Jacob Dineen & Mason David
Scripting for Data Analysis
8/31/2018
Final Project Report

This project will center around collections of baseball data ranging from 1871 - 2014, essentially encompassing all recorded statistics over the course of the history of Major League Baseball. Sean Lahman and a group of researchers are responsible for the collection and storage of most of the main files we will be working with, although there have been some crowdsourcing attempts at expanding the original sets to include such things as college statistics, and various player metadata.

As this is simply a report of our project, we will not be including and code here. All code is submitted in alongside this report in a .ipynb jupyter notebook file, which has code comments and annotation of code chunks, .py (converted), and pdf converted from the raw jupyter notebook.

Roles

Mason and I both wanted to get familiar with different aspects of python that we view as potentially viable options in our own professions. As such, there was no formal declaration of role assignment. Instead, we each branched off and innovated new ways to express what we set out to do, broadly. One of our main goals throughout this project was to be able to analyze, descriptively and predictively, sports statistics. The descriptive task is all encompassing, as we could, and did, go many different ways here - From information retrieval functions, to distribution analysis, to time series evaluation. We decided to both work with free reigns and consolidate the portions of our code that we deemed valuable to our overall analysis, and to the flow of the ultimate story that we are trying to convey. While some of our work was done with base python, we found ourselves wanting to use pandas for most of our operations, and since we are dealing with tabular data it made sense to do so. We relied heavily on Scikit Learn for aspects of this project related to machine learning, as their intuitive functional API makes fast prototyping of predictive models, particularly with Grid Searching, a more simple task. I thought about using Keras or Tensorflow, but I thought that the black box nature of a neural net might detract from one of the sub-stories I was trying to tell, which was to visualize feature ranking when it comes to mapping a set of input features (batting statistics) to a binarized target variable representing hall of fame induction status.

Overall Description of our Program

The below is a chronological flow of what our program accomplishes, chunk by chunk:

1. Scrapes/Fetches .csv URLs for easy reading into pandas dataframes.
2. Loads relevant datasets into individual dataframes.
3. Analysis of data structure. Merges pitching and batting records with player metadata. Drops unnecessary columns, statistics on Null count distribution by feature. Cleaning of null data.
4. Visualizations/time series Analysis
5. Master function for information retrieval on batting/pitching records. Uses pandas to slice/filter copy of original df based on user entered arguments. Example on jupyter notebook: iterate through a list of keystatistics and run each index of the list through the function to output the top 5 players in each statistic, in descending order, all time. This can be toggled per category, statistic, sliced by individual year (defaults to alltime records), and has an option for how many players you want to output.
6. Moving onto modeling, the next chunk merges our aggregated batting dataset with our hall of fame lookup. We filter on the lookup to make sure that we only pull 'players' elected to the hall, and not executives or managers. Then, the target variable is binarized/dummied.
7. This is an interesting classification problem, as the minority class represents less than 1% of the total distribution of our class variable. As such, we use upsampling techniques to randomly draw from our majority class, with replacement, until the length of our minority class is equal to our majority class. We then concatenate our minority with our new majority and have a dead even frequency distribution between hall of famers and non electees. We also explored with downsampling, rather than upsampling, because we though we might be able to better judge a model that hasn't been exposed to 100% of the training data. Some additional metadata columns are removed (playerid, playername) as they are unique to a set of features and don't add any information to our models (similar to how using a sequential feature is misguided because it adds too much variance during induction). We then split our datasets into 2, one for our dependent variable, and one for a collection of our independent features.
8. The next step is breaking our data into train and test split - We use Scikit Learn for this next portion of code. After splitting, we GridSearch through a range of relevant parameters with a gradient boosted classifier - We do this to find the best set of estimators/hyperparameters for our model. After finding that best estimator, we fit on our training set, and score on our testing set. As seen in the script, this results in an accuracy of 99.3%. We also visualize some of the errors that we are making via confusion matrices and classification metrics, all of which can be seen in individual chunks within our script.
9. Now that we have a model, we write a function that takes a user input on player to output hall of fame propensity. This step is a bit interesting because we can't really error catch

on all mismatched strings. Instead, we utilize fuzzy matching to find the closest match to our user input. This function was written to support actual python input through a chatbox, and actual input through a function parameter. An interesting thing we do at this step, within this function, is call our saved pickle file with the model parameters. This means that we we don't have to retrain our model each time that the program is run - Technically the chunk of code that trains the model could be left out of the final script, with the only lines remaining being the call to our directory to load our PKL file with our best estimators for our model. I know that this is often how models are put into production, which drastically reduces the time taken to predict something when inference is needed - Essentially, we'd normally have our training happening in a different script, and only reference the PKL file in our main script.

Collecting the Data

All data is available to us via a github repository. To explore some of text mining/web scraping tactics, and to express some additional learnt skills, we will aggregate the CSV links in using urllib and Beautifulsoup. In our script, we've written a function that takes a url and a base url as parameters, parsing the HTML to render specific tags that contain a specified string, in our case 'csv'. After we can extract that portion of a string, we concatenate it with our base URL to render a full destination that can be read in to a more structured format through the use of pandas. 27 csv links were parsed off the page, and it was brought to our attention that we might need to reduce the overall scale of our assignment. While all of the data together is interesting, it isn't exactly human readable. In short, we decided only to use data pertaining to actual hitting or pitching statistics. Those sets of data are indexed by an ID, which we later needed to merge against a player profile representation, but we thought that our story would be much more accessible at a higher level. *For instance, each of the three datasets that we decided to use, and merge in some form, were shape x by ~30. We could quickly have exploding dimensionality and a lack of interpretability.

```
Scraping Webpage: https://github.com/chadwickbureau/baseballdatabank/tree/master/core
Number of datasets: 27

['https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/AllstarFull.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Appearances.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/AwardsManagers.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/AwardsPlayers.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/AwardsShareManagers.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/AwardsSharePlayers.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Batting.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/BattingPost.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/CollegePlaying.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Fielding.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/FieldingOF.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/FieldingOFsplit.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/FieldingPost.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/HallOfFame.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/HomeGames.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Managers.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/ManagersHalf.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Parks.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/People.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Pitching.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/PitchingPost.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Salaries.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Schools.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/SeriesPost.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/Teams.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/TeamsFranchises.csv',
 'https://raw.githubusercontent.com/chadwickbureau/baseballdatabank/master/core/TeamsHalf.csv']
```

Cleaning the Data

The data that we are dealing with is incredibly structured, and incredibly clean. Reading in to a pandas dataframe was a simple exercise, but we had a bit of munging to do in terms of replacing NaNs with 0's (mainly to not error out our models in later steps), and concatenating some of our string/text based fields for easier user input fuzzy matching in later steps. We also wanted to keep this as statistics heavy as possible, so we dropped many of the player metadata type columns that were present in each of our respective datasets. This allowed for us to use some ipython magic to display our tables in a succinct manner. We also explore with some basic feature engineering, but much of that code was removed from the final output. Much of our work involves using pandas dataframe operations, such as aggregation on groupby methods, dropping columns, merging dataframes/concatenating/removing duplicates, etc. Our first real method of analysis after data read in was value counts of null representations among our features. While we ultimately decided to bulk scrap variables to reduce dimensionality, that was, in part, brought along due to the fact that most of the metadata was null for the keys dating back to the early 20th century. We didn't want to inject an inherent bias in a ML algorithm by having a bunch of null data in half of the century.

Methods of Analysis

Querying statistics

To begin, this project started out as more of an exploratory information retrieval task. Mason and I are *huge* sports fans, so we wanted to see if we could create a program that ultimately outperforms some of the sites out there that aggregate and display statistics based on specified filters/parameters. This portion of the project analyzes data in an aggregate/grouped format. Our data goes down to the year grain, and as such, we have duplicate indices throughout our dataset - A task that pandas is great for. We created a rather complex function with slices on statistics, time, and count of samples returned for the specified parms. We also include some basic error handling that notifies a user on instantiation if they've entered invalid arguments. For instance, say we want the top five players, all time, ranked in descending order by total lifetime hits:

```
Displaying top 5 players sorted by H, Timeslice = Alltime
-----------------------------------------------------------------
```

playerID	FullName	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	BattingAverage
rosepe01	Pete Rose	3562	14053	2165	4256	746	135	160	1314.0	198.0	149.0	1566	1143.0	0.303
cobbty01	Ty Cobb	3035	11434	2246	4189	724	295	117	1937.0	892.0	178.0	1249	357.0	0.366
aaronha01	Hank Aaron	3298	12364	2174	3771	624	98	755	2297.0	240.0	73.0	1402	1383.0	0.305
musiast01	Stan Musial	3026	10972	1949	3630	725	177	475	1951.0	78.0	31.0	1599	696.0	0.331
speaktr01	Tris Speaker	2789	10195	1882	3514	792	222	117	1529.0	432.0	129.0	1381	220.0	0.345

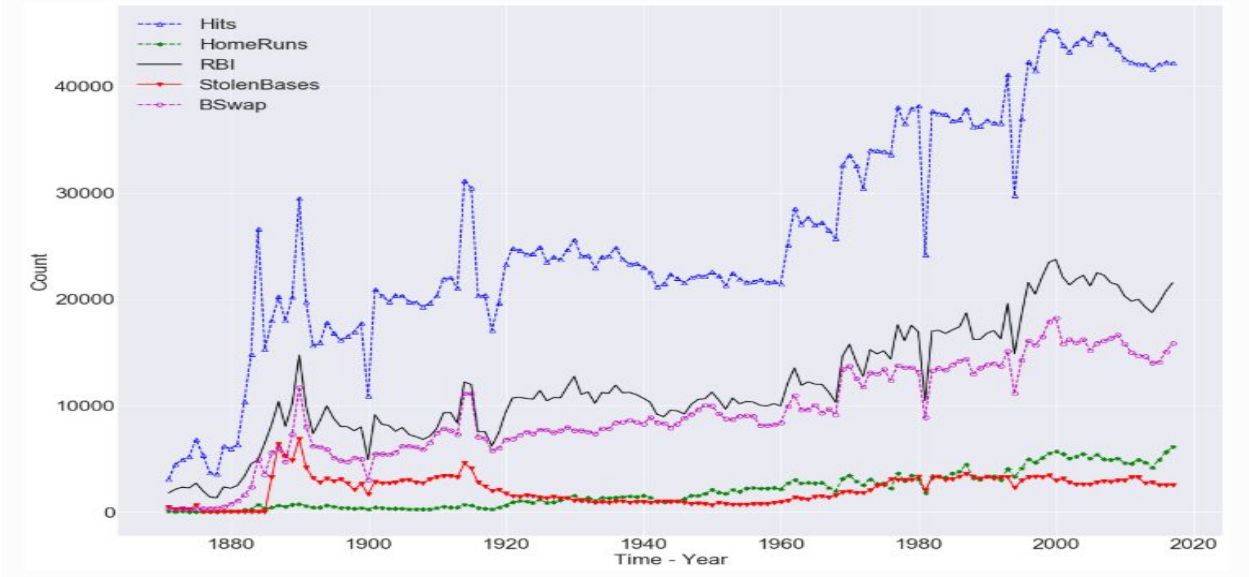
As you'll see in our script, it is possible to manipulate the function to slice this kind of table however you'd see fit. There was a case to be made for displaying all stats, rather than just the key statistic that is entered as an argument - Ultimately, it seems more logical to display all relevant stats as opposed to a single ranked stat, not depriving a viewer or information they might find relevant (ultimately leading to more tinkering, and a longer time on site, which is what websites strive for). We also have an parameter in our function that delineates between categories 'pitcher' and 'hitter'. The above example only outputs hitting stats, while a tweak in the argument yields pitching stats:

```
Displaying top 5 players sorted by W, Timeslice = Alltime
-----------------------------------------------------------------
```

playerID	FullName	W	L	G	GS	CG	SHO	SV	BAOpp	ERA
youngcy01	Cy Young	511	316	906	815	749	76	17	0.240000	2.713043
johnswa01	Walter Johnson	417	279	802	666	531	110	34	0.226667	2.348095
mathech01	Christy Mathewson	373	188	635	551	434	79	28	0.241176	2.648889
alexape01	Pete Alexander	373	208	696	599	437	90	32	0.253810	2.972381
galvipu01	Pud Galvin	365	310	705	688	646	57	2	NaN	2.941765

Visualizations

Because this project centers more around information retrieval and inference we did not place a huge emphasis on exploratory analysis in preparation for modelling, mostly due to time constraints. Nonetheless, because we are looking at, in essence, a way to map a function from a set of input variables to a class pertaining to hall of fame or not hall of fame, it would make the most sense to analyze the differences between the two.



*This graph shows how statistics have changed over time.

Switching back to descriptive stats, as opposed to visualization, we can display the average career stats grouped by hall of fame induction status. This should help to lend credence to the idea that this is a problem that can be modelled:

inducted	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB
0	285.251937	716.501950	92.931713	185.080684	31.375047	6.230510	13.870745	83.447181	14.350735	4.797863	64.727117
1	1673.930435	5854.856522	947.791304	1727.195652	295.178261	74.734783	155.691304	874.286957	158.017391	36.847826	643.804348

Modeling

As noted extensively above, one of our main goals for this project was to predict hall of fame propensity based on career statistics through 2017. In reality, this is a naive approach - Our model will not accurately hall of fame propensity for younger players because we don't have a scaling factor for those active players that would approximate career statistics at the completion of their career. Our model should, however, be fairly accurate for players that are nearing the end of their career or have recently retired. We also wanted to explore with GBC's feature ranking, which exemplifies the probability of a feature appearing in a root node within our forest. The process for preparing our data for a model follows this heuristic:

- Merge/Clean/Group datasets. Here, we tried to look at our batting and pitching records together. Essentially, we'd have a bunch of null batting columns on the observations/players that were pitchers, and vice versa for the players that were hitters - In a way, it's something resembling a hierarchical model.
- Upsample our minority class (class == not hall of fame) so that we can artificially create class balance. This works by randomly sampling from our minority class with replacement until the length of our minority class is equal to the length of our majority class.
- Train test splits. Set test size equal to 0.3 and use stratifed sampling to enforce equal distributions of classes that we created in the previous step.
- Create a dictionary of parameters and grid search through those parameters using an instantiated gradientboostingclassifer.

```
Frequency in Training Set
-------------------------
[[  0 115]
 [  1 115]]
Frequency in Training Set
-------------------------
[[  0 115]
 [  1 115]]
Fitting 5 folds for each of 96 candidates, totalling 480 fits

[Parallel(n_jobs=4)]: Done  52 tasks      | elapsed:    2.8s
[Parallel(n_jobs=4)]: Done 388 tasks      | elapsed:    9.1s
[Parallel(n_jobs=4)]: Done 480 out of 480 | elapsed:    9.8s finished
```

```
---------------------------------------------
Below is the grid searched best estimator:
---------------------------------------------
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.01, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

- Fit our model to our training set and capture our score.

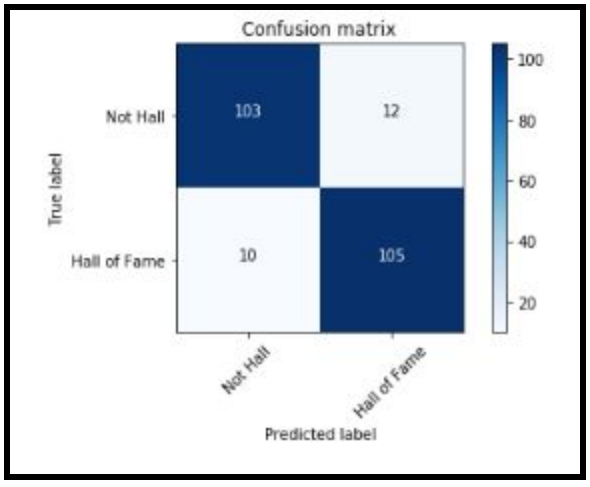
```
-------------------------
Model Score on test set: 0.9043
-------------------------
```

- Display classification report and confusion matrix for error analysis.

```
Classification Report
-------------------------
             precision    recall  f1-score   support

          0       0.91      0.90      0.90       115
          1       0.90      0.91      0.91       115

avg / total       0.90      0.90      0.90       230
```



- Wrap our model up with pickle, and create a function that takes user input and fuzzy matches to find a string match on our player feature. Score that instance by importing our model in real time and output some career statistics along with a probability that a player will make the hall of fame.

```
players = ['Barry Bonds', 'Cy Young', 'Mike Trout']

for i in players:
    hall_of_fame_projection(userinput=False, player_search=i)
```

executed in 8.15s, finished 13:41:25 2018-09-07

```
------------------------------------------------------------------------------------------------------------------------
This program takes a player name as an input, and outputs the players probability of being elected to the baseball hall of fame
------------------------------------------------------------------------------------------------------------------------
Predicting HOF propensity for Barry Bonds
-----------------------------------

 Barry Bonds has a 80.62% chance of making the Baseball Hall of Fame,based on stats through 2017
        Returning career Statistics through 2017
```

	playerID	FullName	2B	3B	AB	BAOpp	BB	CG	ERA	G	...	HR	R	RBI	SB	SHO	SO	SV	W	L	inducted
1552	bondsba01	Barry Bonds	601.0	77.0	9847.0	0.0	2558	0.0	0.0	2986	...	762	2227	1996.0	514.0	0.0	1539.0	0.0	0.0	0.0	0

1 rows × 22 columns

```
------------------------------------------------------------------------------------------------------------------------
This program takes a player name as an input, and outputs the players probability of being elected to the baseball hall of fame
------------------------------------------------------------------------------------------------------------------------
Predicting HOF propensity for Cy Young
-----------------------------------

 Cy Young has a 81.53% chance of making the Baseball Hall of Fame,based on stats through 2017
        Returning career Statistics through 2017
```

	playerID	FullName	2B	3B	AB	BAOpp	BB	CG	ERA	G	...	HR	R	RBI	SB	SHO	SO	SV	W	L	inducted
18810	youngcy01	Cy Young	87.0	35.0	2960.0	0.24	1298	749.0	2.713043	1824	...	156	3492	290.0	29.0	76.0	3007.0	17.0	511.0	316.0	1

1 rows × 22 columns

```
------------------------------------------------------------------------------------------------------------------------
This program takes a player name as an input, and outputs the players probability of being elected to the baseball hall of fame
------------------------------------------------------------------------------------------------------------------------
Predicting HOF propensity for Mike Trout
-----------------------------------

 Mike Trout has a 46.56% chance of making the Baseball Hall of Fame,based on stats through 2017
        Returning career Statistics through 2017
```

	playerID	FullName	2B	3B	AB	BAOpp	BB	CG	ERA	G	...	HR	R	RBI	SB	SHO	SO	SV	W	L	inducted
17298	troutmi01	Mike Trout	200.0	40.0	3399.0	0.0	571	0.0	0.0	925	...	201	692	569.0	165.0	0.0	874.0	0.0	0.0	0.0	0

1 rows × 22 columns

Conclusion/Conclusions about our Data

This project consisted of extensive work in the domain of functional programming, as opposed to statistical analysis. While we touch on some descriptive stats and visualizations throughout our script, our main goal was to create a source where we could easily fetch baseball statistics, and if we wanted we could score a player and see what chances they have of making the hall of fame. We ran into a number of issues working with data from so many different sources - Mainly duplicate entries per ID in one dataset that would mess with the calculations of a pandas group

by function. We used groupbys extensively because we wanted to be able to represent our data at a higher grain than the player level at most points in our script. When we reached the modelling point of our assignment, we had to figure out how to work with a problem that had severe class imbalance, ultimately deciding on downsampling our majority class to enforce uniform distributions. We also take a look at the GBclassifier Feature Ranking, showing us something similar to coefficients in a linear/logit model:

```
features = list(np.round(clf.feature_importances_, 5))
column_names = ['2B', '3B', 'AB', 'BAOpp', 'BB', 'CG', 'ERA', 'G', 'GS', 'H', 'HR', 'R',
       'RBI', 'SB', 'SHO', 'SO', 'SV', 'W', 'L']

dictionary = dict(zip(column_names, features))

print('\tReturning Feature Ranking')
for w in sorted(dictionary, key=dictionary.get, reverse=True):
    print (w, dictionary[w])
```

executed in 9ms, finished 15:28:18 2018-09-07

```
        Returning Feature Ranking
H 0.62934
R 0.28017
G 0.03247
RBI 0.01193
HR 0.01179
AB 0.01093
SV 0.01089
BB 0.00344
3B 0.00308
2B 0.00181
SO 0.00152
SB 0.00114
ERA 0.00073
BAOpp 0.00042
GS 0.00036
CG 0.0
SHO 0.0
W 0.0
L 0.0
```

This is telling us which feature were most impactful in the model - So we can say with moderate confidence that Hits and Runs scored drove a player's hall of fame propensity more than any other statistic.

Continued Work

This could be improved by downsampling our dataset into bins of relevant years, probably excluding anything before the Live Ball era. We also could have integrated some more metadata to see if any of those variables would have an impact on model performance. An interesting task in the near future would be simulating win propensity based on active player stats, probably using monte carlo markov chains (We touched on this a bit in IST718)

Libraries Utilized

Numpy, Pandas, Sklearn, Urllib, BeautifulSoup, OS, Sys, Scipy, itertools, matplotlib, Collections, Seaborn, jupyternbextensions

*Most scripting/prototyping done in Jupyter Notebooks
** Jupyter notebook converted to .py and .pdf upon completion.

References

https://github.com/chadwickbureau/baseballdatabank
Stackoverflow.com
Sklearn Documentation
Pandas Documentation
Syracuse Asynchronous Material - Scripting for Data Analysis