– Independent Work Report Fall 2019 –

Predicting All-NBA Selections from the 2019 NBA Draft

Khatanbuuvei Bold
Adviser: Prof. Ben Raphael

Abstract

This paper describes attempts to build a predictive model that estimates the probability that a new National Basketball Association (NBA) player to be named to an All-NBA team throughout his career, based on his college basketball statistics. In order to tackle this problem, three machine learning models were built based on historical NBA data collected from 1990 to 2015. Though the performance of each model was far from perfect, the models outperformed random guessing by a significant margin and provided interesting insights as to which features of a player impact the probability in question the most.

1. Introduction

The National Basketball Association (NBA) is a multi-billion-dollar entertainment business, and it's in the best financial interest of owners and shareholders of each team to see their team succeed. One of the ways in which teams can achieve this goal is to bet on the potential of young players, who they select at the annual NBA Draft: every summer, the 30 NBA teams draft 60 of the most promising young basketball players, most of them coming from college, with hopes of them becoming star basketball players – these players comprise that year's "draft class". Similarly, it is very interesting for casual sports fans to speculate on the future success of young players, because they want the teams they support to succeed. For instance, the first overall draft pick of the 2019 NBA Draft, Zion Williamson. Williamson has been touted as the next great basketball player, and even his high school basketball highlights have garnered millions of views on YouTube. However, for obvious reasons, the analytical tools teams use for this purpose are kept secret from the general public.

Therefore, a simple tool that is aimed towards casual fans is of great interest to the public. An example would be a tool that predicts whether a given college player will make an All-NBA team,

which is an honor that is awarded to the best players at each position after every season. Such a tool would be well-suited to be used and understood by an average fan, because it uses a familiar award as the benchmark for a player's star potential. Furthermore, since the tool would be used primarily for players who have yet to play a professional game, it should be essential that it only requires that the player has played college basketball.

The main goal of this project was to attempt to use techniques in machine learning to fill this hole. Namely, I set out to build a model that predicts the probability that a player is named to an All-NBA Team at least once throughout his career, based on his college basketball statistics.

2. Background and Related Work

The NBA has become more and more reliant on analytics in recent years[1], as new technologies and resources become available to teams. Many say that as a results of this movement, the game has entirely changed. Of course, teams keep their analytics tools, including their scouting tools, a secret from the public. Aside from hobby projects, there aren't many tools that are available to the public and serve a similar purpose to the goal of this project. The most prominent example is FiveThirtyEight's "RAPTOR Player Projections" [2], that "identifies similar players throughout NBA history and uses them to develop a probabilistic forecast of what a current NBA player's future might look like".

This system defines the potential of players by a metric called the "5 Year Market Value" that is measured in dollars. Additionally, it places players into a certain category, such as "Future All-Star" or "Good prospect". This might be an effort to make it more accessible to the casual fan, but I still found it rather confusing, especially because of the market value metric. In this project, I hope to do two things differently:

[1] See Why Nerds rule the N.B.A.

- Define the potential of players as a probability to make an All-NBA Team, which is both familiar to casual fans and also precisely defined.
- Use college statistics for making the prediction, because that's the only information about the player's abilities that's available immediately after the player is drafted.

3. Approach

I approached this problem as a machine learning classification problem. In classification problems, an algorithm known as the classifier is built by training on some *dataset* where, for each observation, a set of input variables known as the *features*, and the class, sometimes known as the *target* variable, are known. Then, the classifier can be used to predict the class of a new observation where the class is unknown, based on the features of that observation. The canonical example of a classifier is a machine learning algorithm that predicts whether an e-mail is spam or non-spam. In this case, the features might be the word content of the e-mail, and the e-mail address of the sender, and the set of possible classes would be either "spam" or "not spam".

Clearly, how one chooses the dataset, features, and the target variable is one of the most important design choices when building a classifier. For this problem, I decided to train the classifier on the dataset of college basketball players drafted from the years 1990 to 2015, using a set of college statistics and physical measurements as the features, and whether the player made an All-NBA Team throughout his career (encoded as 1 or 0) as the target. The reasons for these decisions are detailed below.

3.1. The Dataset

The primary tradeoff when selecting the right dataset was the count of observations, and the availability and relevance of the features. For example, if older data, such as the draft class of 1960, most statistics are not available simply because they were not recorded back then. Furthermore, the way basketball is played has changed dramatically since then – the most obvious example being the fact that the 3-point line was not introduced to college basketball until the 80's.

However, choosing the dataset solely based on the availability of statistics would decimate the

size of the dataset, because advanced stats such as Win Shares are only readily available starting from 1998[2].

Therefore, I decided to set the cutoff year at 1990, because for every player drafted since, we can reliably find two advanced statistical measures for each player: Free Throw Rate (FTr) and True Shooting % (TS%) (add to appendix). This is because the 3-point line (which the TS% depends on) was universally introduced to college basketball in 1986[3], ensuring that each player has played with the 3-point line for their whole college career, even if they played for 4-years. One final thing to note about the dataset is that it doesn't include each and every draft pick since 1990. Players picked after the 20th draft pick were not considered, because often times, players picked that late into the draft don't play meaningful minutes in the NBA, and their careers are rather short. Furthermore, some players were drafted from foreign basketball leagues, or even straight out of high school, which means they don't have college basketball statistics. These players would be withheld from the dataset.

3.2. The Target Variable

The next consideration was how I choose the target variable. Since the project was essentially meant to be as simple to understand as possible for the average NBA fan, I thought choosing an indicator that is well known and understood would be fitting. The few initial ideas were to use MVP (Most Valuable Player) Awards, or All-Star Selections as the main criteria, but soon recognized some faults with these ideas.

The main problem with using the MVP award as the criteria was that it was far too restrictive, because it's awarded only once per year, to the single best player that season. Since the inception of the award, just three players, namely Kareem Abdul-Jabbar, Michael Jordan and Lebron James have won nearly a quarter of all MVP awards [citation needed]. As a result, using this award as the benchmark for success limits the sample size by too much.

As for All-Star selections, while there are more All-Star players than MVP's, the issue is that

[2] https://www.sports-reference.com/cbb/about/ws.html
[3] https://www.active.com/basketball/articles/the-history-of-the-3-pointer?page=1

often, All-Star selections are based on popularity because it's mainly based on fan votes. As a result, a player that plays for a more populous city might get relatively many votes. Thus, All-Star selections aren't the most objective measures of success.

All-NBA selections, on the other hand, avoid both of these of these problems. Every year, three teams of five players, or 15 players, are named to an All-NBA Team as opposed to just one MVP, which means the sample size is much greater and more diverse. Furthermore, since All-NBA Teams are specifically selected by a panel of experts [citation needed], it is a more objective measure of a player's individual success than an All-Star nomination. Therefore, whether the player made at least one All-NBA Team throughout his career (encoded as 1 or 0) was used as the target variable.

3.3. The Features

The input variables the model considers when making a classification, or the features, was perhaps the easiest decision to make, primarily because the possible features were heavily pre-determined by the dataset. As mentioned above, the cutoff year when determining which draft classes to include in the dataset is 1990. This limits our options for the features to the basic box score statistics of the player, such as points per game. The full list of includes features in this category is:

- **Points Per Game (PPG)**
- **Rebounds Per Game (RPG)**
- **Assists Per Game (APG)**
- **Steals Per Game (SPG)**
- **Blocks Per Game (BPG)**

On top of these basic statistics, we have a few more "advanced" metrics, such as true shooting percentage. Namely, we can find the following "advanced" statistics for each player (see [1] for the methods by which these metrics are calculated):

- **True Shooting Percentage (TS%)** – a measure of shooting efficiency that takes into account 2 point field goals, 3 point field goals, and free throws.

- **Effective Field Goal Percentage (eFG%)** – a field goal percentage that adjusts for the fact that 3 point field goals are worth more points than 2 point field goal.
- **3 Point Attempt Rate (3PAr)** – Percentage of FG attempts from 3 point range.
- **Free Throw Rate (FTr)** – Number of FT attempts per FG attempt.

From these, I decided not to use eFG% and 3PAr, reasons being that eFG% is heavily corelated with TS%, and 3PAr is heavily dependent on the position of the player, and doesn't tell us much information about how well a player can actually score.

Finally, we have some information about the player himself, rather than his in-game statistics. This includes the following:

- **Pick #** - the position the player was picked in in the draft. Lower is better.
- **Strength of Schedule (SOS)** – a measure of how strong the player's opponents were. Higher SOS means the player played against better teams.
- **Height**
- **Weight**
- **Age** (at draft)

The full list of 12 features is summarized in Table 1.

Feature	Description
Pick #	Draft position of the player
RPG	Rebounds per game
APG	Assists per game
SPG	Steals per game
BPG	Blocks per game
PPG	Points per game
SOS	Strength of Schedule
TS%	True Shooting percentage
FTr	Free Throw Rate
Weight	Weight of the player in pounds
Height	Height of the player in inches
Age	Age of the player in years at the time of the draft

Table 1: Features of each player.

3.4. Conclusion

With these design choices, the task at hand is a bit clearer: the goal of the project is to build a model(s) that takes as input 12 numerical values that describe the player and his college basketball performance. Then, the model(s) would output either 1 or 0, which means that it predicts that the player will or won't make an All-NBA Team throughout his career, respectively. In other words, our model would be a *binary* classifier. Furthermore, it would be ideal if the model can also output a probability that the player makes the team.

4. Implementation

The implementation of the project can be roughly broken down into two stages: preparing the data and building the models. The project was implemented mostly in Python, because it is a language that I'm personally comfortable using, and also because it has an extensive array of machine learning and data science packages that are ready to be used out of the box.

4.1. Data Preparation

There were fortunately no major issues while preparing the dataset, because the data necessary to collect was exclusively numerical, and the websites I collected the data from, Basketball Reference and Sports Reference, had a relatively complete and reliable record of NBA Draft data and NCAA basketball statistics, respectively. Furthermore, this stage was made simpler by the fact that the complete dataset wouldn't be too large – it would consist of the statistics for the top 20 picks of the past 25 draft classes, so we would need at most 500 observations. As a result, I didn't run into any serious performance issues, even though very basic web scraping methods were used.

Since dealing with data can get very hairy, the scraping process modularized as much as possible in an effort to reliably automate the process. Namely, the whole process was done with the help of several small Python scripts [A] that interact with each other. The BeautifulSoup library was used in all scripts in order to interact with the markup of the web pages containing the data. The Python scripts and their purposes are broken down as follows:

1. **`all_nba.py`** creates a set of every player named to an All-NBA Team since 1990, and writes the names to a file.
2. **`player.py`** defines a Player class that provides methods for returning the player's statistics from Sports Reference.
3. **`draft.py`** defines a Draft class that uses Player objects to collect stats for every player in that draft class.
4. **`preparedata.py`** creates Draft objects for every year from 1990 to 2015, and using the file created in step 1, creates a `.csv` file consisting of the statistics of every player in the dataset, and whether they made an All-NBA Team or not, encoded as 1 or 0.

The only small issues I ran into was when there were multiple players with the same name, or when the name included accented characters (e.g. Nikola Vučević), in which case I had to find the statistics manually. However, there were only a handful of cases like this, so it wasn't a real issue. The resulting dataset consisted of 445 rows.

	Player	Pick #	RPG	APG	SPG	BPG	PPG	SOS	TS%	FTr	Height	Weight	Age	All-NBA
0	Derrick Coleman	1	12.1	2.9	1.5	2.0	17.9	8.85	0.620	0.747	82	230	23	1
1	Gary Payton	2	4.7	8.1	3.4	0.5	25.7	6.91	0.572	0.299	76	180	22	1
2	Mahmoud Abdul-Rauf	3	2.5	3.2	1.6	0.0	27.8	7.61	0.584	0.317	73	162	21	0
3	Dennis Scott	4	6.6	2.0	1.8	0.9	27.7	10.33	0.593	0.281	80	229	22	0
4	Kendall Gill	5	4.9	3.3	2.2	0.6	20.0	9.89	0.575	0.415	77	195	22	0

Figure 1: The first 5 rows of the dataset.

4.2. Building the Models

(All of the steps in this section are reproducible because the random states are fixed. See Appendix B for the Jupyter Notebook) There are numerous machine learning libraries for Python, but I chose to use `scikit-learn`, because it offers a wide variety of algorithms for many different purposes (including classification), while being relatively simple to use and understand straight out of the box. For example, a very simple way to use a classification algorithm with `scikit-learn` is to load the dataset into a `NumPy`[4] array, split the data into a "train" set and a "test" set for cross validation,

[4] `NumPy` is a package for scientific computing in Python

and use the `fit()` method the classifier interface provides on the training data. Then, we can use the `predict()` method on the test set and get an array with predicted class labels. We don't tune the algorithm in any way before fitting it to the test data, so this doesn't offer the best results. Nevertheless, it's a good place to start to get familiar with `scikit-learn`.

Indeed, the first attempt to build a model in this project was a simple logistic regression model without any tuning whatsoever, using the `LogisticRegression` class that `scikit-learn` provides. Logistic regression is used to model a binary dependent variable using a function that is determined by fitting a logistic function to a given dataset. In other words, after fitting a logistic regression model, we get classifier function that can classify observations into one of exactly two categories. After performing the necessary steps on our dataset of 445 players, the model returned a misleading accuracy score of around 0.87 on the test set, indicating that the classifier correctly predicted the classes 87% of the players. Upon closer inspection, however, it became clear that the model was simply predicting ”0” for every player, and because 87% of the players never made it to an All-NBA Team, the model achieved a high accuracy score. I believe that the untuned logistic regression model always predicted "0", simply because college statistics are only loosely predictive of how successful the player will be in the NBA, and so the safer bet is to always say "0".

In any case, it was clear that it was necessary to tweak the model before training in order to achieve better results. But before that, I needed a better way of evaluating the model than the simple accuracy score. In binary classifiers, we have a few other metrics we can use to get a better sense of the predictive power of the model. Namely, we can use the **recall**, **precision**, and **F1** scores. To understand these metrics, it's useful to first define *false positives* (FP), *false negatives* (FN), *true positives* (TP) and *true negatives* (TN) - as the names imply, FP refers to the number of datapoints the model wrongly predicted as "1", and TP refers to the number of datapoints the model correctly predicted as "1" (FN, TN defined similarly).

Then, recall, precision and F1 are defined as:

$$\text{recall} = \frac{TP}{TP+FN}$$

$$\text{precision} = \frac{TP}{TP+FP}$$

$$\text{F1} = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

For our purposes, *recall* is the fraction of actual star players that the model correctly classifies as star players, while *precision* is the fraction of actual star players within the players that the model thinks are star players. In other words, precision is how reliable a "1" prediction from our model is. *F1* is simply the harmonic mean of precision and recall, so it is a unified measure of the predictive power of a binary classifier. I was now ready to begin optimizing the model, with the goal of increasing the F1 score.

4.2.1. Optimizing the model - There are several things we can do to start optimizing a machine learning model, one of which is *hyperparameter tuning*. Hyperparameters are simply parameters that impact how the model is fit to the data[5], which we can tune before fitting to hopefully achieve better results. A simple way to perform hyperparamter tuning is a "grid search", in which we define the different values every hyperparameter can take, and try every single combination to see which one yields the best results. `scikit-learn` provides the `GridSearchCV` class, which performs a grid search along with cross validation. The "grid" I used to tune the logistic regression model consisted of 6 values for the *C* hyperparameter, and 2 values for the *penalty* hyperparameter, so a grid search would try out 12 different models, and return the best performing model.

On top of hyperparameter tuning (which was the biggest optimization step), there were two major changes I made to the model in an effort to optimize it: standardization and feature selection. Standardization is the process of scaling the features into a normal distribution, and it's useful in this case because different features are on wildly different scales. For instance, while PPG might

[5] An example of a hyperparameter for logistic regression is *C*, the inverse regularization parameter

realistically be anywhere between 0 and 30, FTr is always between 0 and 1. By scaling every feature into a normal distribution, we ensure that the dataset is much more consistent.

Feature selection[6] refers to the process of removing irrelevant features from the model, and only considering the remaining subset of features. This can be useful for a number of reasons, such as reducing complexity of model, making the models easier to reason about. In the case of this project, the dataset was initially 12-dimensional, which I deemed as unnecessarily complex for a classifier that is meant to be relatively simple. Thus, I decided to use the `RFE`[7] (recursive feature elimination) class that `scikit-learn` provides, which was relatively easy to understand, and compatible with `LogisticRegression`. In RFE, features are removed one-by-one recursively based on how much they impact the model, until a target number of features have been reached. Furthermore, we can incorporate RFE into our grid search, by defining the possible numbers of features we want.

The complete pipeline to build the model is illustrated in Figure 2:

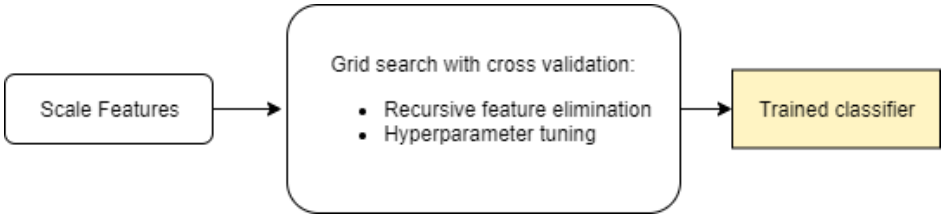

Figure 2: The pipeline to build the model.

After taking these optimization steps, results were much more promising. The accuracy was still at 0.87, but the logistic regression model achieved recall and precision scores of **0.27** and **0.57** respectively, combining for an F1 of **0.36**.

Although this is obviously much better than predicting all 0's, I still didn't have a handle on how good these scores are, so I made a "dummy" classifier that makes predictions randomly based on the distribution of the target variable. In this case, the dummy classifier would predict "0" around 87% of the time, and "1" around 13% of the time. By comparing the scores to those achieved by the dummy model, we can at least know how much better the model predicts than random guessing.

[6] See https://machinelearningmastery.com/an-introduction-to-feature-selection
[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

Finally, I decided to train two more models to the project: Random Forest Classifier (RFC) and Support Vector Classifier (SVC). I will not go into detail as to how I did this, because the steps and the pipeline is identical to that of logistic regression. The scores of the dummy model, an each classifier (after optimization) are summarized in Table 2.

	Accuracy	**Recall**	**Precision**	**F1**
Dummy	0.804	0.133	0.182	0.154
LOG	0.875	0.267	0.571	0.364
RFC	0.866	0.133	0.500	0.211
SVC	0.866	0	undefined	undefined

Table 2: Features of each player.

As seen from the table, the logistic regression model performed considerably better than the dummy classifier, while the improvement for RFC isn't as significant - it does, however, have a much higher precision score, meaning it's more reliable than the dummy classifier. The precision and F1 scores of SVC is undefined, because even after optimization, SVC didn't predict any 1's, so it's actually performing even worse than the dummy.

4.2.2. Class Imbalance - The main problem that caused the default LOG and the optimized SVC models to fail was that the class distribution is imbalanced. Intuitively, there are much fewer star basketball players than average ones. Specifically, 60 of the 445 players in our dataset had made it to an All-NBA Team, meaning only around 13.5% of the players were in class 1. There are various ways to deal with this issue[8], but one of the simpler methods is *oversampling*. Oversampling refers to creating new points that belong to the minority class, which can help re-balance the dataset. A very simple way to do this would be to replicate some of the existing points in the minority class, but doing this might lead to overfitting issues.

A slightly better approach is ADASYN[3], or Adaptive Synthetic Oversampling, which creates "synthetic" points that aren't copies of existing points in the minority class. The ADASYN algorithm is quite complicated, and involves creating new points based on a linear combination of existing minority points within a neighborhood. Fortunately, `imbalanced-learn`, a library

[8]See Handling imbalanced datasets in machine learning

that provides imbalanced machine learning methods to complement `scikit-learn`, implements ADASYN. Thanks to this, we can throw ADASYN into our pipeline (see Figure 3), and see the results immediately. However, it's important to perform oversampling after splitting the dataset into training and testing sets for cross validation, because we want to be able to compare the new results to previous results. This was the final optimization to our model.

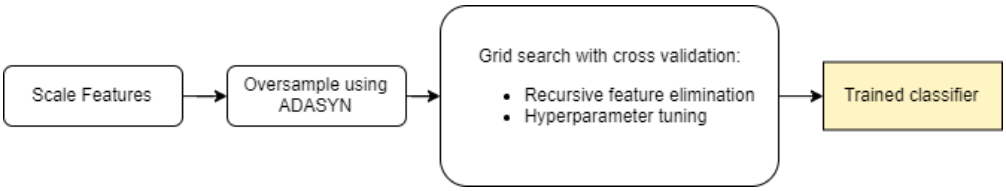

Figure 3: The pipeline to build the model.

5. Evaluation and Results

The scores achieved by using ADASYN are summarized in Table 3. The overall trend was that every model had an increase in recall and a smaller decrease in precision, leading to an overall increase in F1.

	Accuracy	**Recall**	**Precision**	**F1**
Dummy	0.545	0.667	0.179	0.282
LOG	0.598	1.000	0.250	0.400
RFC	0.848	0.400	0.429	0.414
SVC	0.830	0.600	0.409	0.486

Table 3: Features of each player.

The LOG model suffered a heavy decrease in accuracy, because it was now making "1" predictions much more liberally. This also explains the perfect 1.0 recall, but the relatively low precision score of 0.25. Overall, the F1 score had the smallest increase among the models. It's unclear if ADASYN helped or hindered the LOG model considering how the large accuracy decrease.

The RFC model enjoyed a much more significant increase (+0.203) in F1, and had higher recall, because it was making more "1" predictions (12 as opposed to 4 previously). Overall, the model performed much better this time around.

The SVC model was the best performing model, which is surprising considering it was the worst performing model before ADASYN. Although it had slightly lower accuracy than RFC, it had more predictive power considering it made 22 "1" predictions, 9 of which were right. Overall, the model was solid across the board with relatively high accuracy, recall, and precision, resulting in the highest F1 score. We can validate the SVC model by using certain draft years as validation. Figure 4 shows the output for the draft classes of 2010, 2011 and 2012, sorted by draft order (chosen arbitrarily).

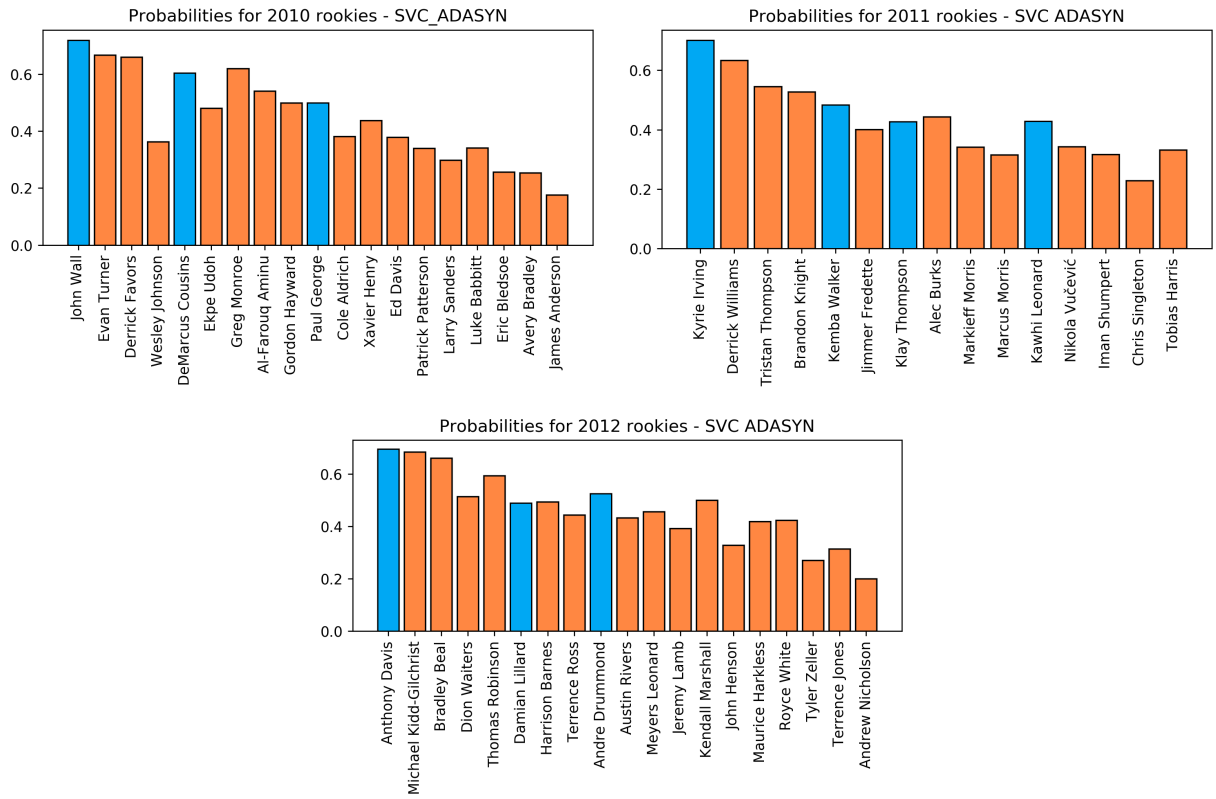

Figure 4: Probabilities predicted by SVC model with oversampling. Blue bar indicates player named to All-NBA Team.

For the cases of All-NBA players like Demarcus Cousins and Paul George in the 2010 draft, Kawhi Leonard in the 2011 draft, and Andre Drummond in the 2012 draft, the model predicted probabilities that are relatively high when compared to their draft neighbors. On the other hand, there are some false positives such as Greg Monroe and Kendall Marshall. All in all, although

the probabilities show an inverse relationship with draft pick #, and some of the relatively high probabilities seem to be reliable.

The main goal of the project was to predict the players who are most likely to make an All-NBA Team from the draft class of 2019. Figure 5 and Table 4 show how the models answered that question:

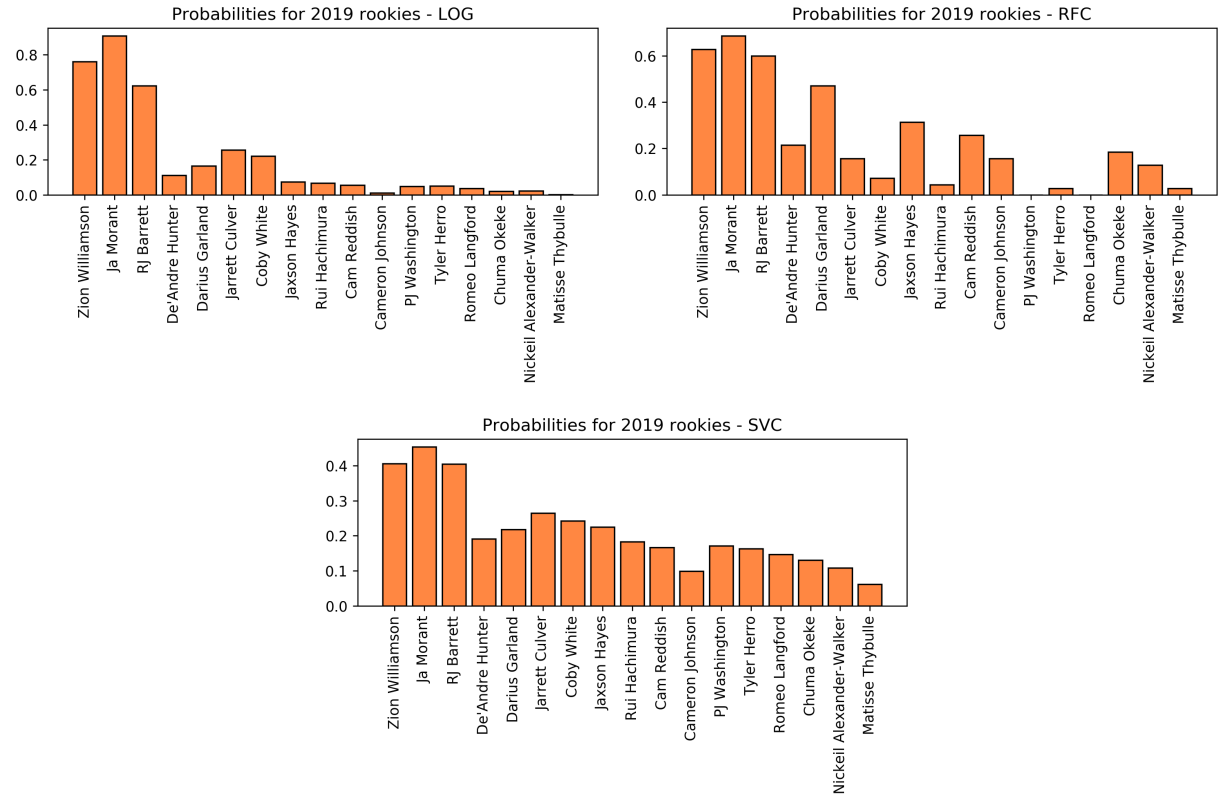

Figure 5: Probabilities that every model predicted for the rookies of the 2019 NBA Draft to make an All-NBA Team

In table 4, I decided to add a column that doesn't include the LOG model, because the LOG model with oversampling seems to undershoot the probabilities of lower draft picks, while overshooting the probabilities of higher draft picks. As a result, the variance of the probabilities is unrealistically low. The SVC and RFC models yield much more realistic probabilities.

Player (Pick)	LOG	SVC	RFC	Avg.	Avg. RFC and SVC
Zion Williamson (1)	0.59	0.74	0.59	**0.64**	**0.66**
Ja Morant (2)	0.60	0.78	0.81	**0.73**	**0.79**
RJ Barrett (3)	0.58	0.71	0.54	**0.61**	**0.63**
De'Andre Hunter (4)	0.52	0.49	0.19	0.39	0.34
Darius Garland (5)	0.54	0.59	0.37	**0.50**	0.48
Jarrett Culver (6)	0.54	0.59	0.43	**0.52**	**0.51**
Coby White (7)	0.54	0.58	0.23	0.45	0.41
Jaxson Hayes (8)	0.52	0.49	0.26	0.42	0.38
Rui Hachimura (9)	0.50	0.44	0.26	0.40	0.35
Cam Reddish (10)	0.50	0.43	0.30	0.41	0.37
Cameron Johnson (11)	0.46	0.28	0.01	0.25	0.15
PJ Washington (12)	0.49	0.40	0.06	0.31	0.23
Tyler Herro (13)	0.50	0.43	0.09	0.34	0.26
Romeo Langford (14)	0.49	0.38	0.17	0.35	0.28
Chuma Okeke (16)	0.47	0.31	0.16	0.31	0.24
Nickeil Alexander-Walker (17)	0.47	0.31	0.19	0.32	0.25
Matisse Thybulle (20)	0.42	0.18	0.07	0.23	0.13

Table 4: Predicted probabilities for rookies of 2019 NBA Draft

6. Conclusion

According to our models, the 2nd pick of last year's draft, Ja Morant, is most likely to make an All-NBA Team at some point in his career. This is very interesting given that he has been the frontrunner to win the Rookie of the Year award of this season so far[9]. Furthermore, the first three draft picks are much more likely to make an All-NBA Team when compared to the rest of the players. If we set the decision probability to .5, then the models think that the following players will make an All-NBA Team throughout their careers:

- **Zion Williamson**
- **Ja Morant**
- **RJ Barrett**
- **Jarrett Culver**

[9] https://www.nba.com/rookie-ladder

Finally, we can look at which features the models thought were most important, to better understand which statistics we should look at when attempting to evaluate a player's future based on his college statistics. To do this, we can chart the coefficients for each selected feature (after RFE) in the LOG and SVC models, and the feature importance of each selected feature for the RFC model. Each of these values are provided by the interfaces of the `scikit-learn` classes for the classifiers.

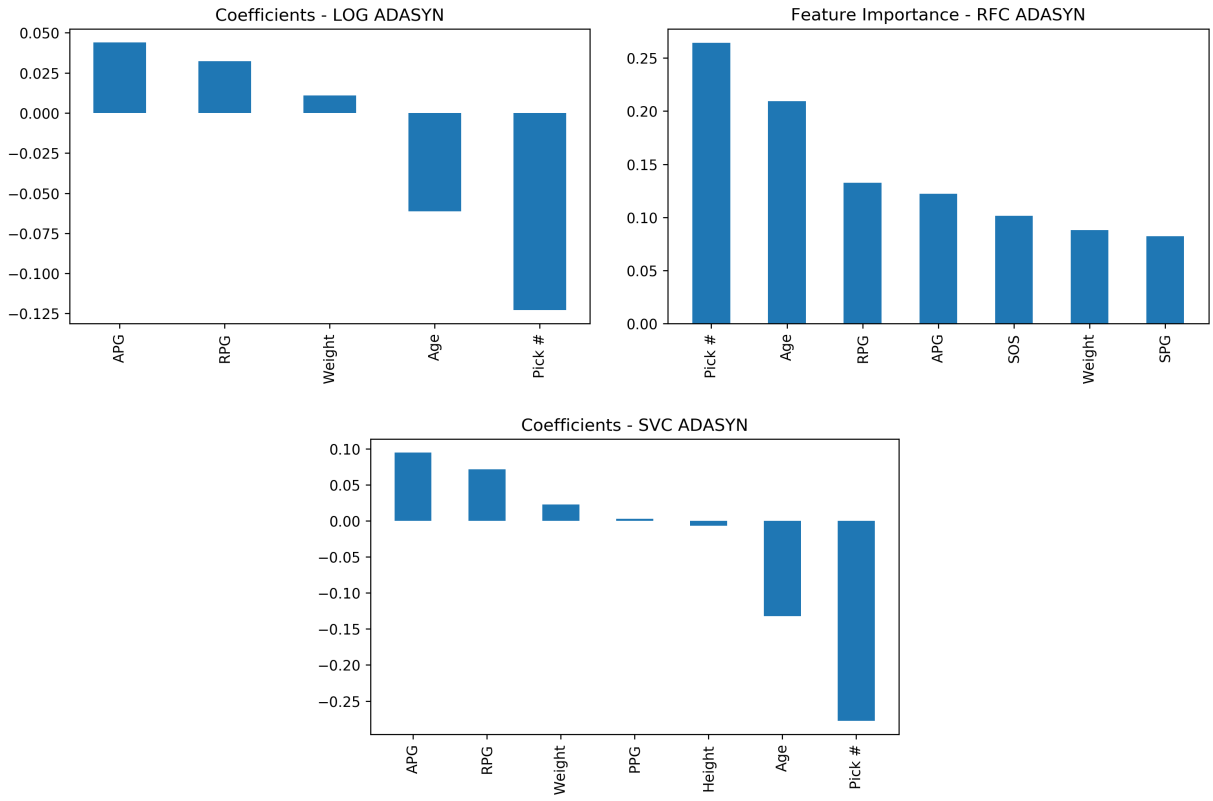

Figure 6: Coefficients and Feature Importances of each selected feature in different models

The models all agree on a few things: draft position (pick) is by far the most important feature in all models, followed by age, then APG and RPG. From these results, we can make draw a few conclusions:

- Draft position being the most important feature shows that teams generally have a good idea of who's going to be good in the future. We should trust the draft.
- Age is also a very important feature. This makes intuitive sense because more talented players tend to enter the league at a younger age.
- Assists per game and rebounds per game are more important than points per game, which wasn't even selected for all three models. Perhaps this implies scoring specialists have relatively low success in the NBA.

References

[1] "Glossary," https://www.sports-reference.com/cbb/about/glossary.html, accessed: 2019-01-05.
[2] "Nba player projections," https://projects.fivethirtyeight.com/2020-nba-player-projections, accessed: 2019-01-08.
[3] H. H. et al., "Adasyn: Adaptive synthetic sampling approach for imbalanced learning." IEEE, 2008.

A. Appendix: Scripts used for scraping

`all_nba.py`

`player.py`

`draft.py`

`preparedata.py`

B. Appendix: Jupyter Notebook

Link to Jupyter Notebook of this project