NBA Player Standalone Game Performance Statistics, Player Position Classification Modeling

ALEX SALCE, MATH574M | Fall 2023 | University of Arizona

Abstract

Background: Data was gathered for individual player standalone game performance from the offical NBA database using *nba_api* Python client. The data was filtered by individual seasons from 2017-2023 to include only player performance features, while excluding any physically identifying features and scaling any enumerated statistics (i.e. non-percentage statistics) by time played in game. A response data set corresponding to the season performance data was generated based upon a selected position (Center, Power Forward, Small Forward, Shooting Guard, Point Guard) and converting player position to a 0/1 binary classification array where 1 indicates selected position and 0 indicates any other position than selected.

Methods: A training data set was generated by first selecting a season (2017-2023), then by selecting the player position of interest and generating a predictor matrix and binary response array. Test data was subsequently generated for each individual season excluding the training season by the same procedure. The training data would train a binary classification model, which would then be used predict position classification using a test season and the misclassification rate would then be calculated. Training error was also captured at each step. This process would be repeated for all other test seasons. The binary classification models used were Linear Discriminant Analysis (LDA), and Logistic Regression. Features selected from a Logistic Regression model were also used to re-filter training and test data to train a new LDA model for analysis and subsequent rerun. Lastly, the filtered data would be used to generate the best AIC and BIC models, and the selected features for the best models would be logged, the data refiltered, and subsequent analysis run once more.

Results: Logistic Regression exhibits best overall performance with this classification problem, but LDA performance is comparable. Feature selection greatly reduced dimensionality and minimally impacted performance of LDA (although there was a slight decrease in performance). Top 5 features were captured dynamically for each training data set in logistic regression to give insight to important features for each position by season. All model results performed best in classifying Centers and Shooting Guards and worst for Small Forwards.

Conclusions: All results made intuitive sense. Feature selection generally favored features that were subjectively reasonable by position. Position classification performance also made sense; the models did best with highly specialized positions and did not do as well with positions that play a variety of roles in a game, and all performed similarly to each other by distinct positon class.

Introduction

I have been a fan of the NBA for a long time. It's a sport that is ever evolving, and not just players, strategy, and talent. Data has also evolved with the sport, particularly in the last decade, as the power of data analytics has been unlocked. Entire companies are now dedicated to sports data and the NBA; sites like basketball-reference.com, StatMuse, StatHead, and Cleaning the Glass are a few popular examples of sports data sites in the NBA fan community. Second Spectrum, a sports data tech company, even has an official partnership with the NBA. Clearly, data is highly valued in sports, and particularly so in basketball.

Today, there is so much more data being captured than a traditional box score; "hustle" stats, positional tracking data and subsequent statistics, and "advanced" box scores to name a few. This new data describes player and team performances in higher fidelity, giving much more descriptive insights for how players and teams really play. And thankfully, much of this data has become accessible to the public. For this project, I wanted to pursue some objective insights for large data sets; both high dimensional and with large sample sizes, and the available NBA data seemed to lend itself well to this pursuit.

asalce@arizona.edu. University of Arizona. github.com/alexsalce.

Motivation and Objective

My project goals were to build tools to collect NBA data, to refine what I collect, and most importantly to leverage the methods we have learned and exercised throughout the semester to generate insightful analysis of the data. In all, I desired a a flexible framework for analyzing any varieties of questions that might be interesting. Given the size of the data, I felt that it made most sense to start small and approach the analysis of the data in a practical and interpretable way. This meant that I should tackle a problem for which I have some intuition, and in the process set up a framework which I can continue to leverage in the future. The question that seemed most practical to start with this data was in the spirit of finding qualitative insights to these "advanced" stats: How well can performance statistics can describe a player? Specifically, can performance statistics alone could be used to identify a player's position?

Terminology

NBA Reference

Links to resources for reference to NBA stats and position terminology.

- NBA Box Score
- Official NBA Stats Glossary (Advanced Stats)
- Basketball Position Descriptions

There are four collections of features that will be used for training and test data.

- Traditional Boxscore - Standard summary stats for each player per game (see "NBA Box Score" above)
- Hustle Stats - Stats that describe "effort" plays not captured by traditional boxscore (see "Hustle Stats" above)
- Advanced Boxscore - "Advanced" stats that typically summarize performance calculated by other performance inputs. Note, as we will see, some of these advanced stats, among others, are highly correlated with other features.
- Player Tracking Stats - Summary statistics calculated from in-game player positional tracking data.
- NBA Seasons will be referenced by only their *start* year, i.e. the "2022" season is referencing the "2022-2023" season.
- Player positions will may be referred to by number, where 1-**PG**, 2-**SG**, 3-**SF**, 4-**PF**, 5-**C**.

Code

Python code chunks add ">" to each line of code and have sand background color. R code chunks do not have ">" and have default chunk background color.

Nomenclature

- Dimensions of the data will be expressed in terms of the number of features (predictors) p, and the number of observations n. A training or test data matrix, for example, will have dimensions $n \times p$, and a response array will have dimensions $n \times 1$.
- "Test Error" and "Train Error" describe the misclassification rates for model predictions against the true

response. Misclassification rate is defined as

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where y_i is the training or test response data and $\hat{y}_i$ is the predicted response data.

- "Binary Classification" and "binary classifiers" in this report implies $0/1$ binary response assignment, rather than $-1/1$ classification typical of some methods.

Data

Data collection, assembly, and refinement were among the most challenging aspects of the project. I was able to find some great resources and ultimately produce a dataset with a satisfactory number of features and observations.

Please note that code chunks provided in this report are only to contextualize attached code and aid in understanding functional approach. These chunks report only the core functionality, and accompanying code will look somewhat different.

Data Collection

The data is sourced directly from the official NBA stats database, using a freely available API client package for Python nba_api [GitHub]. The following endpoints were utilized for data collection.

- PlayerGameLog - Traditional Boxscore
- BoxScoreAdvancedV3 - Advanced Stats
- BoxScoreHustleV2 - Hustle Stats
- BoxScorePlayerTrackV3 - Player Tracking Stats

To queue the endpoints, a dictionary of players was built using the players static endpoint from the API client. Note, for this project, I used only NBA players that are currently active to build my data from the endpoints. Using the CommonPlayerInfo endpoint, I started with building a database of players and profile data (which includes position).

```
> import pandas as pd
> from nba_api.stats.static import players
> from nba_api.stats.endpoints import commonplayerinfo
>
> nba_players = players.get_active_players()
> i=0
> nba_players_info = {}
> for player in nba_players:
>     player_info = commonplayerinfo.CommonPlayerInfo(player_id=player['id'])
>     nba_players_info[i]  = player_info.get_data_frames()
>     i = i+1
> #compile into df for csv export
> data=[]
> df=pd.DataFrame(data)
> for entry in nba_players_info:
>     df_new = nba_players_info[entry][0]
>     df = pd.concat([df,df_new], axis=0)
> df.to_csv(path_or_buf='C:/.../nbaplayers.csv')
```

Using this dictionary of active NBA players, each player's stats were collected from endpoints for every game the player had played for their entire career. An example of this data collection is provided below.

```
> #OMITTED: other imports
> from nba_api.stats.endpoints import boxscoreadvancedv3
>
> #OMITTED: existing data import, exception handling, building dictionaries, etc
>
> #get all advanced stats for active player games
> for GAME_ID in nba_games['Game_ID']:
>     try:
>         #OMITTED: some counter and status code
>         #I needed to add a status ticker, the scrape took a long time to run
>         #and I needed a visual to keep track
>         player_playertrack = boxscoreplayertrackv3.BoxScorePlayerTrackV3(
>           game_id=gameid).player_stats.data
>         player_playertrack_df = pd.DataFrame(player_playertrack['data'])
>         player_playertrack_df = player_playertrack_df.set_axis(
>           player_playertrack['headers'],axis=1)
>         df = pd.concat([df,player_playertrack_df], axis=0)
>
> #OMITTED: exception handling and export to CSV file
```

Some notes about the omitted portions of the code: Wherever possible, I would store data in a CSV and import to keep consistent and avoid redundant scraping. There were some exceptions thrown for certain games during the scrape, which I did not investigate and added to a dictionary/CSV to avoid trying to queue again for subsequent data scrapes; they were filtered out of subsequent scrapes to keep all data consistent. Exception handling was necessary as well as a visual progress meter; some of these scrapes took more than a few hours. This process was repeated for each endpoint, and CSV files were generated for each set of generated data.

Data Assembly and Refinement

Once data was collected from each of the desired endpoints, a separate script joined all datasets on Player_ID and Game_ID, which are unique player and game identifiers. Observations (entire rows) containing any NaN data were dropped during the join.

```
> join_hustle_advanced = pd.merge(nbagames_hustledata,
>                                 nbagames_player_advancedstats,
>                                 how='left', left_on=['Player_ID','Game_ID'],
>                                 right_on = ['personId','gameId'])
> #OMIT other joins
> df = join_allplayer.dropna(subset=['minutes_y'])
```

Lastly, some obviously redundant data was present, and removed "by hand" (using Excel) and stored as 'nbaplayerdatafiltered.csv' before being imported to R.

```
nbaplayerdata <- data.frame(read.csv("data/nba_api/nbaplayerdatafiltered.csv",
    header = T))
```

Once imported in R, some final cleanup is applied before generating Training and Test Data, removing games in which player logged no playing time, conversion of playing time to seconds, partitioning enumerated stats and scaling by seconds played. This is stored in dataframe 'player.stats'.

```
library(lubridate)  #used to convert time played to seconds
# OMITTED: basic filtering of non-performance
# features and other redundancies, also removing
```

```
# seasons prior to 2017 NOTE: 'player.stats' is
# dataframe of filtered data convert 'minutes'
# column to 'seconds
player.stats$minutes <- period_to_seconds(ms(format(as.POSIXct(parse_date_time(x =
 ↪  player.stats$minutes,
    c("HMS", "MS"))), format = "%M:%S")))
# REMOVE any rows with no playing time
player.stats <- player.stats[-which(player.stats$MIN ==
    0), ]
# any nans
player.stats <- na.omit(player.stats)
colnames(player.stats)[grep("minutes", colnames(player.stats))] <- "seconds"
# OMITTED: additional redundant data filtering
# partition enumerated stats to divide by time
# played in game '...' omitted for cleanliness
stats.timenormal.headers <- c("FGM", "FGA", "FG3M",
    "...")
stats.scaleby.time <- player.stats[, which(names(player.stats) %in%
    stats.timenormal.headers)]
# stats scaled by time; this dataframe is
# combined with non-enumerated stats in test and
# train dataframe gen function
player.stats.timenormal <- stats.scaleby.time/player.stats[,
    grep("seconds", colnames(player.stats))]
```

Train and Test Data Generation

```
data.fn <- function(player.stats, player.stats.timenormal, seasons,
    year, position, binary, threeclass) {
    # filter data by year
    stats.player.year <- player.stats[player.stats$SEASON_ID ==
        seasons[(2024 - year), ], ]
    stats.player.year.timenormal <- player.stats.timenormal[player.stats$SEASON_ID ==
        seasons[(2024 - year), ], ]
    # select all non-player characteristic data
    x <- stats.player.year[, 24:ncol(stats.player.year)]
    # remove the columns that will be replaced by time
    # scaled data
    x <- x[, -which(names(x) %in% stats.timenormal.headers)]
    # combine raw stats with time scaled stats
    x <- data.frame(x, stats.player.year.timenormal)
    # get response data
    y <- data.frame(stats.player.year$POSITION)
    # set selected position to 1 for response array
    if (binary == TRUE) {
        # Center
        if (position == 5) {
            y[y == positions[2, ]] <- 1

            y.cv[y.cv == positions[2, ]] <- 1
        }
        # Center-Forward and Forward-Center ...OMIT other
        # IF statements
```

```
        # set others to 0
        y[y != 1] <- 0
        y <- mutate_all(y, function(x) as.numeric(as.character(x)))
    }
    out.data <- list(y, x)
    return(out.data)
}
```

```
# select year and position
train.year <- 2020
pos <- 3
# run data gen function
train = data.fn(player.stats, player.stats.timenormal, seasons,
    train.year, pos, TRUE, FALSE)
# generate training data and response
x.train = train[[2]]
y.train = train[[1]]
# TEST data generated similarly using for loop to gen data
# for other seasons
```

Data Summary Statistics

Table 1:

Dataframe	n	p	Description
nbaplayerdata	126742	122	full dataset captured from nba api
player.stats	126436	99	player stats post R processing
player.stats.timenormal	126436	45	filtered and time scaled stats
x.train x.test	see Table 2	76	test and train data

Table 2:

Season	n
2023-24	3255
2022-23	25774
2021-22	22704
2020-21	18125
2019-20	15836
2018-19	15507

Table 3:

Stat.Category	p
Traditional Boxscore	19
Advanced Boxscore	22
Hustle Stats	15
Player Tracking Stats	20

Model Selection

The original motivation was to determine how well a model could differentiate a player's position from the other positions based solely upon performance statistics from a standalone game. Additionally, the model should have interpretable results; it is important that we can extract some information regarding the features of the player's

performance. Since I have intuition about the sport, having interpretable results would give me a baseline evaluation for whether the model is telling me something reasonable or outrageous.

By nature, this is a classification problem, and we could reasonably select a multiclass or binary classification modeling approach. In this case, I wanted not only to be able to discern whether a player performs like other players of their position in-game, but also to extract the features that best describe how other players of that position are playing. This being the case, I decided to approach as a binary classification problem; the model would classify whether or not a player played like other players for the particular position of interest. Lastly, for the model to be interpretable and return information about the features, candidate models should be supervised learning methods.

Given these parameters, there were some different possibilities that I explored.

Not Selected

- Adaptive LASSO
  - Pros: Feature Selection, Oracle Properties
  - Cons: Prediction performance due to highly correlated predictors (exhibited in our data).
- Adaptive Elastic Net
  - Pros: Handles collinearity issue better than adaptive LASSO.
  - Cons: Not Oracle method, computation time, nonlinear shrinkage.
- SVM
  - Pros: Prediction performance
  - Cons: Non interpretable (hard classifier), computationally prohibitive for data size.

Selected

- Linear Discriminant Analysis (LDA)
  - Pros: Soft classifier, computationally favorable
  - Cons: Not robust against outliers, more assumptions about distribution of data
- Logistic Regression
  - Pros: Soft classifier, Feature selection (significant predictors, information criteria), $\hat{\beta}$ estimates are consistent, distribution of $\hat{\beta}$ converges (data set is large), more robust to outliers
  - Cons: less computationally favorable

It is not too much of a stretch to assume that each feature is normally distributed; each performance statistic is a measure of outcomes of random events in the game with some mean and variance that can be described by the population of players at each position, so both methods make sense to proceed with analysis.

Model Evaluation and Results

Packages

```
# data analysis libraries
library(MASS)  #LDA function
library(lubridate)  #date and time manipulation
library(dplyr)  #data manipulation
library(PredPsych)  #built in 5fold CV function for LDA
library(reshape2)  #melt function used for data manipulation
library(leaps)  #for bestglm
library(bestglm)  #bestglm() function uses leaps and can output best AIC and BIC model, used for
↪  AIC and BIC feature selection
```

Evaluation Procedure

- **LDA and Logistic Regression**
  - Function 'lda.analysis' takes full player stats data as well as desired seasons for analysis, and the desired player position (1 through 5) as inputs. The function loops through each season, generates a train LDA object using 'lda' function, uses the train LDA object to generate a prediction array $\hat{\mathbf{y}}_{test}$, and compares to the true train response array $\mathbf{y}_{\mathbf{train}}$ to calculate TrainErr. A 5 fold CV error is also computed with training data using 'LinearDA' function from the 'PredPsych' package. Then, an embedded loop runs through the other seasons using the train LDA object and test data to generate a $\hat{\mathbf{y}}_{test}$ prediction array. The predicted array is compared to the true test response array $\mathbf{y}_{\mathbf{test}}$ to calculate TestErr. Results are returned in individual dataframes. TestErr dataframe column corresponds to training season, each row logs TestErr for test season prediction with LDA trained by training season.
  - Function 'glm.analysis' uses same procedure for model fit and train & test error as 'lda.analysis', using 'glm' function function for logistic regression fit. The function also evaluates which features are significant for training fit and returns the features retained and dropped at an $\alpha \leq 0.05$ level, as well as top 5 features by significance.
  - Function 'fs.analysis' takes same inputs as 'lda.analysis' and generates both a glm as well as an lda training model object, and the computation time of fitting each model is captured. From the glm object, the significant features at an $\alpha \leq 0.05$ level are stored and the original training and test data matrices are filtered to only the significant features. The logistic regression and LDA analyses are then rerun using the same procedures in 'glm.analysis' and 'lda.analysis', also capturing the computation time of the refit models. The function returns test and train errors for the rerun test data as well as all computation times.
  - Function 'ic.analysis' takes same inputs as 'lda.analysis' and generates a glm training model object. From the glm object, the significant features at an $\alpha \leq 0.05$ level are stored and the original training and test data matrices are filtered to only the significant features. Using the 'bestglm' function from the 'bestglm' package (which utilizes 'leaps' package), the best models based on AIC and BIC are returned, and the coefficients of the best model stored. Note, given the size of the training data, the computation time for the 'bestglm' function can vary from 5 second to 5 minutes, depending on the number of significant features from the original glm fit were retained. The AIC and BIC models could not be generated in any reasonable amount of time without some prior feature selection, hence using the initial feature screening. Due to some issues with code/time, only the computation time of glm and LDA refits using the AIC and BIC features are captured.

Results

See accompanying file 'MATH574M Final Results.xlsx' for all results data

- Error Comparisons
  - Figure 1 graphs summarize the average training and test error over all seasons by position, respectively. The training error graph includes the mean training error and a 1-standard error bar that captures the standard error within training error for a method by each position. The test error graph includes a mean test error point, a smaller set of points for average test error for a particular training model, and 1-standard error bars for each of these smaller points. Although a little noisy,

it captures the test standard error for each training year, and gives a visual for the test error bounds of the results.

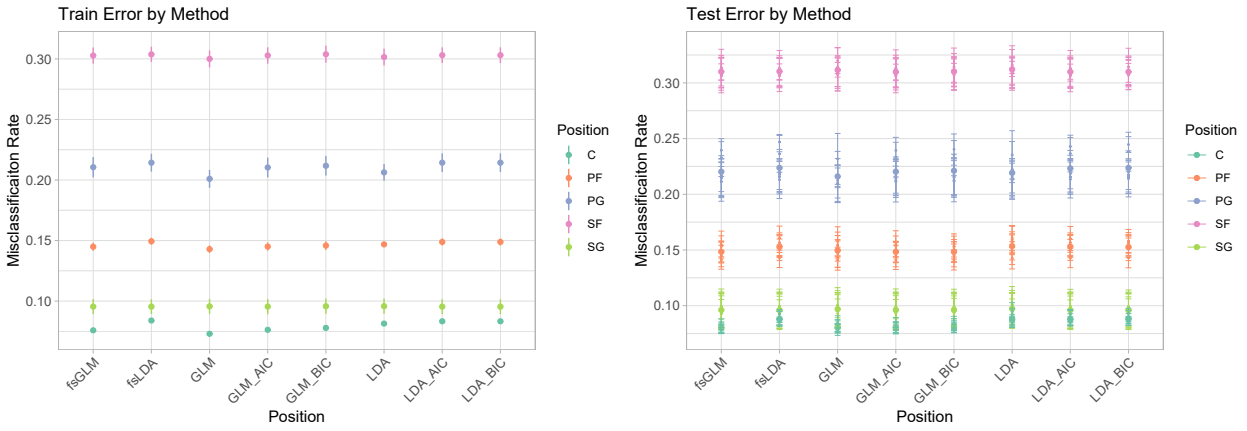

Figure 1: Methods Comparison

- Feature Selection
  - Figure 2 "Feature Selection" summarizes the number of features selected by year for each feature selection method, as detailed in the 'Model Evaluation' section.
  - Figure 2 "Top Features" count features that appear in the top 5 for a position class by season for at least 3 seasons for a given position to give a qualitative description of feature selection by position.

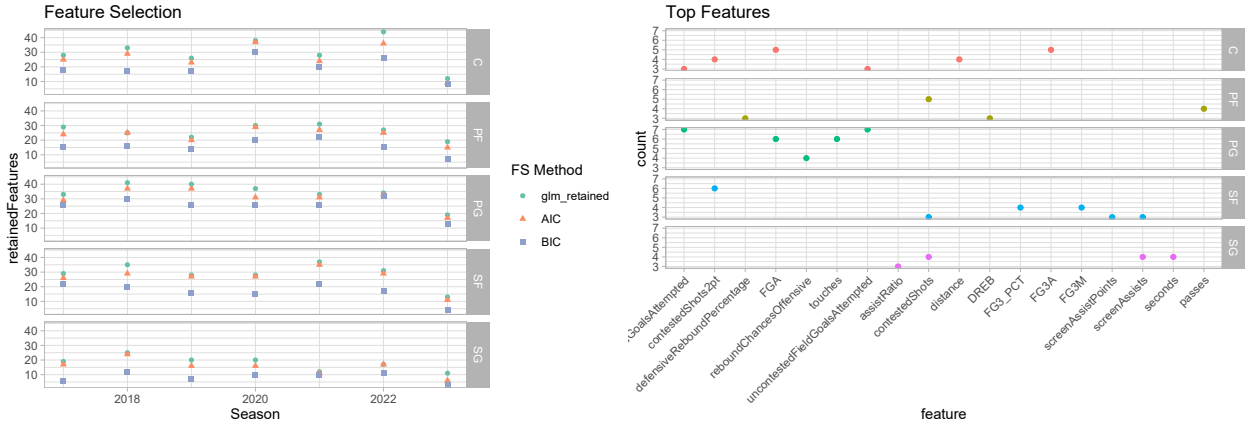

Figure 2: Feature Selection

- Computation Time
  - Figure 3 summarizes the computation time of the model training for each season using features selected by corresponding method.

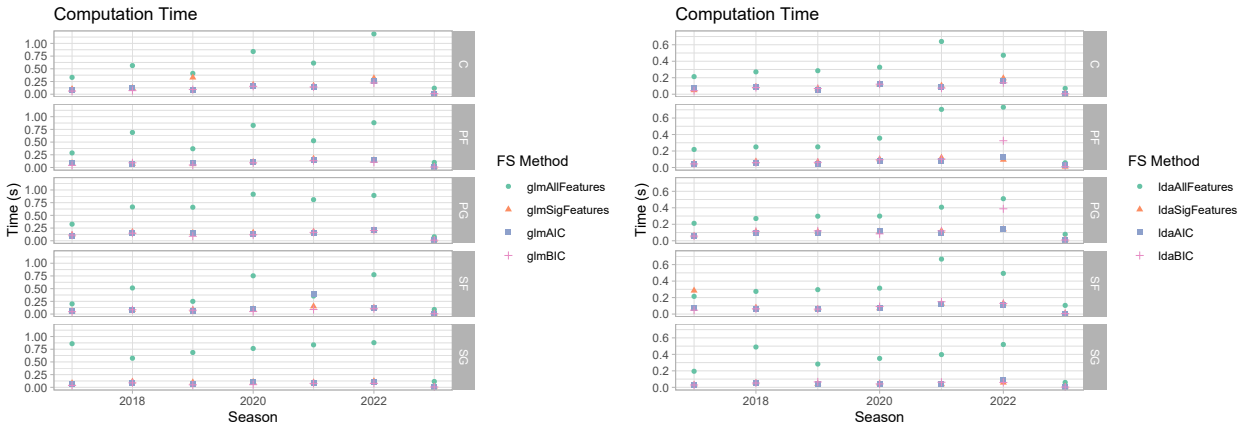

Figure 3: Computation Comparison

Comparisons and Conclusions

Error Comparisons

Overall, both the LDA and Logistic Regression perform fairly well in identifying player position class from game performance data. We can see that in general, feature selection does not impact performance substantially. The standard error for training and test are also not alarming, the results are generally fairly stable.

There are some clear differences in the performance by position class, and I offer some of my own intuition to at least gauge whether this seems reasonable..

- Model error performance by position
  - *Best*: Centers and Shooting Guards ($\sim 10\%$ misclass)

    For a typical basketball team, a Center is easy to spot due to their large stature. But the role that they play within a game is also generally easier to discern than other positions; they are typically near the basket, shooting close and contested shots at the rim, grabbing rebounds, and on defense covering the other team's center who is often doing the same. They are generally less athletic, quick, and often cannot shoot well from very far away from the basket. It seems very reasonable that their performance stats are rather distinct due to overall style of play.

    Shooting Guards may be a little less intuitive to a casual fan, but they often execute highly specialized roles within a team's strategy. They generally play "off ball" (they don't have the ball in their hands much), set screens, assist other players, and "catch-and-shoot" the ball more than other players. They are typically not defensively dominant players generally due to physical limitations like size. So, it's not surprising that their performance statistics may be more distinct than other players.

  - *Fair*: Power Forwards ($\sim 15\%$ misclass) and Point Guards ($\sim 22\%$ misclass)

    Point Guards tend to have the ball the most in the game, and consequently they can have a large variety of offensive statistical performance categories. They shoot, pass, and dribble more than most other positions, and often orchestrate how plays are executed. Defensively, they are often guarding other point guards, and are consequently involved in many defensive actions as well. It is reasonable that, given the variety of statistical categories that they may perform well in for a particular game, it's reasonable to

believe that it may be more difficult to differentiate them.

Power forwards typically play similar roles to Centers, but are generally more athletic. They will often be utilized to grab rebounds, to shoot near the basket, and play defense between the basket and the three point line. There is a little more variety in how power forwards can perform since they can move around the court more as compared to centers, so the classification error rate for this position makes sense.

- *Worst*: Small Forwards ($\sim 31\%$ misclass)

In the modern NBA, by far the most versatile position is the small forward. They generally have enough size, skill, and athleticism to do just about anything that any other position can do. A great example is LeBron James, who is listed as a small forward but can play virtually any position on the court. It's particularly unsurprising, then, that it would be most difficult to discern small forwards from the other positions, so this result is entirely reasonable, but still does surprisingly well!

Feature Selection

We can see features selected for position by year vary, which may imply that some positions play the game a little differently overall each year. The "Top Features" chart shows the features that recur between seasons (3 or more seasons), and the results gave me qualitative confidence in the model's performance, and also gave some interesting insight into what features may best describe each position.

The top features for centers were field goal attempts and 3 point field goal attempts; Centers take the smallest number of shots of any player overall, and especially the smallest number of 3 pointers. Guard play features were reasonable; point guards take many field goal attempts, and generally have the ball the most, and shooting guards set many screens offensively, and also shoot the ball. There was much more variety in the forward positions, so it was a little harder to discern what performance stats were really "most descriptive", but intuitively this is not an unusual finding.

Computational Comparisons

Feature selection clearly improves computational performance over using all of the original features, however there is not a huge benefit beyond the significant features for the further AIC and BIC refinement of the model from a computational perspective. Given the size of the data, the computation time just to acquire the best AIC and BIC model is prohibitive anyway. There are not statistics for these captured in this report, but the time could vary from a few seconds to as long as several minutes to recover a model that may improve feature selection by a small handful of features. LDA and logistic regression models are fairly quick to begin with, so I felt that the best value procedurally was to just capture the significant features from the logistic regression model and leave it at that.

Conclusions

Overall, LDA and logistic regression were quite reasonable models to address the original objective of player classification by standalone game performance statistics. The findings were rather surprising to me in the sense that they were much more reasonable and intuitive that I had expected based on previous experience during the semester with these and other models/data sets. Although this problem itself is fairly novel, the framework that this has provided for future analysis is of great value for my interests. There is certainly plenty of potential for further analysis of these results and beyond, so I look forward to exploring this data in greater depth. Further

analysis using other nonparametric modeling approaches like random forests and PCA would fit some other problems using this data nicely.

For reference, in the footnote of page 1 I provided my department webpage as well as my GitHub as links for where I plan to store future analysis and any elaboration that I may pursue using this data.

References

- Macdonald, B. (2020). "Recreating the Game: Using Player Tracking Data to Analyze Dynamics in Basketball and Football" . Harvard Data Science Review, 2(4). https://doi.org/10.1162/99608f92.6e25c7ee
- Kubatko, Justin; Oliver, Dean; Pelton, Kevin; and Rosenbaum, Dan T. (2007) "A Starting Point for Analyzing Basketball Statistics," Journal of Quantitative Analysis in Sports: Vol. 3 : Iss. 3, Article 1.
- Puranmalka, Keshav. "Modelling the NBA to make better predictions." (2013).
- Jingru Wang and Qishi Fan (2021) "Application of Machine Learning on NBA Data Sets," J. Phys.: Conf. Ser. 1802 032036
- Santos, J., Mendez-Domínguez, C., Nunes, C., Gómez, M.A., Travassos, B. (2020) "Examining the key performance indicators of all-star players and winning teams in elite futsal", International Journal of Performance Analysis in Sport, 20:1, 78-89.
- Hastie, Trevor,, et al. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction". 2nd ed. New York, Springer, 2009.