Impute Players

Matthew Reyers

May 18, 2019

Topic and Idea

Sports are violent. Injuries happen weekly and can persist for weeks or even months on end. Careers are built and destroyed by injuries, sometimes simultaneously. Consider the case of Drew Bledsoe, a former Quarterback for the New England Patriots. Off a hit from New York Jets Linebacker Mo Lewis, Bledsoe suffered a sheared blood vessel which ended his season. He made some rounds through the league but his career did not take back up in New England. Why? Because he had been replaced by a 6th round rookie named Tom Brady. The rest, of course, is history.

I feel that injuries beg the question of "what if"? What if Bledsoe remained healthy? What if Tom Brady remained the backup? Their careers would certainly have followed different trajectories. But what we have instead is one realization of reality and with that we must work.

The purpose of this work is then to explore the missingness of sports. I want to be able to address the performance that would have been missed by a player in an injured season or what can be expected of a player returning from a new injury. These problems naturally suggest a few methods to me: Imputation and Regression.

Data

Before we begin with either of those techniques, I will be using season long box score data for the NHL. The data will be scraped from https://www.hockey-reference.com/leagues and dates back to 1970. This time I have included the scraper code within the write up for easy access.

```r
library(tidyverse)
library(rvest)
library(XML)
library(naniar)
library(VIM)
library(mice)
library(tictoc)
library(missForest)
library(missRanger)
library(png)
```

```r
start_yr <- 1999 # First year where TOI is recorded
end_yr <- 2019
yr_interval <- (start_yr:end_yr)[-(2005-start_yr + 1)] # Skip lockout year
base_url <- "https://www.hockey-reference.com/leagues/NHL_"
suffix <- "_skaters.html"

urls <- paste0(base_url, yr_interval, suffix)

season_grabber <- function(url){
  print(url)
    season_results <- read_html(url) %>%
    html_nodes(xpath = '//*[contains(concat( " ", @class, " " ), concat( " ", "right", " " ))] |
               //*[contains(concat( " ", @class, " " ), concat( " ", "center", " " ))] |
               //*[contains(concat( " ", @class, " " ), concat( " ", "left", " " ))]') %>%
    html_text() %>%
    as_tibble()
}

season_cleaner <- function(season_tbl){
  bad_vals <- c("", "Scoring", "Special Teams", "Assists", "Shot Data", "Ice Time")
  act_col_names <- c("Rk", "Player", "Age", "Tm", "Pos", "GP", "G", "A", "PTS", "PlMi", "PIM",
                     "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc",
                     "TOI", "ATOI")

  # Get player names
  season_tbl <- season_tbl %>%
    filter(!(value %in% bad_vals)) %>%
    mutate(temp_row_names = rep(act_col_names, length.out = dim(.)[1]),
           temp_row_n = (row_number() - 1) %/% length(act_col_names),
           plyr_name = case_when(str_detect(value, "[A-Za-z]+ [A-Za-z]+") ~ value,
                                 TRUE ~ ""),
           num_players = cumsum(!(plyr_name %in% "")),
           player_name = case_when(!(plyr_name %in% "") ~ plyr_name,
                                   TRUE ~ lag(plyr_name, default = ""))) %>%
    select(value, player_name, plyr_name, num_players) %>%
    group_by(num_players) %>%
    mutate(act_player = first(plyr_name),
           row_n = row_number()) %>%
    select(value, act_player, row_n, num_players) %>%
    ungroup()


  return(season_tbl)
}


list_to_vec <- function(list_item){
  # Some lists have garbage input
  if(dim(list_item)[1] == 1){
    return()
  }

  holder <- list_item %>%
    select(value) %>%
    filter(row_number() <= 20) %>% # Only need up to shots
  as_vector()
  return(holder)
}

scrape_to_tbl <- function(scraped){
  # Takes the entire scrape, transforms to a list of vectors, reformats and returns as tibble
  curr <- scraped %>% lapply(list_to_vec)
  names(curr) <- seq_along(curr)
  curr <- Filter(Negate(is.null), curr) %>%
    bind_rows()
  curr.t <- t(curr) %>% as_tibble()
  colnames(curr.t) <- c("Player", "Age", "Tm", "Pos", "GP", "G", "A", "PTS", "PlMi", "PIM",
                        "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc")
  curr.t <- curr.t %>% filter(row_number() > 1) # now that header assigned properly
  return(curr.t)
}

scraper_fn <- function(url){
  return(season_grabber(url) %>%
    season_cleaner() %>%
    split(.$act_player) %>%
    scrape_to_tbl()) %>%
    mutate(Year = str_extract(url, "[0-9]+"))
}
# Other scrape idea
all_data_condensed <- lapply(urls, scraper_fn) %>% bind_rows()
write.csv(all_data_condensed, "scraped_hockey_data.csv")
```

The data is scraped using a combination of RCurl and XML packages. This scraping feels the most natural as I can easily grab html objects from websites and parse the results into nodes. I use the Selector Gadget tool on Chrome to identify the appropriate xPath used in the functions. Finally I convert the nodeset to text as it is otherwise hard to interact with in R.

Imputation

Missing data now becomes the focus of this work. The primary goal of course is to try and understand what would have been in place of this missing data should the player have participated in the given season. Figuring out how to do this requires a few assumptions and thoughts prior to beginning.

Assumption 1: The data is Missing at Random. This assumption allows us to conduct the imputation. It assumes that there is no bias or underlying trend as to why data is missing. An example of where this assumption would not be valid is in conducting surveys of political views but also asking participants to report their income. For one reason or another, respondents may systematically choose to not respond to one or both of the questions. Infering the missing values in situations with non-random missingness then becomes a challenge as groups are underrepresented and may not be homogeneous with other well reported populations. With respect to sports, if we assume injuries to be a random event as they most commonly are then we can reasonably make this assumption. Note that this is a softer assumption than Missing Completely At Random.

Assumption 2: The last season for a player in the league is fixed (maybe?)

Assumption 3: A season is counted as missed if the player played in fewer than 20 games. The reason for the reduction can be anything as I don't have a database on who was holding out for contracts and who was injured.

Imputation then has two steps: testing imputation techniques to identify the method that works best for the given data followed by actually imputing the data.

Testing involves gathering all of the complete records available. The assumptions above have laid out the filter criterion and so I will filter the data accordingly.

```r
frac_NA <- 0.25
set.seed(1)

complete_seasons <- read_csv("scraped_hockey_data.csv", col_types = cols()) %>%
  filter(GP > 20) %>% # Can also filter rookies by group, arrange, filter !first(year)
  mutate(row_n = row_number())
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
missing_seasons <- read_csv("scraped_hockey_data.csv", col_types = cols()) %>%
  filter(GP <= 20) %>%
  replace_with_na_at(.vars = c("GP", "G", "A", "PTS", "PlMi", "PIM",
                     "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc"),
                     condition = ~is.numeric(.x))
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
fake_missing <- complete_seasons %>%
  sample_frac(frac_NA) %>%
  replace_with_na_at(.vars = c("GP", "G", "A", "PTS", "PlMi", "PIM",
                     "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc"),
                     condition = ~ is.numeric(.x))

impute_season_set <- complete_seasons %>%
  filter(!(row_n %in% fake_missing$row_n)) %>%
  bind_rows(fake_missing) %>%
  arrange(X1) %>%
  select(-row_n, -X1) %>%
  mutate(Player = as.factor(Player),
         Tm = as.factor(Tm),
         Pos = as.factor(Pos))
```

After the filtering and NA manipulation we have the data as desired. The sampling fraction of 25% is based on the actual number of missing rows (4273) we have versus the total number of rows (17377). Now to impute. Well, almost. There are a few methods for imputation, I'll cover them quickly.

Nearest Neighbour

Imputation with Nearest Neighbours is reasonably approachable. On the outset we specify a neighbourhood size k. This is usually done through a process of cross validation. The k symbolizes the number of neighbours to compare a record with missing information against. The neighbourhood is then populated with the records that are more closely related to the record of interest. Here the columns that will determine the nearest neighbour are the ones that are not missing: Player name, team, age, and year of play. A player will then naturally have their neighbourhood populated by their previous seasons and those of teammates or players of the same position in the same year. These seem reasonable fits at the surface.

Now to perform the Nearest Neighbour imputation.

```
kNN_imputed_set <- kNN(impute_season_set, k = 40)
View(test_imp_2)

# Cross validate to choose K
test_k <- seq(5, 50, by = 5)
knn_impute_list_cv_5_50 <- list()
for(i in test_k){
  print(paste0("Running with neighbourhood size ", i))
  knn_impute_list_cv_5_50[[(i %/% 5)]] <- kNN(impute_season_set, k = i)
}
```

```
# Test the results to identify best imputation
  # Truth is complete_seasons
  # Imputed sets are knn_impute_list_csv
comp_fn <- function(knn_results){
```

```r
  # Saved in wrong spot, handle errors
  if(length(knn_results) == 0){
    return(0)
  }
  # Get squared error
  only_imp_data <- knn_results[, 1:21] %>%
    arrange(Year, Player) %>%
    select(c("GP", "G", "A", "PTS", "PlMi", "PIM",
             "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc")) %>%
    scale()

  only_act_data <- complete_seasons %>%
    select(c("GP", "G", "A", "PTS", "PlMi", "PIM",
             "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc")) %>%
    scale()

  error <- only_imp_data - only_act_data
  tot_error <- sum(error^2)

  return(tot_error)
}
```

```r
error_tracker <- lapply(knn_impute_list_cv, comp_fn) %>% as_vector() %>%
  as.data.frame() %>%
  filter(. > 0) %>%
  ggplot(aes(x = 5:15, y = .)) +
  geom_point() +
  xlab("Number of Neighbours") + ylab("Total Square Error") +
  ggtitle("Comparison of Error and Neighbourhood Size for kNN") +
  theme_bw()


bigger_error_tracker <- lapply(knn_impute_list_cv_5_50, comp_fn) %>%
  as_vector() %>%
  as.data.frame() %>%
  filter(. > 0) %>%
  ggplot(aes(x = seq(5, 50, by = 5), y = .)) +
  geom_point() +
  xlab("Number of Neighbours") + ylab("Total Square Error") +
  ggtitle("Comparison of Error and Neighbourhood Size for kNN") +
  theme_bw()
```

Running the above code can be time consuming, at least on my device. Instead I have saved the results and will now load them for plotting. Unfortunately the plot failed to save. Feel free to run the code yourself to obtain the plot, just know that the run time for all of the kNN settings was more than just a few minutes. The description that follows is still accurate to the plot and the conclusion that is reached.

The plot above demonstrates the effect neighbourhood size has on Total Square Error. More neighbours tends to mean better estimates though at the cost of run time. I have run it for neighbourhood sizes of 5 to 15 and for 5 to 50. The biggest decreases occur in the first few steps though the estimates get better as the neighbourhood size grows larger. I will use the neighbourhood size of 40 due to it being relatively close (3%) to the minimum error value achieved and because of the large quantity of missing values we need to impute for each missing row.

Multiple imputation

There is some obvious skipping going on here as there are many other individual imputation techniques (hot/cold deck, random, mean, etc.). These techniques have quickly become outdated as better computational techniques have become available. kNN is still in frequent use because of its slightly smarter approach that comes at a minor cost to run time. It is also easy enough to teach and for that has become a staple. There are other techniques out there, however, that have become excessively popular in recent years. One of these is multiple imputation.

Multiple imputation takes advantage of an abundance of computing power. The algorithm imputes values with a process that will generate some random error but does so multiple times. These multiple imputations create multiple imputed data sets. The real imputed set is then the one made from combining these other data sets, providing both an approximately unbiased estimate and a measure of uncertainty in our estimation. Considering most computers are easily capable of handling multiple imputation and that it is conveniently implemented in multiple statistical softwares, it is not unreasonable to see why this method is popular.

Fortunately, multiple imputation is readily available in R through the well-cited 'mice' package. With so many missing values to impute it makes sense to use chained equations. The tuning parameter in this method is the number of sub data frames to create. I will use 10 here because the process of tuning is computationally enormous and I do not have the resources necessary to push for what may be minor improvements.

```r
tic()
mice_impute <- mice(impute_season_set, m = 10)
toc() # 238564.27 seconds
```

```r
# Now use the entirety of the imputation to gauge error
  # Need to first create a function to calculate the degree of error
error_fn <- function(mice_dataset){

  # Get squared error
  only_imp_data <- mice_dataset %>%
    arrange(Year, Player) %>%
    select(c("GP", "G", "A", "PTS", "PlMi", "PIM",
             "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc")) %>%
    scale()

  only_act_data <- complete_seasons %>%
    select(c("GP", "G", "A", "PTS", "PlMi", "PIM",
             "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc")) %>%
    scale()

  error <- only_imp_data - only_act_data
  tot_error <- sum(error^2)

  return(tot_error)
}

# Testing
completed_mice_imp <- read_csv("mice_5.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   Player = col_character(),
##   Tm = col_character(),
##   Pos = col_character(),
##   PS = col_double(),
##   SPerc = col_double()
## )

## See spec(...) for full column specifications.
```

```
check <- error_fn(completed_mice_imp)

# Get error for whole set
m <- 1:10
mice_impute_error <- data.frame()
for(i in m){
  mice_file <- read_csv(paste0("mice_", i, ".csv"), col_types = cols()) %>% select(-X1)
  mice_impute_error[i,1] <- error_fn(mice_file)
}

avg_error <- mean(mice_impute_error$V1)

# Compare
knn_res <- read_csv("min_error_5_50.csv", col_types = cols()) %>% select(-X1) %>% comp_fn()
```

Comparison

Each of the two imputation techniques take a while to run with MICE being especially slow in the given setting. To justify such a run time, we would need to see error that is magnitudes better on the fake missing data. The error rates for each are as follows. Instead what we observe is kNN obtaining a scaled error of 1.6348894×10^5 while the average error in a run of MICE was 1.6927303×10^5. The differences are negligible in terms of error rate but exceptional in terms of run time. When there are this many missing values in any given row, MICE slows down to the point that it may not be useful. It is a fantastic method for other imputation needs. It just does not quite fit the necessity of this comparison. For that reason I will continue with kNN.

A new thought

While writing this report, I remembered another technique I came across called Missing Forest Imputation. Simply put, imputed values are determined by a Random Forest trained on the complete data. Fitting is done based on an Out of Bag Error meaning that there is no extra leg work to do prior to running this approach.

```
corr_df <- complete_seasons %>% select(-X1, -row_n) %>%
    mutate(Player = as.factor(Player),
         Tm = as.factor(Tm),
         Pos = as.factor(Pos)) %>%
  as_data_frame()

# Having some issues, convert the character columns to factors a priori
factor_impute_set <- impute_season_set %>%
  mutate(Player = as.factor(Player),
         Tm = as.factor(Tm),
         Pos = as.factor(Pos))

mf_imp_res <- missRanger(factor_impute_set, pmm.k = 3, seed = 1234, verbose = 1)
```

```
##
## Missing value imputation by random forests
```

```
##
## iter 1:  ................
## iter 2:  ................
## iter 3:  ................
## iter 4:  ................
```

```
mf_imp_error <- error_fn(mf_imp_res)
```

This was a lucky find as the Missing Forest imputation took a few seconds to run and generated an error of 8.804701×10^4 as compared to the error from k-Nearest Neighbours of 1.6348894×10^5. Considering the error for the missing forest is about half that of the k-Neart Neighbours, I will go on to do the actual data set with missing forests. If there are other great imputation methods out there, feel free to let me know or go about it yourself with this data.

The Real Imputation

```
scraped_set <- read_csv("scraped_hockey_data.csv", col_types = cols()) %>%
  mutate(row_n = row_number())
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
real_imp_valid <- scraped_set %>%
  filter(GP > 20)

missed_2005 <- real_imp_valid %>%
  filter(Year %in% c(2004, 2006)) %>%
  group_by(Player) %>%
  summarize(n = n()) %>%
  filter(n == 2) %>%
  select(Player) %>%
  unique(.) %>%
  left_join(real_imp_valid %>% filter(Year %in% 2004)) %>%
  mutate(Year = 2005, row_n = 1000000) # Filler row_n value for filtering below
```

```
## Joining, by = "Player"
```

```
real_imp_invalid <- scraped_set %>%
  filter(GP <= 20) %>%
  bind_rows(missed_2005) %>%
  replace_with_na_at(.vars = c("GP", "G", "A", "PTS", "PlMi", "PIM",
                     "PS", "EVG", "PPG", "SHG", "GWG", "EVA", "PPA", "SHA", "S", "SPerc"),
                     condition = ~ is.numeric(.x))

real_imp <- scraped_set %>%
  filter(!(row_n %in% real_imp_invalid$row_n)) %>%
  bind_rows(real_imp_invalid) %>%
  arrange(X1) %>%
  select(-row_n, -X1) %>%
  mutate(Player = as.factor(Player),
         Tm = as.factor(Tm),
         Pos = as.factor(Pos))

real_imp_set <- missRanger(real_imp, pmm.k = 3, seed = 1234, verbose = 1) %>%
  mutate(Pts = G + A)
```

```
##
## Missing value imputation by random forests
##
## iter 1:  ................
## iter 2:  ................
## iter 3:  ................
## iter 4:  ................
```

Fewer iterations were required in this step as the algorithm terminates at the first step in which the OOB error does not decrease. Again the run time was quick and hopefully efficient. One small adjustment is made for logical reasons to correct Points to Goals plus Assists. The imputation was reasonably accurate at guessing this but it made a few errors.

The Results

The first step will be to calculate the new totals for players from 1999 onwards.

```
new_stats <- real_imp_set %>%
  arrange(Year, Player) %>%
  group_by(Player) %>%
  summarize(GamesPlayed = sum(GP),
            Goals = sum(G),
            Assists = sum(A),
            Points = sum(PTS),
            PointShare = sum(PS),
            Shots = sum(S),
            ShootPerc = mean(SPerc),
            FirstYear = first(Year),
            LastYear = last(Year))

orig_stats <- scraped_set %>%
  arrange(Year, Player) %>%
  group_by(Player) %>%
  summarize(GamesPlayed = sum(GP),
            Goals = sum(G),
            Assists = sum(A),
            Points = sum(PTS),
            PointShare = sum(PS),
            Shots = sum(S),
            ShootPerc = mean(SPerc),
            FirstYear = first(Year),
            LastYear = last(Year))

change_in_stats <- new_stats %>%
  left_join(orig_stats, by = c("Player")) %>%
  mutate(GamesPlayed = GamesPlayed.x - GamesPlayed.y,
            Goals = Goals.x - Goals.y,
            Assists =  Assists.x - Assists.y,
            Points = Points.x - Points.y,
            PointShare = PointShare.x - PointShare.y,
            Shots = Shots.x - Shots.y,
            ShootPerc = ShootPerc.x - ShootPerc.y,
            FirstYear = FirstYear.x,
            LastYear = LastYear.x) %>%
  select(Player, GamesPlayed,
            Goals,
            Assists,
            Points,
            PointShare,
            Shots,
            ShootPerc,
            FirstYear,
         LastYear)

new_stats %>% arrange(desc(Points))
```

```
## # A tibble: 3,389 x 10
##    Player GamesPlayed Goals Assists Points PointShare Shots ShootPerc
##    <fct>        <int> <int>   <int>  <int>      <dbl> <int>     <dbl>
##  1 Joe T~        1541   411    1062   1474       160.  2924      13.4
##  2 Jarom~        1483   619     648   1267       157.  4594      13.7
##  3 Jarom~        1228   493     744   1237       148.  3841      12.5
##  4 Sidne~         943   446     770   1216       149.  3063      14.4
##  5 Alex ~        1084   658     553   1211       164.  5234      12.6
##  6 Patri~        1665   560     632   1192       136.  4296      13.3
##  7 Maria~        1381   535     627   1162       147.  4318      12.4
##  8 Marti~        1233   417     681   1098       122.  3103      13.1
##  9 Henri~        1406   253     842   1095       111.  1955      12.8
## 10 Danie~        1114   410     648   1059       126.  3162      12.9
## # ... with 3,379 more rows, and 2 more variables: FirstYear <dbl>,
## #   LastYear <dbl>
```

```
orig_stats %>% arrange(desc(Points))
```

```
## # A tibble: 3,389 x 10
##    Player GamesPlayed Goals Assists Points PointShare Shots ShootPerc
##    <chr>        <int> <int>   <int>  <int>      <dbl> <int>     <dbl>
##  1 Joe T~        1511   410    1061   1471       160.  2891      14.0
##  2 Jarom~        1402   591     627   1218       151.  4436      13.1
##  3 Sidne~         943   446     770   1216       149.  3063      14.4
##  4 Alex ~        1084   658     553   1211       164.  5234      12.6
##  5 Jarom~        1152   465     721   1186       141.  3676      12.1
##  6 Patri~        1583   538     596   1134       129.  4072      13.4
##  7 Maria~        1302   525     608   1133       144.  4219      12.5
##  8 Henri~        1330   240     830   1070       109.  1856      12.8
##  9 Danie~        1306   393     648   1041       123   3474      11.2
## 10 Marti~        1134   391     642   1033       115.  2884      13.0
## # ... with 3,379 more rows, and 2 more variables: FirstYear <int>,
## #   LastYear <int>
```

```
new_player <- new_stats %>% arrange(desc(Points)) %>% select(Player) %>% head(n = 10)
orig_player <- orig_stats %>% arrange(desc(Points)) %>% select(Player) %>% head(n = 10)

res <- bind_cols(orig_player, new_player)
names(res) <- c("Original", "Imputed")
res
```

```
## # A tibble: 10 x 2
##    Original          Imputed
##    <chr>              <fct>
##  1 Joe Thornton       Joe Thornton
##  2 Jarome Iginla      Jarome Iginla
##  3 Sidney Crosby      Jaromir Jagr
##  4 Alex Ovechkin      Sidney Crosby
##  5 Jaromir Jagr       Alex Ovechkin
##  6 Patrick Marleau    Patrick Marleau
##  7 Marian Hossa       Marian Hossa
##  8 Henrik Sedin       Martin St. Louis*
##  9 Daniel Sedin       Henrik Sedin
## 10 Martin St. Louis* Daniel Alfredsson
```

The results are about what would be expected. The players with the most points since 1999 do little in swapping positions. We see that Jaromir Jagr passes both Sidney Crosby and Alex Ovechkin on the back of what would likely have been a strong campaign in the lockout season. He compensated for this a bit in real life, putting up 123 points in 2005-2006, but the whole point of this analysis is to shed light on the what ifs of the past 20 years.

Other movers include the pushing up of Martin St. Louis, a player who missed both the lockout season and the 2013-2014 season. Even nearing the end of his career, St. Louis was still amassing 60 points per season routinely. Giving him these two extra years could have reasonably provided him with 120 or more points. It is not surprising to see him shift up. Daniel Alfredsson also makes an appearance on this list, though this is more due to the imputation giving him a good lock out year and Daniel Sedin a bad one. Either way, the two of them rightfully deserve a spot on this list as two tremendous players.

The Pros

This analysis has lead to an interesting new data set, connecting the real world results with the what ifs that are left unanswered. I won't go too far into the differences in this write up, perhaps that is best left for another time. This report also accomplished the introduction of imputation to those that may not be familiar with it while also thining the pack of options when it comes to imputation. Finally, as a sanity check, the top players still hold the top spots and no well-established players are moved intensely. This suggests that small corrections are being made, as per expectations.

The Cons

Players that had many years of flickering between the NHL and minor leagues or had careers completely thrashed by injuries had many seasons imputed. Unsurprisingly, their results were imputed based on NHL player data. This means that even if they were fluctuating because they were not NHL caliber, they were imputed as if they were a true NHL player. Household names like Mike Blunden and Corey Potter each had more than 400 games imputed. The suggestion is then naturally to be wary when dealing with the lower ranked players.

Summary

Imputation is fun and useful in sports. We can identify appropriate imputation techniques by creating fake missing data and testing for minimum error. Through the testing in this situation, missing forest imputation proved to be most effective. Applying this to the NHL, we were able to create a reasonable data set with which to compare players as if they had not missed seasons in their career due to lockouts or injury.

The code and data sets, as always, will be made public through my github page: https://github.com/mreyers/sports_analytics_data. I left a lot of exploration out of this paper to encourage others to go play. Have fun, and message me on Twitter or whatever medium you are using when you find something interesting!

Thanks for reading,

Matthew Reyers