PREDICTING THE HEIGHT OF NBA PLAYERS BASED ON PERFORMANCE AND CLUSTERING THEM

Mihaly Hanics
Central European University
Quellenstrasse 51, 1100 Vienna, Austria
e-mail: hanics_mihaly@student.ceu.edu

ABSTRACT

The National Basketball Association hosts the most prestigious basketball league in the world: the NBA. Getting into the NBA is super hard even compared to other sports due to being almost totally limited to North America and the “incestuous” college draft system, and the players who get to join are already special, “chosen” players. Watching some stars shine in games, we can see that indeed players tend to be more singular and rely much more on their talent, than in soccer where you are expected to train and rely on a much broader skillset, having “more average” statistics. In this project, the aim was to find what is the correlation between a player’s physical attributes and performance, and then make predictions on one off the other. In particular, I used machine learning methods to predict the height of a player based on statistics of how they play, mostly from their “behavior” on the pitch. Does the habit of shooting from a far range typically mean that the player is small (as smaller players may tend to rely on their shooting skills, rather than powerfully scoring a point from close range)?
Implemented algorithms include a decision tree, random and ensemble forests, gradient descent methods, and a clustering algorithm. It turns out that for this data, an “averaging” ensemble forest is the most proper classifier.
For further research, predictions should be made on weight too, and vice versa: from physical attributes, predict some statistics of performance. All code and data can be found on GitHub: me9hanics/ml-nba-height-from-performance.

1 INTRODUCTION

Given both career and seasonal performance statistics of an NBA player, can we predict his height? It would be interesting to see how well we could predict a player’s physical attributes from performance, particularly playstyle. Some related work on this dataset has been done by Tal Boger, a Ph.D. student at John Hopkins University, founder of Dribble Analytics, he used machine learning algorithms (mostly LDA and Logistic Regression) on this dataset to predict the MVP each year, predict best defenders, etc., often when working on defenders, using height (and wingspan) as a predictor.
Like Tal, I also worked with the sub-official NBA API. It gathers information from the nba.com website, fetching data through various player and team stats endpoints. This gave me enough information to work with, and build a model that 49.5% of the times can predict the height of a player with at most 1-inch error. This ensemble model, and other models can be later used to show that there is a playstyle and physical attribute correlation in the NBA.

2 DATA

The data is totally fetched from the official NBA website [1]. The fetching process and code is available in the file *fetch_players.ipynb*, and the stored data is available in 4 files: *career.csv*, *career_filtered.csv*, *player_bios.cv*, *player_bios3.csv*. The tool used to fetch the data is the nba-api package, available on pip. Using this, I fetched two types of data: player career stats, using the *PlayerCareerStats* endpoint, and seasonal player data including biographical information through the LeagueDashPlayerBioStats endpoint. The fetching was done on all players who have played in the previous 20 seasons, fetching all seasonal data from 2003-04 till 2022-23. All information about gathering the data is available in the respective notebook.

2.1 Data description

The seasonal performance data from 2003-04 till 2022-23 is contained in career_filtered.csv, which after loading correctly in *ml.ipynb* as a Pandas dataframe has 9778 instances of player-season data, with 27 attributes including age, team, education, country, minutes and games played, and performance statistics, including points, shots scored and attempted, defensive stats (this set of data does not contain height and weight as an attribute yet). The most important variables from this dataset are the number of offensive and defensive rebounds, which is not available in the other dataset. Aside from IDs and names, all attributes are numerical.
The player bio dataset contains more biographical data of each player, but also seasonal. It contains height, weight, draft year, round and number, and more general statistics such as net rating, usage rate. For each season, the players playing in that season are included in the seasonal data, and I collected this data for all seasons between 2003-04 and

2022-23. The number of attributes is 24, but most attributes overlap with the other dataset.
After combining the two sources and filtering for most relevant attributes, combining multiple one season stats for each player (e.g., player has played for two teams in one season) excluding players who played less than the length of 3 matches in a season, I had 8322 instances with 14 attributes. This was used on most learning models, with the exception of weight and typically age. An instance:

PLAYER_ID	SEASON_ID	HEIGHT_INCH	WEIGHT
1630639	*2022-23*	*78.0*	179.0
AGE	MIN	OREB_PCT	DREB_PCT
22.0	*217*	*0.046*	*0.152*
FGA_PM	FG_PCT	FG3A_PM	FG3_PCT
0.406	*0.663*	*0.230*	*0.267*
FTM_PM	BLK_PM	PF_PM	TS_PCT
0.018	*0.0*	*0.101*	*0.589*

As you can see, because of multiple seasons data for most players, the data is actually longitudinal. The handling of this is described below, but this could be analyzed with change-over-time data statistical methods alongside machine learning models.
Important to mention that while height was a numerical variable, its values only included integers. This led to most models interpreting it as a categorical variable, which may be a major decrease in accuracy.

2.2 Data understanding

A description and explanation of variables:

- MIN: minutes
- FG, FG3, FT: Field goal (2 point goals), field goal 3 pointers, free throws ("penalty throws", 1 point) M: made (scored), A: attempted, PCT: percentage
- REB/OREB/DREB: Offensive/defensive rebounds
- AST: Assists, AST_PCT: Assist percentage
- STL: Steals
- BLK: Blocks
- TOV: Turnovers (negative statistic)
- PF: Personal fouls (negative statistic)
- PTS: Points scored

Bio:

- DRAFT: Each year before the season starts, 60 college newcomers can be drafted by teams. More "promising" players are drafted earlier
- GP, GS: Games played, games started
- NET_RATING: Offensive rating - defensive rating For a player: measure for how many goals a team scores with him vs. allow in. Can be negative.
- USG_PCT: Usage percentage, estimate of participation in team plays
- TS_PCT: True shooting percentage. Measure for how well a player shoots.

Player age is also a statistic, and it is an interesting one.
We are trying to predict height, and for almost all players, with age that doesn't change, therefore age should not be a factor. However, age is a major keypoint in how well one does (the stats show that players play the best at age 26-28), and we are predicting height based on performance, so somehow age does infact matter indirectly? Should we include it then? Or should we adjust performance based on age (e.g. if players play 10% better at 28 then at 22, then we should reduce their performance stats by 10 percent)? The way I tackled this is that I included age as an attribute for most models, and used less age-dependent attributes. I typically relied on "attempts percentage" attributes, as that describes less how well a player plays, but more "habits", which may come from a player's physical attributes (a small agile player might try to attempt more dribbles, how many of them are successful depends on the player's skill, but attempts don't).

A similar question arises: if we have longitudinal data, data of players from different seasons, and our task isn't typically to predict trends (like how a player's career develops) but to predict something constant throughout the time interval (career start and end), should we use approaches designed longitudinal data, should we combine all season data for a player into one averaged career-average data, or keep it as it is? I choose the last option. My argument against combining the data into one career-long average is that this way, we would have much less instances and more importantly much less diversity in stats.
When combining, we'd have only around 1200 instances to analyze. This would mean that we have to oversample our instances. Oversampling is a dangerous tool as it creates bias, and with these limited instances of many attributes, I would advise against it. The little "variance" of player's stats over multiple seasons gives in fact a more general outlook than averaging and resampling the same data about as many times as many seasons the player played.
As a tradeoff, our models "oversample" players who have played multiple seasons, the rate of oversampling linear to how many seasons they played between 2003 and 2023. But from my point of view, even if this'd mean less accuracy on players who "come and go" from the NBA, it makes the models be better predictors on players who have the ability to stay in the NBA for many seasons, who are "more interesting". This may even be more important than a higher percentage of accuracy when trying to find connections between physical attributes and performance.

Going back to the most variables, it is important to consider which ones are going to benefit us in building models, and why. Here is a collection of what we "need" and what we don't need:

- **Position**: A very much wanted information. A player's position on the court can play a significant role in statistics and can "suggest" the player's physical attributes. However, there was no data particularly on which position a player is playing (preferably). This is because it can change game to game, the NBA does not store players' preferred position and to obtain such info I'd have to fetch team lineup from all matches (and consider all substitutes) and look at where

each player plays most commonly. Due to the limitations of the API (most API calls are rejected simply because of too frequent requests, and high traffic on the NBA website) this is not feasible. This in fact is such a problem, that developers have made multiple ML models to predict the position of players (Tal [2] used simple KNN, this categorizes 11-12 types of positions, arguing against the typical 5). So I have to work without this attribute.
(Just to get a feel, Wikipedia describes some of the positions respectively as: “typically the team’s shortest player” (PG), “the big man, usually the tallest on the floor” (center), small and power forward are two major positions).
- **Age**: As I said, it doesn't correlate with height, but it correlates with performance on which we predict height. Especially for clustering. I think for "attempts" stats, age should not really matter however. I include it just to show that it will not matter much for classifying.
- **FG, FG3, FT**: I would drop FT_PCT, it's a "throwing skill". FTA is not relevant, FTM may matter a little (from 5 meters to score a goal into a basket above, additional 30cms of height can matter).
For field goals, my thought was "made" is more important than attempt as it suggests success, and made/attempt ratio would suggest even better how successful the player is. But one of the key ideas I wanted to know is whether "smaller" players tend to play different than other players: do they throw more from far away, as they struggle to penetrate from close, or it's exactly the opposite: they use their agility to score from close? For that, attempts per minute may be a better predictor. I decided on first using attempts alongside made/attempt ratio, we may get some results like short players are less successful in scoring 2-pointers.
- **REB:** I would either drop OREB, DREB, and keep REB(_PM) with position, or keep only OREB+DREB. Since we don't have position, I keep OREB and DREB as indicators on "position". (As we see later on, these turned out to be the most important attributes).
- **BLK:** I keep this attribute, because some short players even have 0 blocks after many matches.
- **TOV:** drop this information, not height related probably
- PF: maybe relevant, I'll keep it at first.
- **STL:** Not sure to keep it or not, but from data it spans really thin, probably not a good predictor so I drop this.
- **AST:** Drop all. I believe it is more of a skill/team stat. (Although might have some significance, like in soccer the small, fast, agile wingers cross the ball to the big, strong striker who can head it in, and assist stats are correlated with position, but there is too much “noise” from how well the receiver can finish, or his behaviours.)
- PTS: Not needed, we have other points stats
- DRAFT: I'd skip. Initially, I thought it can find extreme cases like Giannis being 6'11, but he was only 15th pick.
- NET_RATING: Skip. **USG:** Undecided, **TS:** I'd keep.
- Anything else: skip.

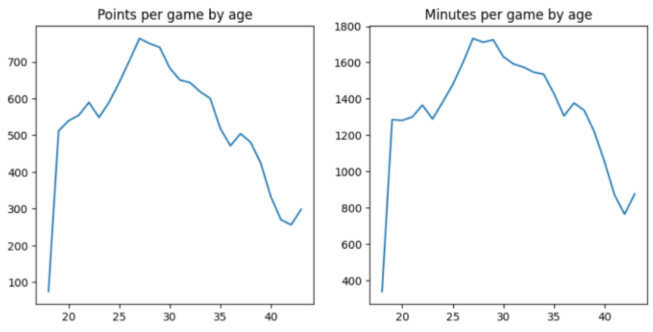
The target variable, PLAYER_HEIGHT_INCHES has values from 65 to 89 (all values have an instance, except 66,67, 68), and is an integer in all cases. As mentioned, modeling it as a categorical variable may have decreased accuracy.

2.3 Data preprocessing

The steps I took in preparing the data:
Career:
1) Select the relevant attributes
2) Combine multiple one-season player data into one seasonal data (the case when a player plays for 2+ teams in one season)
3) Filter out instances with less than 3 matches of gametime (144 minutes)
4) Normalize adequate attributes by minutes
Biographical:
1) Select the relevant attributes
2) After-analysis-preprocessing
3) Combine data with career data

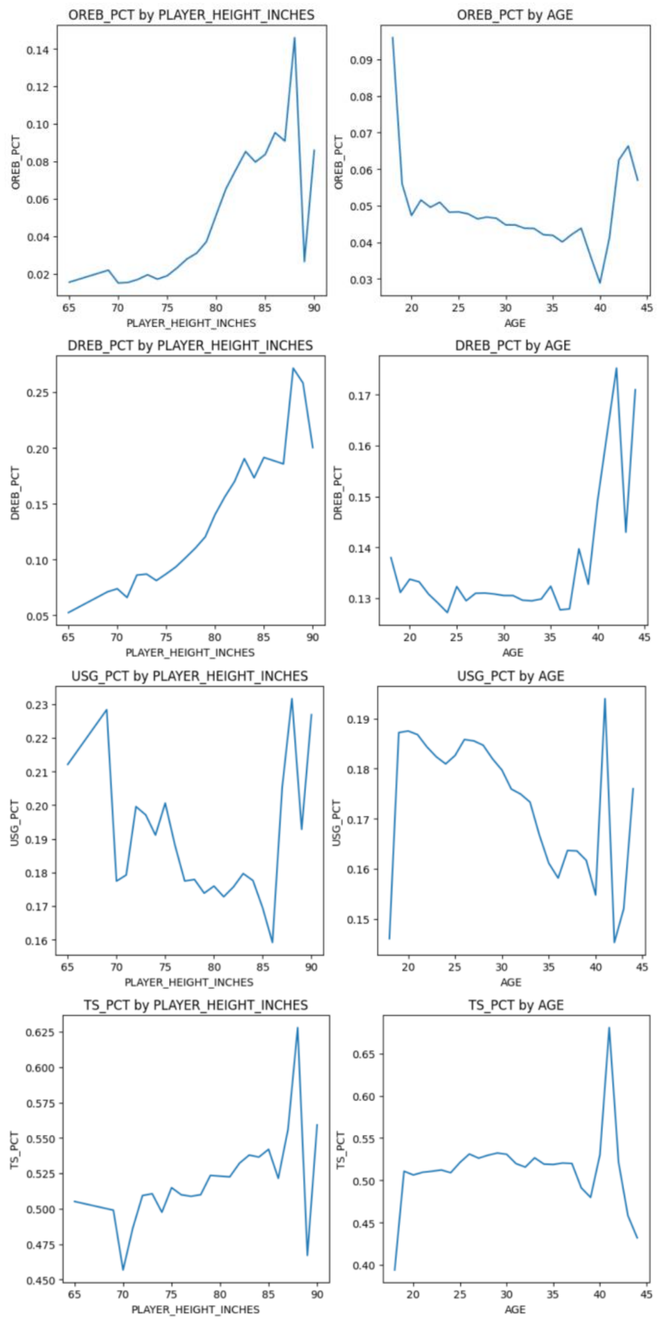
After-analysis-preprocessing:
I found while looking at attribute-by-age plots that “too” young and “too” old players are outliers, for which I filtered them. I also found that very heavy players are sub-outliers too, but I did not filter them, as I found it part of the task to predict their height too.

Normalization:
A visual proof that we have to normalize some stats by minutes:



I normalized most statistics by minutes, except the ones where it would not make sense.
What I found interesting is that plotting average performance by age does not change that much. However, plotting a specific player’s performance by age, we do see the trend that they improve in their 20s. How is that? From what I see, what actually matters is not how old a player is. It is about how many seasons of experience they have. Some players get drafted at 19, some get into the NBA later in their careers. Usually the “late joiners” bring down the averages.

After evaluating the possible correlations between height and various attributes (by plotting them with *matplotlib* and *Seaborn*), I have found that usage rate (USG_PCT) probably is not a good indicator, and I dropped it.



3. MACHINE LEARNING METHODS USED

The models can be split into three categories: a tree, random and ensemble forests, and gradient descent models, these are the approaches I tried. Aside from that, to help better understand the connections between physical attributes and performance, I ran a carefully designed clustering algorithm, which led to clustering players into 5 groups, 3 based on position, and 2 "skillgroups".
For training and testing the models, I split players into a training and testing category, so we don't get the same player in both test and training data (with different season data). For the tree and gradient descent, I ran grid search cross-validated hyperparameter tuning.

3.1 Brief description of the methods used

1) Decision tree
As a first step, I explored a problem with a hyperparameter-tuned decision tree to gather information on about how good accuracy we can get and what aspects matter.
I used entropy as criterion, as it gives back the most "information giving" attributes.
2) Random forests: This is based on the tree, also entropy based, to improve accuracy. Includes classifier, regressor.
3) Ensemble forests: Created a self-made ensembler based on certain parameter decision trees, one time I used "mode" classification as if the variable was categorical, and one time averaging the guesses of the decision trees.
4) Clustering: I carefully renormalized the parameters to try to make the bias as low as possible, but I doubled the height values to make its weight higher, and cluster similar height players more together.
5) Gradient descent methods: Tried just plain gradient descent, "badly parametrized" gradient descent, and well cross-validated gradient descent. Most importantly, I also tried gradient regression, and not just classification.

3.2 Brief description of the evaluation criteria

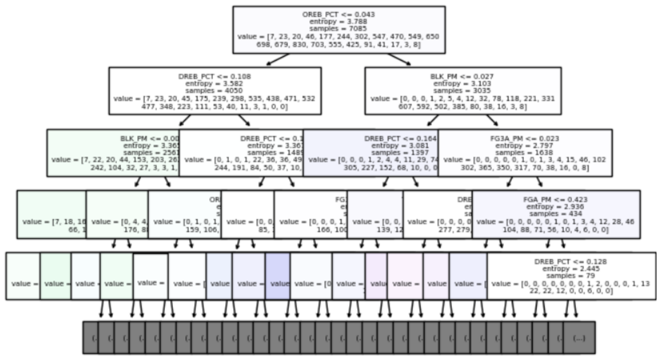
For evaluating the results, I measured the accuracy of the models on the test set, but not just plain accuracy, 1-,2-,3-inch accuracy and the largest size error. I also chose models on best cross-validation score. For a quality measure, I computed F1 score. With these, we can find the most accurate, most "close", smallest maximum error and simplest working models.
To visualize the results of each model, I used error histograms, as I found it simpler to interpret than confusion matrices.

4. EXPERIMENTS

The setup for finding the best methods was lengthy: firstly, fit a decision tree classifier for setting initial expectations on accuracy and have an outlook on what attributes matter the most, then try a random forest and ensemble forests to gather better accuracy. In the next step, I clustered players together, into 5-groups with high emphasis on height to find more connection between attribute values and certain heights. Lastly, I ran two very long (one failed) hyperparameter tunings to gradient descent models to find the possibly best performing models. Sadly due to the unexpectedly long over 110 hours of computing on hyperparameters, I did not have time to compute the best parameters for gradient regressors, only for classifiers.
I used the scikit-learn library for running models and evaluation, and further used the shap library for Shapley-scores.

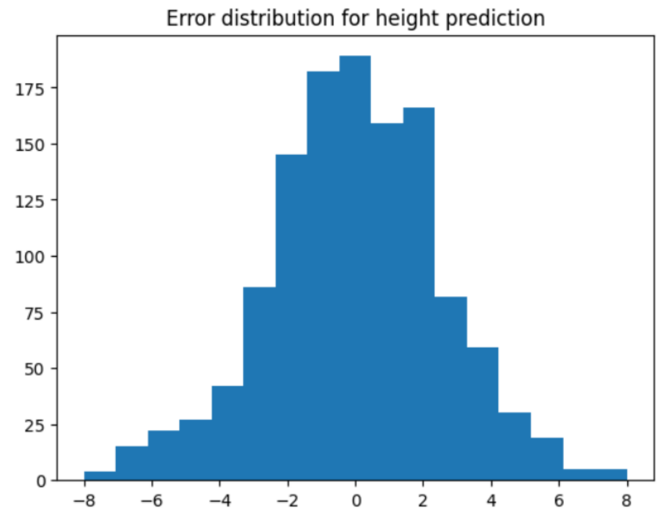
Decision tree

The parameter space given for hyperparameter tuning of the tree includes small and large values for max_depth, and small and large values for minimum samples split. I used Grid Search Cross Validation, the found best parameters: *max_depth = 9, min_samples_split = 30*. After fitting a tree with these parameters, we find that OREB and DREB are most key. A not so clear-out plot is what we get:


OREB_PCT <= 0.043
entropy = 3.788
samples = 7085
value = [7, 23, 20, 46, 177, 244, 302, 547, 470, 549, 650 698, 679, 830, 703, 555, 425, 91, 41, 17, 3, 8]
DREB_PCT <= 0.108
entropy = 3.582
samples = 4050
value = [7, 23, 20, 45, 175, 239, 298, 535, 438, 471, 532 477, 348, 223, 111, 53, 40, 11, 3, 1, 0, 0]
BLK_PM <= 0.027
entropy = 3.103
samples = 3035
value = [0, 0, 0, 1, 2, 5, 4, 12, 32, 78, 118, 221, 331 607, 592, 502, 385, 80, 38, 16, 3, 8]

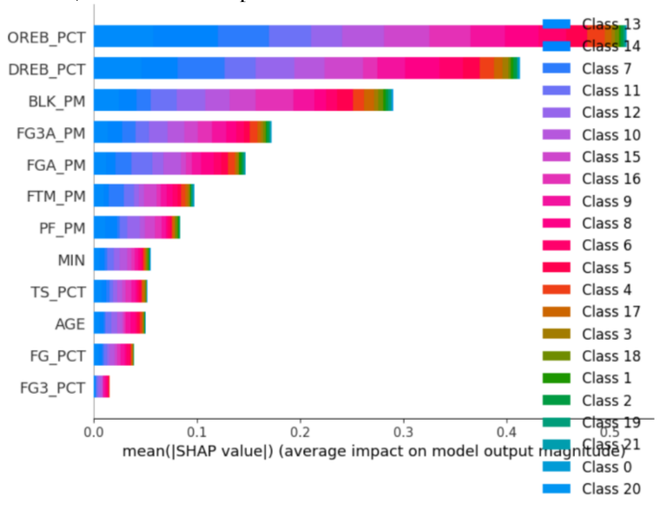
Maybe after zooming one can see the many splits based on rebound values. Blocks are also good indicators. Attempts seem to be less relevant; age is "nowhere".

The accuracy turned out to be 15.28%. I kind of expected better, but we guess correctly 1 in 6 times, so it's not that bad. Let's see how close we got to the answer on average:



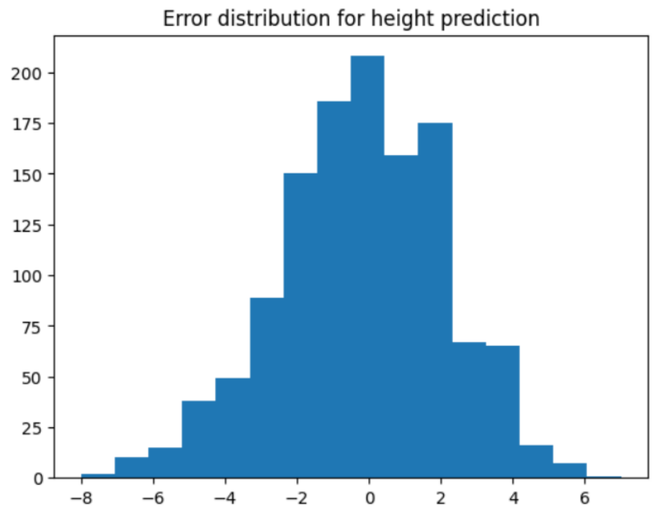
42.85% of the times we are in 1-inch accuracy, 67.99% of the times we are in 2 inches, 4 out of 5 times we are within 3 inches. And we even did not have an error above 8 inches! That is SUPER interesting. If we'd categorize the heights into 3-4 categories, we'd have very good results for predicting them. I did not expect this, I imagined there'd be rare but large errors, for a player very tall or very small, and for classifying we are supposed to be even more prone to make big misses occasionally. (The height span is 69 inches to 90.)

To see what made up the predictions, let's see the Shapley values. (These are from a game theoretical concept, basically, the prediction is an outcome of a "game", and the attributes are "players", the Shapley values describe how much each player attributes "locally" to the outcome. This is a used concept in Explainable AI.) We see that rebounds stole the show, blocks are up there strong, and attempts are also among the better predictors, rather made goal stats. As we see on the right, the height values are put into different classes, which is not optimal.



Random forest

I used similar parameters for forests as for the tree. 100 estimators seemed like a nice fit, for depth, I took a 3 levels deeper depth to enable for "deeper" trees to in the ensembling, and lowered the minimum samples split number by 10 to allow smaller splits too. This classifier resulted in 16.8% accuracy, but 44.7% 1-inch accuracy, 71% for 2 inches, 83.6% for 3 inches. The maximum positive error shrinked to only 7 too, and apart from 3 cases, we are in the (-7,+6) range.

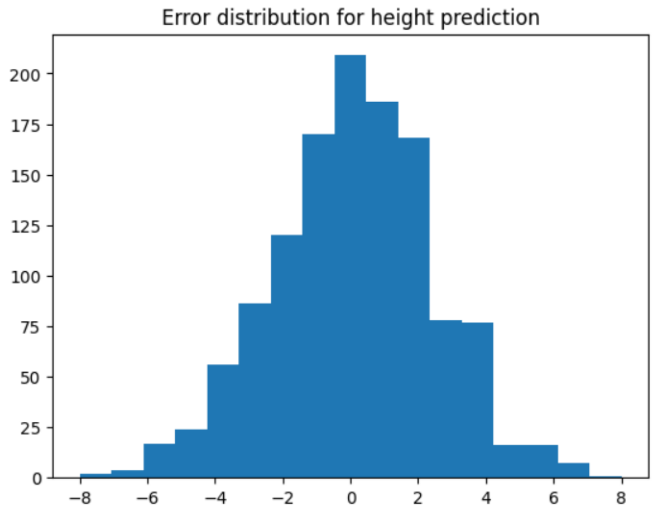


In all measures, the forest performs is 1.5-3% better than the tree itself.

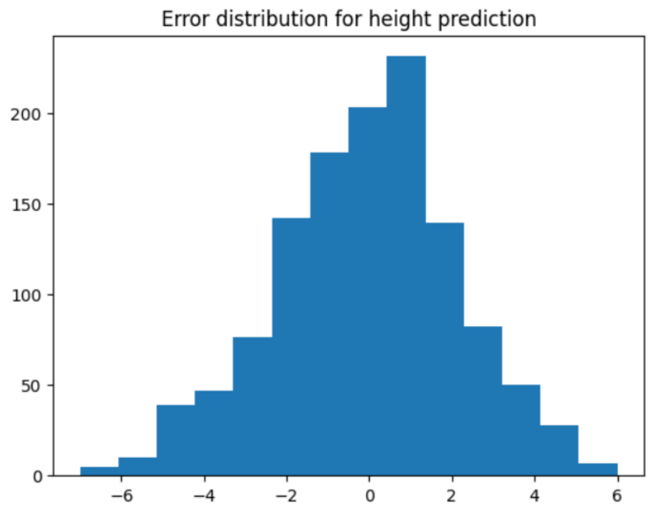
What is surprising is that the number of +2 errors increased (whilst the number of +1 errors decreased). It shouldn't be like this, probably if we use weights (and give more weight to the picked decision tree) this part could be improved a bit. To aid this, I decided to build my own ensembler. Based on the later ideas, I decided to make a random forest regressor model too. It only guessed in 1-inch range 35.5% of the times and did poorly, just like other regressors, but it had the best range: [-7, 4].

Ensemble forests

I took the previous homework code, and created a class MyLimitedForest, built on fitted decision trees. Firstly I just counted the mode of the predictions from the trees, in the second case I averaged the guesses. Ensembler 1:



Accuracy on this case is 0.169, but it varies from run to run. The model does "mixed". The accuracy only here is highest, in other cases it was the lowest with around 15.0%, but 1-inch accuracy is consistently higher than the random forest, above 45%. This suggests to me an "averaging height predictions", maybe the 1-over/1-lower guesses will somewhat cancel each other out. That is what Ensembler 2 was built upon:



Whilst surprisingly the "perfect accuracy" (0-inch-error) isn't the best, and we see a shift towards much more to +1 errors (which is intriguing, see how we have much more +1 errors than correct guesses), the ±1 inch accuracy is the highest by far, ±2 and ±3 inch accuracy is also the highest. Safe to say, this is the best as of yet, and it can be "tricked" into better perfect guess by choosing a 1-higher guess than the average.

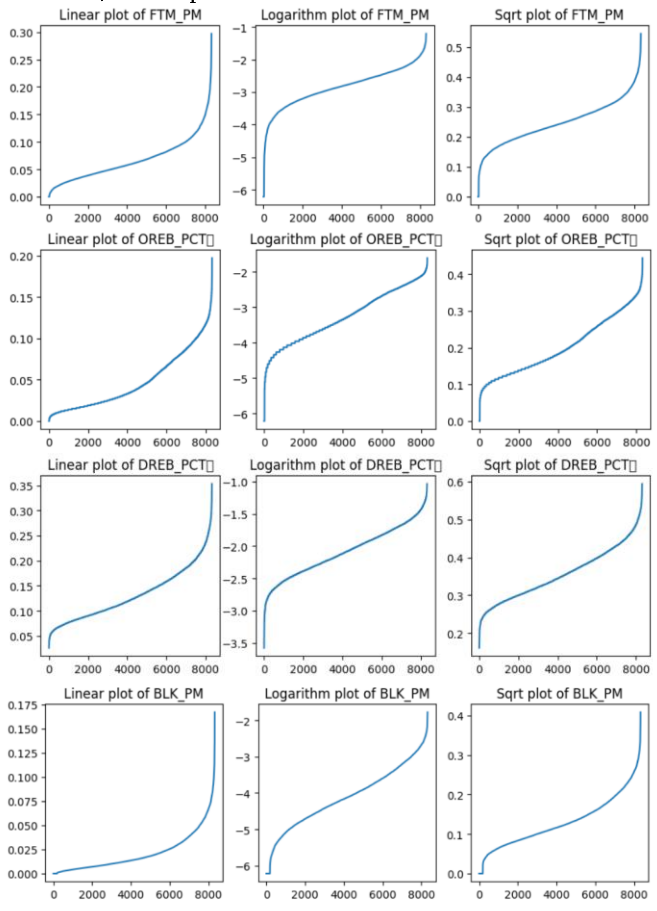
Trying with different values it does indeed vary quite a lot, and I occasionally got 17+% for perfect accuracy, but the 1-inch accuracy is always the highest at 49%. As the perfect accuracy is lower for the simple ensembler predict, the average is near the highest, suprisingly. The "+1 bias" is indeed consistently happening.

If I'm being honest, I expected better "perfect accuracy", but the +-1 inch accuracy is very much better than expected. It's just surprising that we guess 1 inch lower/higher 16-17% of the times, just like we guess the correct height. You'd think that they'd more be like 22-11-11 or so. But the accuracy of this model strongly suggests using regression instead of classification, which sadly I only realized very late.

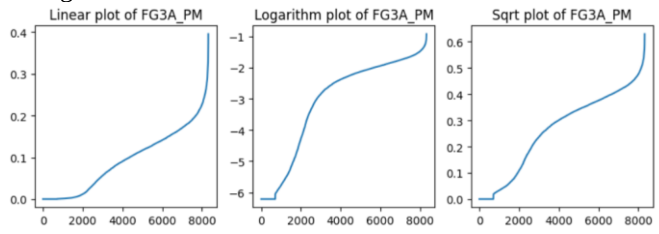
Clustering

As said, it was a goal to find relationships this way too between height and playstyle. Height had double weight as to other attributes, age got filtered. To normalize properly, I looked at the distribution of some attributes, and applied logarithm and square root when needed to get a more linear distribution (for normalizing).
I did not consider age and minutes played.

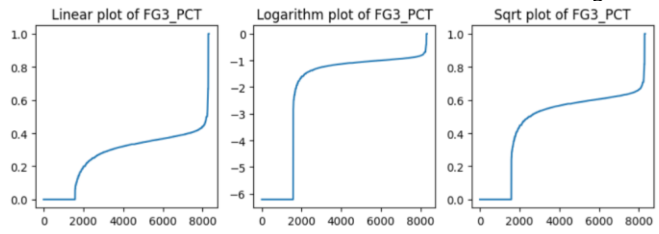
I took the logarithm for free throws made per minute, rebounds, blocks per minute.



I took the square root of values for 3-pointer attempts, as the rightmost distribution seem to fit a linear line most:



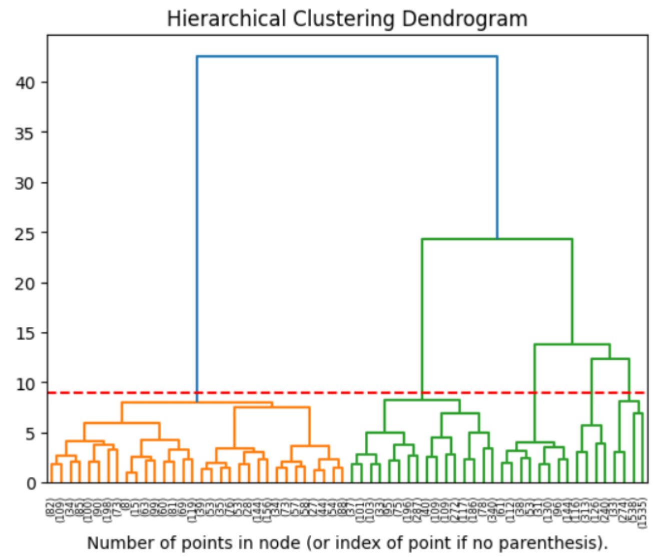
I removed FG3 %, as IT'd create more biased clustering:



Unpleasant looking. In the first case, 0 and 1 values would cluster together because everything else varies too much from them, in the second and third one, the 0 values would cluster. I cannot keep this as it is, has to be dropped.

I also tried fitting an inverse-sigmoid on TS_PCT, but it did not improve the distribution enough, so I just used it as it is. Lastly, I doubled the value in player height, to bias it towards similar height players.

The dendogram:



I cut it at the red line as it seems like a good separation, thus we got 5 clusters. The clusters are: Group 1 on the far left with 2304 instances, the most disconnected, Groups 2, 3, 4 are "fairly similar" (three most right) with 2380, 665, 795 instances respectively, Group 0 is also quite disassociated. Let's see average values on the groups:

ID	HT.	WT.	OREB	DREB	PF	FGA
0	74.7	193.8	0.019	0.085	0.078	0.339
1	82.3	247.5	0.091	0.180	0.123	0.283
2	79.0	217.6	0.035	0.119	0.088	0.297
3	82.4	239.3	0.059	0.177	0.101	0.379
4	77.4	210.7	0.025	0.116	0.072	0.464
	TS	FG%	FG3A	FG3%	FTM	BLK
0	0.521	0.417	0.123	0.339	0.064	0.006
1	0.545	0.509	0.008	0.106	0.069	0.039
2	0.524	0.424	0.118	0.325	0.049	0.015
3	0.552	0.457	0.124	0.336	0.078	0.033
4	0.553	0.444	0.160	0.351	0.110	0.014

Let's evaluate the averages:

Height, Weight: Group 1 and 3 are the big guys (height similar, group 1 is a bit heavier), group 0 is lightweight, group 2 and 4 are medium. Group 4 is also somewhat more in the lighter category.
Rebounds: We can indeed see that the two "big" groups have substantially more rebounds than the other groups. What makes a difference is that whilst their defensive rebound percentages are quite similar, the offensive ones are not: the biggest group (group 1) has more than 150% of offensive rebounds compared to the group 3.
Shooting percentage: Nearly similar with the exception that small guys and group 2 does a little less well.
Minutes: Group 4 plays a lot, much more than other groups. The biggest guys play the least, along with group 2. Little guys and group 3 play similarly much.
FGA_PM: Group 4 has exceptional stats. Group 1 and 2 do the least, small guys also do not try much.
FG_PCT: Small guys and group 2 do worst, the biggest guys stand out. Group 4 do similar to group 3.
FG3A_PM: Group 4 does the most by quite an amount, small guys do as much as group 3, I expected more from them. But the biggest guys almost never throw a 3-pointer. I highly assume they are players from one position, that is close to the basket.
FG3_PCT: Similar statistics to all groups except the biggest guys.
FTM_PM: Group 4 does best again, and group 2 do worst.
BLK_PM: The big guys block most, small guys almost never, once every 160 minutes.
PF_PM: The big guys foul the most but considerably low amounts, group 4 does best.

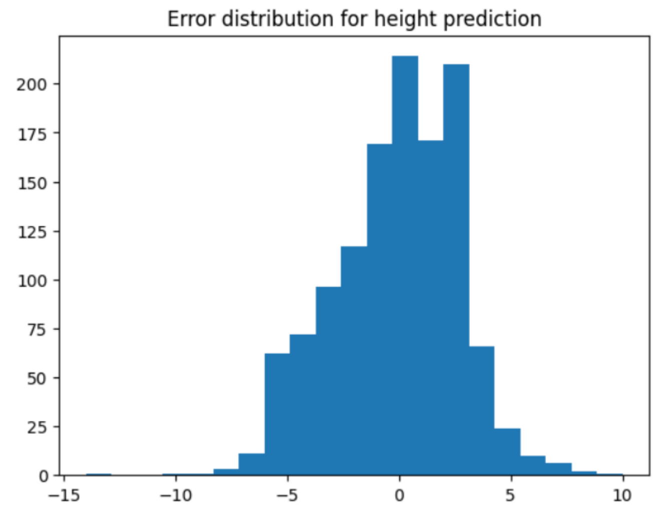
Other: Age is mostly similar. Seems like it doesn't matter.
For minutes, group 4 plays a lot, much more than other groups. The biggest guys play the least, along with group 2.

Takeaway:
- The cluster analysis actually separated fairly closely to positions. Big big guys are surely centers, group 3 are power forwards, and small guys are supposed to be the point guards. But since point guards, shooting guards and small fowards (remaining positions) are so versatile that the cluster algorithm separated thse players into rather "small guys, bad and good players" from what I see from the stats.

Gradient descent

To conclude the research, finding a very strong algorithm to tackle the problem would be ideal. As gradient boosting is the “best” for this type of tabular data (alongside random forests), the goal is to construct a booster stronger than all previous models.
The first thought was to remove the unnecessary variables to cut down on overfitting, so I first dropped age, personal fouls and total shot percentage, and ran the gradient descent classifier algorithm with max depth equal to 1 on this data. (I used 1000 estimators, with 0.1 learning rate.)
The result was “clf g0”, a 15.76% accurate classifier, with 44-66-81% accuracies per extra inch error, which is not bad, but a bit underwhelming, although estimators start as dummies, which I assume is weak. Trying without restricting depth actually gave the worst result of the runs: 14.5% score, so I dumped that.

Turns out that according to the scikit-learn documentation [3], gradient boosting is not really sensitive to overfitting, therefore it’s a good idea to go back to the original, and run a classifier with the exact same parameters. This resulted in a suprisingly good 17.3% result, which is again good with perfect accuracy, but bad with 1-,2-,3- inch accuracy, in the latter two, even worse than the tree. So while this model may have the highest F1 score so far, don’t be mislead as it’s not that close when it doesn’t hit the target. We experience again the common overshooting prediction, but here the common overshoot is +2.



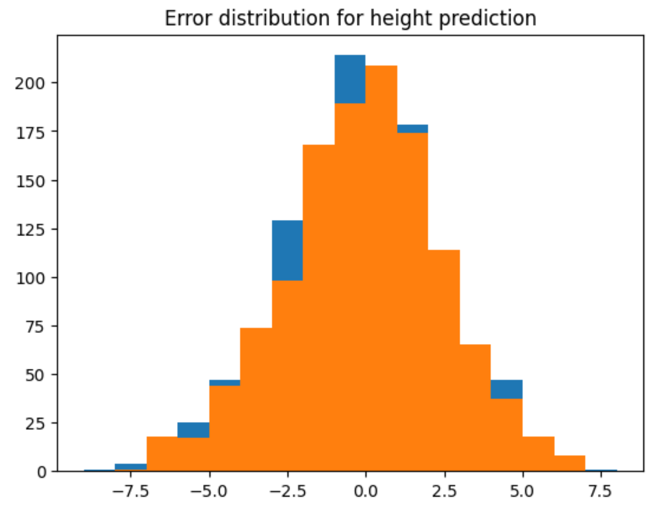
After reading [3] and [4], I have developed a deeper knowledge on how to choose parameters for my booster. On one part, they recommend trying out the following parameters: learning rate < 1, typically learning rate = 0.2, with subsample < 1, typically 0.5. A faster version with comparable accuracy is max_features = 2. In any case, they recommend always cross-validating, so that is what I did. I found that you can further train a previous classifier with gradient descent, I’m sure to use it on our previous random forest classifier.

Wrong parameters for gradient descent

This painful part of the project took the most time, and it was the result of a tiny mistake which cost over 66 hours, during which I did not notice the mistake made. When defining a parameter space for my hyperparameter-tuning, I actidentally gave the GSCV method the tree parameter space as an input, which resulted in huge computation, and wrong results: I only got two hyperparameter values, the maximum depth and minimum samples split, but I wanted n_estimators, subsample, max_features, and possibly max_depth.
I still used these parameters, running with other values and played around to find that the best results are when I don’t use other classifiers as an initial estimator. However, the results were still not good: for that classifier (without initialization), the accuracy is 16.17%.

Right parameters, regressor

After running the correct cross-validation for 50 hours, finishing 2 hours before the deadline, I could test the best parameter gradient descent classifier, with the highest expectations, thinking „this is IT”, and it turned out to be weak: 14.56% accuracy.
To hopefully save the situation, I tried regressors with the same parameters, both initialized and uninitialized, but they were not as accurate as wished either: 32-33% 1-inch accuracy, with around ±8 maximum error.



Non-player data splitting results

Out of curiosity, I tried splitting data on training and testing totally randomly, with players being in both datasets (with different season data) whether the cause of gradient descent doing poorly is bias. The answer is no: the classifier model is the worst, but surprisingly, the regressor model is the best of its kind.
Classifier: 8% (!) accuracy, only 25% 1-inch accuracy, having both -15 and +15 errors.
Regressor: 33.9% 1-inch accuracy, [-8,12] error range.

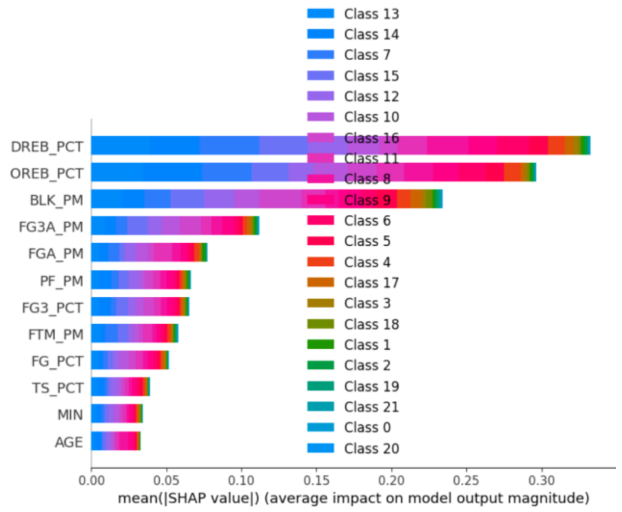
5. VISUALIZATION

The results from the predictors:

Model	Data	Accuracy score	+/-1 inch %	F1 score	Error range
Decision tree	All	15.28%	42.8%	0.149	-8,7
Random forest classifier	All	16.81%	44.7%	0.149	-8,6
Random forest regressor	All	-	35.5%	-	**-7,4**
Ensemble forest 1	All	16.89%*	45.7%	0.139	-8,7
Ensemble forest averaging	All	16.41%	**49.4%**	0.161	**-7, 5**
Gradient descent 0	Filtered	15.76%	44.1%	-	-9,11
Gradient descent 1 classifier	All	**17.30%**	44.8%	**0.171**	-14,9
Gradient descent 2 classifier	All	16.17%*	44.0%	0.138	-8,11
Gradient descent 3 regressor	All	-	33.9%	-	-8,12
Gradient descent 4 classifier	Shuffled	8%	25.0%	0.080	-15,15
Gradient descent 5 regressor	Shuffled	-	37.3%	-	-10,8

*Not consistent enough, often below 16% with other runs.

The differences between the gradient descent models: 0 is an initial run on a more filtered data, 1 is ran on the original data, 2 has deemed "bad" parameters but picked the best model out of multiple runs, 3 has "good" (hyper)parameters and is a regressor, and the last 2 are ran on data where players may appear in both the training and the test set, but with different season data.



The above diagram is a bit "ruined" due to the class labels, this plot of what attributes mostly impact the outcome in our random forest model is a good representation of the most important aspects.

6. CONCLUSION

This project led to the "best I know" predictions of player height in the NBA. On the internet, I did not find any approaches that exhausted a large database and used some models to predict height, but any biographical statistic predicted from performance stats is rare, the vica versa is more common. It could help understand how the "playstyles" in the NBA are biased on physical attributes, leading to a big question: "are you, Mr. Basketball Player playing on the position you always wanted, or where your boss put you because of your size?"

What this project showed me on the toolside, are that:

- Hyperparameter tuning is very useful, may be even most on gradient descent which is very sensitive to initial parameters, but when finding a global minimum or anything close to it is not enough to predict your target, then you need to get creative.
- Before you build the final model(s), you have to play with the data and play with "starting models". You have to experiment with what works, what works best, often cross-validate, check what correlates and what doesn't.
- Scikit-learn also gives general advice on machine learning techniques, not just description of the methods of their library.
- When the target variable can be numerical, or in case of classifying we can define some type of distance between classes, these can be really powerful tools. Even to just define some simple distance between classes.
- One has to be very careful when splitting their data manually… If it is biased in any way, that can hurt many model performances.

Further research can be to predict the weight of players alongside height, in scikit-learn there are options to do multi-variate prediction.

References

[1] N. B. Association, „NBA Advanced Stats,” [Online]. Available: https://www.nba.com/stats. [2023].

[2] T. Boger, „Defining NBA players by role with k-means clustering,” [Online]. Available: https://dribbleanalytics.blog/2019/04/positional-clustering/.

[3] scikit-learn, „GradientBoostingClassifier,” [Online]. https://scikit-learn.org/stable/modules/generated /sklearn.ensemble.GradientBoostingClassifier.html. [2023].

[4] scikit-learn, „Ensembles: Gradient boosting - subsampling,” [Online]. https://scikit-learn.org/stable/modules /ensemble.html#subsampling. [2023].