Machine Learning Applications in MLB Pitch Classification

STA 5363 Final Project

Caleb Fox

12/9/22

Table of contents

Introduction

Baseball pitchers use different grips and put spin on the ball to make a pitch move in numerous directions at varying speeds. These pitches generally fall into three categories: fastballs, breaking balls, and changeups. Each type of pitch can then be broken down into subcategories. The different types of fastballs include four seam fastballs, two seam fastballs, cuters, splitters, and sinkers. Breaking balls include curveballs and sliders, although both can be thrown with multiple grips. Finally, changeups are often simply classified as a changeup but are thrown in different styles. Each pitcher has approximately two to five pitches they typically throw, sometimes blurring the lines between the types of pitches.

The trained eye can often detect the pitch type being thrown, but most casual fans will struggle to differentiate between pitch types while watching a game. Without knowing what pitches a pitcher typically throws (the pitcher's arsenal), it is even more difficult to identify specific pitch types being thrown. Much of the strategy in a baseball game is in the pitch selection, and knowing what pitches are being thrown makes a game much more interesting to watch. For this reason, TV broadcasts display the pitch type in real time on the broadcast. However, accurately classifying pitch types in real time is not a trivial problem.

The varying types of movement in a pitch are caused by the seams of the baseball interacting with the air as the ball spins. A fastball has backspin which causes the ball to resist gravity, and a curveball has topspin casuing the ball to break sharply downwards at the last second. The pitch speed, spin direction, spin rate, and many other variables can then identify a specific pitch. Since 2015, Major League Baseball has implemented the Statcast system which measures every event in every game. A neural network is then applied to classify the pitch type in real time which is provided to the broadcast. (insert section talking about the MLB algorithm) The goal of this paper is to replicate a similar pitch classification to the system used by the MLB and to compare the performance of a neural network with other machine learning models. We then will apply this model to estimate a pitcher's pitch arsenal.

The structure of this paper is as follows. In section two we will outline the data source and the models selected for the analysis. In section three we will discuss the results of the models and compare their performance. In section four we will discuss the results and future work that could improve the models, and we will close with a conclusion in section five.

Methods

Data

We collected data from all pitches in Major League Baseball in the 2022 season. The data is publically available at baseballsavant.mlb.com, and we downloaded the data using the baseballr package. There were 689,020 piches in the 2022 MLB season, which presented significant computa-

tional challenges for the analysis. To reduce computation time we randomly sampled 25% of the pitches in 2022 to create our data set of 172,255 pitches. The models were able to converge to an acceptable accuracy on the smaller dataset, so we would only expect marginal improvements when training the models on the entire dataset. Although, for optimal classification, all available data should be used. We then used a 70-30 train-test split to train and test the models.

Models

Before performing classification on the specific pitch type, we want to evaluate the models in performing simple pitch type classification (fastball, breaking ball, and changeup). These pitch type groups are more distinguishable than the specific pitch types, so we would expect all the models to perform well. We then will apply the same models to perform classification of the specific pitch type, and compare the prediction accuracy of each model for each pitch type.

For each classification problem we will compare the performance of a neural network, k-nearest neighbors, a random forest, and extreme gradient boosting. We also trained a support vector machine in the model building stage but chose to remove it from the analysis due to computational time and similar performance to all other models. The neural network was a simple feed-forward neural network with two hidden layers containing 32 nodes each, and we held out 20% of the data for validation in fitting the network. For the other three models, we used 10-fold cross-validation to select optimal tuning parameters.

After finding the model that most accurately classifies MLB pitches, we will use the model to estimate a pitcher's pitch arsenal. The models are trained to identify 7 different pitch types, so a pitcher could potentially throw between 1 and 7 types of pitches. Using all of a given pitcher's pitches in 2022 we will performing k-means clustering for 1 through 7 clusters. We will then apply the classification model previously selected to classify all pitches the pitcher threw The classifciations for each cluster will be compared and used to assign a pitch type to each cluster. This will provide an estimate for the pitcher's pitch arsenal. We will demonstrate this process for one pitcher, but it could be applied to all pitchers in the model and used to improve classification.

Results

Exploratory Data Analysis

We begin the analysis of the results with an exploratory data anlaysis. Figure 1 shows the counts of each pitch type, with the color representing the general pitch type. The rare pitch types such as the eephus and knuckleball were removed from the data as the pitches are almost never thrown in modern baseball. Four-seam fastballs were by far the most common pitch thrown, followed by sliders as the second most common. Note that major league pitchers have started throwing more sliders instead of curveballs in recent years, which is represented in the data. Splitters and

knuckle-curves are by far the least represented pitch type, potentially providing a challenge in classification. Knuckle-curves are effectively the same as a curveball, the only difference is they are thrown with the pointer finger tucked under the baseball. As the difference is only in grip and not in characterstics of the pitch, we will combine knucle-curves with curveballs and leave distinguishing between the two to the individual pitcher. Splitters on the other hand are wildly different from any other pitch, but a small group of MLB pitchers rely heavily on the pitch. We then will attempt to classify splitters, but the small sample size may decrease the accuracy of classifying splitters.

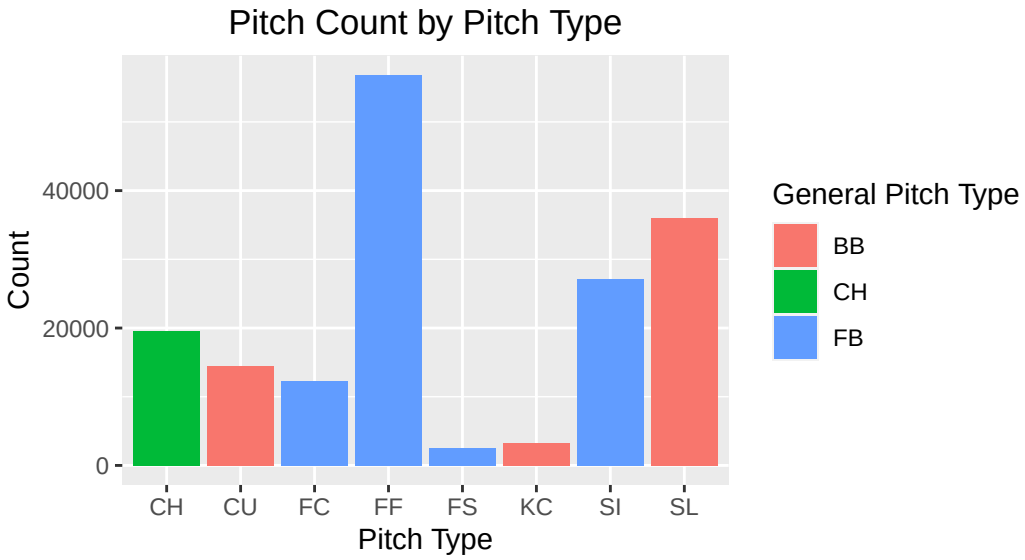

Figure 1: Bar chart of the counts of each pitch type. Colors represent the general pitch type.

Before performing classification it is necessary to visualize the data and ensure the groups appear separable on some of the predictor variables. This is impossible to do for all 15 variables of interest, but we can examine a few variables which we would expect to show signifcant differences between the pitch types. Figure 2 shows a scatterplot of the pitch speed and the pitch spin rate grouped by the general pitch type. We see the three general pitch types are mostly separable by these two variables. Once the other variables are included, the models should then easily identify the general pitch groups. We then expect high classification accuracy for the general pitch type.

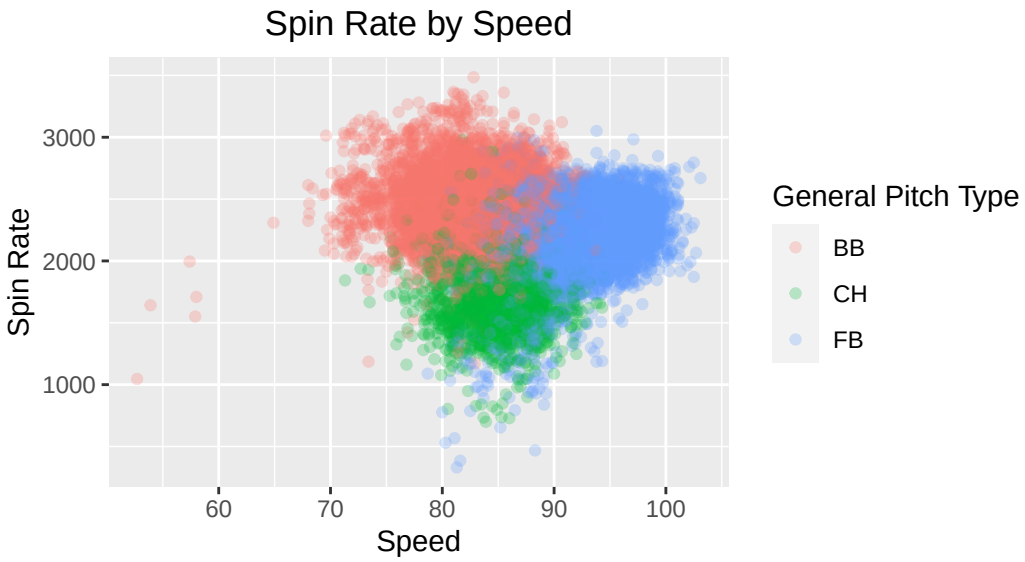

Figure 2: Scatterplot of spin rate by pitch speed grouped by the general pitch type.

While it is good to know the general pitch type is likely identifiable from the data, classifying the specific pitch type is of greater interest. Reproducing the same scatterplot but grouping the color by specific pitch type shows the greater challenge of the specific pitch type classification. Figure 3 shows the specific pitch types significantly overlap across spin rate and pitch speed. To identify the specific pitch types we then expect to require a more complicated model including more variables. Figure 4 shows a similar plot, but instead plotting the horizontal and vertical break of the pitch. We observe combining all the predictor variables may result in a model that can accurately classify specific pitch types.

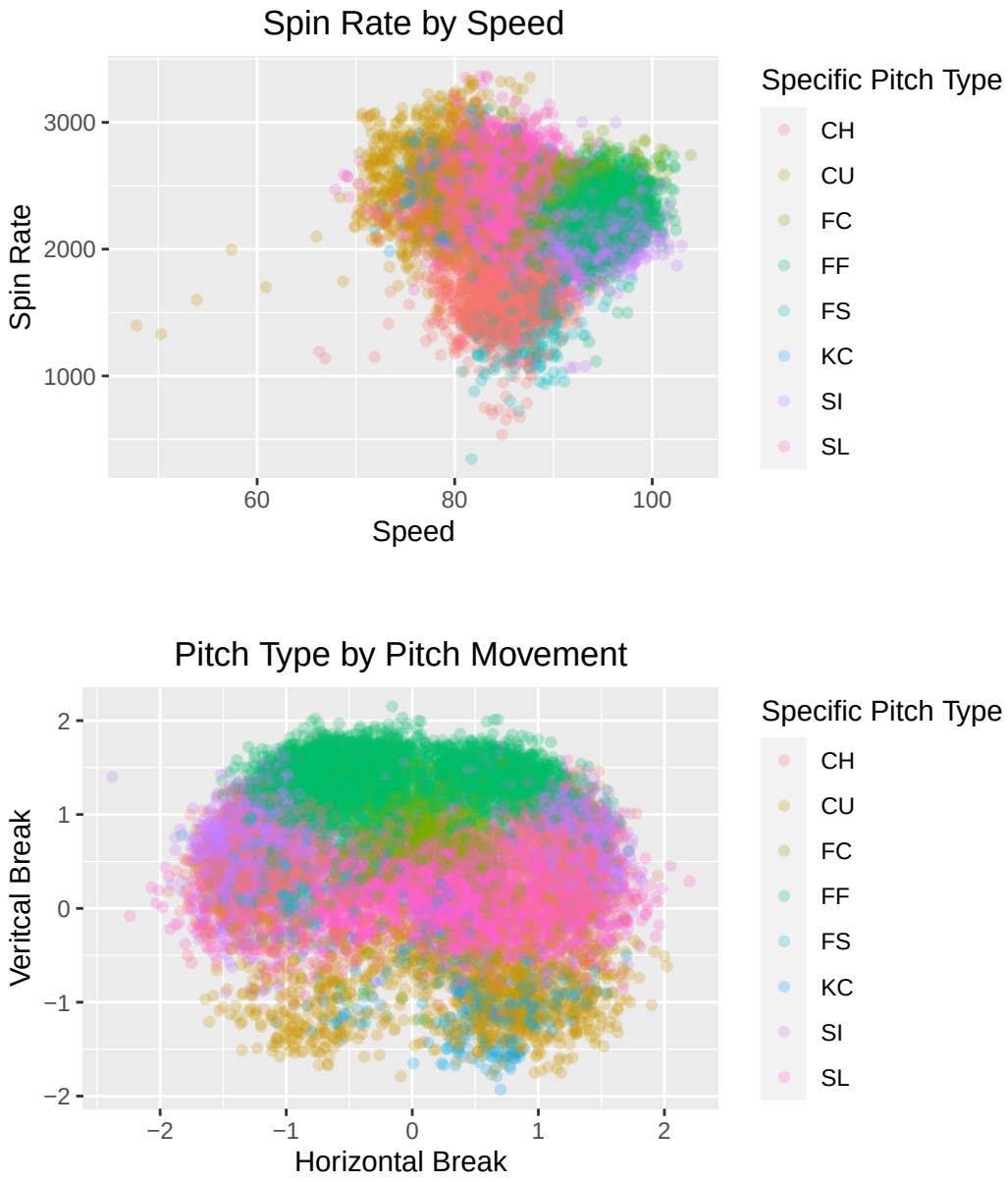

Figure 3: Scatterplot of horizontal and veritcal break grouped by the specific pitch type.

General Pitch Type Classification

We then fit each of the models to perform classification of the general pitch type. Table 1 shows the accuracy for each model. All models had a testing accuracy greater than 95%. Extreme gradient boosting had a slightly higher accuracy than the other models, but it was not a large difference.

Table 1: General pitch type model accuracy

Model	Accuracy
RF	0.958
KNN	0.954
NN	0.953
XGB	0.963

Table 2 shows the accuracy of each model for each pitch type. The models had high accuracy for all three general pitch types, but all the models had the lowest accuracy on changeups. These results show potential for the extreme gradient boosting model to outperform the neural network used by the MLB, although we expect to see lower accuracy when classifying the specific pitch types.

Table 2: General pitch type model accuracy by pitch type

Pitch_Type	RF	KNN	NN	XGB
FF	0.959	0.956	0.957	0.965
BB	0.962	0.959	0.956	0.967
CH	0.941	0.929	0.922	0.947

Specific Pitch Type Classification

Next, we fit the same models to classify the specific pitch type. Table 3 shows the testing accuracy for each model. There is a slight reduction in the overall accuracy as we increased the number of groups for classification from 3 to 7. Again, extreme gradient boosting outperformed the other models. Surprisingly, the neural network had the lowest testing accuracy at 89%.

Table 3: Specific pitch type model accuracy

Model	Accuracy
RF	0.913
KNN	0.900
NN	0.890
XGB	0.921

The decrease in accuracy likely comes from the difficulty in differentiating the different types of fastballs, and differentiating curveballs and sliders. Table 4 shows the test accuracy of each model given for each pitch type. Splitters were much more difficult to classify than other pitch types, but extreme gradient boosting was able to more accurately classify splitters than the other models. Cutters also had lower testing accuracy than most of the pitch types. Cutters are considered fastballs but are also closely related to sliders, so it is reasonable for cutters to be misclassified as either a four-seam fastball or a slider.

Table 4: Specific pitch type model accuracy by pitch type

Pitch_Type	RF	KNN	NN	XGB
FF	0.967	0.962	0.949	0.968
FC	0.702	0.717	0.760	0.753
SI	0.907	0.866	0.892	0.912
FS	0.617	0.536	0.471	0.684
CH	0.949	0.931	0.946	0.949
CU/KC	0.878	0.887	0.890	0.900
SL	0.923	0.903	0.841	0.922

Pitch Arsenal Estimation

After training a pitch classification model with relatively high accuracy, we want to apply the model to estimate a pitcher's pitch arsenal. We will demonstrate this process with Lucas Giolito, a pitcher for the Chicago White Sox, but the method could be used to estimate the pitch arsenal of any MLB pitcher. Lucas Giolito throws 4 pitches, a four-seam fastball, changeup, slider, and a curveball. If our method is successful, then it should estimate Lucas Giolito's pitch arsenal as these four pitches. However, Giolito only throws his curveball 3% of the time, so the model may not detect the curveball as part of his pitch arsenal.

Our model assumes 7 pitch types, so a pitcher's arsenal could contain anywhere from 1 to 7 pitches. To identify potential pitch types thrown by a pitcher, we performed k-means clustering on the pitches for $k = 1, 2, \ldots, 7$. A plot of the within cluster sum of squares is shown in figure 5. The plot would reasonably indicate Giolito throws 3 pitches. Although we know he actually throws 4 pitches, we will assume he only throws three pitches.

Table 5: Lucas Giolito pitch cluster type estimation.

cluster	FS	CH	CU	SL	FF	FC	SI
1	0	0.99	0.00	0.01	0.00	0.00	0.00
2	0	0.00	0.01	0.77	0.20	0.01	0.00
3	0	0.00	0.07	0.04	0.89	0.00	0.01

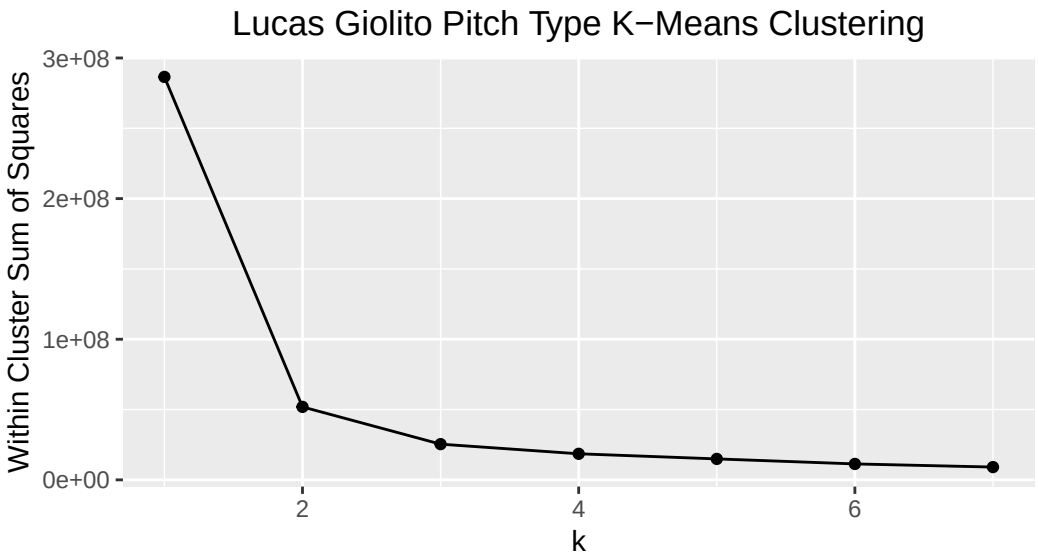

Figure 4: Within cluster sum of squares plot for k-mean clustering of Lucas Giolito pitch types. The plot indicates he only throws 3 types of pitches.

Under the assumption that Giolito throws three pitches, we then want to classify each of the clusters. We used the extreme gradient boosting model to classify all of Giolito's pitches from the 2022 season, and then counted the proportion classified as each pitch within each cluster. These results are shown in table 5. Cluster 1 is classified as a changeup, cluster 2 is classified as a slider, and cluster 3 is classified as a four seam fastball. These classifciations are the true pitches we know Giolito throws. It is also of note that 7% of pitches were classified as curveballs which could indicate Giolito also throws a curveabll, which we know is true.

Discussion

The primary goal of this project was to compare performance of different machine learning models in classifying MLB pitch types. The MLB uses a neural network to perform classification live during games, so it is of interest to explore if other models could outperform a neural network.

All four of the models applied resulted in good classification, although extreme gradient boosting outperformed all other models. Surprisingly, the neural network performed the worst of the models tested. Increasing the sample size could help improve the performance of the neural network, especially as the neural network struggled to identify splitters. Increasing the sample size by only including more splitters could help to improve the accuracy of the models without slowing down the training of the models by including all the data.

The secondary goal of the project was to estimate the pitch arsenal for a given pitcher. Due to time constraints we did not perform pitch arsenal estimation for every pitcher in the data, but we demonstrated how the process could be implemented for a single pitcher. Applying k-means clustering and then using the extreme gradient boosting model to classify each cluster provided a good estimate of Lucas Giolito's pitch arsenal. Our method did not perfectly identify his four pitches as he rarely throws a curveball, but we were able to correctly classify his three primary pitches. Other clustering methods could be used to better identify pitch types that are rarely thrown by a pitcher.

The initial results from this analysis showed pitch types can be accurately identified using extreme gradient boosting, and without knowing anything about a pitcher we can estimate their pitch arsenal. However, these results can still be improved. Combining the pitch arsenal estimation with the pitch type classification could potentially improe the accuracy of the model. If we believe a pitcher only has three pitches in their arsenal, then limiting the output of the classification model to those pitches will likely improve the accuracy of the model. Other simple improvements could also improve the model accuracy. Training separate models for left and right handed pitchers could improve model accuracy, as well as increasing the sample size to include more values for the less frequently thrown pitches.

Conclusion

Baseball fans are accustomed to seeing the type of pitch being displayed immediately after the pitch is thrown on tv, but fans rarely consider how this classification is performed. Major League Baseball does not provide a detailed explanation of their neural network, but the data used for this classification is publicly available through statcast. We then wanted to compare the performance of different machine learning algorithms at classifying pitch types. Extreme gradient boosting outperformed all other models, including a neural network we applied to the classifciation. Extreme gradient bossting better classified the less common pitches such as splitters. This model then indicates it can be used to estimate a pitcher's pitch arsenal, which can in turn be sued to better classify pitches. Further development of these models could continue to improve accuracy and potentially outperform the neural network applied by Major League Baseball.

Code Appendix

General Pitch Type Classification

```
# load packages -------------------------------------------------------

library("here")
library("caret")
library("keras")
library("tensorflow")
library("reticulate")
library("baseballr")
library('fastDummies')
library('randomForest')
library('e1071')
library('furrr')
library('doParallel')
library('tictoc')
library("tidyverse"); theme_set(theme_minimal())



# Read Data -----------------------------------------------------------

# dates <- c(seq(as.Date("2022-04-07"), as.Date("2022-06-17"), by="days"),
#   seq(as.Date("2022-06-22"), as.Date("2022-10-07"), by="days"))
#

# plan(multisession(workers = availableCores()))
# tb <- dates %>%
#   future_map(~ statcast_search(.x, .x), .progress = TRUE)
#
# plan(sequential)

 tb <- read_rds('data_list.rds')

 id <- map(tb, ~ .x$pitch_type %>% is.character()) %>% unlist()
 pitch_data <- bind_rows(tb[id])

 pitch_data <- pitch_data %>%
   select(
     pitch_type,release_speed,release_pos_x,release_pos_z,zone,
```

```
    p_throws,balls,strikes,pfx_x,pfx_z,plate_x,plate_z,vx0,vy0,
    vz0,ax,ay,az,release_spin_rate,spin_axis
  ) %>%
  drop_na() %>%
  filter(pitch_type %in% c('CH','CU','FC','FF','FS','KC','SI','SL')) %>%
  mutate(pitch_type_group = case_when(
    pitch_type %in% c('FC','FF','FS','SI') ~ 1,
    pitch_type %in% c('CU','KC','SL') ~ 2,
    pitch_type == 'CH' ~ 3
  ))
# write_rds(tb, file = "data_list.rds")
write_rds(pitch_data, file = "pitch_data_simple.rds")



# Train Test Split ----------------------------------------------------------
set.seed(123)
pitch_data <- read_rds('pitch_data_simple.rds')

pitch_data <- pitch_data %>%
  # mutate(pitch_type_fb = pitch_type_group == "FB") %>%
  select(-c('pitch_type', 'zone', 'p_throws', 'balls', 'strikes'))

#If you do not have an ID per row, use the following code to create an ID
tb <- pitch_data %>% mutate(id = row_number()) %>% sample_frac(.25)
#Create training set
train <- tb %>% sample_frac(.7)
#Create test set
test  <- anti_join(tb, train, by = 'id') %>% select(-id)
train <- select(train, -id)


# EDA -----------------------------------------------------------------------

# pitch_data %>%
#   ggplot(aes(pitch_type_group)) +
#     geom_bar()
#
# pitch_data %>%
#   sample_frac(.1) %>%
#   ggplot(aes(release_spin_rate, spin_axis, color = as.factor(pitch_type_group))) +
#     geom_point(alpha = .25)
#
```

```
# pitch_data %>%
#   sample_frac(.1) %>%
#   ggplot(aes(release_speed, pfx_x, color =  as.factor(pitch_type_group))) +
#     geom_point(alpha = .25)




# Neural Network ------------------------------------------------------------
tic()

train_matrix <- model.matrix(pitch_type_group ~ . , train)[, -1] %>%
  scale()

train_label <- train %>% pull(pitch_type_group) %>% to_categorical() %>% .[,-1]

test_matrix <- model.matrix(pitch_type_group ~ . , test)[, -1] %>%
  scale()

test_label <- test %>% pull(pitch_type_group)

nn_mod <- keras_model_sequential() %>%
  layer_dense(units = 32, input_shape = ncol(train_matrix), activation = "relu") %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 3, activation = "softmax")

summary(nn_mod)


nn_mod %>% compile(loss = "categorical_crossentropy",
                   optimizer = "adam",
                   metrics = c("accuracy"))

trained_nn <- nn_mod %>% fit(
  x = train_matrix, # using for prediction
  y = train_label, # predicting
  batch_size = 50, # how many samples to pass to our model at a time
  epochs = 45, # how many times we'll look @ the whole dataset
  validation_split = 0.2) # how much data to hold out for testing as we go along

write_rds(nn_mod, "models/nn_mod.rds")

test_preds <- predict(nn_mod, test_matrix, type = "response")
```

```
test_pred_labels <- k_argmax(test_preds) %>% as.numeric() + 1

nn_acc <- mean(test_pred_labels == test_label)

nn_pitch_acc <- tibble(predicted = test_pred_labels, actual = test_label) %>%
  group_by(actual) %>%
  summarise(accuracy = mean(predicted == actual))



write_rds(nn_pitch_acc, "Simple/nn_pitch_acc.rds")
write_rds(nn_acc, "Simple/nn_acc.rds")
```

```
# K-Nearest Neighbor ------------------------------------------------------
knn_train <- train %>%
  select(-pitch_type_group)

knn_train_lab <- pull(train, pitch_type_group) %>% factor()

cl <- makePSOCKcluster(15)
registerDoParallel(cl)
knnFit <- caret::train(knn_train, knn_train_lab,
                  method = "knn",
                  preProcess = c("center", "scale"),
                  tuneLength = 10,
                  trControl = trainControl(method = "cv"))
```

```
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)


write_rds(knnFit, "models/knnFit.rds")


knnFit <- read_rds("models/knnFit.rds")

knn_test_pred <- predict(knnFit, newdata = test)

knn_acc <- mean(knn_test_pred == test_label)

knn_pitch_acc <- tibble(Predicted = knn_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(knn_pitch_acc, "Simple/knn_pitch_acc.rds")
write_rds(knn_acc, "Simple/knn_acc.rds")









# Random Forest ------------------------------------------------------------


cl <- makePSOCKcluster(15)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
                     number = 10,
                     search = "random",
                     verboseIter = TRUE)
forestFit <- caret::train(factor(pitch_type_group) ~ ., data = train,
```

```
                    method = "rf",
                    trControl = ctrl,
                    tuneLength = 15,
                    metric = "Accuracy")
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)

write_rds(forestFit, "models/forestFit.rds")



forestFit <- read_rds("models/forestFit.rds")

forest_test_pred <- predict(forestFit, newdata = test)

forest_acc <- mean(forest_test_pred == test_label)

forest_pitch_acc <- tibble(Predicted = forest_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(forest_pitch_acc, "Simple/forest_pitch_acc.rds")
write_rds(forest_acc, "Simple/forest_acc.rds")

toc()



# XGBoost -----------------------------------------------------------

set.seed(123)
cl <- makePSOCKcluster(12)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
                     number = 10,
                     search = "random",
                     verboseIter = TRUE)
xgbFit <- caret::train(factor(pitch_type_group) ~ ., data = train,
                       method = "xgbTree",
                       trControl = ctrl,
```

```
                      tuneLength = 15,
                      metric = "Accuracy")

stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)


write_rds(xgbFit, "models/xgbFit.rds")

xgbFit <- read_rds("models/xgbFit.rds")

xgb_test_pred <- predict(xgbFit, newdata = test)

xgb_acc <- mean(xgb_test_pred == test_label)

xgb_pitch_acc <- tibble(Predicted = xgb_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(xgb_pitch_acc, "Simple/xgb_pitch_acc.rds")
write_rds(xgb_acc, "Simple/xgb_acc.rds")







# Support Vector Machine ------------------------------------------------



cl <- makePSOCKcluster(15)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
                     number = 50,
                     search = "random",
                     verboseIter = TRUE)
svmFit <- caret::train(factor(pitch_type_group) ~ ., data = train,
```

```
                          method = "svmRadial",
                          trControl = ctrl,
                          tuneLength = 10,
                          metric = "Accuracy")
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)

write_rds(svmFit, "models/svmFit.rds")

svmFit <- read_rds("models/svmFit.rds")

svm_pred <- predict(svmFit, newdata = test)

svm_acc <- mean(svm_pred == test_label)

svm_pitch_acc <- tibble(Predicted = svm_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(xgb_pitch_acc, "Simple/svm_pitch_acc.rds")
write_rds(xgb_acc, "Simple/svm_acc.rds")


toc()
```

Specific Pitch Type Classification

```
# load packages ------------------------------------------------------------
library("here")
library("caret")
library("keras")
library("tensorflow")
library("reticulate")
library("baseballr")
library('fastDummies')
library('randomForest')
library('e1071')
library('furrr')
library('doParallel')8
```

```
library('tictoc')
library("tidyverse"); theme_set(theme_minimal())


# Read Data ---------------------------------------------------------------

# dates <- c(seq(as.Date("2022-04-07"), as.Date("2022-06-17"), by="days"),
#   seq(as.Date("2022-06-22"), as.Date("2022-10-07"), by="days"))
#

# plan(multisession(workers = availableCores()))
# tb <- dates %>%
#   future_map(~ statcast_search(.x, .x), .progress = TRUE)
#
# plan(sequential)
#
# tb <- read_rds('data_list.rds')
#
# id <- map(tb, ~ .x$pitch_type %>% is.character()) %>% unlist()
# pitch_data <- bind_rows(tb[id])
#
# pitch_data <- pitch_data %>%
#   select(
#     pitch_type,release_speed,release_pos_x,release_pos_z,zone,
#     p_throws,balls,strikes,pfx_x,pfx_z,plate_x,plate_z,vx0,vy0,
#     vz0,ax,ay,az,release_spin_rate,spin_axis
#   ) %>%
#   drop_na() %>%
#   filter(pitch_type %in% c('CH','CU','FC','FF','FS','KC','SI','SL')) %>%
#   mutate(pitch_type_group = case_when(
#     pitch_type %in% c('FC','FF','FS','SI') ~ 1,
#     pitch_type %in% c('CU','KC','SL') ~ 2,
#     pitch_type == 'CH' ~ 3
#   )) %>%
#   mutate(pitch_type = case_when(
#     pitch_type == 'FF' ~ 1,
#     pitch_type == 'FC' ~ 2,
#     pitch_type == 'SI' ~ 3,
#     pitch_type == 'FS' ~ 4,
#     pitch_type == 'CH' ~ 5,
#     pitch_type %in% c('CU','KC') ~ 6,
#     pitch_type == 'SL' ~ 7
#   ))
```

```
#
# write_rds(tb, file = "data_list.rds")
# write_rds(pitch_data, file = "pitch_data.rds")



# Train Test Split ----------------------------------------------------------
tic()
set.seed(123)
pitch_data <- read_rds('pitch_data.rds')

pitch_data <- pitch_data %>%
  # mutate(pitch_type_fb = pitch_type_group == "FB") %>%
  select(-c('pitch_type_group', 'zone', 'p_throws', 'balls', 'strikes'))

#If you do not have an ID per row, use the following code to create an ID
tb <- pitch_data %>% mutate(id = row_number()) %>% sample_frac(.25)
#Create training set
train <- tb %>% sample_frac(.7)
#Create test set
test  <- anti_join(tb, train, by = 'id') %>% select(-id)
train <- select(train, -id)




# Neural Network ------------------------------------------------------------

train_matrix <- model.matrix(pitch_type ~ . , train)[, -1] %>%
  scale()

train_label <- train %>% pull(pitch_type) %>% to_categorical() %>% .[,-1]

test_matrix <- model.matrix(pitch_type ~ . , test)[, -1] %>%
  scale()

test_label <- test %>% pull(pitch_type)

nn_mod <- keras_model_sequential() %>%
  layer_dense(units = 32, input_shape = ncol(train_matrix), activation = "relu") %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 7, activation = "softmax")
```

```
summary(nn_mod)


nn_mod %>% compile(loss = "categorical_crossentropy",
                   optimizer = "adam",
                   metrics = c("accuracy"))

trained_nn <- nn_mod %>% fit(
  x = train_matrix, # using for prediction
  y = train_label, # predicting
  batch_size = 50, # how many samples to pass to our model at a time
  epochs = 45, # how many times we'll look @ the whole dataset
  validation_split = 0.2) # how much data to hold out for testing as we go along

# write_rds(nn_mod, "models/nn_mod2.rds")

# nn_mod2 <- read_rds("models/nn_mod2.rds")


test_preds <- predict(nn_mod, test_matrix, type = "response")
test_pred_labels <- k_argmax(test_preds) %>% as.numeric() + 1

nn_acc <- mean(test_pred_labels == test_label)

nn_pitch_acc <- tibble(predicted = test_pred_labels, actual = test_label) %>%
  group_by(actual) %>%
  summarise(accuracy = mean(predicted == actual))



# write_rds(nn_pitch_acc, "Complex/nn_pitch_acc.rds")
# write_rds(nn_acc, "Complex/nn_acc.rds")







# K-Nearest Neighbor ------------------------------------------------------

train_label <- train %>% pull(pitch_type)
```

```
test_label <- test %>% pull(pitch_type)


knn_train <- train %>%
  select(-pitch_type)

knn_train_lab <- pull(train, pitch_type) %>% factor()

cl <- makePSOCKcluster(12)
registerDoParallel(cl)
knnFit <- caret::train(knn_train, knn_train_lab,
                       method = "knn",
                       preProcess = c("center", "scale"),
                       tuneLength = 10,
                       trControl = trainControl(method = "cv"))
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)


write_rds(knnFit, "models/knnFit2.rds")


knnFit2 <- read_rds("models/knnFit2.rds")

knn2_test_pred <- predict(knnFit2, newdata = test)

knn_acc <- mean(knn2_test_pred == test_label)

knn_pitch_acc <- tibble(Predicted = knn2_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(knn_pitch_acc, "Complex/knn_pitch_acc.rds")
write_rds(knn_acc, "Complex/knn_acc.rds")
```

```
# Random Forest ------------------------------------------------------------


cl <- makePSOCKcluster(14)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
                     number = 10,
                     search = "random",
                     verboseIter = TRUE)
forestFit <- caret::train(factor(pitch_type) ~ ., data = train,
                          method = "rf",
                          trControl = ctrl,
                          tuneLength = 15,
                          metric = "Accuracy")
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)

write_rds(forestFit, "models/forestFit2.rds")

forestFit2 <- read_rds("models/forestFit2.rds")

forest_test_pred <- predict(forestFit2, newdata = test)

forest_acc <- mean(forest_test_pred == test_label)

forest_pitch_acc <- tibble(Predicted = forest_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(forest_pitch_acc, "Complex/forest_pitch_acc.rds")
write_rds(forest_acc, "Complex/forest_acc.rds")


# Gradient Boosting --------------------------------------------------------

set.seed(123)
cl <- makePSOCKcluster(14)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
```

```
                      number = 10,
                      search = "random",
                      verboseIter = TRUE)
xgbFit <- caret::train(factor(pitch_type) ~ ., data = train,
                 method = "xgbTree",
                 trControl = ctrl,
                 tuneLength = 15,
                 metric = "Accuracy")

stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)


write_rds(xgbFit, "models/xgbFit2.rds")

xgbFit2 <- read_rds("models/xgbFit2.rds")

xgb_test_pred <- predict(xgbFit2, newdata = test)

xgb_acc <- mean(xgb_test_pred == test_label)

xgb_pitch_acc <- tibble(Predicted = xgb_test_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(xgb_pitch_acc, "Complex/xgb_pitch_acc.rds")
write_rds(xgb_acc, "Complex/xgb_acc.rds")



# Support Vector Machine ------------------------------------------------


cl <- makePSOCKcluster(12)
registerDoParallel(cl)
ctrl <- trainControl(method="cv",
                      number = 5,
                      search = "random",
                      verboseIter = TRUE)
svmFit <- caret::train(factor(pitch_type) ~ ., data = train,
```

```
                        method = "svmRadial",
                        trControl = ctrl,
                        tuneLength = 10,
                        metric = "Accuracy")
stopCluster(cl)

env <- foreach:::.foreachGlobals
rm(list=ls(name=env), pos=env)

# write_rds(svmFit, "models/svmFit2.rds")

svmFit <- read_rds("models/svmFit2.rds")

svm_pred <- predict(svmFit, newdata = test)

svm_acc <- mean(svm_pred == test_label)

svm_pitch_acc <- tibble(Predicted = svm_pred, Actual = test_label) %>%
  group_by(Actual) %>%
  summarise(Accuracy = mean(Predicted == Actual))


write_rds(xgb_pitch_acc, "Complex/svm_pitch_acc.rds")
write_rds(xgb_acc, "Complex/svm_acc.rds")


toc()


# Pitch Reproitoire Prediction ----------------------------------------

tb <- read_rds('data_list.rds')

id <- map(tb, ~ .x$pitch_type %>% is.character()) %>% unlist()
tb <- bind_rows(tb[id])

predict_pitch_rep <- function(k, player_data, player_kmeans) {
  player_data %>%
    mutate(cluster = player_kmeans[[k]]$cluster, pred = predict(xgbFit2, newdata = player_dat
    mutate(pred = factor(case_when(
      pred == 1 ~ 'FF',
      pred == 2 ~ 'FC',
      pred == 3 ~ 'SI',
```

```
      pred == 4 ~ 'FS',
      pred == 5 ~ 'CH',
      pred == 6 ~ 'CU',
      pred == 7 ~ 'SL'), levels = c('FF', 'FC', 'SI', 'FS', 'CH', 'CU', 'SL'))) %>%
    group_by(cluster, pred) %>%
    summarise(count = n()) %>%
    group_by(cluster) %>%
    mutate(perc = count/sum(count)) %>%
    ungroup() %>%
    select(-count) %>%
    pivot_wider(names_from = pred, values_from = perc, values_fill = 0)
}

pitch_rep <- function(first, last) {

  player_id <- baseballr::playerid_lookup(last_name = last, first_name = first) %>%
    dplyr::pull(mlbam_id)

  player_data <- tb %>%
    filter(pitcher == player_id) %>%
    select(
      pitch_type,release_speed,release_pos_x,release_pos_z,zone,
      p_throws,balls,strikes,pfx_x,pfx_z,plate_x,plate_z,vx0,vy0,
      vz0,ax,ay,az,release_spin_rate,spin_axis
    ) %>%
    drop_na() %>%
    filter(pitch_type %in% c('CH','CU','FC','FF','FS','KC','SI','SL')) %>%
    mutate(pitch_type_group = case_when(
      pitch_type %in% c('FC','FF','FS','SI') ~ 1,
      pitch_type %in% c('CU','KC','SL') ~ 2,
      pitch_type == 'CH' ~ 3
    )) %>%
    mutate(pitch_type = case_when(
      pitch_type == 'FF' ~ 1,
      pitch_type == 'FC' ~ 2,
      pitch_type == 'SI' ~ 3,
      pitch_type == 'FS' ~ 4,
      pitch_type == 'CH' ~ 5,
      pitch_type %in% c('CU','KC') ~ 6,
      pitch_type == 'SL' ~ 7)) %>%
    select(-c('pitch_type','pitch_type_group', 'zone', 'p_throws', 'balls', 'strikes'))
```

```
  player_kmeans <- map(1:7, ~ kmeans(player_data, centers = .x, nstart = 25, iter.max = 50))


  wss <- map_dbl(1:7, ~ player_kmeans[[.x]]$tot.withinss)

  kmeans_plot <- tibble(k = 1:7, wss = wss) %>%
    ggplot(aes(k, wss)) +
    geom_line() +
    geom_point()

  list(plot = kmeans_plot, pitch_pred = map(1:7, predict_pitch_rep, player_data, player_kmea
}


dylan_cease <- pitch_rep("Dylan", "Cease")
sandy_alcantara <- pitch_rep("Sandy", "Alcantara")
lucas_giolito <- pitch_rep("Lucas", "Giolito")

# write_rds(lucas_giolito, "lucas_giolito.rds")
```