NBA Prop Bet Analyzer

Tomas Kersulis (tk492), Tanay Menezes (tm452), Uday Kolla (uk46)
AI Keywords: Machine Learning, Multiclass Classification, Regression, Reinforcement learning
Application Setting: Sports Betting, Fantasy Basketball
Other Students used in evaluation of GUI: Mario Xerri (max3) and Puxuan Yang (py64)
Presentation Link:
https://drive.google.com/file/d/1zwR2YoBgfBkpVxbpN3IZNjrhRYbsH3tG/view?usp=share_link

Project Description:

In this project we planned to create a model that would be able to accurately predict NBA players statistics in the regular season. We would then be able to compare these predictions to available prop lines on various sports betting websites and see how our model would fare against these prop lines. For more context, a prop line is a number set by a betting sportsbook for a bet on a specific player's statistic for a game. Bettors then either gamble on the player having more or less of that statistic for that specific game. We knew this was going to be an ambitious task as sportsbooks want to maximize profits and therefore strategically set their lines based on their own models for which they hire data scientists to create. Therefore, it is extremely difficult for bettors to profit off of sports prop lines because of the accuracy of the prop lines. In the end, our goal with this project was to develop an interactive user interface where a person can access our model and use this information to decide the most favorable prop bets.

The data sources that we used for our project came from two main sources. For statistics for every game played by every NBA player for the last three NBA seasons, we used https://www.basketball-reference.com/. This was an invaluable data source as it provided the models many useful features that included both basic and advanced NBA statistics. To evaluate our models, we compared our model's predictions to previous prop betting lines available at various sports betting books using https://www.bettingpros.com/nba/odds/player-props/. For the predictions of player statistics we ended up creating three separate artificial intelligence models that each predicted the three main statistics with prop lines available: points, rebounds, and assists. We created and tested each model separately and in the final user interface, gave users the option to pick which model they wanted to use for their predictions.

Our first model was a "reinforcement learning" type of model. We wanted to include one model that we built from scratch. This model used a feature-based representation with weights for each feature to represent the state. In this case, the features were a player's specific basic statistics (i.e. points, rebounds, assists, etc.) and advanced statistics (ie. true shooting percentage, usage rate, etc.) for a game, the opposing team's basic and advanced statistics in the form of an exponential moving average, and the inactive players for that player's team in that game. Each one of these features had a weight associated with it and using the dot product between the feature values and weights, we were able to predict a player's statistic for a future game. To predict one of points, rebounds, or assists, we made sure to exclude that statistic from the player's features and weights. For the following example of how we trained, validated, and tested our model assume we are predicting points, although the same process holds for rebounds and assists. To train our model, we started with all weights being 0. Then for each player we iterated over their first n - 20 games of the 2023 NBA season. We used a "Q-Learning" type of update system that calculated the difference from our predicted points for that game to the actual points scored by that player for that specific game. We then updated all of the weights using $w_i \leftarrow w_i + \alpha*\text{difference}*f_i(k)$ for game k. As specified earlier, the features f(k) for each game k included the opposing team's statistics in the form of an exponential moving average. During training, we also computed these features for each team for the first n - 20 games using $of_i \leftarrow of_i + \alpha f_i(k) - \alpha(of_i)$

where of_i is the opposing team feature we are computing and $f_i(k)$ is the opposing team statistic for feature i in game k. of_i of the specific team that the player is facing in game k is then used for the opposing team's features in the player's features. For the inactive player features, we simply had R number of features for each player, where R is the number of players on that specific player's team. Each one of these R features corresponded to a specific teammate. For each game, we set the feature corresponding to a teammate to β if that player did not play that game and 0 if he did. Lastly, during training we also computed an exponential moving average of the players features related to their own basic and advanced statistics, which would be used during prediction using $f_i \leftarrow f_i + \alpha f_i(k) - \alpha f_i$. After training on each NBA player, we learned each player's specific weights for the features. We then validated our training from the 20th to 11th to last games of an NBA player's season. Our training model had a total of four hyperparameters: α used in weight updating, α used in the player's own statistics exponential moving average, α used in the opposing team's feature exponential moving average, and the β we set the features to related to a game's inactive players. In validation, we randomly tested the performance of training on 100 randomly selected NBA players and used a randomized search technique to find the best tuple of four hyperparameters that minimized the squared sum of the difference between predicted and actual points scored for the 100 random players. Finally, we tested our model on the last 10 games of each player's 2023 NBA season using our model with the optimized hyperparameters.

Our second model was a support vector machine. While the reinforcement learning model involved a regression task, the support vector machine was for classification: specifically, if one had a model that accurately classified each instance into the correct point value that a player would score in that instance's game, it would be a valuable model for prop betting. On its own, a support vector machine is suited for a binary classification task, so multiple models must be manipulated in order to translate this to multiclass classification. Rather than training a model for each class to separate training examples that were in the class from those not in the class in a "one vs. rest" type of model, we trained an SVM for each half-point value (0.5, 1.5, …), where each svm's task was to classify training examples whose label point value went "over" or "under." This choice was made because of the reasoning that prop bet lines are predictions of a median outcome, and any model that comes up with a better median prediction would beat the prop lines in the long run. A "one vs. rest" type of SVM model would work if a probability was assigned to each class and a discrete probability distribution was then built from those class probabilities, but the main issue with this is that probabilities are not directly provided by a support vector machine, though there are computationally expensive techniques to compute a number representing a probability. Since we really only care about the median of this underlying distribution, by opting for the approach of training an SVM for each half point value, at inference time, we can step linearly among the SVM models in decreasing order, and the first SVM to classify a test point as part of the positive/"over" class is indicative of the underlying median being reached. This resulted in training 38 (for the points model) binary classification models (from 0.5 to 37.5) where the task was classifying into over or under the half point value inherent

to the model. We employed kernelization of our SVM in order to capture any non-linearities in our data that a linear support vector machine would not be able to account for. Specifically, we used the radial basis function kernel because of its high performance in general and the fact that it only has one parameter, gamma, which reduced the hyperparameter search space. In addition, we opted to use soft-margin SVMs, since they allow for some training misclassification in order to provide a more generalizable model at test time. Given that a player's output in any game is a random variable, we did not want to achieve high training performance as it would likely come at the cost of test performance. A large task for this support vector machine model was feature engineering, as our data did not come in a clean dataset with the features we wanted. In contrast to the reinforcement learning model where the player's game stats were simply pulled from the game log to use as features, for the support vector machine, we had to extract features from our web scraped data. The advantage of this feature engineering is that it provided a richer representation of the player: one such example is the inclusion of player priors, which were features containing descriptive data about the player prior to the season of the instance. While a description of all the features is not necessary because it is well-documented in the codebase, each of the 100+ features belongs to one of four categories: player priors, season data from the player, recent form of the player, and data specific to the instance, such as opposing team quality. A drawback of the SVM was that it took a long time to train: about an hour for each model, for 1000 training points. For this reason, only data from the 2022 entire season was used for training, while the test set was the last 10 games of the 2023 season of each player, which is by design exactly the same as the test set of the reinforcement learning model. This left the rest of the 2023 season as a validation set. The training process involved a hyperparameter grid search for gamma and C, the two hyperparameters of the kernelized soft-margin SVM. In validation for each model, the geometric mean of the F1-score (the harmonic mean of recall and precision) of both classes was calculated in order to find which pair of hyperparameters yielded the best score, which turned out to be C=1.0 and gamma=0.01, for all stat types. We kept this consistent for all of the sub-models such that a lower classifier wouldn't predict a positive class while a higher one predicts the negative class. During test time, we used these models that were trained on these hyperparameters and about 10,000 instances. As for actual training, we formulated the dual form of the soft margin SVM, and along with a computed kernel matrix on the training examples, formulated the corresponding quadratic programming convex optimization problem. We leveraged the cvxopt package to solve this optimization problem and retrieve the alpha values for each training instance, which were the main values needed at test time.

Our third model was a Random Forest Regressor. A Random Forest is an ensemble learning method that constructs many decision trees at training time and outputs the average prediction of the individual trees. This model is particularly effective as it mitigates overfitting, a common pitfall of decision trees, while maintaining high accuracy. Our model utilized player-specific basic and advanced statistics, such as points, rebounds, assists, actual shooting percentage, and usage rate, among others, as features. These were combined with the opposing team's statistics in the form of an exponential moving average and data about the inactive players

for that player’s team in a given game. Based on these features, the goal was to predict a player's statistics for a future function. We used a custom function, 'convert_to_valid_value,' to ensure all feature values were valid and compatible with our model. This function handled edge cases and transformed all feature values into a float data type. Our Random Forest model's training, validation, and testing process were encapsulated in the 'run_model' function. We employed the 'train_test_split' function to partition our dataset into training and testing sets, ensuring we had unseen data to evaluate our model's performance. To ensure uniformity and fairness in model training, we standardized our features using the 'StandardScaler' from the sklearn—preprocessing module. We used the 'GridSearchCV' method to optimize our model, which performed an exhaustive search over specified parameter values for the estimator. This helped us determine the best combination of hyperparameters, such as the number of trees in the forest (n_estimators), the maximum depth of the trees (max_depth), the minimum number of samples required to split an internal node (min_samples_split), and the minimum number of pieces needed to be at a leaf node (min_samples_leaf).

We trained our model using the training data and the best hyperparameters identified through grid search. The performance of our model was then evaluated on the testing data. We used three key metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and R2 Score. These metrics provided us with a comprehensive understanding of our model's accuracy and predictive power. After training, we saved our model using the joblib library for future use, ensuring our efforts could be utilized for future predictions without retraining. We also logged the feature importances, which gave us an insight into the features that most significantly influenced the model's predictions. This model was a comprehensive implementation of the Random Forest Regressor, tailored to predict NBA player statistics. In addition, it demonstrated the strength of ensemble learning methods and their ability to handle complex, multi-dimensional data.

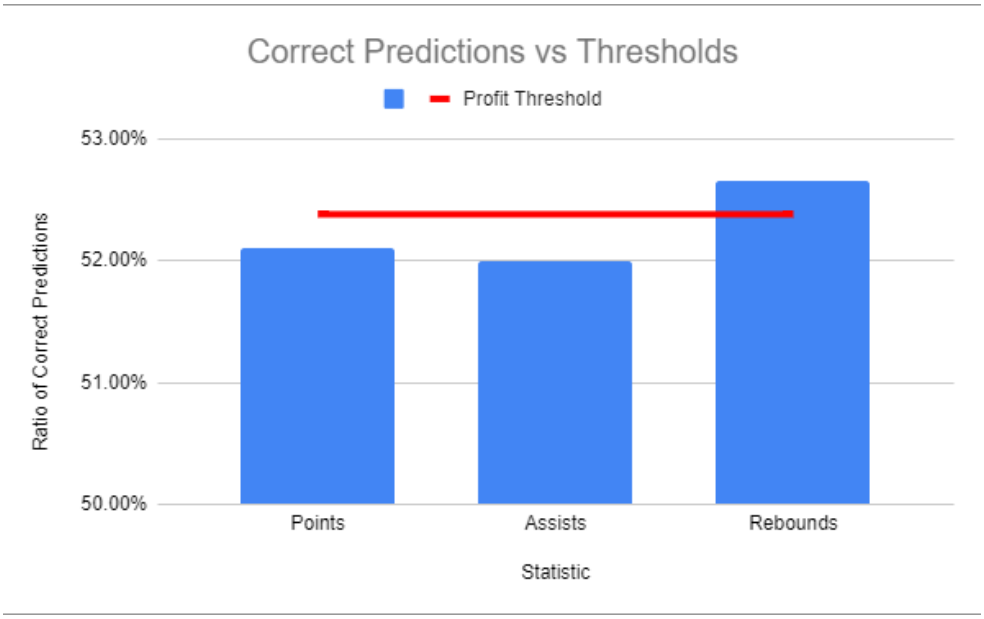
At the end, we created an interactive user interface. This GUI gave users the option to enter any NBA player and pick any of the three statistics: points, assists, and/or rebounds as well as any of the three models. The GUI then outputs the predicted, actual, and prop lines for the up to three statistics chosen in the last ten games played by that player in the 2023 NBA season. It displays the dates of each of these ten games and whether or not these predictions ended up winning or losing the bet for the model selected.

Evaluation:

As stated earlier, we separately created and evaluated each one of our three models. We also evaluated our final user interface by asking several outside students about the accessibility and ease of use of our GUI. The evaluation of our three models was largely quantitative, whereas the evaluation of our interface was mostly opinion-based by outside students.

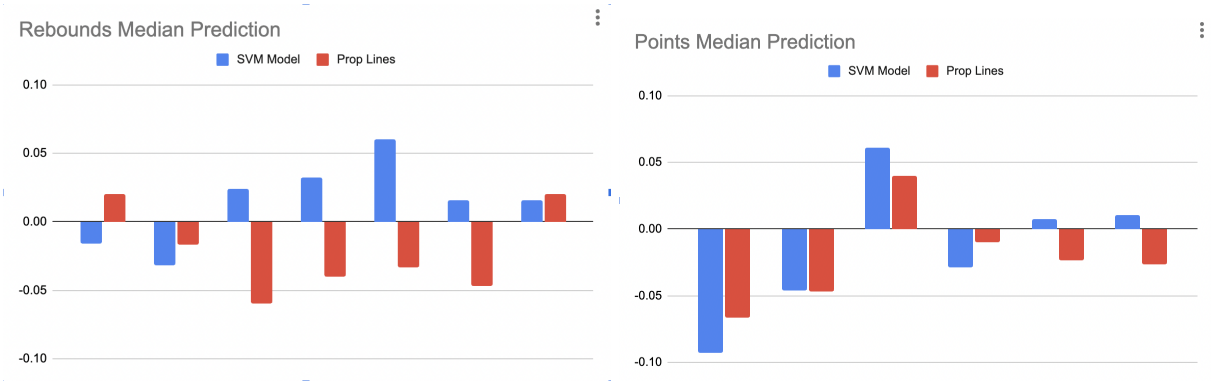
For evaluation of our reinforcement learning model, we used several different metrics. We evaluated the last ten games of the 2023 NBA season for all NBA players and computed how much on average each player differed in points, rebounds, and assists per game from our

predicted output. We also computed the percentage of games that our model correctly predicted over or under the corresponding prop line for that game and statistic. This is our most important metric as this is primarily what was set out for at the beginning of our project since we wanted to analyze prop lines themselves. In the end, the average difference in points per game was 2.906, the average difference in assists per game was 0.845, and the average difference in rebounds per game was 1.259. We believe that this is pretty promising because it shows that our model was relatively close to the real statistic numbers produced by players. There is a lot of variance in game to game statistics for NBA players, so the fact that we were able to keep the average difference for rebounds and assists to around 1 per game and points to 3 per game, was a success. However, this does not tell us much with respect to our model's performance, because our main goal was to find out how well our model could predict prop bets. So, In terms of our prop line metric, the model correctly predicted 52.10% of point prop lines, 52.00% of assist prop lines, and 52.65% of rebound prop lines. At first glance these numbers may seem like a huge success because if we get over half of the prop bets picked correctly, our model should be able to profit in the long-run. However, it is important to note that on average most prop bets are given at -110 odds. This means that in order to profit $10 on a bet, you must wager $11. This is how sportsbooks make their money since they get to basically keep $1 for every $11 bet if there is the same amount of money bet on both sides of the prop line. This means that in order to actually profit, a bettor must get at least 52.38% of their bets correct. Therefore, as shown in the figure below, our rebounding predictions barely exceed this number and our points and assists predictions barely miss profitability. Nevertheless, we still believe that these numbers are a huge success because it is virtually impossible to profit off of prop betting or else sportsbooks would be driven out of business. The fact that this model got so close to the threshold of profit for two of the statistics, and even exceeded it for rebounds is very promising. The model was substantially better than a 50/50 chance of getting the prediction correct and if it weren't for the sportsbooks "house tax" described earlier, our model would be extremely profitable.



For evaluation of our SVM model, we found a much wider range of results than for the reinforcement learning model. Of course, the most important metric in our evaluation was the models' performance against the prop lines on the test set. Specifically. a higher or lower model output than the given prop line was simulated as an over or under "bet," respectively, and the

outcome determined whether the bet was winning or losing. After totaling up these simulated bets on about 1800 instances for each stat type (points, rebounds, assists), the points SVM model classified 48.1% correctly (i.e won 48.1% of its bets), while the assists SVM model classified 55.02% correctly, and the rebounds model 52.38% correctly. While the points model had a disappointing outcome, on the whole, this was a very good outcome of results: 2 of the 3 models achieved the aforementioned baseline for long-term profitability over the sportsbooks, even with the typical implicit cut the sportsbooks take with their odds. From another perspective, if one were randomly guessing over or under, they would have a 0.001% chance to do as well or better on the assists test set, and a 2.15% chance to do as well or better on the rebounds test set, as defined by a calculation of the upper cumulative distribution function of a binomial distribution with the given parameters. Over all the stat types including the poor-performing points model, a 51.83% accuracy was achieved on 5491 samples, which has a likelihood under random guessing of 0.3%. While this quantitative analysis is the most important for evaluation on our project, a deeper property of these models inherent to their intended task can be explored. Ultimately the SVM model's main task was to output a median predicted outcome. On the surface, it seems as this has been achieved even for the poor performing points model: of all of the predictions for which the outcome was not equal to the prediction, 51.37% outcomes were over the prediction, while 48.6% were under, nearly an even split. However, while this even split property is implied by a good predictive median model, it does not imply a good model on its own. As an example, one could predict the historical median every time, say 12 for points, and see an even split at test time. A stronger indicator of a good median predictor would be if this property held at a finer granularity of each prediction value. In other words, if the equal split property held when the prediction is 1, as well as when the prediction is 2, and so on, this would be stronger evidence of a good predictor. We can analyze this from the results of the test set, though the limited data means analysis at values with low counts of predictions will be subject to low sample sizes. Because of this limitation, rather than doing this analysis at each value, we do it at 6 ordered bins of values, where each bin contains an equal number of predictions. Below, we plot the rate at which outcomes went over the predicted value, beyond or below 50%, for both the model prediction and the actual prop line, since it represents a median as well.



Above, we can contrast the plots for the points model, which performed poorly, and the rebounds model, which performed well. The objective is to have the tightest margins possible to 0, which

represents that half the outcomes went over and half went under. The SVM models are shown in blue. From inspection, it appears that the rebounds SVM model has a consistent tighter bound on this metric, and this is the case: in the aggregate, the SVM rebounds model had a 0.18 sum (while the corresponding prop lines had a 0.21 sum), while the SVM points model had a 0.24 sum (while the corresponding prop lines had a 0.21 sum again). From this analysis, it is not surprising that the rebounds model performed better against the prop lines than the points model: it performed better as a median predictor. Despite these seeming successes with the models, it should be noted that these results do not imply profitability in the prop betting market. Because rebounds and assists are distributed on a much smaller scale than points, odds are often more "juiced" than an odds offering -110, (odds of -130 and -140 are common) as sportsbooks must stick to a half-point value even though there is a lean in one direction or another, though not large enough to move the line a whole point. This means that these points and rebounds models' results are not as impressive as if a points model were to have that classification accuracy, as the minimum hit rate for profitability to overcome the vig is on average higher for rebounds and assists. Not only does one have to beat these oddsmakers, but one would have to beat them by a significant margin defined by the vig. In addition, our models cannot take into account the human element that the sportsbooks will adjust for when these line are offered, such as a coach telling the media that a player has a minutes restriction in an upcoming game due to returning from injury, or a player who typically scores 25 points going into a game for which he needs 30 points to break a record. There are many of these instances which makes the likelihood of long-term profitability using an artificial intelligence model even more impossible. All in all, I would evaluate the SVM's performance as a moderate success.

To evaluate our GUI, we asked two other Cornell students, Mario Xerri and Puxuan Yang, to use our interface model after giving them background information about what a prop bet is and what information the interface is displaying. Both students described the interface as visually appealing at first glance and liked the way that the buttons looked with the NBA logo in the background. However, both students also stated that after inputting the parameters into the interface and seeing the displayed results, the interface seemed a little cluttered. One student mentioned that a table may have been a better way to display the data. The other student said that the information took a bit of concentration to comprehend when first seeing it. They also did have some good things to say about the displayed data such as their appreciation for the scrollbar and the small pictures that helped them easily see whether our model predicted over or under and whether its prediction was correct. Overall, they said that although the data may have looked a little confusing at first, once they actually took some time to look into it, they were able to understand the data. We also asked the students about the ease of use of the interface. They both commented that they were able to figure out how to use the interface quickly because of the short instructions at the top as well as the messages they received upon an invalid input. We believe that this GUI was a success because it was able to convey the data that we wanted to present and although the students may have thought that it was cluttered at first, they both were able to comprehend the data shortly thereafter. Most importantly, we wanted to create an interface that

would be easy to use and both students were able to quickly figure out how to use the interface and interpret the data it presented.

Overall our goal was to provide a resource that would be valuable to bettors in their decisions on prop lines. We believe that we achieved this goal: in the quantitative analysis of our models above, we found the reinforcement learning and SVM models to be much better than a random guess at a highly significant level. Although we mentioned that it is unlikely these models would be profitable on their own mainly due to the sportsbook house tax, a bettor that is already betting on prop lines could use our models in aggregate to make an informed decision, along with their evaluation of the human factors that our models can't take into account, before placing a wager. With our GUI, we demonstrated the ease in which this could happen: a bettor would simply have to type in the name of the player and the stat type and model type to get these valuable predictions. Our original goal of the project was for a user to have the capability to enter our interface and be able to see prop bets and our model's predictions for the current day NBA games. These predictions could be listed in order of confidence and give users accurate predictions for what they prop bets they should consider betting on that day. In a setting in which the NBA regular season was still ongoing, this would be possible with our models as our models use past game data to predict each game's statistics. However, since the NBA regular season is over, there are no more prop betting lines available and therefore we simulate this by showing what this theoretical user interface could look like for a specific player for their last 10 games on this past regular season. In addition to this theoretical interface, we therefore were able to include the results of the games as well since these games already happened. Consequently, we believe that the combination of our models and interface give a good representation of the practical use of our project in the real world.

References:

NBA Statistics Data: https://www.basketball-reference.com/
NBA Prop Bet Data: https://www.bettingpros.com/nba/odds/player-props/
NBA Player Prior Data: https://projects.fivethirtyeight.com/2023-nba-player-projections/
Software Resources: PyQt5 for GUI, sklearn.preprocessing for scaling the data, cvxopt for solving quadratic programming problems for the SVM, selenium and BeautifulSoup to data scrape prop bet and NBA statistic data