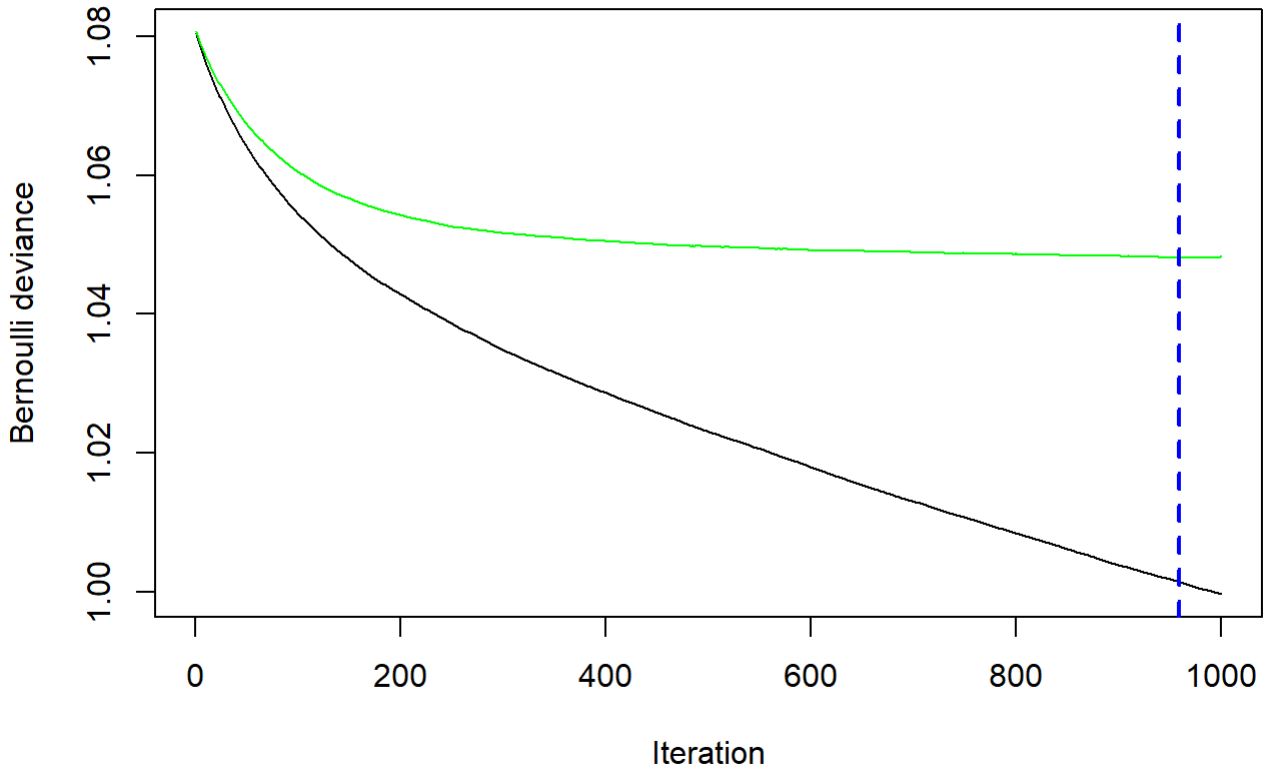

```
####################FC
#Select only useful columns
df_19_FC_2 <- sc_2019_FC %>%
  select(Stuff_columns)

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_FC_2$play <- as.logical(as.integer(df_19_FC_2$play)-1)

##Binary model
SimpleGBMModel_FC_2 <- gbm(formula = play ~ . ,
                           distribution = "bernoulli",
                           data = df_19_FC_2,
                           n.trees = 1000,
                           #maximum depth of each tree
                           interaction.depth = 10,
                           #learning rate
                           shrinkage = 0.008,
                           cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_FC_2 <- gbm.perf(SimpleGBMModel_FC_2, method = "cv")
```



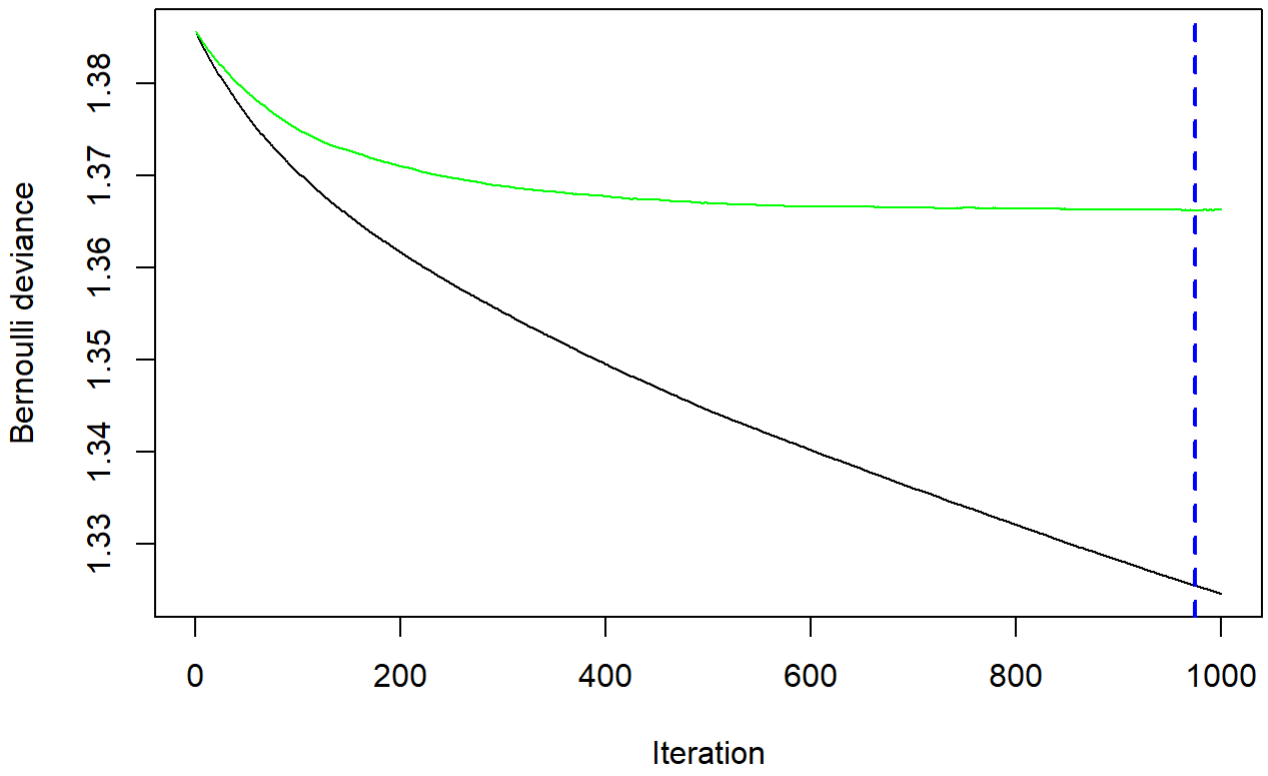

```
####################CU
#Select only useful columns
df_19_CU_2 <- sc_2019_CU %>%
  select(Stuff_columns)

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_CU_2$play <- as.logical(as.integer(df_19_CU_2$play)-1)

##Binary model
SimpleGBMModel_CU_2 <- gbm(formula = play ~ . ,
                           distribution = "bernoulli",
                           data = df_19_CU_2,
                           n.trees = 1000,
                           #maximum depth of each tree
                           interaction.depth = 10,
                           #learning rate
                           shrinkage = 0.008,
                           cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_CU_2 <- gbm.perf(SimpleGBMModel_CU_2, method = "cv")
```




```
####################SI
#Select only useful columns
df_19_SI_2 <- sc_2019_SI %>%
  select(Stuff_columns)

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_SI_2$play <- as.logical(as.integer(df_19_SI_2$play)-1)

##Binary model
SimpleGBMModel_SI_2 <- gbm(formula = play ~ . ,
                           distribution = "bernoulli",
                           data = df_19_SI_2,
                           n.trees = 1000,
                           #maximum depth of each tree
                           interaction.depth = 10,
                           #learning rate
                           shrinkage = 0.008,
                           cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_SI_2 <- gbm.perf(SimpleGBMModel_SI_2, method = "cv")
```




```
####################SL
#Select only useful columns
df_19_SL_2 <- sc_2019_SL %>%
  select(Stuff_columns)

#Mutate play as TRUE if in play and FALSE if swing and miss
df_19_SL_2$play <- as.logical(as.integer(df_19_SL_2$play)-1)

##Binary model
SimpleGBMModel_SL_2 <- gbm(formula = play ~ . ,
                           distribution = "bernoulli",
                           data = df_19_SL_2,
                           n.trees = 1000,
                           #maximum depth of each tree
                           interaction.depth = 10,
                           #learning rate
                           shrinkage = 0.008,
                           cv.folds = 4)

#Optimal ntree number
ntree_opt_cv_SL_2 <- gbm.perf(SimpleGBMModel_SL_2, method = "cv")
```



Steps 6 ~ 8: Adding Pitching+ and Stuff+ to 2021 MLB data.

Step 6: Function used in Step 7 to convert predicted probability of contact to Pitching+ and Stuff+ values (The input data for this function needs to be at least a full season of MLB data to normalize properly and obtain meaningful results).

```
#Function to convert predicted probability of contact to Pitching+ and Stuff+
prob_to_pitch_stuff <- function(data, pred_pitch, pred_stuff) {
  #Adding Pitching+ to data
  data$prob_contact_by_pitch <- pred_pitch

  z_score_pitch <- scale(data$prob_contact_by_pitch, center = median(data$prob_contact_by_pitc
h))

  percentile_pitch <- sapply(z_score_pitch, pnorm)

  percentile_pitch_adj <- 200 - percentile_pitch * 100 * 2

  data$Pitching_plus = percentile_pitch_adj

  #Adding Stuff+ to data
  data$prob_contact_by_stuff <- pred_stuff

  z_score_stuff <- scale(data$prob_contact_by_stuff, center = median(data$prob_contact_by_stuf
f))

  percentile_stuff <- sapply(z_score_stuff, pnorm)

  percentile_stuff_adj <- 200 - percentile_stuff * 100 * 2

  data$Stuff_plus = percentile_stuff_adj

  return(data)
}
```

Step 7: Function for adding Pitching+ and Stuff+ to MLB data using models created in Step 3 and 4.

```
#Function that adds Pitching plus and Stuff plus into data
pitching_stuff_generator <- function(df) {

  #Adjust horizontal data to account for R and L pitchers, filter data to inplay or swing and mi
ss, remove na value
  df_valid <- df %>%
    mutate(release_pos_x_adj = ifelse(p_throws == "R", release_pos_x, - release_pos_x),
           pfx_x_adj = ifelse(p_throws == "R", pfx_x, - pfx_x),
           spin_axis_adj = ifelse(p_throws == "R", spin_axis, 360 - spin_axis)) %>%
    filter(description %in% c("hit_into_play", "swinging_strike", "swinging_strike_blocked"),
           !is.na(release_speed),
           !is.na(release_pos_z),
           !is.na(release_pos_x_adj),
           !is.na(release_extension),
           !is.na(pfx_x_adj),
           !is.na(pfx_z),
           !is.na(release_spin_rate),
           !is.na(spin_axis_adj))

  #Add pitcher's fastest pitch type info to data using pitcher's 2019 season info
  df_valid_with_fastest <- left_join(df_valid, data_pitcher_fastest)

  #Add velocity and movements difference of pitcher's data
  df_valid_with_fastest_diff <- df_valid_with_fastest %>%
    mutate(
      release_speed_diff = ifelse(pitch_type != fastest_pitch_type, release_speed - fast_velo, -
Inf),
      xmov_diff_adj = ifelse(pitch_type != fastest_pitch_type, pfx_x_adj - fast_pfx_x_adj, -In
f),
      zmov_diff = ifelse(pitch_type != fastest_pitch_type, pfx_z - fast_pfx_z, -Inf))

  #Obtaining probability of contact with location
  ################ FF
  df_FF <- df_valid_with_fastest_diff %>%
    filter(pitch_type == "FF")

  Predictions_FF <- predict(object = SimpleGBMModel_FF,
                            newdata = df_FF,
                            n.trees = ntree_opt_cv_FF,
                            type = "response")

  df_FF$prob_contact_by_pitch <- Predictions_FF

  #Obtaining probability of contact without location
  ################ FF
  Predictions_FF_2 <- predict(object = SimpleGBMModel_FF_2,
                              newdata = df_FF,
                              n.trees = ntree_opt_cv_FF_2,
                              type = "response")

  df_FF$prob_contact_by_stuff <- Predictions_FF_2
```

```
#Repeat the process for offspeed pitches

################ CH
df_CH <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "CH")

Predictions_CH <- predict(object = SimpleGBMModel_CH,
                          newdata = df_CH,
                          n.trees = ntree_opt_cv_CH,
                          type = "response")

df_CH$prob_contact_by_pitch <- Predictions_CH

Predictions_CH_2 <- predict(object = SimpleGBMModel_CH_2,
                            newdata = df_CH,
                            n.trees = ntree_opt_cv_CH_2,
                            type = "response")

df_CH$prob_contact_by_stuff <- Predictions_CH_2


################ FC
df_FC <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "FC")

Predictions_FC <- predict(object = SimpleGBMModel_FC,
                          newdata = df_FC,
                          n.trees = ntree_opt_cv_FC,
                          type = "response")

df_FC$prob_contact_by_pitch <- Predictions_FC

Predictions_FC_2 <- predict(object = SimpleGBMModel_FC_2,
                            newdata = df_FC,
                            n.trees = ntree_opt_cv_FC_2,
                            type = "response")

df_FC$prob_contact_by_stuff <- Predictions_FC_2

################ CU
df_CU <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "CU")

Predictions_CU <- predict(object = SimpleGBMModel_CU,
                          newdata = df_CU,
                          n.trees = ntree_opt_cv_CU,
                          type = "response")

df_CU$prob_contact_by_pitch <- Predictions_CU

Predictions_CU_2 <- predict(object = SimpleGBMModel_CU_2,
                            newdata = df_CU,
```

```
                            n.trees = ntree_opt_cv_CU_2,
                            type = "response")

df_CU$prob_contact_by_stuff <- Predictions_CU_2

############### SI
df_SI <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "SI")

Predictions_SI <- predict(object = SimpleGBMModel_SI,
                          newdata = df_SI,
                          n.trees = ntree_opt_cv_SI,
                          type = "response")

df_SI$prob_contact_by_pitch <- Predictions_SI

Predictions_SI_2 <- predict(object = SimpleGBMModel_SI_2,
                            newdata = df_SI,
                            n.trees = ntree_opt_cv_SI_2,
                            type = "response")

df_SI$prob_contact_by_stuff <- Predictions_SI_2

############### SL
df_SL <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "SL")

Predictions_SL <- predict(object = SimpleGBMModel_SL,
                          newdata = df_SL,
                          n.trees = ntree_opt_cv_SL,
                          type = "response")

df_SL$prob_contact_by_pitch <- Predictions_SL

Predictions_SL_2 <- predict(object = SimpleGBMModel_SL_2,
                            newdata = df_SL,
                            n.trees = ntree_opt_cv_SL_2,
                            type = "response")

df_SL$prob_contact_by_stuff <- Predictions_SL_2


#Changing probability of contact to Pitching+ and Stuff+
data_FF <- prob_to_pitch_stuff(df_FF, Predictions_FF, Predictions_FF_2)
data_CH <- prob_to_pitch_stuff(df_CH, Predictions_CH, Predictions_CH_2)
data_FC <- prob_to_pitch_stuff(df_FC, Predictions_FC, Predictions_FC_2)
data_CU <- prob_to_pitch_stuff(df_CU, Predictions_CU, Predictions_CU_2)
data_SI <- prob_to_pitch_stuff(df_SI, Predictions_SI, Predictions_SI_2)
data_SL <- prob_to_pitch_stuff(df_SL, Predictions_SL, Predictions_SL_2)
```

```
  rbind(data_FF, data_CH, data_FC, data_CU, data_SI, data_SL)
}
```

Step 8: Obtaining 2021 MLB Statcast data with contact probabilities using the function in Step 7.

```
#Obtain MLB 2021 season contact probabilities to compare and normalize with TrackMan data
sc_2021 <- read_csv("C:/Users/12244/STAT430/statcast/2021.csv")
mlb_2021 <- pitching_stuff_generator(sc_2021)
```

Steps 9 ~ 13: Getting Pitching+ and Stuff+ for TrackMan (college) data.

Step 9: Obtaining TrackMan data from UIUC database.

```
query = "SELECT * FROM tm_pitches
         WHERE Season = 2023 OR Season = 2022
         AND TaggedPitchType != 'Undefined';"
con = connect_db("uiuc")
pitches = dbGetQuery(con, query)
dbDisconnect(con)
```

```
## [1] TRUE
```

Step 10: Function for adjusting TrackMan data to match Statcast column names.

```
  #Auto or Tagged?
trackman_adjuster <- function(trackman_data) {
  #Adjust column names that are used to fit with statcast column names
  df_adjusted <- trackman_data %>%
    mutate(pitcher = Pitcher,
           pitch_type = case_when(AutoPitchType == "Fastball" ~ "FF",
                                  AutoPitchType == "Cutter" ~ "FC",
                                  AutoPitchType == "Slider" ~ "SL",
                                  AutoPitchType == "Curveball" ~ "CU",
                                  AutoPitchType == "Changeup" ~ "CH",
                                  AutoPitchType == "Sinker" ~ "SI"),
           p_throws = ifelse(PitcherThrows == "Right", "R", "L"),
           release_pos_x = - RelSide,
           release_pos_z = RelHeight,
           release_extension = Extension,
           release_speed = RelSpeed,
           pfx_x = - HorzBreak / 12,
           pfx_z = InducedVertBreak / 12,
           release_spin_rate = SpinRate,
           spin_axis = SpinAxis,
           plate_x = - PlateLocSide,
           plate_z = PlateLocHeight,
           description = case_when(PitchCall == "InPlay" ~ "hit_into_play",
                                   PitchCall == "StrikeSwinging" ~ "swinging_strike",
                                   TRUE ~ " "))
    #Adjust horizontal data to account for R and L pitchers, filter data to inplay or swing and
miss, remove na value
    df_valid <- df_adjusted %>%
    mutate(release_pos_x_adj = ifelse(p_throws == "R", release_pos_x, - release_pos_x),
           pfx_x_adj = ifelse(p_throws == "R", pfx_x, - pfx_x),
           spin_axis_adj = ifelse(p_throws == "R", spin_axis, 360 - spin_axis)) %>%
    filter(!is.na(release_speed),
           !is.na(release_pos_z),
           !is.na(release_pos_x_adj),
           !is.na(release_extension),
           !is.na(pfx_x_adj),
           !is.na(pfx_z),
           !is.na(release_spin_rate),
           !is.na(spin_axis_adj))
  return(df_valid)
}
```

Step 11: Function for adding player's fastest pitch info for TrackMan data (similar to Step 2).

```
#Use all track man data given in the past
fastest_pitch_info_getter <- function(trackman_data) {

  data_adjusted <- trackman_adjuster(trackman_data)

  dt <- setDT(data_adjusted)

  dt1 <- data.table(pitcher = 0, fast_velo = 0, fast_pfx_x_adj = 0, fast_pfx_z = 0, fastest_pitc
h_type = " ")

  for (p in unique(dt[["pitcher"]])) {
    data_p <- dt[pitcher == p]
    if ("FF" %chin% unique(data_p)[["pitch_type"]]) {
      data_FF <- data_p[pitch_type == "FF"]
      dt1 <- rbindlist(list(dt1, list(p, mean(data_FF[["release_speed"]]),
                                          mean(data_FF[["pfx_x_adj"]]), mean(data_FF[["pfx_
z"]]), "FF")))

    } else if ("SI" %chin% unique(data_p)[["pitch_type"]]) {
      data_SI <- data_p[pitch_type == "SI"]
      dt1 <- rbindlist(list(dt1, list(p, mean(data_SI[["release_speed"]]),
                                          mean(data_SI[["pfx_x_adj"]]), mean(data_SI[["pfx_
z"]]), "SI")))

    } else if ("FC" %chin% unique(data_p)[["pitch_type"]]) {
      data_FC <- data_p[pitch_type == "FC"]
      dt1 <- rbindlist(list(dt1, list(p, mean(data_FC[["release_speed"]]),
                                          mean(data_FC[["pfx_x_adj"]]), mean(data_FC[["pfx_
z"]]), "FC")))
    }
  }

  dt1 <- dt1[-1,]

  #Data frame with pitcher and his fastest pitch info
  trackman_data_pitcher_fastest <- as.data.frame(dt1)

  return(trackman_data_pitcher_fastest)
}

#Trackman data with player's fastest pitch info added
trackman_data_adjusted_with_pitcher_fastest <- fastest_pitch_info_getter(pitches)
```

Step 12: Function used in Step 13, which obtains the MLB scaled predicted contact probability of the input TrackMan data, by comparing it to 2021 MLB contact probabilities obtained in Step 8. This function also converts this predicted probability of contact to Pitching+ and Stuff+ values.

```
#Function to convert predicted probability of contact to Pitching+ and Stuff+
trackman_prob_to_pitch_stuff <- function(trackman_data, pred_pitch_trackman, pred_stuff_trackma
n) {

  #Return null if input data doesn't contain that pitch type
  if (nrow(trackman_data) == 0) {
    return(NA)
  }

  pitch = trackman_data$pitch_type[1]

  mlb_2021_by_pitch <- mlb_2021 %>%
    filter(pitch_type == pitch)

  contact_prob_by_pitch_2021 <- mlb_2021_by_pitch$prob_contact_by_pitch
  contact_prob_by_stuff_2021 <- mlb_2021_by_pitch$prob_contact_by_stuff

  # Adding Pitching+ to data
  pred_pitch_trackman_mlb <- c(pred_pitch_trackman, contact_prob_by_pitch_2021)

  z_score_pitch <- scale(pred_pitch_trackman_mlb, center = median(pred_pitch_trackman_mlb))

  percentile_pitch <- sapply(z_score_pitch, pnorm)

  percentile_pitch_adj <- 200 - percentile_pitch * 100 * 2

  #Extract percentiles from TrackMan data and add it to TrackMan data
  trackman_data$Pitching_plus = percentile_pitch_adj[1:nrow(trackman_data)]

  #Adding Stuff+ to data
  pred_stuff_trackman_mlb <- c(pred_stuff_trackman, contact_prob_by_stuff_2021)

  z_score_stuff <- scale(pred_stuff_trackman_mlb, center = median(pred_stuff_trackman_mlb))

  percentile_stuff <- sapply(z_score_stuff, pnorm)

  percentile_stuff_adj <- 200 - percentile_stuff * 100 * 2

  #Extract percentiles from TrackMan data and add it to TrackMan data
  trackman_data$Stuff_plus = percentile_stuff_adj[1:nrow(trackman_data)]

  return(trackman_data)
}
```

Step 13: Function for adding MLB scaled Pitching+ and Stuff+ to TrackMan data using the models from Step 4 and 5.

```
#Function that adds Pitching plus and Stuff plus into data
pitching_stuff_generator_trackman <- function(df) {

  #Adjust TrackMan data to fit statcast column names and add required columns
  df_valid <- trackman_adjuster(df)

  #Add pitcher's fastest pitch type info to data
  df_valid_with_fastest <- left_join(df_valid, trackman_data_adjusted_with_pitcher_fastest)

  #Add velocity and movements difference of pitcher's data
  df_valid_with_fastest_diff <- df_valid_with_fastest %>%
    mutate(
      release_speed_diff = ifelse(pitch_type != fastest_pitch_type, release_speed - fast_velo, -
Inf),
      xmov_diff_adj = ifelse(pitch_type != fastest_pitch_type, pfx_x_adj - fast_pfx_x_adj, -In
f),
      zmov_diff = ifelse(pitch_type != fastest_pitch_type, pfx_z - fast_pfx_z, -Inf))

  #Obtaining probability of contact with location
  ################ FF
  df_FF <- df_valid_with_fastest_diff %>%
    filter(pitch_type == "FF")

  Predictions_FF <- predict(object = SimpleGBMModel_FF,
                            newdata = df_FF,
                            n.trees = ntree_opt_cv_FF,
                            type = "response")

  df_FF$prob_contact_by_pitch <- Predictions_FF

  #Obtaining probability of contact without location
  ################ FF
  Predictions_FF_2 <- predict(object = SimpleGBMModel_FF_2,
                              newdata = df_FF,
                              n.trees = ntree_opt_cv_FF_2,
                              type = "response")

  df_FF$prob_contact_by_stuff <- Predictions_FF_2

  #Repeat the process for offspeed pitches

  ################ CH
  df_CH <- df_valid_with_fastest_diff %>%
    filter(pitch_type == "CH")

  Predictions_CH <- predict(object = SimpleGBMModel_CH,
                            newdata = df_CH,
                            n.trees = ntree_opt_cv_CH,
                            type = "response")

  df_CH$prob_contact_by_pitch <- Predictions_CH
```

```
Predictions_CH_2 <- predict(object = SimpleGBMModel_CH_2,
                            newdata = df_CH,
                            n.trees = ntree_opt_cv_CH_2,
                            type = "response")

df_CH$prob_contact_by_stuff <- Predictions_CH_2


############### FC
df_FC <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "FC")

Predictions_FC <- predict(object = SimpleGBMModel_FC,
                          newdata = df_FC,
                          n.trees = ntree_opt_cv_FC,
                          type = "response")

df_FC$prob_contact_by_pitch <- Predictions_FC

Predictions_FC_2 <- predict(object = SimpleGBMModel_FC_2,
                            newdata = df_FC,
                            n.trees = ntree_opt_cv_FC_2,
                            type = "response")

df_FC$prob_contact_by_stuff <- Predictions_FC_2

############### CU
df_CU <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "CU")

Predictions_CU <- predict(object = SimpleGBMModel_CU,
                          newdata = df_CU,
                          n.trees = ntree_opt_cv_CU,
                          type = "response")

df_CU$prob_contact_by_pitch <- Predictions_CU

Predictions_CU_2 <- predict(object = SimpleGBMModel_CU_2,
                            newdata = df_CU,
                            n.trees = ntree_opt_cv_CU_2,
                            type = "response")

df_CU$prob_contact_by_stuff <- Predictions_CU_2

############### SI
df_SI <- df_valid_with_fastest_diff %>%
  filter(pitch_type == "SI")

Predictions_SI <- predict(object = SimpleGBMModel_SI,
                          newdata = df_SI,
                          n.trees = ntree_opt_cv_SI,
                          type = "response")
```

```
  df_SI$prob_contact_by_pitch <- Predictions_SI

  Predictions_SI_2 <- predict(object = SimpleGBMModel_SI_2,
                              newdata = df_SI,
                              n.trees = ntree_opt_cv_SI_2,
                              type = "response")

  df_SI$prob_contact_by_stuff <- Predictions_SI_2

  ################ SL
  df_SL <- df_valid_with_fastest_diff %>%
    filter(pitch_type == "SL")

  Predictions_SL <- predict(object = SimpleGBMModel_SL,
                            newdata = df_SL,
                            n.trees = ntree_opt_cv_SL,
                            type = "response")

  df_SL$prob_contact_by_pitch <- Predictions_SL

  Predictions_SL_2 <- predict(object = SimpleGBMModel_SL_2,
                              newdata = df_SL,
                              n.trees = ntree_opt_cv_SL_2,
                              type = "response")

  df_SL$prob_contact_by_stuff <- Predictions_SL_2



  #Changing probability of contact to Pitching+ and Stuff+
  data_FF <- trackman_prob_to_pitch_stuff(df_FF, Predictions_FF, Predictions_FF_2)
  data_CH <- trackman_prob_to_pitch_stuff(df_CH, Predictions_CH, Predictions_CH_2)
  data_FC <- trackman_prob_to_pitch_stuff(df_FC, Predictions_FC, Predictions_FC_2)
  data_CU <- trackman_prob_to_pitch_stuff(df_CU, Predictions_CU, Predictions_CU_2)
  data_SI <- trackman_prob_to_pitch_stuff(df_SI, Predictions_SI, Predictions_SI_2)
  data_SL <- trackman_prob_to_pitch_stuff(df_SL, Predictions_SL, Predictions_SL_2)

  data_full <- rbind(data_FF, data_CH, data_FC, data_CU, data_SI, data_SL) %>%
    filter(!is.na(pitch_type))
  return(data_full)
}

#Add Pitching+ and Stuff+ to TrackMan data
trackman_df <- pitching_stuff_generator_trackman(pitches)
```

Steps 14 ~ 15: Obtaining college scaled Pitching++ and Stuff++ for TrackMan data.

Step 14: Function used in Step 15 for scaling the obtained MLB level Pitching+ and Stuff+ data to college level by comparing it to all the available TrackMan data obtained in last part of Step 13.

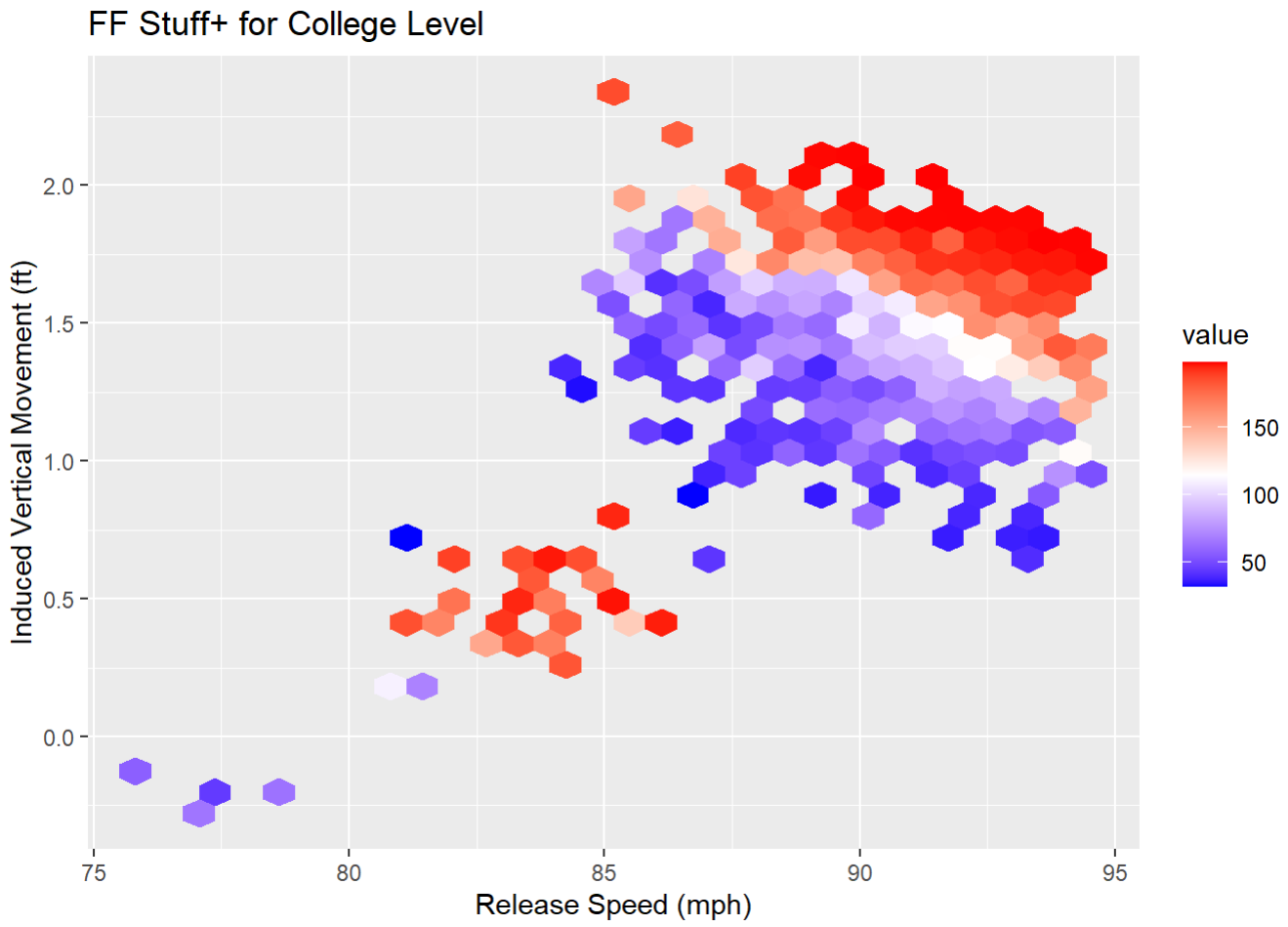
```
#Add Pitching++ and Stuff++ (normalized Pitching+ and Stuff+ based on TrackMan data) to TrackMan
data
scale_to_college <- function(input_data) {

  #Return null if input data doesn't contain that pitch type
  if (nrow(input_data) == 0) {
    return(NA)
  }

  pitch = input_data$pitch_type[1]

  #Filter TrackMan data to the input pitch type
  trackman_df_pitch <- trackman_df %>%
    filter(pitch_type == pitch)

  pitching_plus_trackman <- trackman_df_pitch$Pitching_plus
  stuff_plus_trackman <- trackman_df_pitch$Stuff_plus

  pitching_plus_input <- input_data$Pitching_plus
  stuff_plus_input <- input_data$Stuff_plus

  # Adding Pitching++ to data
  pitch_input_trackman <- c(pitching_plus_input, pitching_plus_trackman)

  z_score_pitch <- scale(pitch_input_trackman, center = median(pitch_input_trackman))

  percentile_pitch <- sapply(z_score_pitch, pnorm)

  percentile_pitch_adj <- percentile_pitch * 100 * 2

  #Extract percentiles from TrackMan data and add it to TrackMan data
  input_data$Pitching_plus2 = percentile_pitch_adj[1:nrow(input_data)]

  #Adding Stuff+ to data
  stuff_trackman_mlb <- c(stuff_plus_input, stuff_plus_trackman)

  z_score_stuff <- scale(stuff_trackman_mlb, center = median(stuff_trackman_mlb))

  percentile_stuff <- sapply(z_score_stuff, pnorm)

  percentile_stuff_adj <- percentile_stuff * 100 * 2

  #Extract percentiles from TrackMan data and add it to TrackMan data
  input_data$Stuff_plus2 = percentile_stuff_adj[1:nrow(input_data)]

  return(input_data)
}
```

Step 15: Function that adds Pitching++ and Stuff++ to data. Function in Step 12 is used to add MLB scaled Pitching+ and Stuff+. Then, the scaling technique in Step 14 is used to convert those values into college level Pitching++ and Stuff++ values.

```
#Add Pitching++ and Stuff++ to TrackMan data
pitching2_stuff2_generator_trackman <- function(df){

  #Add Pitching+ and Stuff+ to input data
  trackman_data2 <- pitching_stuff_generator_trackman(df)

  scaled_data_FF <- scale_to_college(trackman_data2 %>% filter(pitch_type == "FF"))
  scaled_data_CH <- scale_to_college(trackman_data2 %>% filter(pitch_type == "CH"))
  scaled_data_FC <- scale_to_college(trackman_data2 %>% filter(pitch_type == "FC"))
  scaled_data_CU <- scale_to_college(trackman_data2 %>% filter(pitch_type == "CU"))
  scaled_data_SI <- scale_to_college(trackman_data2 %>% filter(pitch_type == "SI"))
  scaled_data_SL <- scale_to_college(trackman_data2 %>% filter(pitch_type == "SL"))

  scaled_data <- rbind(scaled_data_FF, scaled_data_CH, scaled_data_FC, scaled_data_CU, scaled_da
ta_SI, scaled_data_SL) %>%
  filter(!is.na(pitch_type))

  return(scaled_data)
}
```

Testing using TrackMan and MLB data; sample hexagon plots to verify our model.

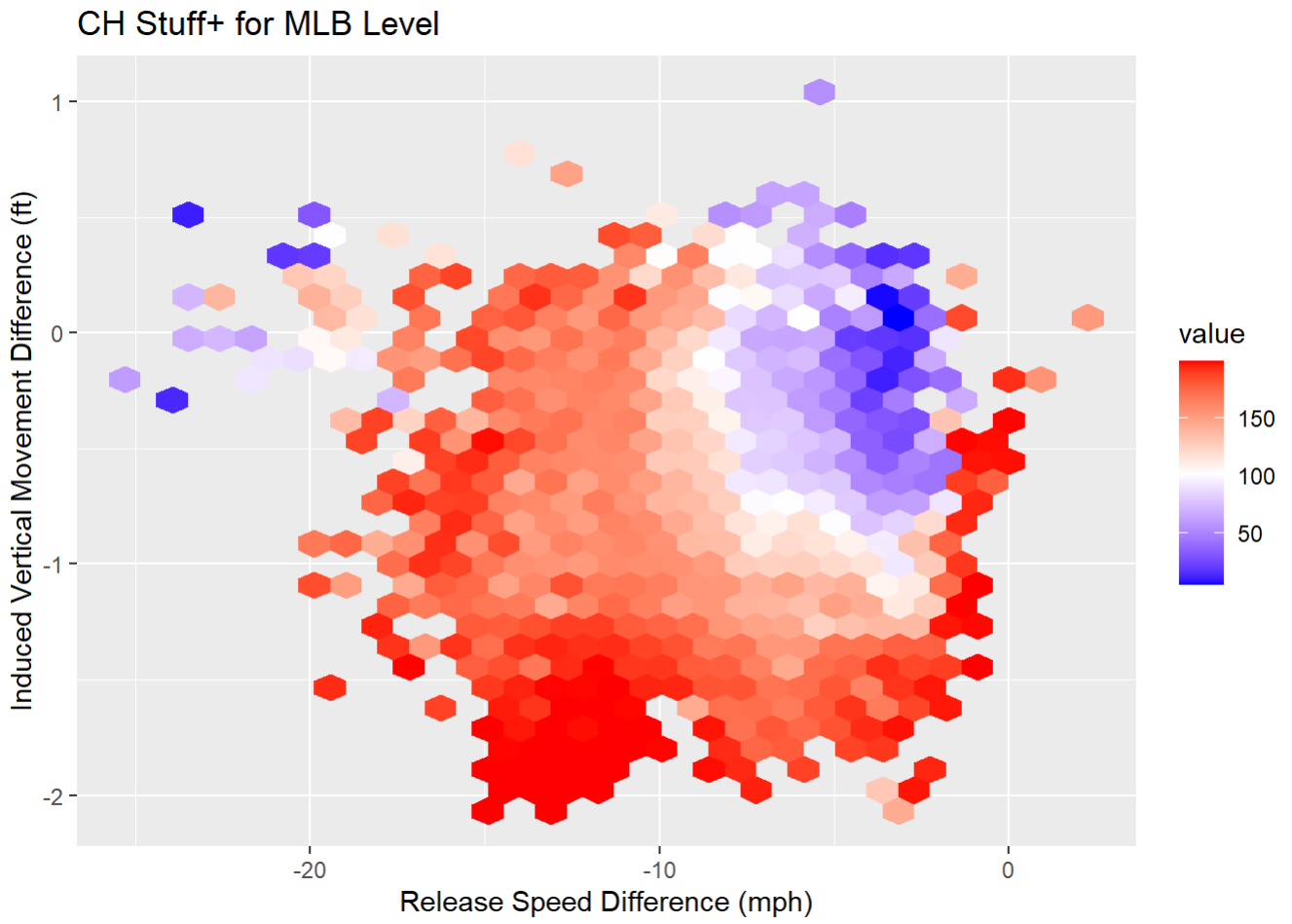
```
#Generating testing data from TrackMan data
testing_df <- pitching2_stuff2_generator_trackman(head(pitches, 2000))

#Fastball Stuff+ for college level
ggplot(testing_df %>% filter(pitch_type == "FF"), aes(release_speed, pfx_z)) +
  stat_summary_hex(aes(z = Stuff_plus2)) +
  scale_fill_gradientn(colours = c('blue', 'white', 'red')) +
  ggtitle("FF Stuff+ for College Level") + xlab("Release Speed (mph)") + ylab("Induced Vertical
Movement (ft)")
```



Comments

This hexagon plot depicts Stuff+ for fastballs scaled to college (D1) level, with x-axis depicting release speed and y-axis depicting induced vertical movement, color coded by Stuff+ values. As expected, a fastball that has high velocity and high vertical movement is classified with a high Stuff+ value.

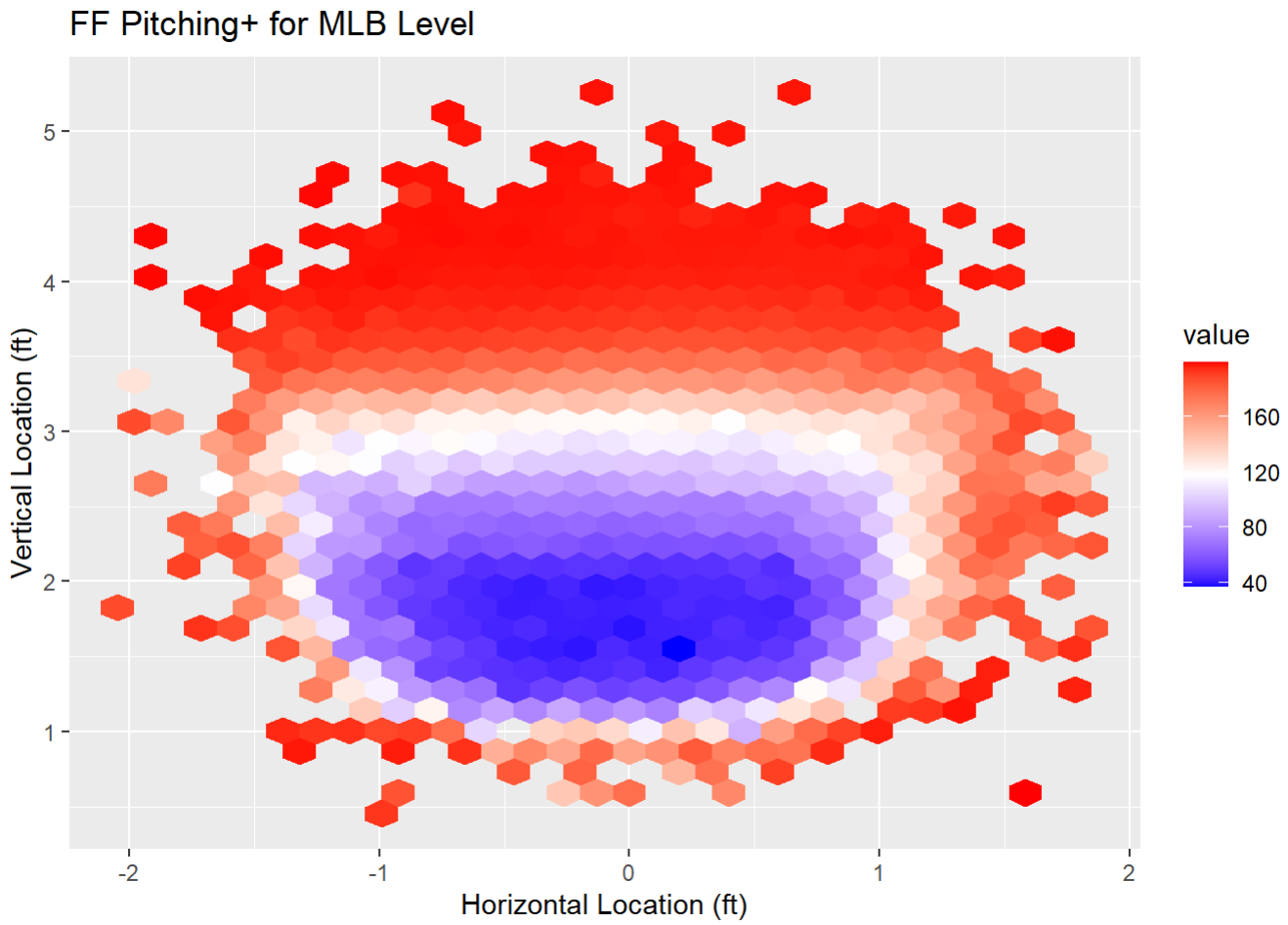
```
#Change-up Stuff+ for MLB Level
ggplot(mlb_2021 %>% filter(pitch_type == "CH"), aes(release_speed_diff, zmov_diff)) +
  stat_summary_hex(aes(z = Stuff_plus)) +
  scale_fill_gradientn(colours = c('blue', 'white', 'red')) +
  ggtitle("CH Stuff+ for MLB Level") + xlab("Release Speed Difference (mph)") + ylab("Induced Ve
rtical Movement Difference (ft)")
```



Comments

This hexagon plot depicts Stuff+ for change-ups scaled to MLB level, with x-axis depicting release speed difference from pitcher's fastest pitch type and y-axis depicting induced vertical movement difference from pitcher's fastest pitch type, color coded by Stuff+ values. As expected, a change-up that has high velocity difference and high vertical movement difference is classified with a high Stuff+ value.

```
#Fastball Pitching+ for MLB Level
ggplot(mlb_2021 %>% filter(pitch_type == "FF"), aes(plate_x, plate_z)) +
  stat_summary_hex(aes(z = Pitching_plus)) +
  scale_fill_gradientn(colours = c('blue', 'white', 'red')) +
  ggtitle("FF Pitching+ for MLB Level") + xlab("Horizontal Location (ft)") + ylab("Vertical Loca
tion (ft)")
```



Comments

This hexagon plot depicts Pitching+ for fastballs scaled to MLB level, with x-axis depicting horizontal location and y-axis depicting vertical location of the pitch, color coded by Pitching+ values. As expected, a fastball that is placed high in the zone has a high Pitching+ value. The worst location to throw a fastball according to this plot is middle low area.