NBA Points Prediction Model

Colin Lei
colin.y.lei@gmail.com

June 21, 2021

This project applied machine learning techniques on a team's average stats to predict the number of points a team will score in a NBA game. This project also explored various sports betting strategies. Predictions were combined with an optimized betting strategy and tested on NBA games from 2012-2021 as an evaluation of their real world effectiveness.

Introduction

The three primary ways to bet on a basketball game is through the moneyline, the spread, and the over-under. The simplest type of bet is the moneyline which is just a bet on which team will win a game. The spread acts as a points handicap and is a bet on which team will 'win' after applying this handicap. Functionally it is a bet on how many points a team will win or lose by. The over-under (O/U) is a bet on whether the total points scored in a game will be over or under a certain number.

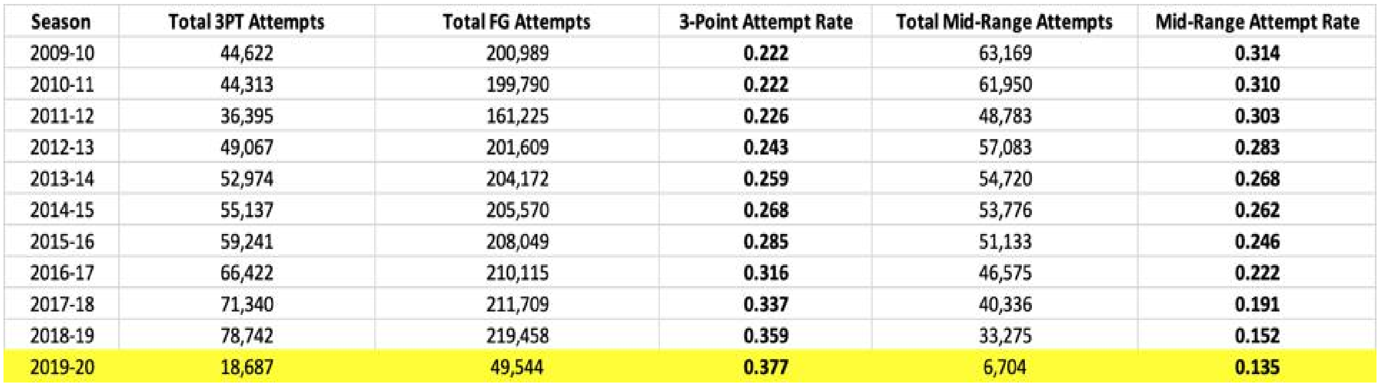
Understanding the three types of bets, it can be seen that if a model could predict the number of points both team will score individually, then it should also be able to make winning bets. The model primarily used a team's average stats over the past x number of games. These stats included traditional stats and advanced stats as defined by the NBA. The bulk of this project was built in Python and employed regression algorithms from popular machine learning libraries, *scikit-learn* (*8*) and *XGBoost* (*4*).

Game Data

The data for the initial model, Phase 1, was taken from a Kaggle dataset containing NBA game stats (*6*). The creator of the database stated that the original data was taken from the official NBA stats website before being combined and sorted.

The dataset contained rows of all the regular season and playoff games from 2004 to 2021. Each row contained identifying information, points (pts), field goal percentage (fg pct), free throw percentage (ft pct), three point field goal percentage (fg3 pct), assists (ast), and rebound (reb) for both the home and away team. E.g. the number of rebounds the away team grabbed was stored in column **REB_away**.

Figure 1 below was taken from an article (*9*), which detailed how the 3pt shot had gotten progressively more popular since the 2012 season.



Season	Total 3PT Attempts	Total FG Attempts	3-Point Attempt Rate	Total Mid-Range Attempts	Mid-Range Attempt Rate
2009-10	44,622	200,989	**0.222**	63,169	**0.314**
2010-11	44,313	199,790	**0.222**	61,950	**0.310**
2011-12	36,395	161,225	**0.226**	48,783	**0.303**
2012-13	49,067	201,609	**0.243**	57,083	**0.283**
2013-14	52,974	204,172	**0.259**	54,720	**0.268**
2014-15	55,137	205,570	**0.268**	53,776	**0.262**
2015-16	59,241	208,049	**0.285**	51,133	**0.246**
2016-17	66,422	210,115	**0.316**	46,575	**0.222**
2017-18	71,340	211,709	**0.337**	40,336	**0.191**
2018-19	78,742	219,458	**0.359**	33,275	**0.152**
2019-20	18,687	49,544	**0.377**	6,704	**0.135**

Figure 1: FG types 2009-19

There was a drastic jump in 3pt attempt rate starting from 2012-13. By the 2019 season, the 3pt attempts rate had seen a 67% increase compared to the 2011 season. This style change could cause concept drift within the model, specifically with the weighing of **fg3_pct**. Thus, the model was trained and tested on a subset of games, beginning in the 2012 season to the current 2020 season.

Feature Engineering

Since team's average stats were not provided they had to be calculated for each game. For starters, a log of previous games for the home team of the current game, e.g. Toronto Raptors, was extracted. Then the old game stats were sorted into two types, **_gotten** and **_allowed**. Gotten stats represent the stats that the Raptors recorded, e.g. Raptors *got* 30 assists last game. Allowed stats represent the stats that the Raptors' opponents recorded, e.g. Raptors *allowed* the other team to score 120 points last game. Finally, the gotten and allowed stats were mean-averaged and appended to the original data set. E.g. the Raptors' average field goal percentage over the past x games was stored in column **fg_pct_gotten_h**. The same process was applied to the away team and their averages were similarly stored just with an **_a** suffix.

Multiple datasets were created using [8, 20, 40, 60, 80] previous games (prev_count) to test whether a team's recent performance or their season-long performance functioned as better predictors.

Odds Data

Data for 2012 to 2021 betting lines were taken from an archive (2). A few entries had missing information which was manually patched in with data from other betting sites.

A betting line is the specific number for which one can bet on either side of. These lines can change as a response to new information, such as an injury update. Closing lines are the last snapshot prior to bets being locked in and are generally the most accurate predictions. So to challenge our model, lines for the spread and total O/U were taken from the **Close** column. The odds for spread and O/U are always -110 (assuming 10% vigorish) meaning $110 bet to win $100. Moneyline bets do not have a 'line' but rather have different odds for the home and away team which were stored in their own separate columns.

Using the following formula, an estimate of the bookmaker's prediction for how many points each team would score was obtained.

$$\frac{\textit{total game points}}{2} \pm \textit{spread} = \textit{total team points}$$

The bookmaker's points prediction was used as a baseline model, dubbed Vegas model, to compare performance metrics with.

Merged Data

The games dataset and betting line dataset were sorted, matched, and merged using game date and team names. Each row held information for a single game. Since the goal was to predict points scored by each individual team, each row was duplicated. The stats for home, **_h**, and away, **_a**, were swapped for every other row. Effectively, each team was treated as the 'home' team when the model was trying to predict their score. Their opponent was treated as the 'away' team regardless of the teams' actual status. A feature column **is_home** was added to preserve the true home/away status. The target variable was stored in **pts_home**.

Data Exploration

Figure 2 below contains subplots of histograms overlapped with kernel density estimators for each explanatory and target variable. The plots showed that most variables were normally distributed and all were within expected ranges. **pts_gotten** and **_allowed** had three peaks, likely due to the league average for points scored per game varying each year (*1*). This variation actually occurred with every stat but generally to a lesser degree. A few distributions, such as **ast_gotten**, had a slight skew due to outliers. A quantile transformation was applied on training and testing datasets to make all distributions more Gaussian.

Prior to creating the features, some games with outlier values were verified using the official records from the NBA and an independent archive Basketball Reference. E.g. on Jan 19, 2018 the Lakers did actually shoot 14% (2/14) from the free throw line vs. league average ∼77%. All spot checks passed; thus, there was no reason to believe that the data had any unnatural outliers that required removal.

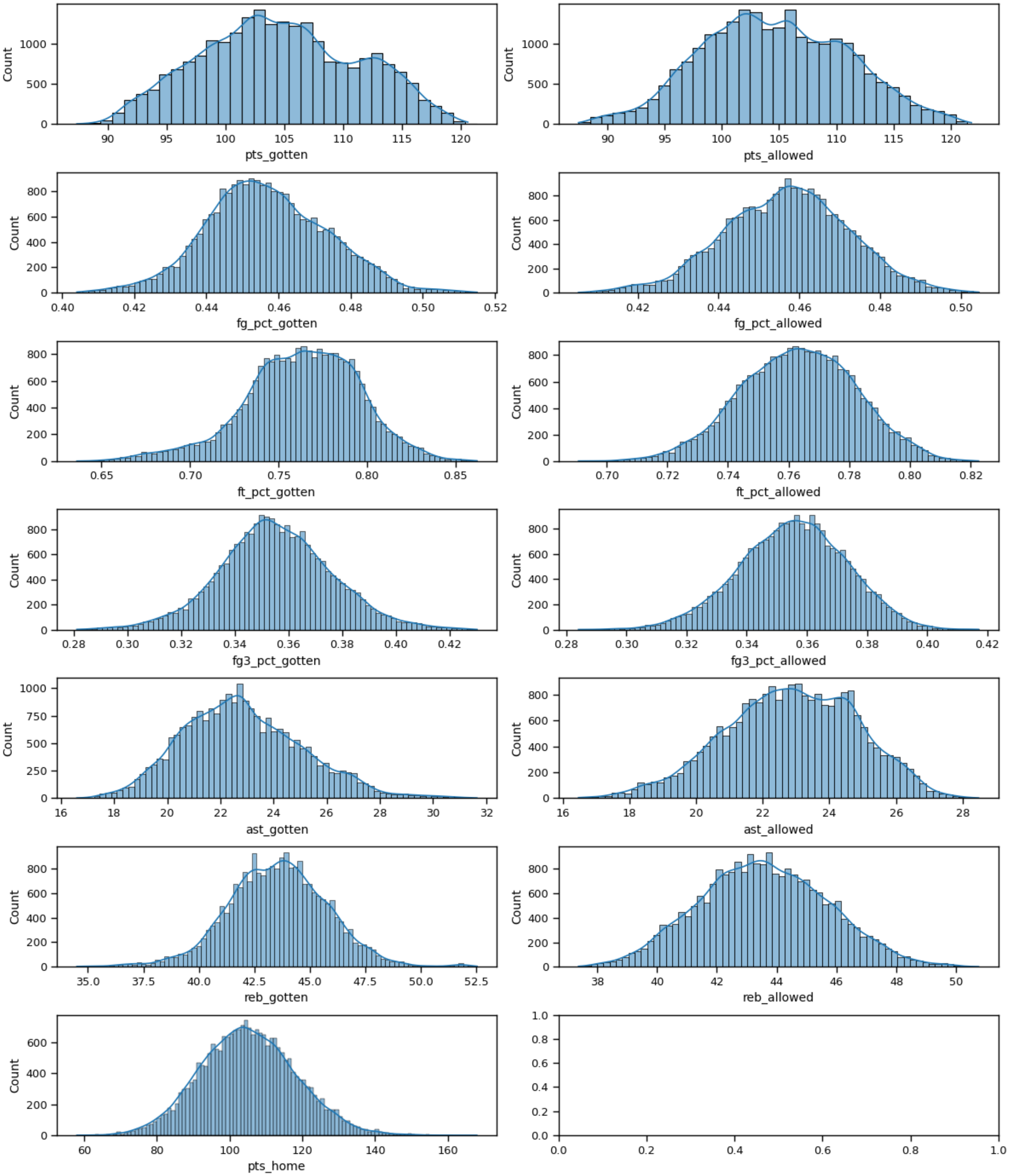


Figure 2: Histogram and KDE of target and features

The correlation heat map in Figure 3 indicated that some explanatory variables were moderately to highly correlated with each other. This was not surprising as one can expect a stat like **fg_pct_gotten**, which measures how often a team scores on their shots, to have a high correlation with **pts_gotten**, points scored. However, no correlation pair was greater than 0.8. So while highly correlated, every feature still offered some unique information.

It is also important to note that all of these variables are real basketball stats that provide context and interpretability to the model. E.g. it is known that **fg_pct** and **ast** are highly correlated to **pts**. However, if a team was looking to improve their game, saying "We need to get more points!" is far less informative than saying "We are moving the ball around and getting a lot of assists but our shot conversion percentage is still lower than league average. We need to analyze which type of shots are bringing our average down and practice those." Thus, only features that offered nearly no unique information and had extremely high correlation to each other, > 0.95, were removed.

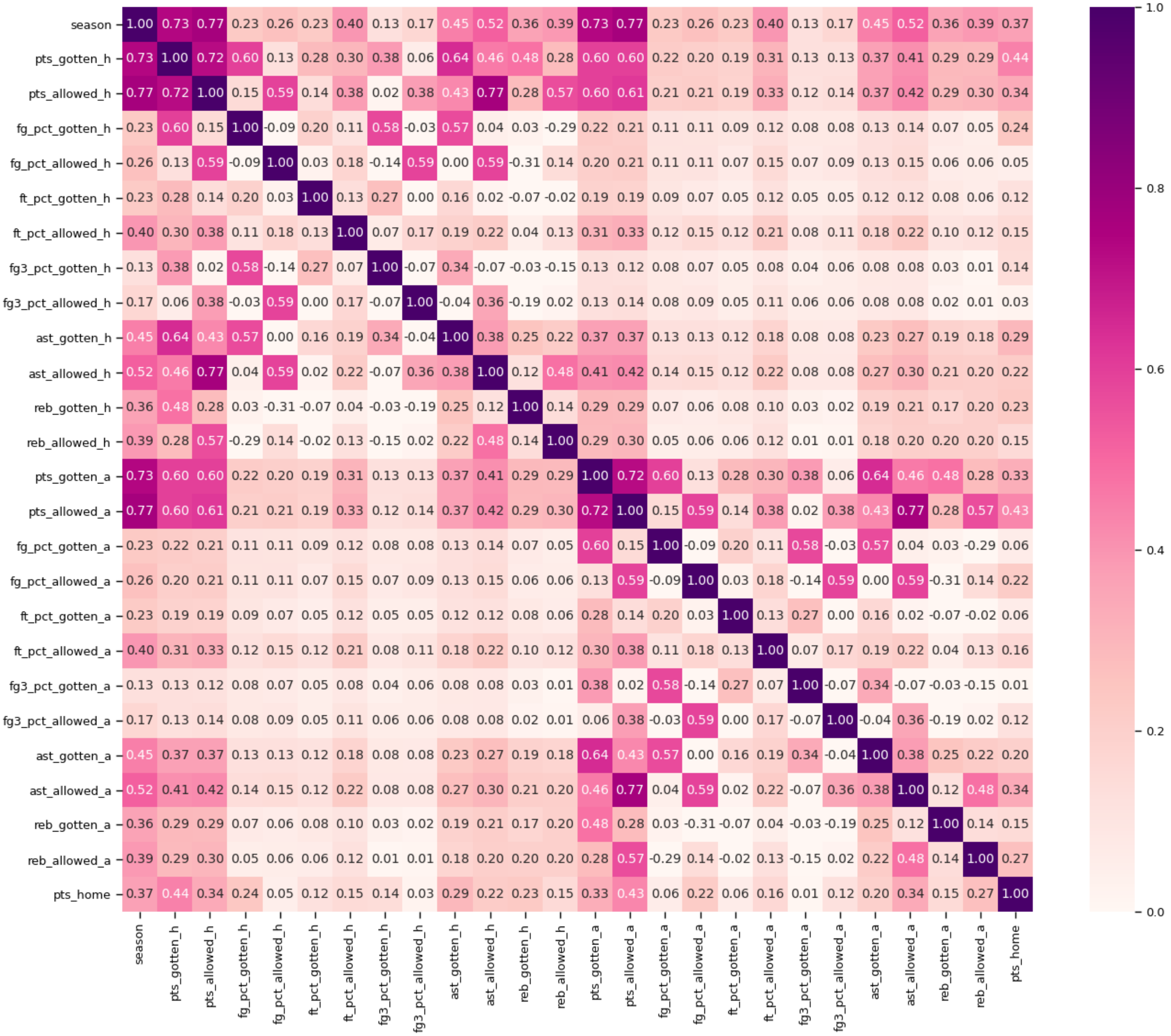


Figure 3: Correlations of target and features with each other (prev_count = 40)

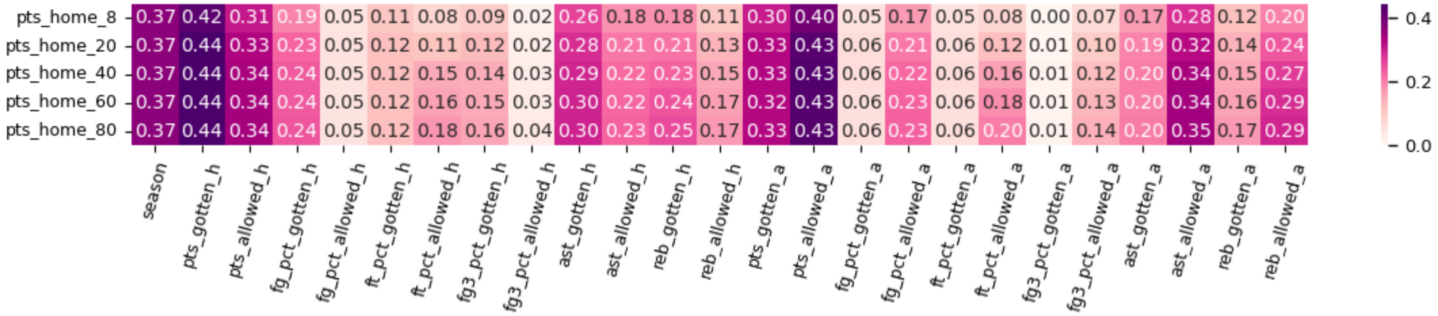


Figure 4: Correlations of features to target for varying prev_count

Figure 4 above compared the correlations of the dependent variable and the independent variables generated by [8, 20, 40, 60, 80] previous games. The correlations were mostly similar but there were a few variables that got stronger as more previous games were used, such as **fg3_pct_allowed_a**.

Most of the stats which did correlate with **pts_home** made sense. E.g. rebounds grabbed by home team and rebounds allowed by the away team can predict how many extra possessions home team gets which translated to extra points. Low correlation stats included **fg3_pct_allowed_h** and **fg3_pct_gotten_a** which can both help to predict away team's 3pt field goal percentage and, as expected, likely do not help much with predicting home team's points.

Interesting ones were **pts_allowed_h** and **pts_gotten_a** which should predict how many points away team scores; yet, they were somewhat correlated with how many points home team scores. This could be because our NBA teams are both literally and figuratively in the same league. Thus, expected points scored by either team in the same game should not differ too much. Figure 5 showed that 75% of the time the final score was within 16 points.

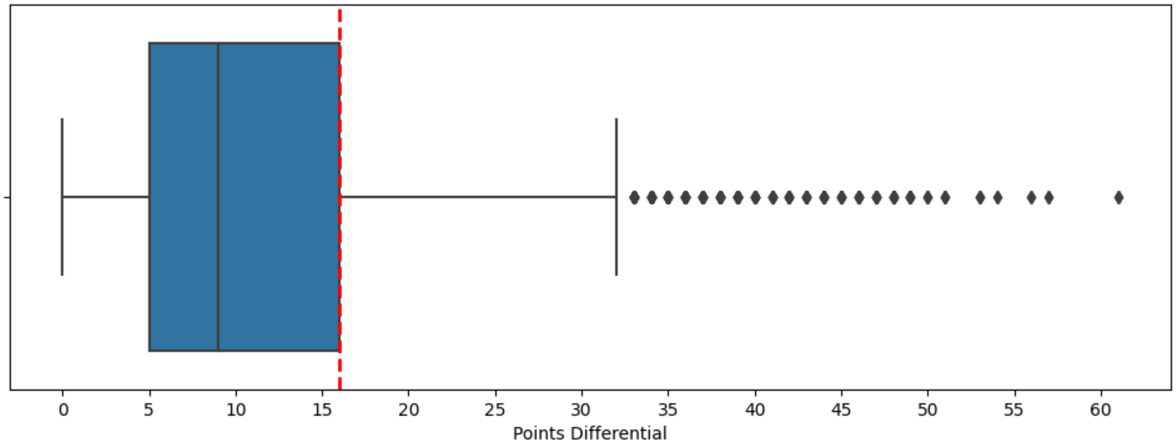


Figure 5: Boxplot of point differentials each game

Any variable that had an absolute correlation value < 0.05 across all previous game counts were removed, as those were unlikely to be useful explanatory variables across any hyperparameter setting. A threshold of 0.05 seems low but the highest correlation was only 0.44 with a mean of 0.20. Additionally, the averages built using prev_count = 80 were used as features for the models since they had the strongest correlation with the target variable.

Train Test Splits

Due to the chronological nature of the dataset, it would be neither logical nor realistic to train on newer data in order to predict older data. Thus, the first training set included games from the 2012 season to 2014 and was tested on the 2015 season. The 2015 test set was then added to the training set, the model was retrained, then tested on the 2016 season, and so forth, up to and including the current 2020 season. The training set was further split 70/30 into training and validation. The validation set included games that came from the same seasons the model was trained on, and was expected to perform much better than the test set would.

Phase 1 Validation Results

For Phase 1, models were trained using six regression algorithms from *scikit-learn* and one from *XGBoost*, and then tested on the validation set. Hyperparameters were initialized using mostly default values along with some sensible guesses. The primary performance metrics were mean absolute error (MAE) and explained variance percentage. Additionally, each pair of predictions for a single game was compared to the betting line and if the difference exceeded a certain threshold, a $100 dollar bet was placed. E.g. if the threshold was 3 and the O/U was 200, the sum of the pair of predictions must have been >203 or <197 to warrant a bet. Dollars won or lost was tracked independently for each type of bet.

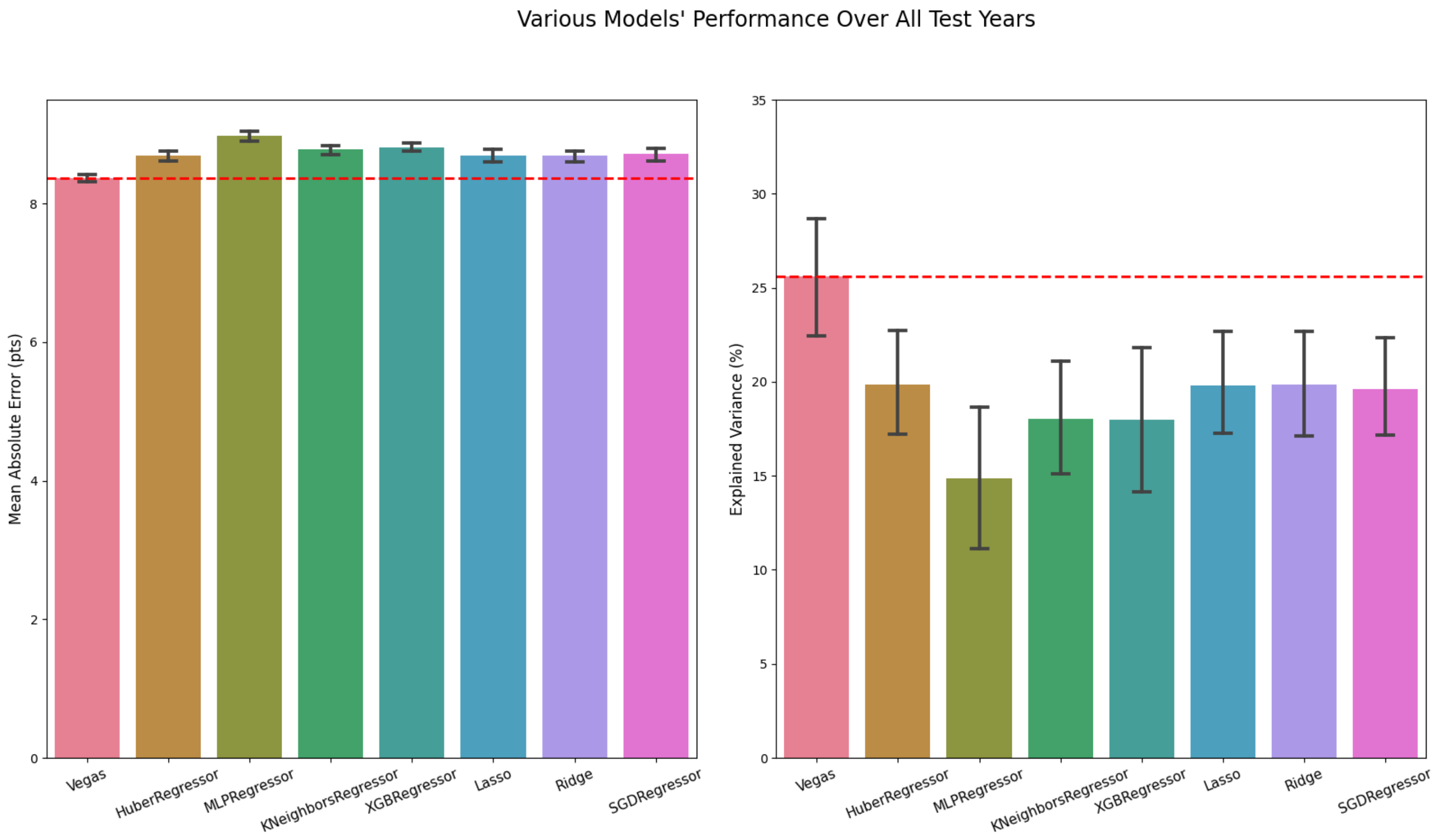


Figure 6

Results were grouped and plotted in Figure 6. None of the models had lower MAE or higher explained variance than the baseline Vegas model in any of the test years.

With the exception of K-nearest neighbours' (KNN) $6,920 winnings on O/Us, all spread, O/U, and moneyline bets resulted in cumulative losses in the tens of thousands when a difference threshold of 0 was used. A similar pattern occurred with a threshold of 1 as seen in the plot below.

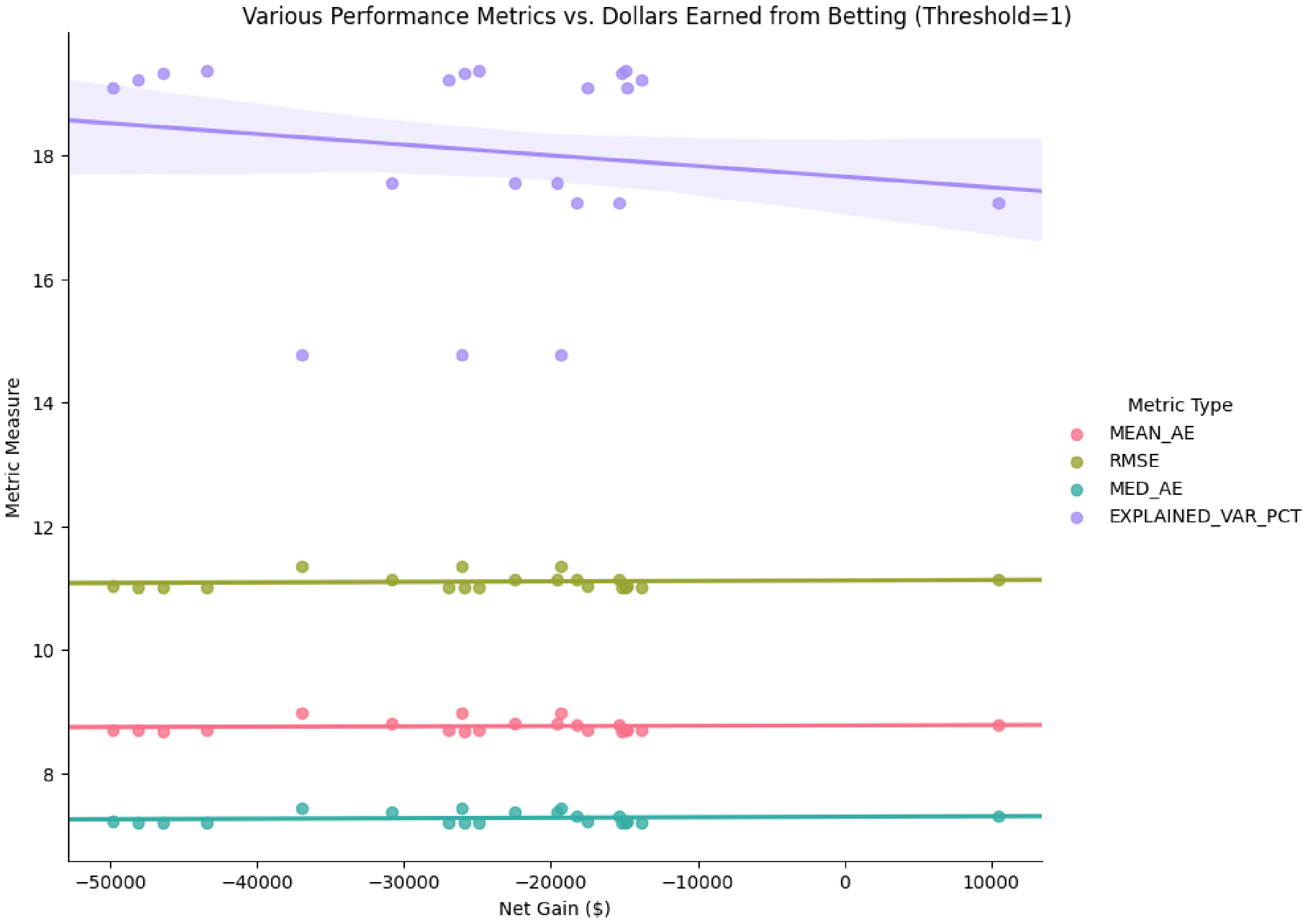
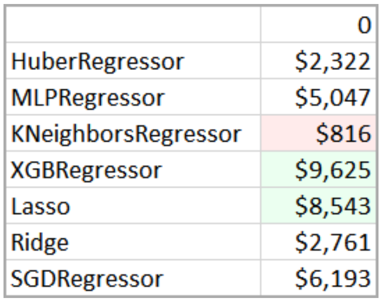


Figure 7

Figure 7 shows the relationship of various model's performance metrics and the cumulative dollars won from the three types of bets. One would expect that predictions with lower error or higher explained variance scores would win more bets but that was not necessarily the case. Unfortunately, money earned from betting was extremely noisy and betting with one model that had the exact same error as another model could either net you $10,450 or -$49,880 over six years. Difference hyperparamter settings and bet thresholds within the same model also yielded varying results but to a lesser degree. The difference intra-model was somewhere in the ballpark of a few thousand instead of tens of thousands.

After the disappointing results, a modified betting strategy was employed where moneyline bets were only placed on the underdog team. The theory was that teams who were predicted to lose offered better odds than favourites. E.g. if underdogs were +200, a $100 dollar bet would return $200 which means a bettor only needs to win $33.\overline{3}\%$ of bets to break even. While betting on a -200 favourite requires a bettor to win $66.\overline{6}\%$ of the time. Furthermore, strictly betting on underdogs would lead to less bets placed which would lessen a bettor's exposure to the 10% vigorish or 'commission fee' that bookmakers take for each bet. Figure 8 shows the promising result of this strategy with every model turning a profit. However, the exact amount of money won remained noisy.



	0
HuberRegressor	$2,322
MLPRegressor	$5,047
KNeighborsRegressor	$816
XGBRegressor	$9,625
Lasso	$8,543
Ridge	$2,761
SGDRegressor	$6,193

Figure 8: Dollars earned from moneyline bets on underdogs only (threshold = 0)

Phase 1 Improvements

When tested on the validation set, Phase 1 models were able to turn a profit on underdog moneyline bets but none were able to predict points scored better than Vegas could. Thus, some improvements were made; primarily, more stats were added to the dataset. Traditional and advanced box scores for each game were taken directly from the NBA using the *nba-api* package (*7*). The number of stats tracked increased from 6 to 33.

In an attempt to reduce outliers, overtime games, which had inflated countable discrete stats due to extra play time, were normalized to a regular 48 minute game. However, bets were still evaluated using the actual results of the games. After normalization, the same process of data cleaning, feature creation, feature preprocessing, and dataset splitting were repeated on the new dataset.

Phase 2 Validation Results

The Phase 2 models performed slightly better on the validation set compared to Phase 1 and can be observed in Figure 9. Due to the normalization of **pts_home** in outlier overtime games, Vegas' model also improved by a similar amount. Therefore, indicating that improvements of all models were likely due to overtime normalization and not due to the new feature data.

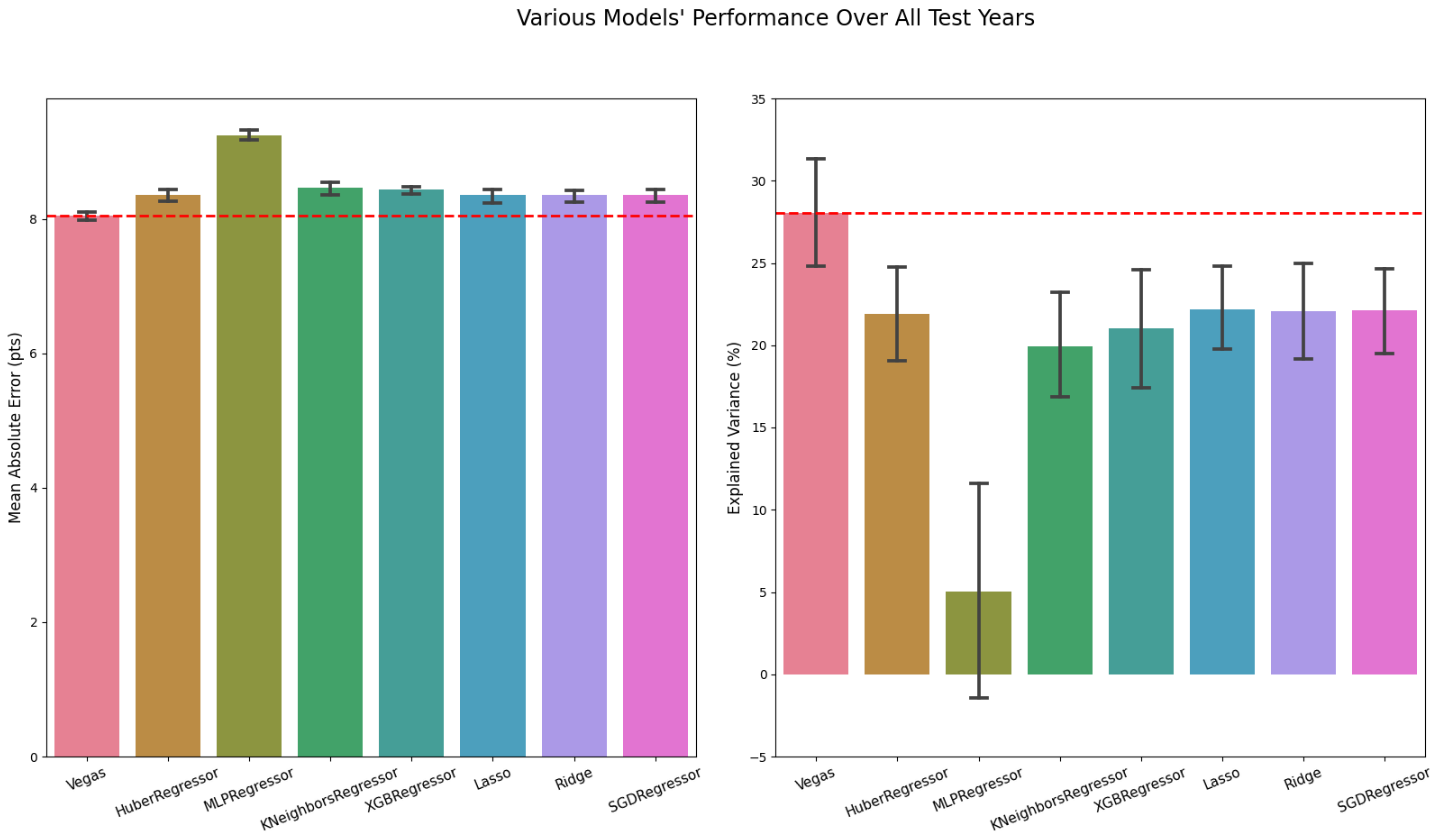
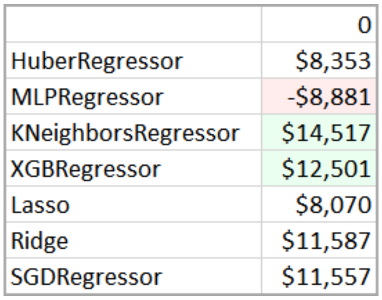


Figure 9

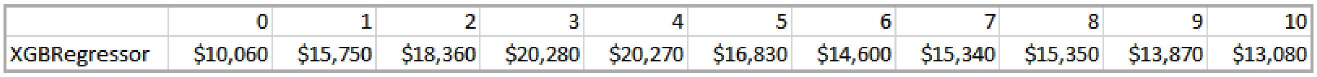
For underdog moneyline betting, nearly every model made more money in Phase 2.



	0
HuberRegressor	$8,353
MLPRegressor	-$8,881
KNeighborsRegressor	$14,517
XGBRegressor	$12,501
Lasso	$8,070
Ridge	$11,587
SGDRegressor	$11,557

Figure 10: Dollars earned from moneyline bets on underdogs only (threshold = 0)

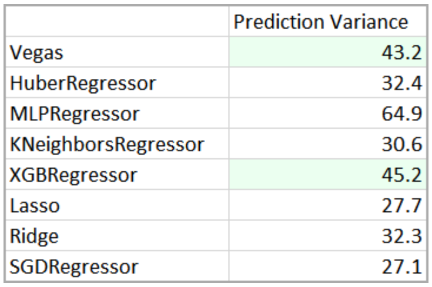
Instead of KNN as was in Phase 1, XGBRegressor (XGBoost) was the sole model to produce consistently profitable results for O/U bets. Other models were too inconsistent as they turned a profit on certain thresholds but not others. Furthermore, these profitable thresholds were not consistent across Phase 1 and 2.



	0	1	2	3	4	5	6	7	8	9	10
XGBRegressor	$10,060	$15,750	$18,360	$20,280	$20,270	$16,830	$14,600	$15,340	$15,350	$13,870	$13,080

Figure 11: Dollars earned from O/U bets across multiple thresholds

While XGBoost did not perform the best on the MAE and explained variance measurements, it was more consistently profitable than other models across both phases. The difference between the lowest error model, Lasso, and XGBRegressor was $|8.355 - 8.441| \approx 0.09$. To put this into context, the target had a median of ∼105, so Lasso predicting 0.09 points better was relatively meaningless especially when not backed up by betting results.



	Prediction Variance
Vegas	43.2
HuberRegressor	32.4
MLPRegressor	64.9
KNeighborsRegressor	30.6
XGBRegressor	45.2
Lasso	27.7
Ridge	32.3
SGDRegressor	27.1

Figure 12: Mean prediction variance across all years

A possible explanation for this behaviour is that XGBoost's prediction variance is closest to Vegas'. Other models with less variance likely made predictions which were too conservative, while MLPRegressor had more variance and likely made predictions that were too extreme. Thus, XGBoost was chosen as the final model that will be optimized and used to predict the test set.

Optimization

Hyperparameter tuning was done using the NSGA-II algorithm (5) in *Optuna* (3) to optimize for both MAE and explained variance percentage. A notable change was going from mean squared error loss to Pseudo-Huber loss, which is more robust and closer to absolute loss. The model was retuned after every season across five random seeds and the parameters were averaged before being fed into the final model.

The moneyline betting strategy had two hyperparameters which were manually optimized using predictions from Phase 2's XGBoost. The first being the difference threshold which was expanded to test [-5, 10] with steps of 1. The second being the lowest odds the model would still be willing to bet on. [-110, -105, 100, 105, 110] were chosen to be tested because -110 is technically even odds prior to the vigorish being applied and is also the odds for all spread and O/U bets.

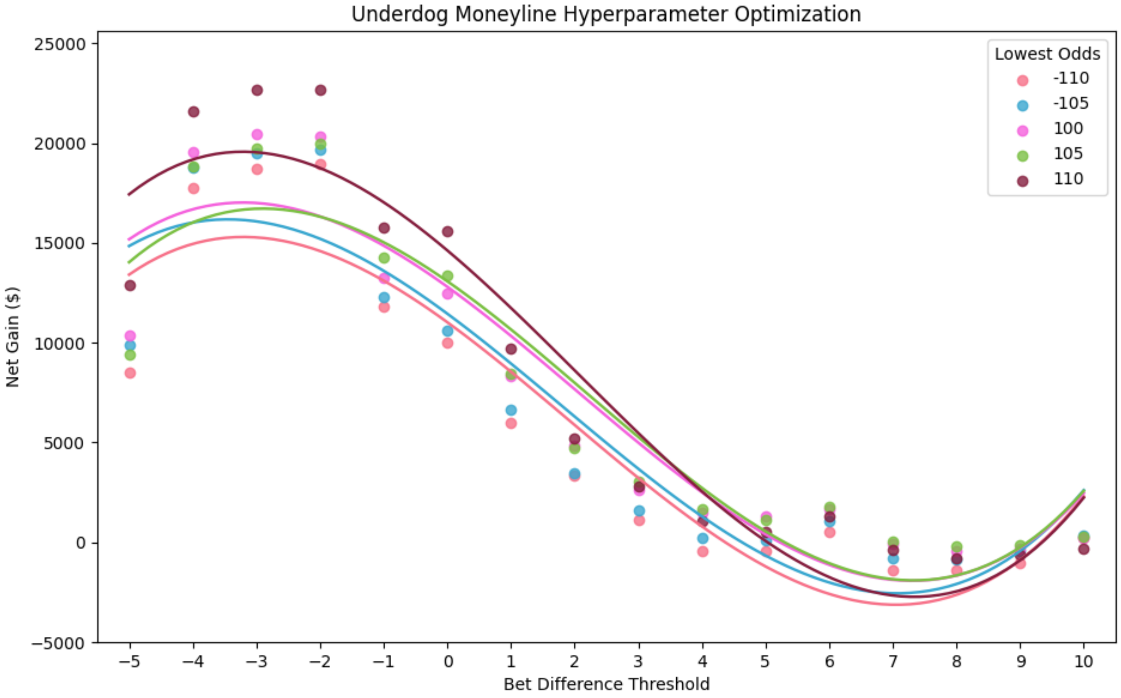


Figure 13

From the plot in Figure 13, it appeared that a negative difference threshold performed better than a positive one. This was an unexpected outcome because a negative difference, e.g. -1, indicated that team A was predicted to defeat team B by 1 point. Thus, it made no sense to moneyline bet on team B since the model did not project them to win. Upon further inspection, since the model was only placing bets on underdogs there were times were the offered odds were extremely high, possibly too high. E.g. if team B has +500 odds, this implies that Vegas estimates team B would lose against team A $83.\overline{3}\%$ of the time. Vegas' spread would also be -11 in this case, while the model's predicted difference would only be -1. So while the model believes that team B is still more likely to lose, it believes they would lose only 52% of the time. Now if team B's true loss probability is closer to the model's prediction, say 60%, then +500 odds is too generous an offer. Over time and over many of these 'value' or 'mispriced' bets, the model would be profitable despite betting on projected losers.

The peak, at difference threshold = -3 and lowest odds = 110, was chosen as hyperparameters for the final model's betting strategy.

Final Phase Test Results

Using the optimized hyperparameters, a regression XGBoost model was trained yearly over five random seeds. The models individually predicted games scores for the following season, which was the test set, and the median of the five predictions was used as the final prediction.

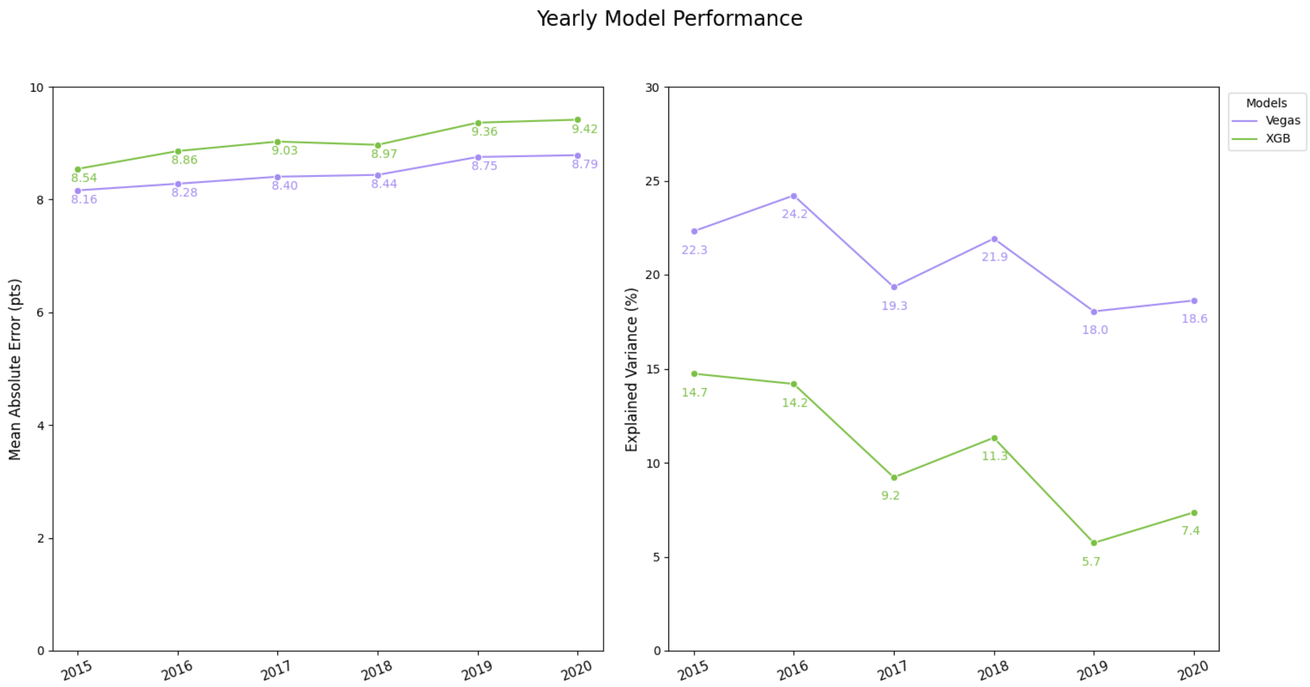


Figure 14

The final model was still unable to best Vegas' model, generating 6.6% more error and explaining 50% less variance. The only profitable betting strategy was underdog moneyline betting and Figure 15 plots the yearly performance. 2018 was the only net loss year but did not perform significantly worse on MAE and explained variance. This likely suggests that the model was simply unlucky that year and that underdogs lost more games than they should have.

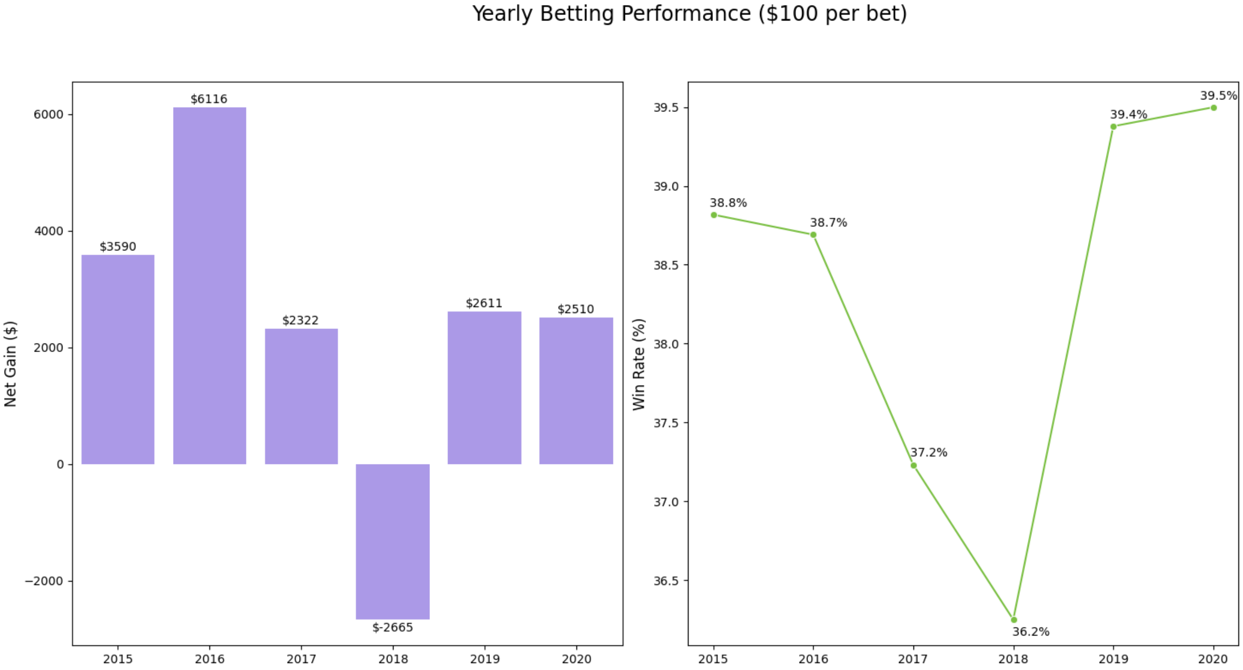


Figure 15

Conclusion

The optimized Pseudo-Huber XGBoost model was unable to predict each team's points more accurately than bookmaker's baseline model. Despite the model being off by an average of 9 points per team, it was able to generate a cumulative $14,484 through a underdog moneyline betting strategy, which had an emphasis on finding mispriced bets.

Features were created using traditional and advanced team stats, for both the home and away squads, which were averaged across 80 previous games. To improve points prediction, some amount of individual player stats would be necessary. One aspect that needs accounting for is each game's player lineups. Average team stats do not provide an accurate representation of a team if their star player switched teams or gotten injured since the time those averages were built. Unfortunately, gathering and applying player data is a complicated and time consuming task.

To improve betting results, a deeper dive into mispriced value bets can be done. Only mispriced underdogs were considered but favourites should be considered too. The difference between bookmaker's implied win percentage and the model's predicted win percentage can be compared and if it exceeds a certain amount, then a bet can be deemed as 'mispriced'. Going further, if one's focus was strictly betting then a better method may be to train a classification model on bet outcomes. The new model could be weighted based on the odds offered so that it would prioritize accurately predicting more profitable outcomes.

Sports betting is a multibillion dollar industry, so naturally bookmakers know what they are doing when setting lines. Finding winning NBA bets is a challenging endeavour but definitely possible if one knows how and where to look.

References

1. *NBA League Averages - Per Game.* https://www.basketball-reference.com/leagues/NBA_stats_per_game.html. *Retrieved May, 2021.*

2. *NBA Scores And Odds Archives.* https://www.sportsbookreviewsonline.com/scoresoddsarchives/nba/nbaoddsarchives.htm. *Retrieved May, 2021.*

3. Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. *Optuna: A next-generation hyperparameter optimization framework. In* Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining *(2019).*

4. Chen, T., and Guestrin, C. *XGBoost: A scalable tree boosting system. In* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining *(2016), KDD '16, ACM, pp. 785–794.*

5. Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. *A fast and elitist multiobjective genetic algorithm: Nsga-ii.* IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.*

6. Lauga, N. *Nba games data.* https://www.kaggle.com/nathanlauga/nba-games. *Version 6, Retrieved May, 2021.*

7. Patel, S. *nba-api Python Package.* https://www.basketball-reference.com/leagues/NBA_stats_per_game.html. *Version 1.1.9.*

8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research 12 *(2011), 2825–2830.*

9. Young, S. *The NBA's 3-Point Revolution Continues To Take Over.* Forbes *(2019).*