NBA Game Score Difference Prediction

Paul Jacob
Springboard Data Science Career Track

Date: September 1, 2022

Executive Summary

In this project, we aimed to build a good NBA game score difference prediction model. This was done using a random forest regressor for the prediction of NBA game score difference. The usefulness of our model was evaluated using a learned betting policy. Our model yielded a 70% betting accuracy and 45% ROI in the 2017-18 season. Betting on a bootstrapped sampling of our 2017-18 season sportsbook games suggested that 90% of the time, we expected a ≥22% ROI.

Introduction

NBA sportsbook bettors use a variety of strategies in attempts to turn a profit, e.g. subjective handicapping and finding microholds, 0% holds, or negative holds in synthetic markets. The purpose of our NBA game score difference prediction model is to offer the data point of a good model. This way, NBA sportsbook bettors can supplement bet techniques and other data they collect before choosing to place a bet.

The NBA is the top basketball league by betting volume with spread bets as its most popular bet type. A spread bet requires prediction of the margin one team beats another team by–called the score difference.

In this report, we take you through the process of developing our game score difference prediction model [1] and the evaluation of the goodness of our model.

Definition: synthesizing a 0% hold can be accomplished by buying spread prices at two different sportsbooks to give a fair zero cut to the bookie bet.
Definition: subjective handicapping is assigning an advantage or disadvantage by score compensation.

Data Collection

In our approach, we collected NBA game data at both the team and player level: team box scores, team advanced box scores, player advanced box scores, player injury reports, player inactive lists, and team city coordinates. Also, we collected sportsbook betting data to evaluate our model predictions. Sources collected from include Kaggle.com, nba_api, and Wikipedia.

Data Wrangling

From this, we filtered for the 2010-11 thru 2017-18 seasons, as well as cleaned data at both the team and player level.

Exploratory Data Analysis and Feature Engineering

Our predicted variable was the score difference. As shown in Figure 1, the score difference distribution was bimodal. An NBA game was either won or lost such that no game resulted in a tie.

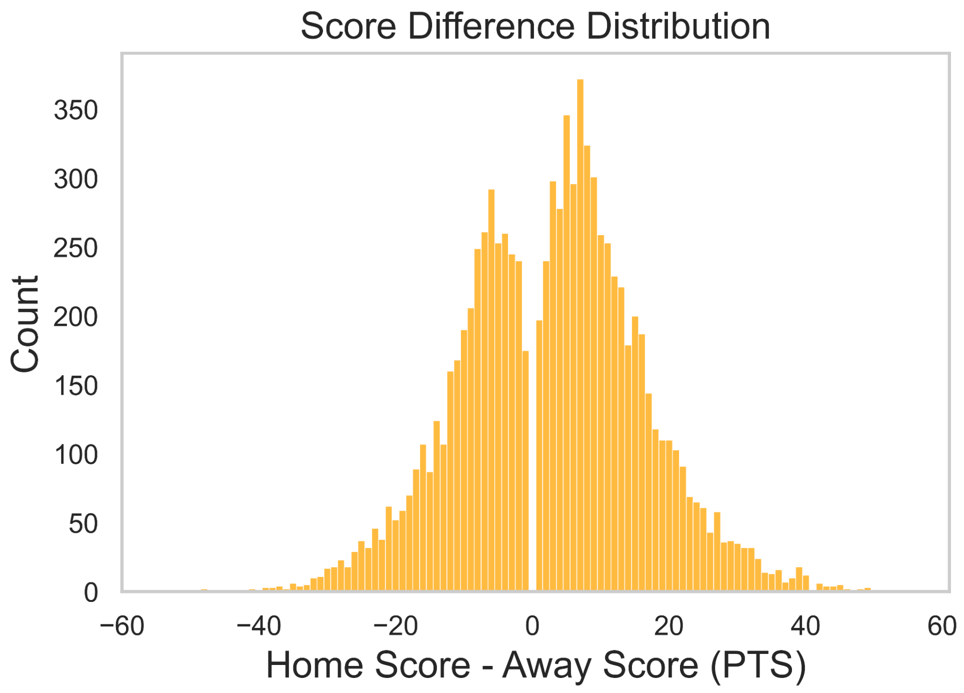


Figure 1. The score difference distribution for seasons 2010-2011 to 2017-2018.

Definition: Score difference is the home team score minus the away team score.

Definition: ‘score_difference_a’ is the game point difference between both teams ordered alphabetically by team abbreviation, e.g. an NBA game result of BOS 90 and MIA 98 has a ‘spread_a’ of -8.

In our model, we experimented with hundreds of features for NBA game score difference prediction. *Here, we mentioned the features of greatest predictive importance (as dictated by the random forest regressor Gini importance).*

We grouped team difference features into four major categories which included: box score, advanced box score, lost contribution, and schedule. The **box score** feature category included stats such as team points, 3-point field goal attempts, personal fouls, and field goal percentage. The **advanced box score** feature category included stats such as team net rating, player impact estimate, effective field goal percentage, and true shooting percentage. The **lost contribution** feature category included instances where a player did not play. The **schedule** feature category included team strength of schedule and travel distance.

Box Score and Advanced Box Score Feature Categories

The simplest feature engineering was for the team box score stat and team advanced box score stat features. This process is shown in Figure 2.

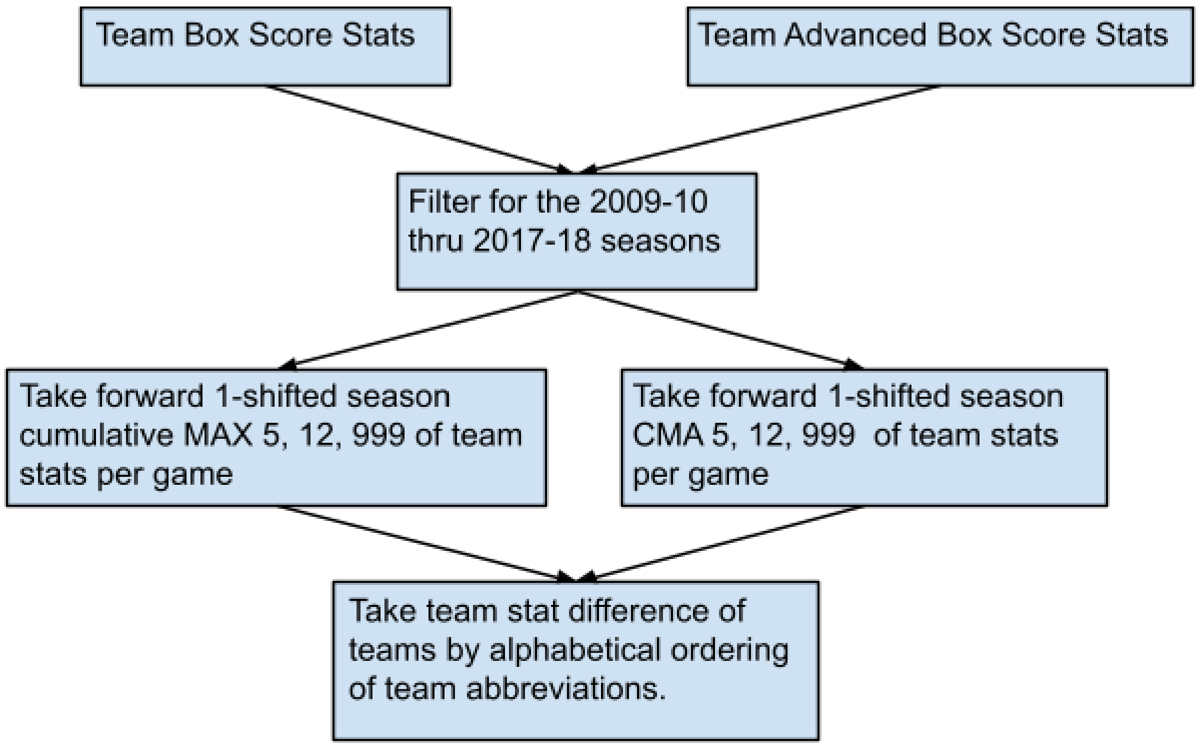

Team Box Score Stats
Team Advanced Box Score Stats
Filter for the 2009-10 thru 2017-18 seasons
Take forward 1-shifted season cumulative MAX 5, 12, 999 of team stats per game
Take forward 1-shifted season CMA 5, 12, 999 of team stats per game
Take team stat difference of teams by alphabetical ordering of team abbreviations.

Figure 2. The team box score and advanced box score stat feature engineering diagram.

For example, for the team advanced box score stat net rating, we have the feature ‘net_rating_cma999_diff'. This feature is the difference of team net ratings before the current game, where team net rating is the number of points scored per 100 possessions minus the number of points allowed per 100 possessions. The relationship between ‘net_rating_cma999_diff’ and ‘score_difference_a’ can be seen in Figure 3.

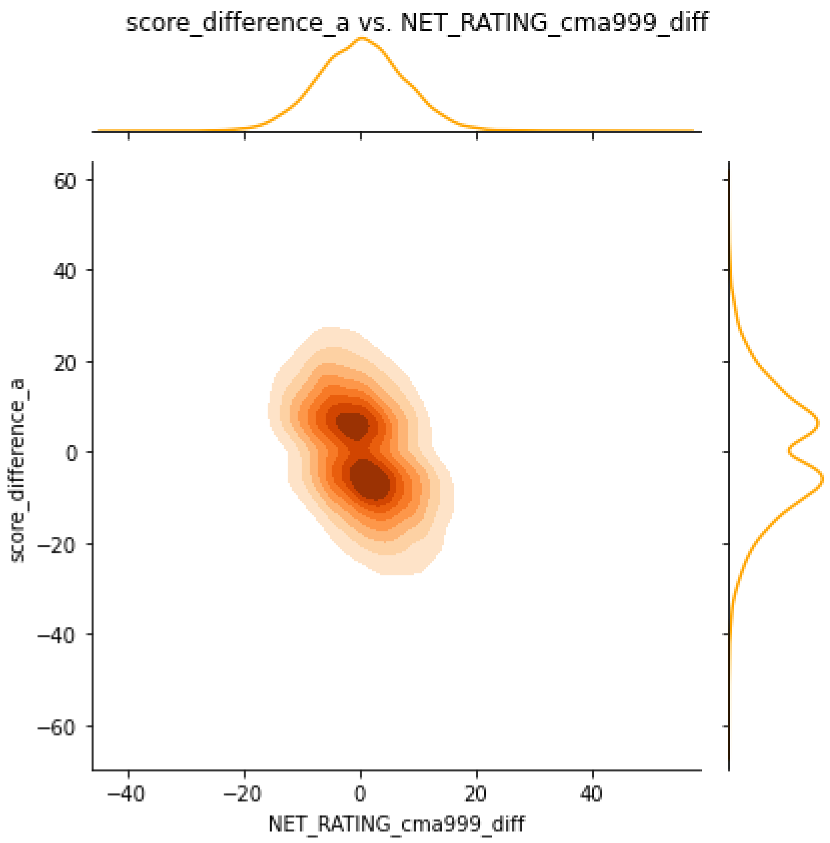


Figure 3. The ‘score_difference_a’ vs. ‘net_rating_cma999_diff’ kernel density estimate (KDE) plot.

Lost Contribution Feature Category

A lost contribution is when a player did not play and can be viewed as a disruption in team chemistry.

Using the injury list (where players are listed as relinquished or acquired) in conjunction with the NBA per game player advanced box score stats and the per game player inactive data, a player missing a game due to injury who performed well in one of the advanced stats had predictive importance (i.e. random forest regressor Gini importance) in the outcome of games. These were lost contributions and an example of this player lost contribution for the effective field goal percentage advanced box score stat is shown in Figure 4.

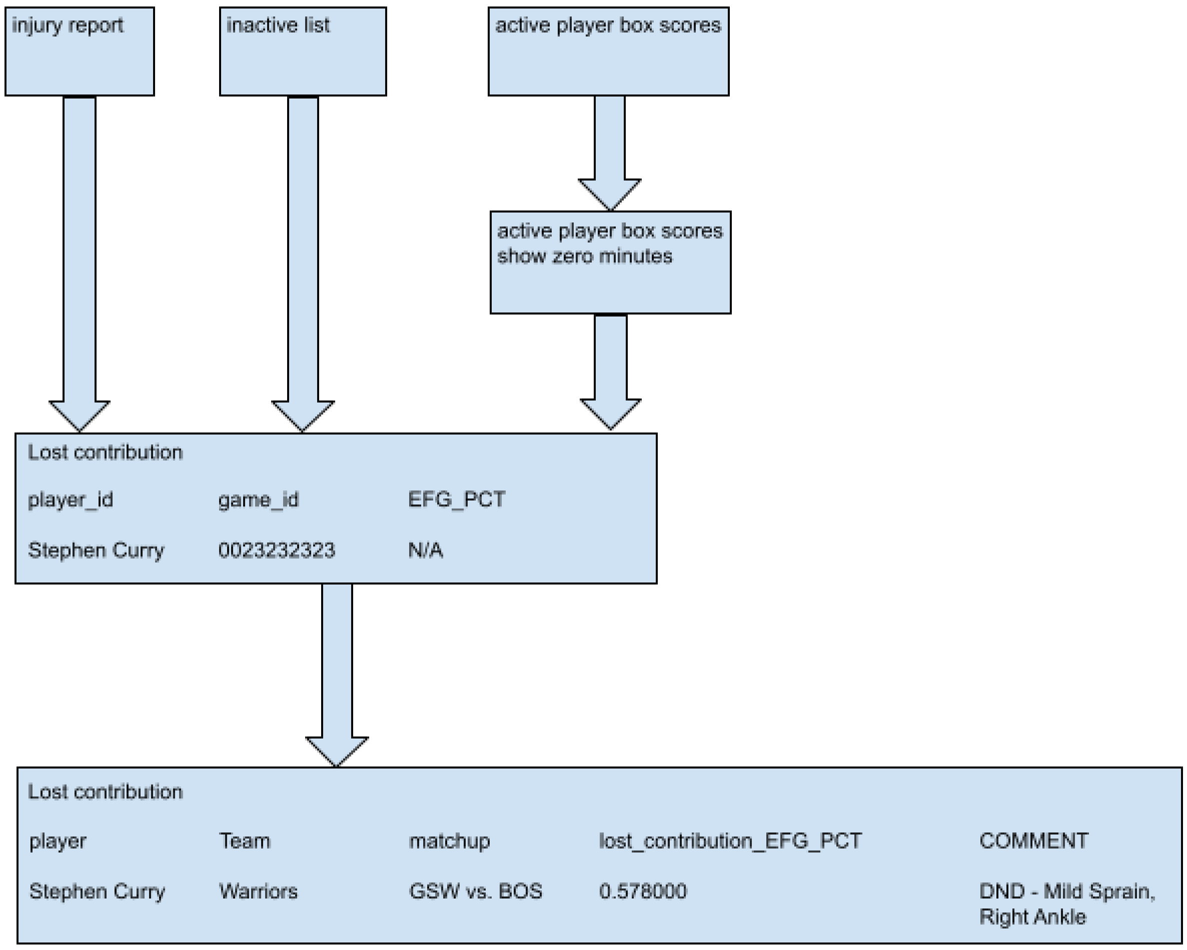


Figure 4. Player lost contribution diagram calculation for the example of Stephen Curry. Here, Curry has a lost contribution of 0.578 for the stat of effective field goal percentage for a game between the Golden State Warriors and the Boston Celtics.

Following the extraction of per game player lost contribution, we wanted the per game team lost contribution. The feature 'lc_sum_NET_RATING_cmax999_diff' was the team max sum of player net ratings of players shown with zero minutes, as inactive, or as listed relinquished on the injury list per game for all games that season before the current game.

A programming perspective of our team player lost contribution feature engineering was the following:

1. Take the CMA per each player box score advanced stat for NBA games the player played and was not injury reported in for seasons 2009-10 thru 2017-18
2. This CMA is used to fill forward to games the player did not play in and is named their lost contribution.

3. Take the per game SUM, MEAN, and MAX of each team's players' box score advanced stat CMA from this process to get lost contribution per game, e.g. 'lc_sum_NET_RATING' and 'lc_mean_EFG_PCT'.
4. Add missing teams with no lost contribution per game and fill NaN value lost contribution box score advanced stats with 0.
5. Take the 1-shifted cumulative max 5, 12, 999 per team season for each box score advanced stat CMA forward filled lost contribution.
6. Following the calculation of team lost contribution, we then take the difference by team abbreviation alphabetical ordering.

Of our features calculated, the lost contribution features included the most steps. An intuitive description from a programming perspective is shown in Figure 5.

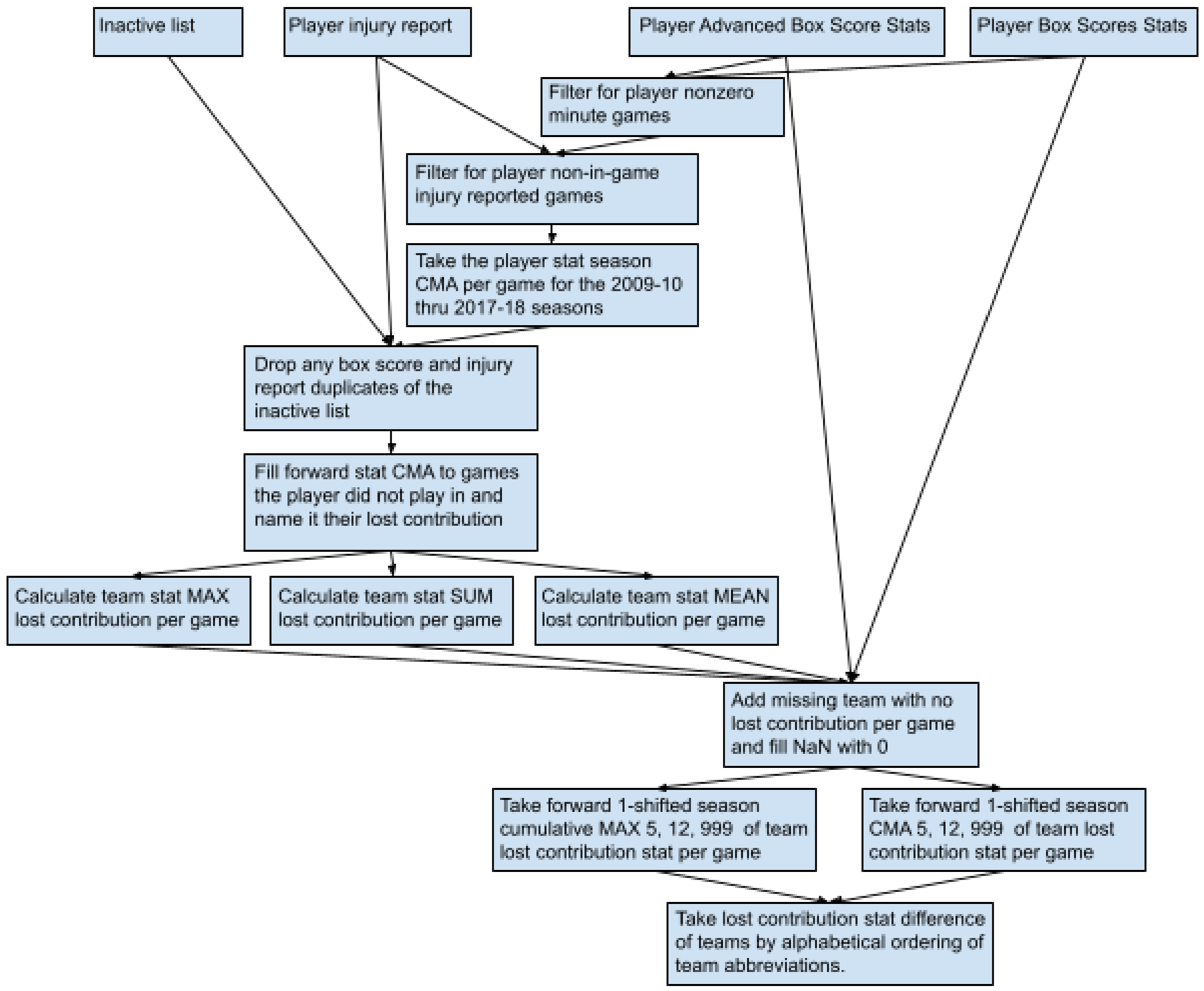

Inactive list
Player injury report
Player Advanced Box Score Stats
Player Box Scores Stats
Filter for player nonzero minute games
Filter for player non-in-game injury reported games
Take the player stat season CMA per game for the 2009-10 thru 2017-18 seasons
Drop any box score and injury report duplicates of the inactive list
Fill forward stat CMA to games the player did not play in and name it their lost contribution
Calculate team stat MAX lost contribution per game
Calculate team stat SUM lost contribution per game
Calculate team stat MEAN lost contribution per game
Add missing team with no lost contribution per game and fill NaN with 0
Take forward 1-shifted season cumulative MAX 5, 12, 999 of team lost contribution stat per game
Take forward 1-shifted season CMA 5, 12, 999 of team lost contribution stat per game
Take lost contribution stat difference of teams by alphabetical ordering of team abbreviations.

Figure 5. The team lost contribution difference calculation diagram.

In ranking the lost contribution features, we looked to the top 20 features by random forest mean decrease in impurity (MDI). Seven of these top features were lost contribution features and their ordering by random forest Gini importance are shown in Figure 6.

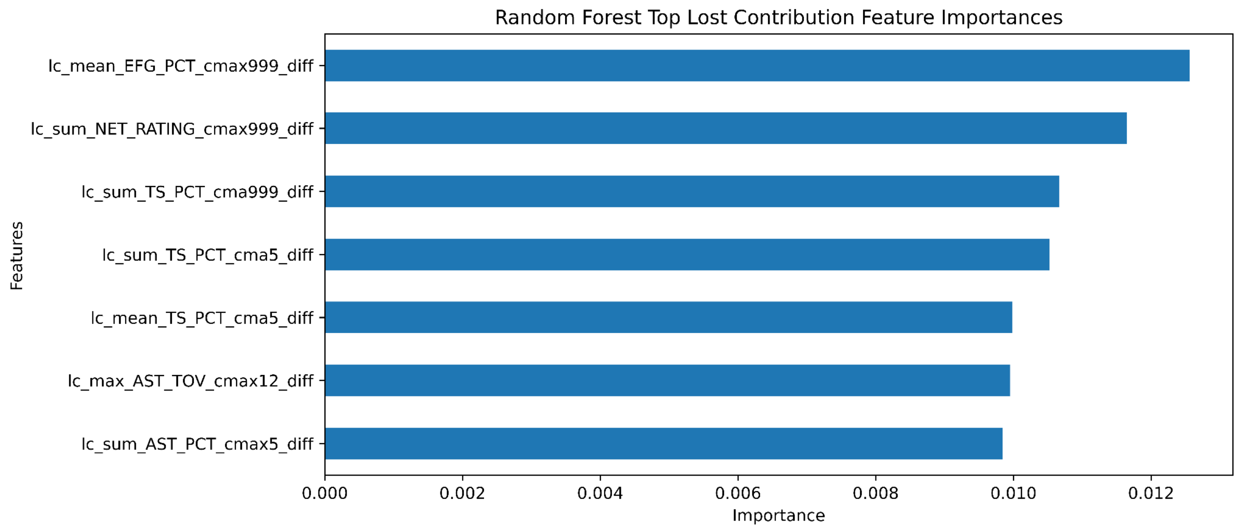


Figure 6. Lost contribution features ordered by random forest Gini importance.

Our top lost contribution feature by Gini importance was 'lc_mean_EFG_PCT_cmax999_diff' and was a disruption in the scoring task chemistry. This feature is plotted against the target variable 'score_difference_a' in Figure 7.

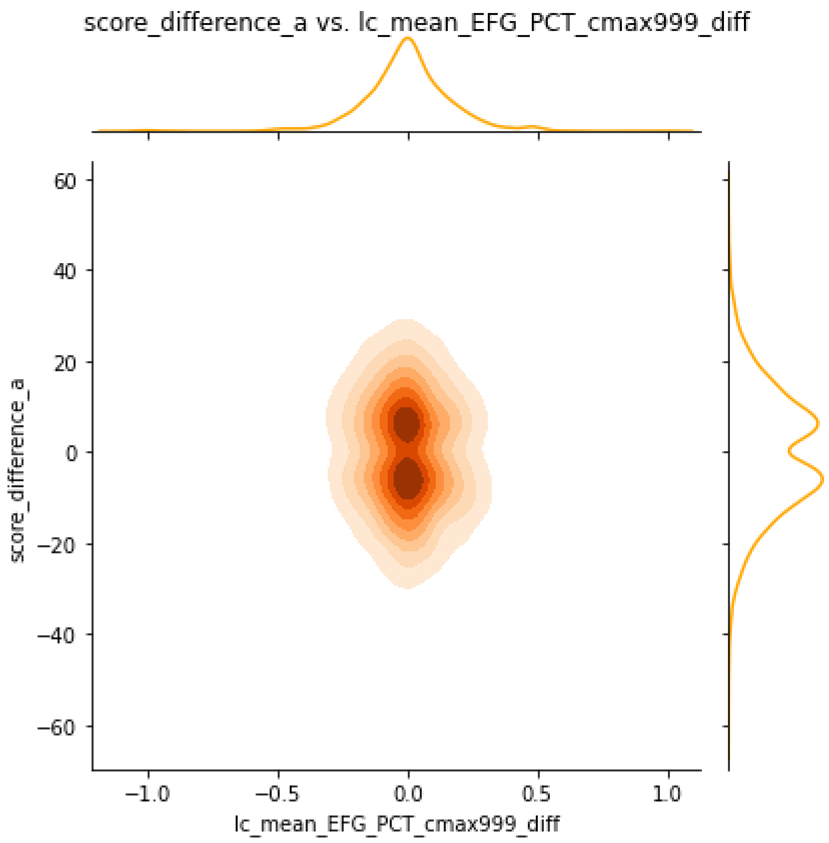


Figure 7. The kernel density estimate (KDE) plot of ‘score_difference_a’ vs. ‘lc_mean_EFG_PCT_cmax999_diff’.

Of the lost contribution features, ‘lc_sum_NET_RATING_cmax999_diff’ was ranked the second most important for score difference prediction in the random forest regressor. Its KDE plot with score difference is shown in Figure 8.

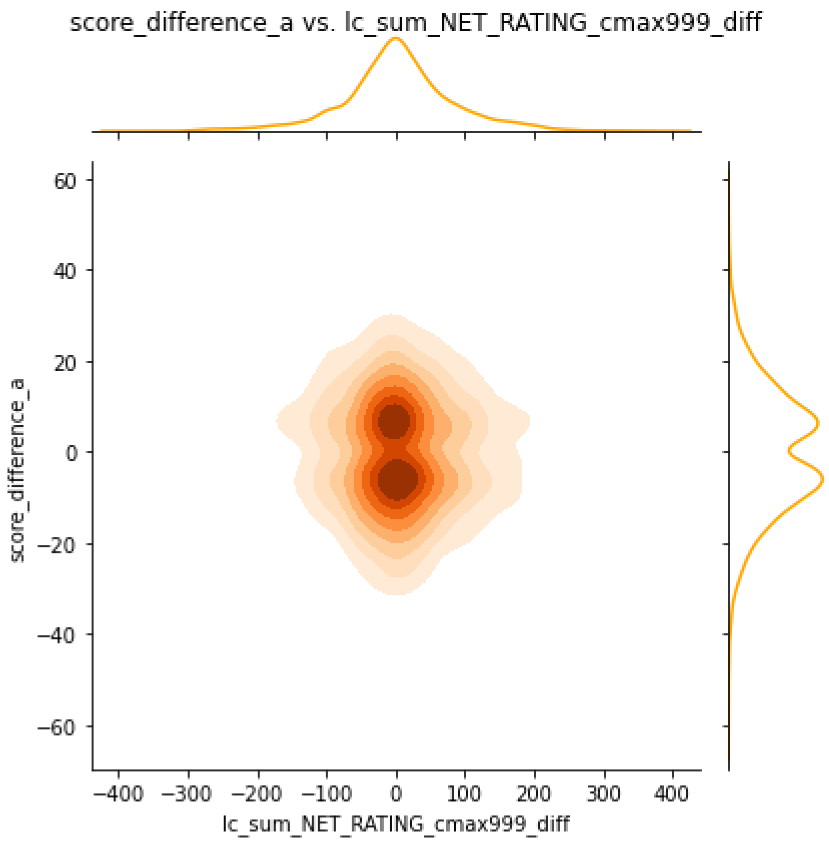


Figure 8. Kernel density estimate (KDE) plot of ‘score_difference_a’ vs. ‘lc_sum_NET_RATING_cmax999_diff’.

Schedule Feature Category

The strength of schedule is how difficult or easy a team’s opponents compare to another team’s opponents.

Another schedule feature was the travel distance difference. The feature ‘travel_distance_diff’ was the team difference in distance traveled from the team city of origin to the game city ordered by team abbreviation, e.g. GSW 0 and CAV 2154 is 2154.

Definition: Here, strength of schedule is (2 * (Opponent’s Offensive Rating) + Opponents’ Opponents Offensive Rating) / 3.

Travel distance difference and its relationship with score difference is shown in Figure 9.

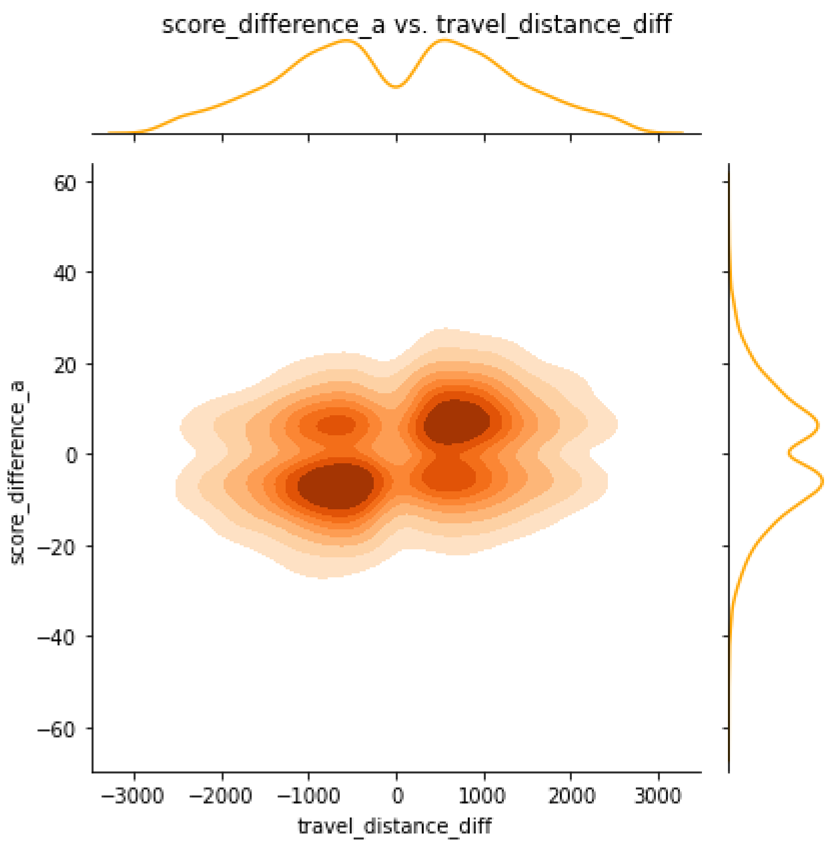


Figure 9. The ‘score_difference_a’ vs. ‘travel_distance_diff’ kernel density estimate (KDE) plot.

Modeling

Random Forest Regressor Modeling Summary

Following the creation of some features, we developed a random forest regressor. This was done by first selecting hyperparameter values for ‘criterion’, ‘bootstrap’, and ‘max_features’. Next, we performed ‘RandomForestRegressor’ hyperparameter tuning using different ‘max_depth’, ‘min_samples_splits’, and ‘n_estimators’ value combinations with ‘GridSearchCV’. Following, we built a ‘RandomForestRegressor’ with all the train data and selected the top features. If more features remained to be used in a ‘RandomForestRegressor’, we repeated this process and again got the top features. Combining top features of separate ‘RandomForestRegresssor’s, we dropped duplicate-like features and built another ‘RandomForestRegressor’ until all top features were used in our time series ‘RandomForestRegressor’ collection.

Random Forest Regressor Feature Selection

Before running models, we selected some features for training random forest regressors. Here, preference was given to smaller datasets to keep random forest regressor training runtimes low.

Random Forest Regressor Hyperparameter Selection

Before hyperparameter tuning, we selected parameters that were empirically strong for a random forest regressor.

One was bootstrapping our train data in building each decision tree.

The second was using all of the train data features for building each decision tree. Both of these hyperparameters are the ‘RandomForestRegressor’ default values in the scikit-learn library [25].

Lastly, we set the criterion equal to ‘absolute_error’ [26]. Using a mean absolute error [28] was suitable because we wanted to reduce this metric in our random forest prediction. This way, decision tree splits were taken by a max standard deviation reduction where the standard deviation is the impurity measured by the mean absolute error of score difference of each node split on a feature [26][27].

Random Forest Regressor Hyperparameter Tuning

Of the scikit-learn library, ‘GridSearchCV’ was used to tune our ‘RandomForestRegressor’ hyperparameters. This random forest regressor grid search 5-fold cross-validation was performed using hyperparameters ‘n_estimators' with array value 200; ‘max_depth' with array values 5, 10, 20, 1000; and 'min_samples_split' with array values 50, 100, 1000. Here, max depth and minimum number of samples to make a split (on a decision tree) regulate the height for each of the 200 decision tree estimators. Though this 5-fold cross-validation introduced data leakage, it was shown to be effective in selecting good best hyperparameters.

We also performed time series cross-validation random forest regressor grid search hyperparameter tuning. For our final random forest regressor, we took the best mean coefficient of determination R^2 score of random forest regressors and got 10, 100, 200 for max_depth, min_samples_splits, and n_estimators respectively. Note: comparing

‘RandomForestRegressor’ hyperparameter performance by MAE instead of R^2 would be preferred here.

Random Forest Regressor Top Feature Selection

Keeping runtimes to reasonable levels is good. To this end, we selected the top 50 random forest regressor features by feature importance for use in a following random forest regressor. This feature selection was done using the scikit-learn ‘RandomForestRegressor’ built-in mean decrease in impurity (MDI), i.e. Gini impurity.

By collecting the top features, we filtered out for less important features and just kept those with the most predictive power.

These filtered larger datasets to smaller datasets was done because in combining just the top features, we reduced following RandomForestRegressor training runtimes.

Random Forest Regressor Combine Features

We now combined the top features of datasets of these two or more ‘RandomForestRegressor’s. In our final time series cross-validation random forest regressor collection, this led us to combine sub-datasets to make 100 features to learn from.

Random Forest Regressor Feature Elimination

Of these 100 features, there were some very high feature correlations. Because some of our top features behaved very similarly and in some cases were just inverses of each other, these duplicate-like features were dropped. Here, **strength_of_schedule_diff**, **win_pct_diff**, and **opp_win_pct_diff** are all highly correlated, with the latter two as inverses of each other. We just kept **strength_of_schedule_diff**.

Similarly, **NET_RATING_cma999_diff**, **score_difference_cma999_diff**, and **E_NET_RATING_cma999_diff** are all highly correlated, with the first two as inverses of each other. We just kept **NET_RATING_cma999_diff**.

After high correlations were removed our top 100 features became our top 88. Our final random forest regressor top 10 features by Gini impurity are shown in Figure 10.

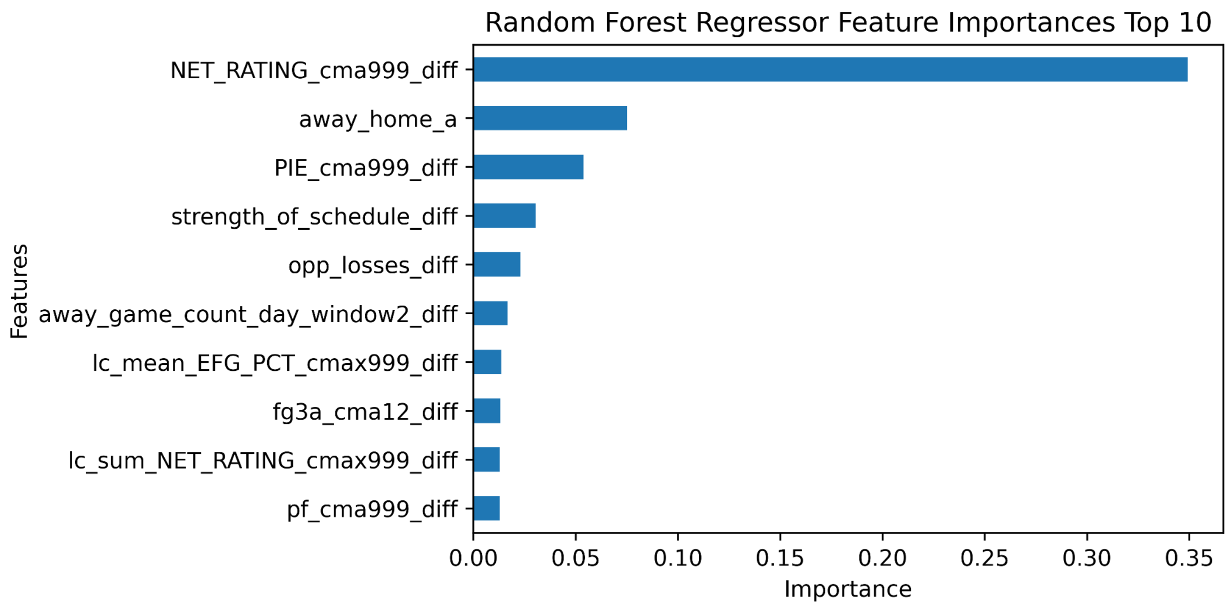


Figure 10. The random forest regressor top 10 features by Gini importance after eliminating very high correlations.

Checking the Pearson correlation coefficients among our top 10 features, we see no feature pair exceeded a correlation magnitude greater than 0.91 as shown in Figure 11.

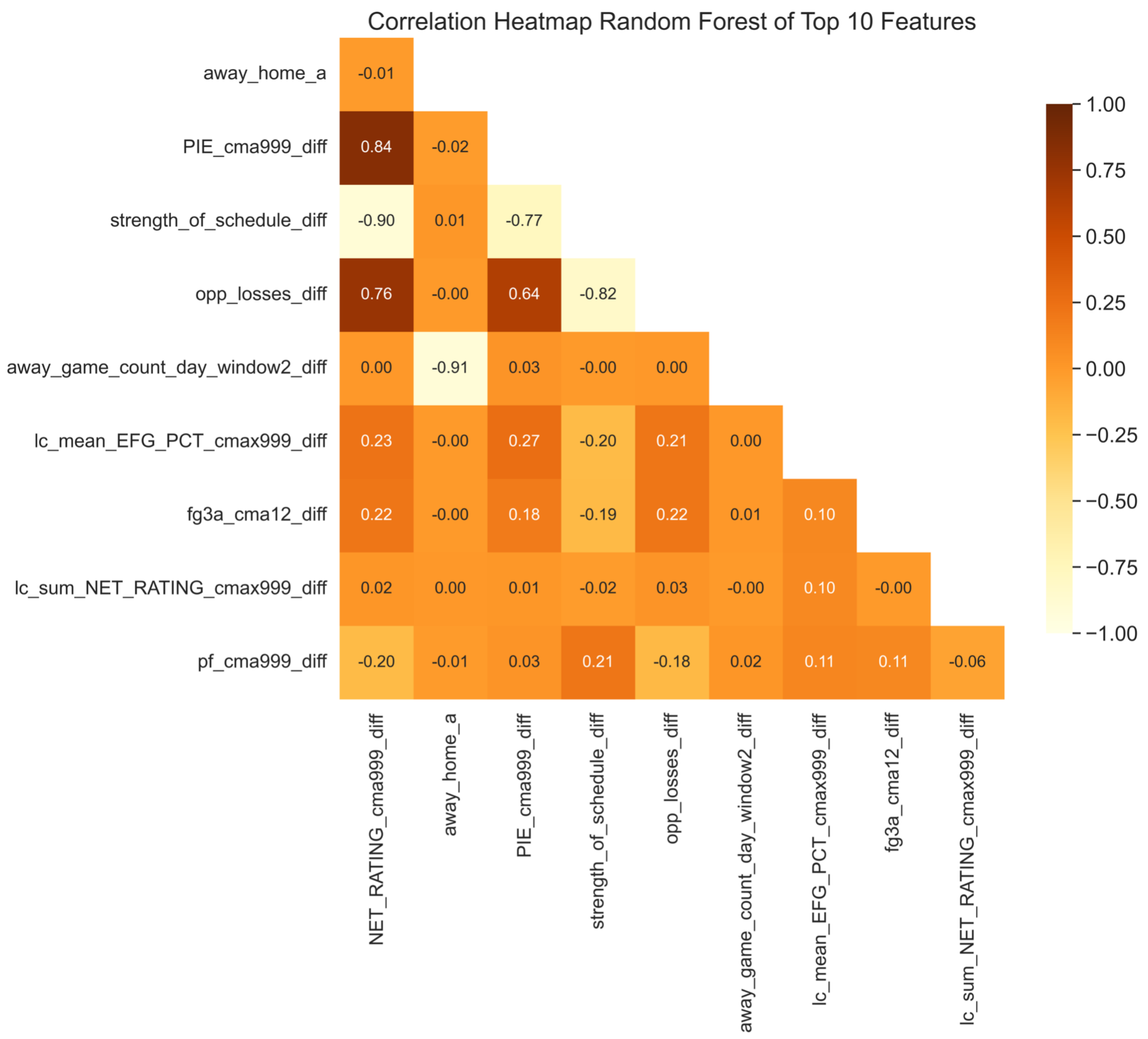


Figure 11. The Pearson correlation coefficient heatmap of the random forest top 10 features by importance.

In some model iterations of removing duplicate-like features, 'travel_distance_diff' was kept. Here, it was removed. Features 'away_home_a' and 'away_game_count_day_window2_diff' took its place in the top 10 random forest regressor features by Gini impurity.

Random Forest Regressor Time Series Cross-Validation

As our game score difference prediction model was developed, we restructured our random forest regressor model learning to time series since this is the nature of the desired outcome, i.e. to use historical data to predict future game score differences.

A complete segmenting of data per random forest regressor is shown in Figure 12.

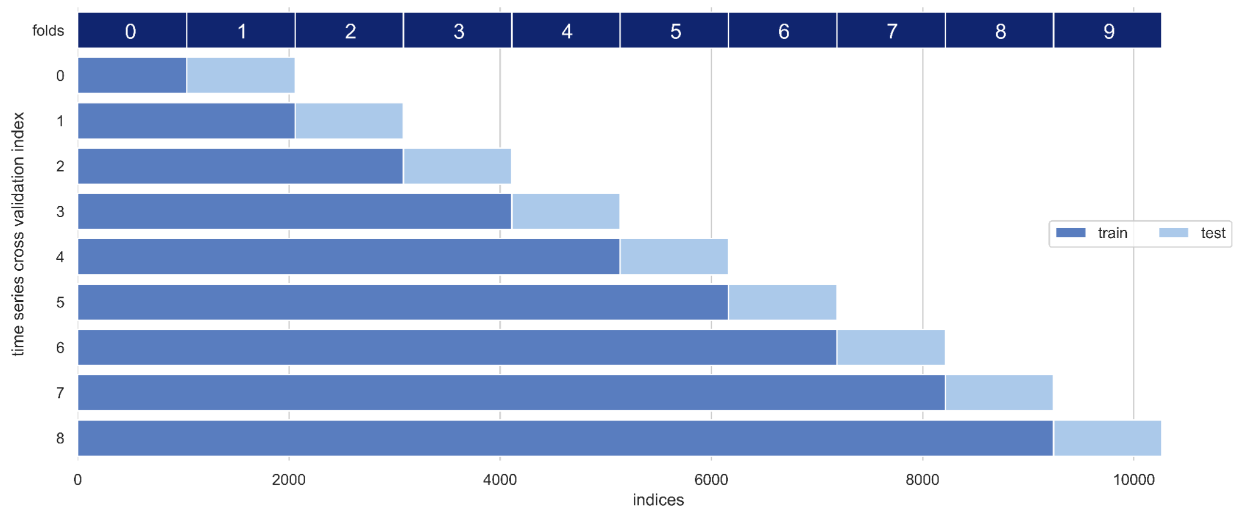


Figure 12. Time series cross-validation train and test indices of time series NBA games for seasons 2010-11 thru 2017-18.

This time series cross-validation was important because we used the time series cross-validation random forest regressor collection score difference predictions later in the betting policy heuristic.

Model Performance

Of the 8 NBA seasons the first 8214 games (i.e. folds 0 thru 7) were used as train and the next 1026 games of the 2016-17 season (i.e. fold 8) as test. Using the random forest regressor top features, we collected MAE and MAPE metrics for a random forest, gradient boosting, linear regression model as shown in Table 1. Because the random forest regressor performed the best in the MAE metric, it was used for score difference prediction in our betting policy.

Model Metrics

	MAE	MAPE
Random Forest	10.415	135.57
Gradient Boosting	10.516	132.951
Linear Regression	10.534	118.23

Table 1. Metrics MAE and MAPE for random forest, gradient boosting, and linear regression models on fold 8.

The random forest regressor development summary diagram is shown in Figure 13.

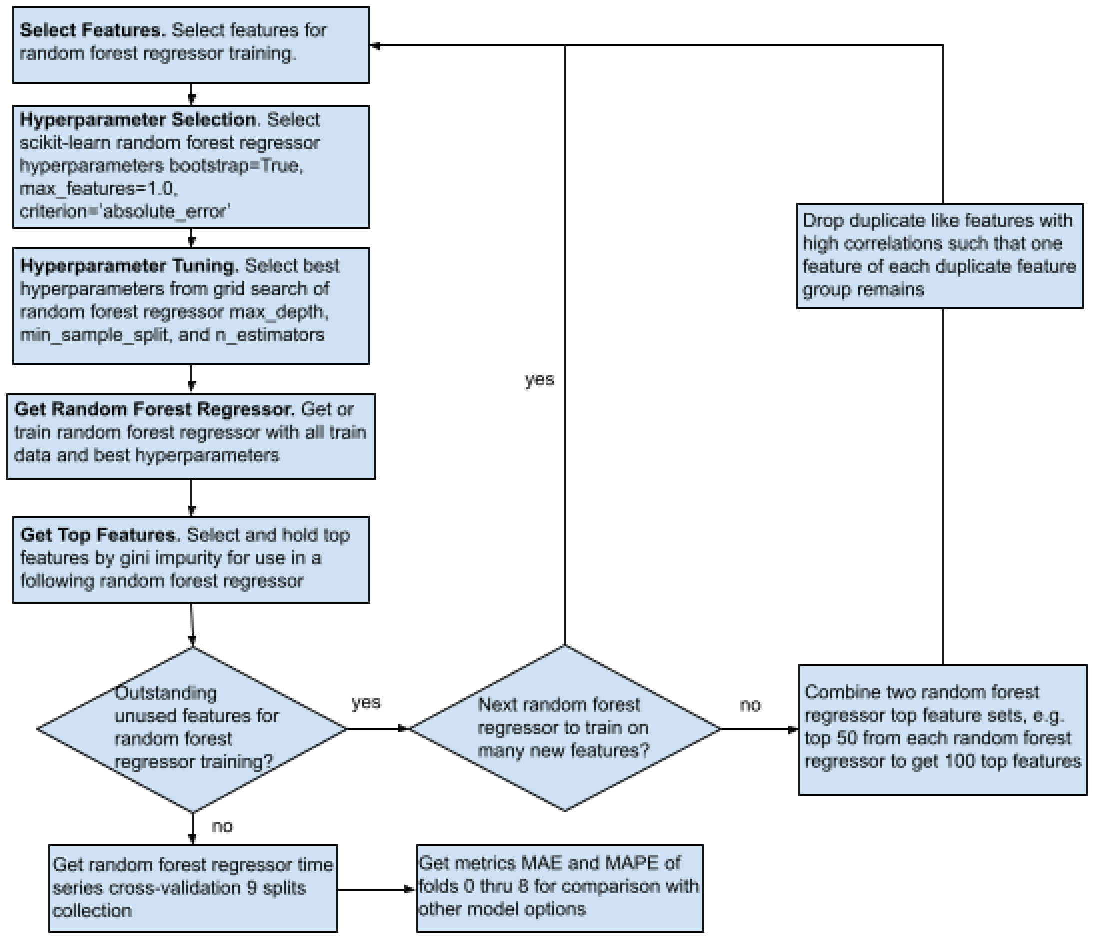

Select Features. Select features for random forest regressor training.
Hyperparameter Selection. Select scikit-learn random forest regressor hyperparameters bootstrap=True, max_features=1.0, criterion='absolute_error'
Hyperparameter Tuning. Select best hyperparameters from grid search of random forest regressor max_depth, min_sample_split, and n_estimators
Get Random Forest Regressor. Get or train random forest regressor with all train data and best hyperparameters
Get Top Features. Select and hold top features by gini impurity for use in a following random forest regressor
Outstanding unused features for random forest regressor training?
yes
Next random forest regressor to train on many new features?
yes
no
Combine two random forest regressor top feature sets, e.g. top 50 from each random forest regressor to get 100 top features
Drop duplicate like features with high correlations such that one feature of each duplicate feature group remains
no
Get random forest regressor time series cross-validation 9 splits collection
Get metrics MAE and MAPE of folds 0 thru 8 for comparison with other model options

Figure 13. The random forest regressor development summary diagram.

Betting Policy

Betting Policy: Overview

To evaluate the usefulness of our predictions, we placed NBA game spread bets using a betting policy. This was accomplished by learning which policies worked and which did not by betting on past games called the learn heuristic.

First, we ran betting policy combinations of a prediction confidence interval, absolute value of spread lower limit, and price break even upper limit on the train data. Second, we took our learnings from our run on the train data and ran a top betting policy combination on the train and validation data. We repeated this step until we had a heuristic method.

The time series cross-validation data segmentation for the learn and test heuristic are shown in Figure 14.

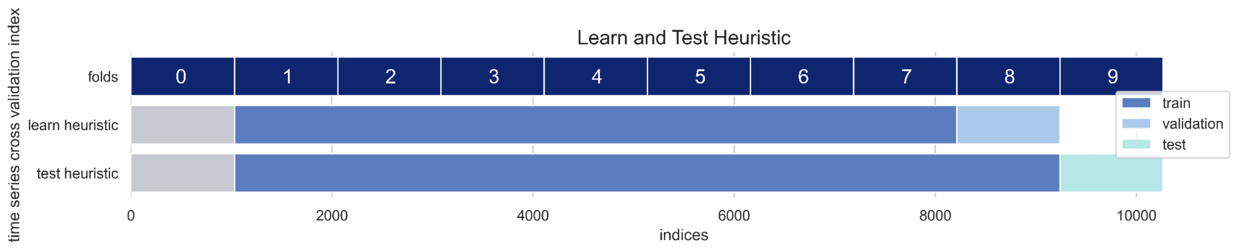


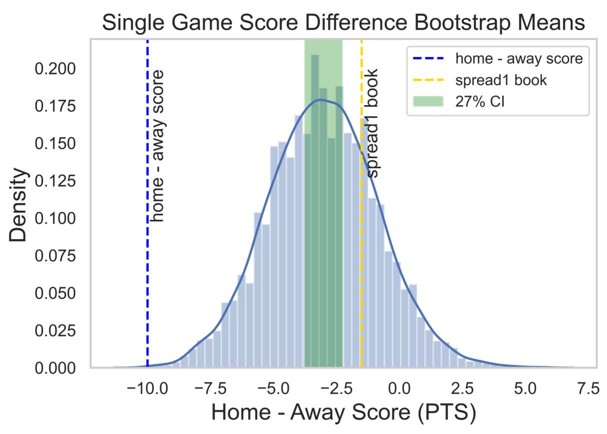
Figure 14. Time series cross-validation for learn and test heuristic.

Betting Policy: Feature Engineering

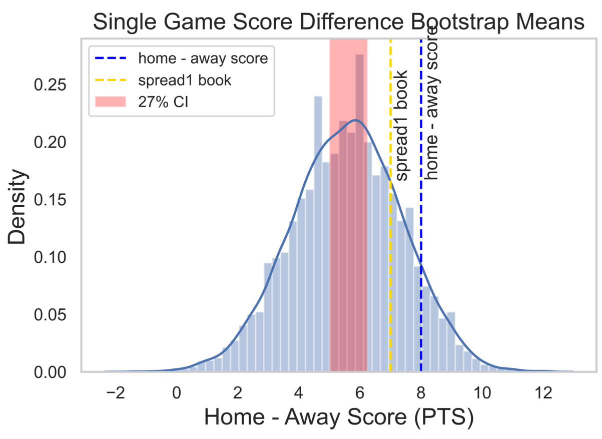
The first betting policy criterion we created was the game score difference confidence interval. We now had predictions from time series cross-validation folds 1 thru 8 as shown in Figure 14. A score difference distribution for each game was created by bootstrap sampling the random forest decision tree predictions. From these game score difference distributions, we formed confidence intervals 1 thru 100.

Under the assumption the sportsbook ‘spread1’ and ‘spread2’ are negative inverses of each other, we described how bets were filtered by the confidence interval betting policy criterion in Figure 16.

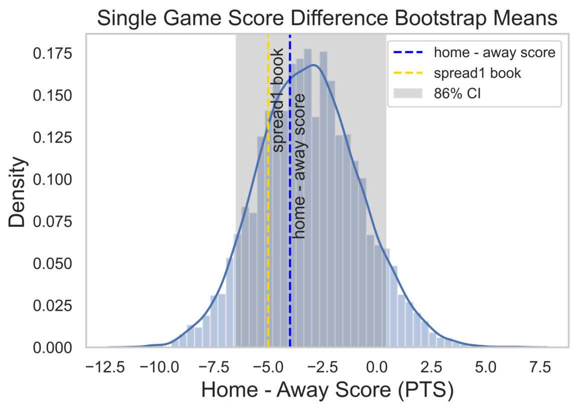
For each game, a confidence interval that contained the sportsbook ‘spread1’ value, it was a decision to not place a bet on that game as shown in Figure 16c. The confidence interval not containing the sportsbook ‘spread1’, permitted the betting on that game as shown in Figure 16a and Figure 16b.



(a)



(b)



(c)

Figure 15. (a) A bet was placed because the confidence interval was outside the sportsbook ‘spread1’ and winning because the predicted score difference is in the direction from ‘spread1’ to score difference. (b) A bet was placed because the confidence interval is outside the sportsbook ‘spread1’, but losing because the predicted score difference is not in the direction from ‘spread1’ to score difference. (c) A bet was not placed because the confidence interval contains the sportsbook ‘spread1’.

The second betting policy criterion was the absolute value of ‘spread1’ lower limit. Taking the NBA sportsbook ‘spread1’ values from folds 0 thru 7, we had the absolute value of ‘spread1’ lower limit values for our betting policy criteria.

The third betting policy criterion was the price break even upper limit. Taking the NBA sportsbook price break even values from folds 0 thru 7, we had the price break even upper limit values.

Betting Policy: Learn Heuristic

We wanted to create a generalization for our betting policy using either a machine learning model or heuristic. We used a heuristic.

The learn heuristic steps were the following:

1. Select combinations of criteria:
    - Prediction confidence interval: 1%, 2%, …, 100%.
    - Absolute value of spread lower limit: 0.4878, 0.4950, ..., max.
    - Price break even upper limit: 0, 0.5, …, 22.0.
2. Run betting policy combinations on the train data.
3. Select the top 5% of betting policy combinations by win percentage with a >2% of games bet on.
4. Select a betting policy by criterion value count plot.
5. Run a betting policy combination on each of the train and validation data.
6. Confirm the betting policy has acceptable accuracy and >2% of games bet on, otherwise return to 4.

The learn heuristic can also be described as the flowchart shown in Figure 17.

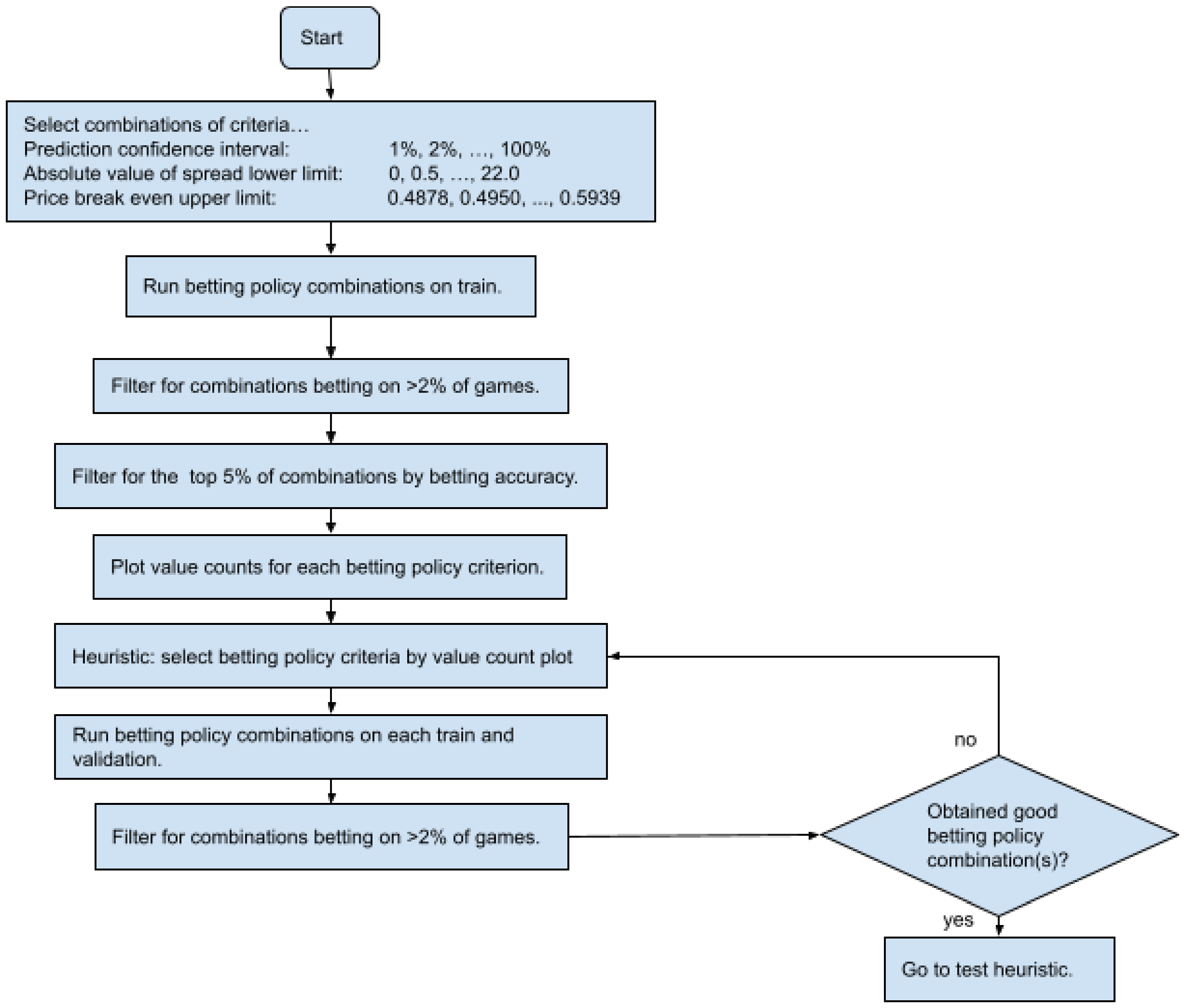

Start
Select combinations of criteria…
Prediction confidence interval: 1%, 2%, …, 100%
Absolute value of spread lower limit: 0, 0.5, …, 22.0
Price break even upper limit: 0.4878, 0.4950, ..., 0.5939
Run betting policy combinations on train.
Filter for combinations betting on >2% of games.
Filter for the top 5% of combinations by betting accuracy.
Plot value counts for each betting policy criterion.
Heuristic: select betting policy criteria by value count plot
Run betting policy combinations on each train and validation.
Filter for combinations betting on >2% of games.
Obtained good betting policy combination(s)?
no
yes
Go to test heuristic.

Figure 17. The learn heuristic flowchart diagram.

Betting Policy: Test Heuristic

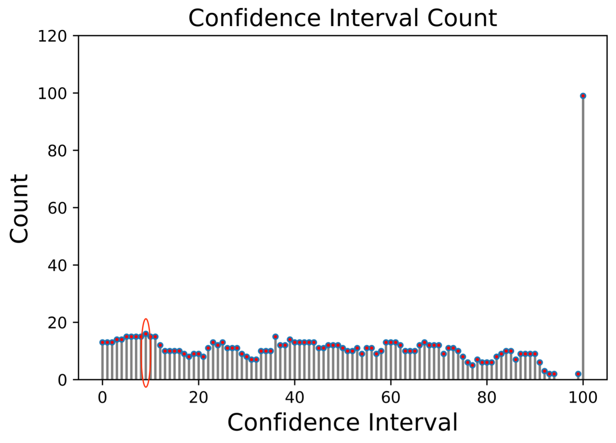
We learned a heuristic and we wanted to test it.

Here, we ran betting policy combinations on the train data, but then applied our heuristic method to select our betting policy. This betting policy was run on the test data to get our results.

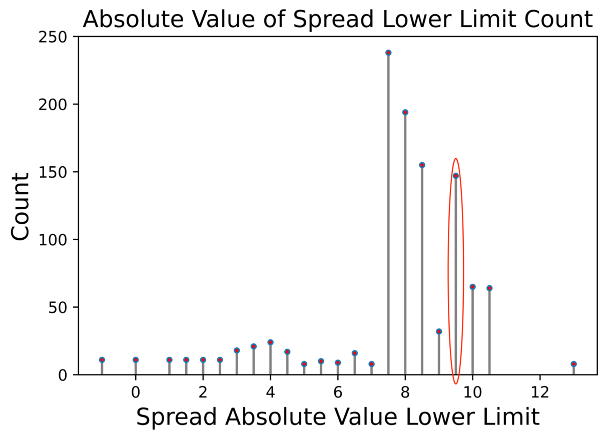
Our test heuristic steps were the following:
1. Learn Heuristic steps 1-4.
2. Run the betting policy combination on the train data.
3. Confirm the betting policy has an acceptable accuracy and >2% of games bet on, otherwise return to step 1. Learn Heuristic step 4.

4. Run this betting policy combination on the test data.

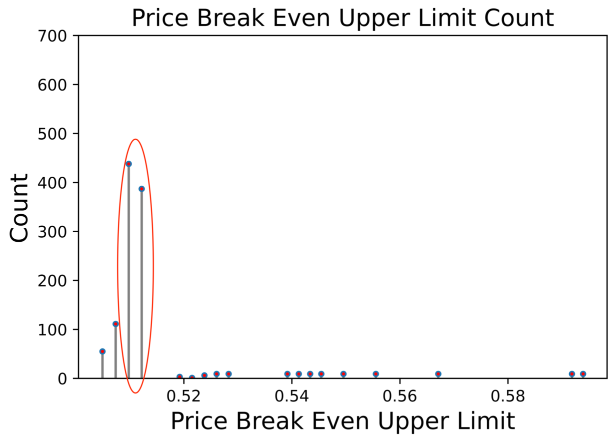
In Figure 18., we show the betting policy criteria selection when we test the heuristic.



(a)



(b)



(c)

Figure 18. The betting policy criteria were selected by the value count plots per the learned heuristic.

Betting Policy: Results

For the case of placing bets on fold 9 according to our random forest regressor and betting policy, we analyze our outcome.

We took the approach of using a rule of thumb 1% account balance per bet. As shown in Figure 19. our betting policy placing bets on fold 9 has a 45% ROI.

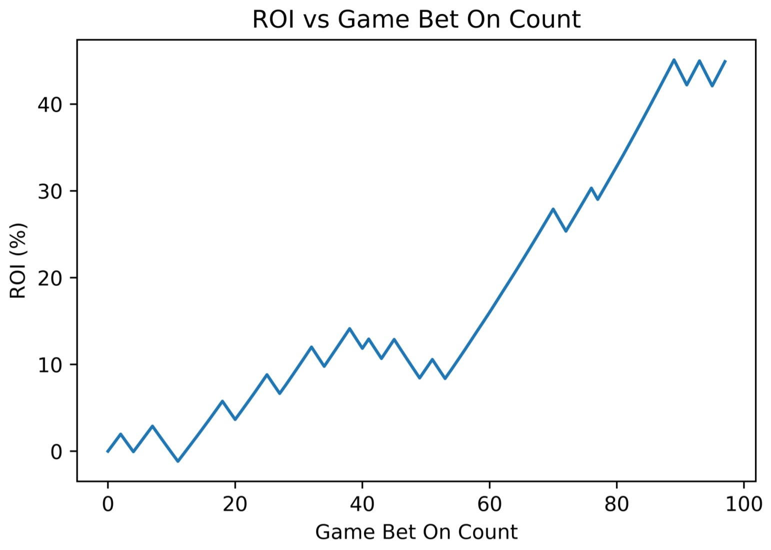


Figure 19. Implementing a 1% of the account balance per bet over the 2017-18 season, our betting policy had a 45% ROI.

In the situation where the underlying probabilities were unknown with many variables to account for, analyzing our outcome by betting on a nonparametric bootstrap sampling was appropriate.

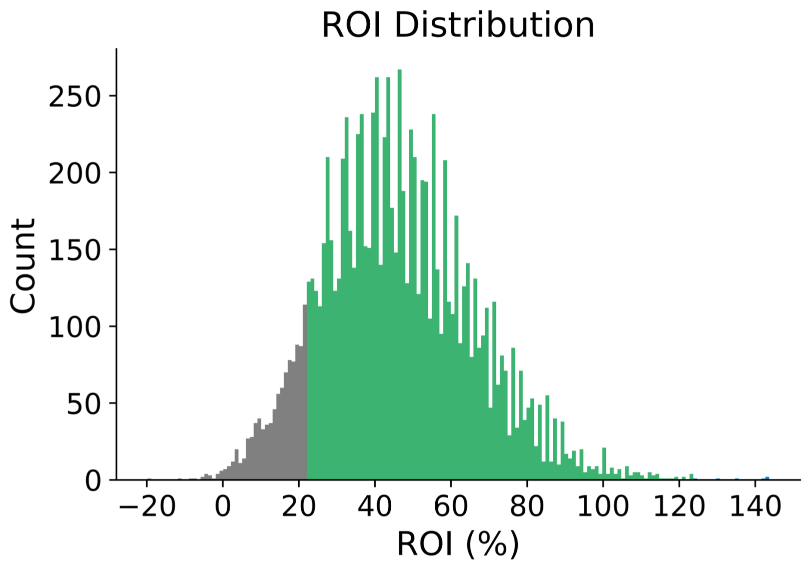
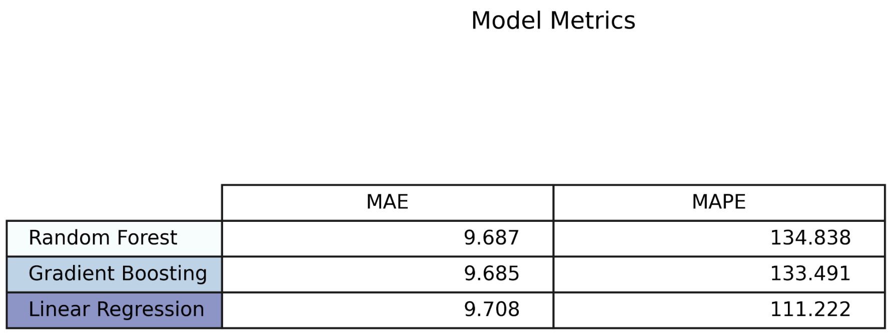


Figure 20. Placing a 1% of the account balance per bet on a nonparametric bootstrap sampling of 2017-18 season NBA sportsbook games suggested that 90% of the time a ≥22% ROI was expected.

Model: Results

Our best model, the random forest regressor, predicted an NBA game score difference on average within 9.687 points. This is similar to that of linear regression (selecting k-best) and gradient boosting at 9.708 and 9.685 shown in Table 2.



Model Metrics

	MAE	MAPE
Random Forest	9.687	134.838
Gradient Boosting	9.685	133.491
Linear Regression	9.708	111.222

Table 2. Model prediction metrics for the random forest, gradient boosting, and linear regression models on fold 9.

Conclusion

Implementing a game score difference prediction model and simple betting policy over the 2017-18 season showed a 45% ROI. This was evidence a good model for NBA game score difference was developed and use is permissible for NBA sportsbook spread betting.

Further testing the robustness of our model by collecting NBA game data up to the present is desired, such that NBA games of not just the past but the future can be bet on, furthering the practical importance of our game score difference prediction model.

However, further testing the robustness of our model by collecting historical NBA game data to the present for prediction on future NBA games is desired.

Future Challenges

Over time, ROI is expected to decrease because eat bet is information for the sportsbook to learn from on how to set the spread. In other words, the sportsbook is learning what you're learning.

Appendix: Future Work

Some future work includes collecting NBA game data up to the present. This way, NBA games of not just the past but the future can be bet on, furthering the practical importance of our game score difference prediction model.

Also, adding more categorically different features and modifications to our current features may serve invaluable to increasing the predictive accuracy of NBA game score difference.

Replacing 5-fold cross-validation hyperparameter tuning with time series cross-validation (for predicting score difference using just historical data) is desired to avoid data leakage.

Fix coefficient of determination scoring of the time series cross-validation random forest regressor validation on folds 6,7,8 to use MAE scoring instead because it is the metric we are optimizing for spread betting.

References

[1]
https://www.kdnuggets.com/2021/10/machine-learning-model-development-operations-principles-practice.html