"THIS GAME IS IN THE FRIDGE": PREDICTING NBA GAME OUTCOMES

Jesse A Rodríguez, jrodrig@stanford.edu

Abstract

Five machine learning models are investigated along with a novel feature design to predict the outcomes and popular betting metrics of NBA basketball games. The most predictive feature design consisted of the statlines of the top 3 players of each team in their last 3-4 games. The support vector machines classifier performed the best in predicting game outcomes, achieving 62.6% accuracy on the test set. The logistic regression and quadratic discriminant analysis models were 57.8% and 56.4% accurate respectively. The exponential neural network (NN) score predictor achieved 59.1% accuracy and reproduced the score distribution of the test set well, indicating that it may be effective in predicting the over/under for NBA games. The softmax NN score predictor only achieved 57.6% accuracy but it managed to reproduce the margin of victory distribution well. Ultimately, a mismatch between the train set and the dev/test sets as well as an overall lack of data likely led to the unremarkable performance of the five models.

Motivation and Related Work

Of all the areas to which machine learning has been applied during its rise to ubiquity, one of the most popular has been predictive modeling for sporting events[1]. Though it may seem frivolous, the recent legalization of sports betting in 2018 has made correctly predicting the outcomes of sporting events quite lucrative[2]. For my CS 229 project, I decided to take a sport that I love and know very well, NBA basketball, and try to create a machine learning model that can outperform prior models, the best of which lie in the 60-70% range[3][4][5][6] in predicting game outcomes, which is comparable to human experts[6][7]. These approaches place an emphasis on team-level metrics that are averaged over whole seasons or several games. Prior machine learning approaches, particularly that of ref. [6] where the authors utilize a fusion of four NN classifiers via a Bayesian network, have seen accuracies in excess of 70% using team-level metrics. While this out-performs the experts, it doesn't allow for the prediction of final scores which gives access to several key betting metrics. This study also used a dataset of only 620 games, so the model is not likely to generalize well as the stlye of NBA play has been evolving so rapidly as of late. This project aims to address these issues with a feature design focused on recent performance of individual players.

Predicting the outcome of sporting events in general can be considerably difficult, especially in sports with a low number of scoring opportunities. Basketball is a unique sport in that there are many scoring opportunities throughout each game, reducing the influence of randomness on game outcomes. In this work, several ML models are trained to predict winners via both binary classification and final scores, with the latter allowing for the additional prediction of point spreads, over/unders, and other popular sports bets/metrics. Choosing final scores as the objective is ambitious, as the randomness mentioned before has an opportunity to manifest itself in predicting something with such a high variance, but the values generated can still be effective in predicting the common metrics above.

Dataset and Features

The base dataset for this work is simply the complete statline for every player in each game from the 2012-2013 NBA season to the 2017-2018 season. I have chosen a 80/10/10 split for train/dev/test, which results in dataset sizes of about ∼5800/725/725 games depending on which feature parameters are chosen, as I will explain shortly. In order to ensure that the model cannot 'see the future' (i.e. that the model is trying to predict the outcomes of games that occurred after games in the training set) and also that the test and dev sets are pulled from the same distribution, the train set is simply the first 80% chunk of the games by chronological order and the test/dev set examples are sampled uniformly from the remaining 20%. Histograms for individual team score,

combined scoring (relevant for over/under bets), and point spread for the train/dev/test sets are included below in Figure 1. We can see already that the train set has a slightly different distribution than the test/dev sets, most evident in the per game scoring where the mean of the distribution is shifted upward ∼10 points in the dev/test sets. This is likely due to the recent change in pace and style of play in the NBA à la the Golden State Warriors post-2016 where teams take and make much more 3 pointers than before and thus have much higher scoring games. The distributions of the point spreads (margin of victory) seem to be fairly consistent, however. We will discuss the potential impact of these factors later.

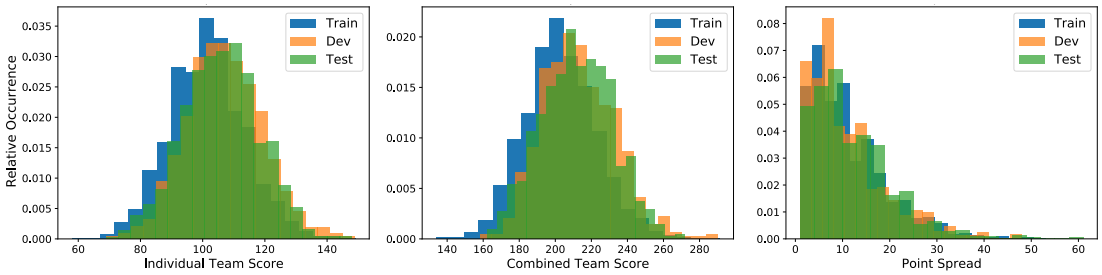

Figure 1: *Histograms for individual team scores (left), combined scores (middle) and point spreads (right) from the NBA game corpus used in this project.*

The main aspect of my approach that differs from previous attempts at final score prediction or even just game outcomes in the past is feature engineering. Up until now, all of the prior attempts at predicting NBA game outcomes that I found used team-level metrics (total points, rebounds, assists, etc.) averaged over entire seasons[8][9][10][11]. While this does lead to prediction accuracies of game outcomes on par with the experts, I felt that this data is just too simplistic to accurately predict the outcomes of games, and particularly their scores. For example, what if one player will not be playing due to injury, or is just returning from one? What if one player is having an incredible hot streak (many NBA fans will remember 'Linsanity', for example)? What if the team itself is in a slump? These are the edge cases that I aimed to capture with my feature design.

My feature/label set up is as follows: Each feature vector contains the statlines of the top n_p players (ranked by scoring) that are on the roster for the game in question for each team in their personal previous n_g games. This means that if a player is just returning from an injury, we pull from the last n_g games they played before being injured. For example, when you consider the performance in the previous 5 games of the top 8 players and omit no stats (17 total), you have feature vectors of length 1360. The player statlines for each team are stacked home team first, so as to encode the home-court advantage, and then standardized by subtracting the mean and dividing by the standard deviation. The labels are simply the final scores of each team with the order corresponding to the order in which the statlines are stacked. If a binary classifier is being trained, the labels are converted to 1's and 0's where 1 corresponds to home team win. An example of an un-normalized/standardized feature vector for $n_g = 1$ and $n_p = 2$ is found below, where THP is top home player, 2HP is second-ranked home player, and TAP is top away player, and the stats you see are the statlines for the players in question from the last game that they played:

$$\underbrace{[\ \underbrace{111}_{\text{home score}}\ ,\ \underbrace{102}_{\text{away score}}\ ]}_{\text{label}}\ \underbrace{[\ \underbrace{29}_{\text{THP PTS}}\ ,\ \underbrace{8}_{\text{THP REB}}\ ,\ \underbrace{5}_{\text{THP AST}}\ ,\ \ldots\ \underbrace{17}_{\text{2HP PTS}}\ ,\ \underbrace{3}_{\text{2HP REB}}\ ,\ \ldots\ \underbrace{26}_{\text{TAP PTS}}\ ,\ \underbrace{4}_{\text{TAP REB}}\ ,\ \ldots]}_{\text{feature vector}}$$

The code to produce these datasets from player box scores stored in csv files can be found at

http://github.com/JesseRodriguez/CS230_229_FinalProject. The Box Score data must be in the same format as the Kaggle dataset found at http://www.kaggle.com/pablote/nba-enhanced-stats. The Refridgerator class (found in RNBA.py in the repo), as I've called it (this is a reference to the late, great NBA commentator Chick Hearn), takes these box scores and produces a dataset for different values of n_g and n_p specified by the user. Below, we sweep these values to search for an optimal combination to maximize predictive power of the model.

Methods

In this work, five distinct machine learning models are explored. The first and most simple model tested is a logistic regression model implemented via SKLearn[12]. Logistic regession is a simple binary classifier that utilizes the sigmoid function to calculate the probability of one class over the other given training data. The feature vectors for each example are dotted with a set of model parameters and then substituted into the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}} \in (0, 1)$ to give the desired probability. The second model tested is support vector machines, which are a set of ML models that aim to maximize the 'distance' between the training examples and the hyperplane that correctly classifies them. The full mathematical formulation can be found on the SVM page within SKLearn's website[12]. The third model tested was a Gaussian discriminant analysis classifier which assumes $p(x|y)$ is distributed according to a multivariate normal distribution and predicts the class that yields a higher likelihood of the data. In this case, the covariance matrices for each class are allowed to be different and thus the model draws quadratic decision boundaries.

For the score predictor, 2 neural network (NN) architectures were tested. A NN stacks layers of units similar to the logistic regression classifier with various non-linear activation functions in place of the sigmoid, and then fits the parameters using a cost function along with an optimization algorithm like gradient descent. The first NN architecture is forked; containing 6 fully connected layers with a fork after layer 3 and an output layer with an exponential activation for each fork. The second architecture contains 9 fully connected layers with a fork after layer 6. The output layer is a softmax classifier (a function that assigns a probability to a number of possible classes; a generalization of logistic regression) where the classes are the possible final scores of each team. In our dataset, the highest score observed is 149 and the lowest is 58, leading to two output layers of size 92 (one for each fork). Due to the small size of the dataset, HPC resources were not required and all models were trained on a personal laptop with SKLearn or Keras[14]. A diagram of each NN model along with the number of nodes in each layer and expressions for their output layer activations and losses is included below.

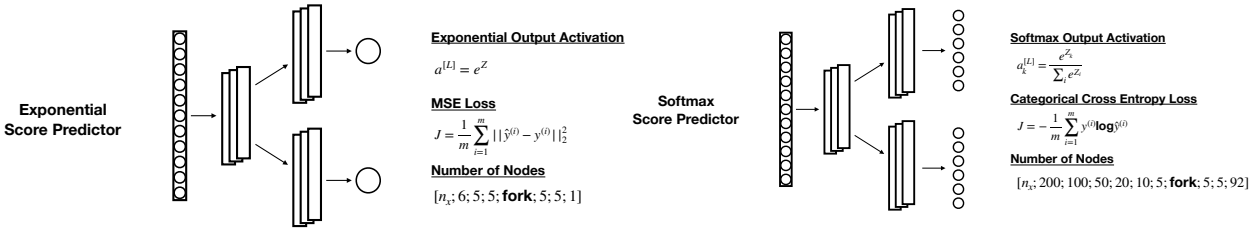

Figure 2: *NN architectures for the two model types explored in this work.*

Results and Discussion

Feature Development

To see which feature design would lead to a model with the greatest predictive power, I swept over all combinations of n_g and n_p in the range of 1 to 10 and trained each model discussed above.

The results of this sweep are found below in Figure 3 where the models are evaluated according to whether or not they predicted the outcomes of the games correctly.

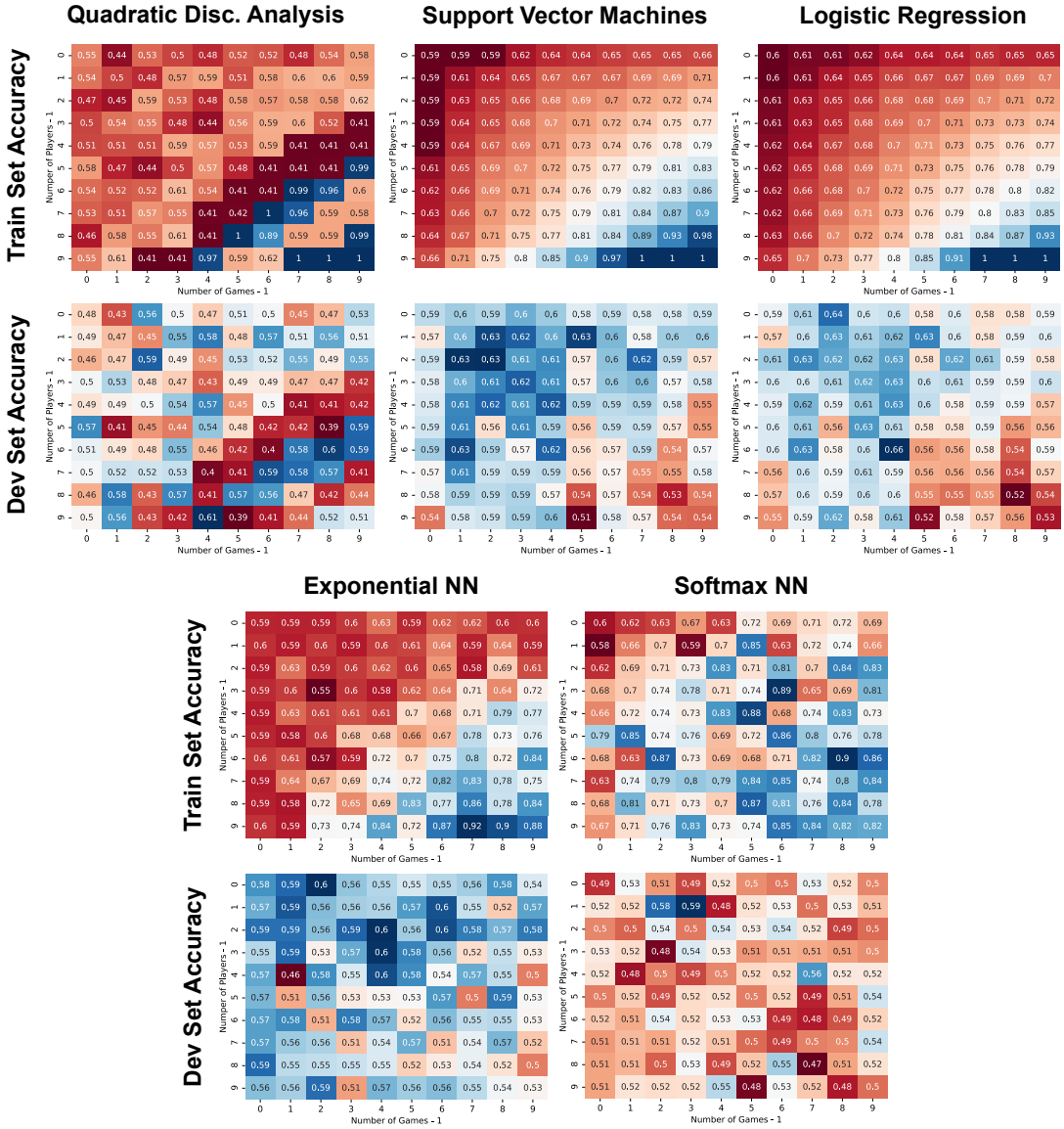

Figure 3: *Dev set performance of each machine learning model for each n_g, n_p pair.*

We can see immediately that of the two score predictors, the exponential score predictor is far superior in predicting game outcomes. The QDA model tends to struggle on both the train and dev set, indicating that the assumption of a gaussian distribution of the training data is problematic. The SVM and LR models perform very similarly, consistently performing in excess of 60% on the dev set and therefore reaching the realm of human experts in performance. It is clear through evaluation of these values, however, that all the models have bias and variance problems, and tend to overfit to the training set for large values of n_g and n_p. In the case of the NNs, regularization methods were employed to no avail. Ultimately, however, it may just be a fundamental mistmatch of the train set to the dev set that causes the problems we see here. As was mentioned before, unfortunately the NBA saw quite the revolution in style of play right about at the time of the

train/dev split, so any model trained on this corpus is somewhat destined to be biased.

Test Set Performance

Based on the dev set performance results above, n_g, n_p pairs were selected for each model for testing. Accuracies on the test set are found below:

Model	n_g,n_p pair	Train Set Accuracy	Dev Set Accuracy	Test Set Accuracy
QDA	(4,10)	97%	61%	56.5%
SVM	(3,3)	65%	63%	62.6%
LR	(5,7)	72%	66%	57.8%
Exponential NN	(4,3)	62%	60%	59.1%
Softmax NN	(4,2)	59%	59%	57.6%

Thus, we have some some issues with overfitting to the dev set (this likely arises from choosing islands of good dev set performance on the heat maps), but overall we have comparable performance on the test set. In addition, we take a look at the distributions of the scores predicted by the exponential and softmax NN models as compared to the true test set labels.

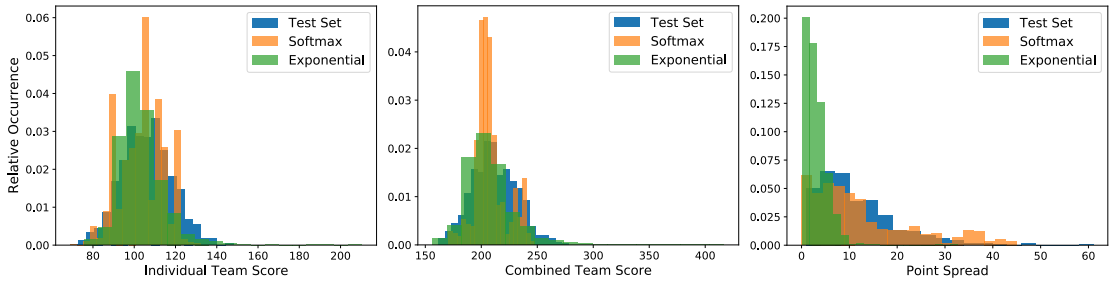

Figure 4: *Histograms for individual team scores (left), combined scores (middle) and point spreads (right) from the true test set values and the predictions from each model.*

Interestingly, the predicted score distributions produced by the exponential model look much better than the sigmoid (despite some evidence of overfitting to the train set as the mean is shifted down about 10 points as we see above), but the point spread distribution produced by the sigmoid is much more representative. This suggests that the sigmoid architecture may be more conducive to predicting point spreads while the exponential model is more suited to predicting game outcomes and over/unders, though a more rigorous investigation of these metrics is needed to say for sure.

Conclusion and Future Work

The most predictive feature design consisted of the statlines of the top 3 players of each team in their last 3-4 games. The support vector machines classifier performed the best in predicting game outcomes, achieving a performance on the test set on par with human experts. The exponential score predictor reproduced the score distribution of the test set quite well, indicating that it may be effective in predicting the over/under for NBA games. The softmax score predictor performed relatively poorly on the previous two tasks but managed to reproduce the margin of victory distribution remarkably well. Ultimately, a mismatch between the train set and the dev/test sets as well as an overall lack of data likely led to the unremarkable performance of the three models.

In the future, omission of some stats along with the inclusion of other stats that might be quite predictive and weren't present in this dataset (such as +/- or unique player IDs), would likely improve performance by shrinking the size of the feature vectors and making the information present more potent. Ultimately, however, the main issue that needs to be addressed is the lack of training data and the mismatch between the train and dev/test sets. To tackle both of these problems, it may be effective to artificially generate data via the new (and quite sophisticated) NBA basketball video games. These games simulate entire seasons of the NBA remarkably well, and using them to generate an immense dataset to train on may lead to very high prediction accuracy. Overall though, I believe the work presented here is a good first step toward considering a new dataset paradigm in NBA basketball prediction, and it is primed to be built upon.

References

[1] : Bunker, R. et al., *A Machine Learning Framework for Sport Result Prediction.* Applied Computing and Informatics Vol. 15, Issue 1, Pg. 27-33. (2019)

[2] : Ryan, S. et al., *Application of Bayesian Network to total points in NBA Games.* Industrial and Manufacturing Systems Engineering Conference Proceedings and Posters. 179. (2019)

[3] : Lin, J. et al., *Predicting National Basketball Association Winners.* CS 229 Final Project. (2014)

[4] : Torres, R., *Prediction of NBA games based on Machine Learning Methods.* University of Wisconsin-Madison. (2013)

[5] : Jain, S. et al., *Machine Learning Approaches to Predict Basketball Game Outcome.* 3rd International Conference on Advances in Computing, Communication & Automation. IEEE. (2017)

[6] : Loeffelholz, B. et al., *Predicting NBA Games using Neural Networks.* Journal of Quantitative Analysis in Sports. Vol. 5 Iss. 1, Article 7. (2009)

[7] : AccuScore, *The Leader in Sports Forecasting*, https://accuscore.com/basketball/nba-picks

[8] : Uudmae, J., *CS 229 Final Project: Predicting NBA Game Outcomes.* CS 229 Final Project. (2017)

[9] : Avalon, G. et al., *Various Machine Learning Approaches to Predicting NBA Score Margins.* CS 229 Final Project. (2016)

[10] : Thabtah, F. et al., *NBA Game Result Prediction Using Feature Analysis and Machine Learning.* Annals of Data Science 6(1):103–116. (2019)

[11] : Bucquet, A., Sarukkai, V., *The Bank Is Open: AI in Sports Gambling.* CS 229 Final Project. (2018)

[13] : scikit-learn.org, *Scikit Learn* (accessed 2019)

[13] : D. P. Kingma, J. Lei Ba. *Adam : A method for stochastic optimization.* arXiv:1412.6980v9 (2014)

[14] : tensorflow.org, *Keras* (accessed 2019)

CS 230 FINAL PROJECT REPORT (SHARED PROJECT, I am still the sole author)

Abstract

Three NN architectures are proposed along with a novel feature design to predict the outcomes and popular betting metrics of NBA basketball games. The most predictive feature design consisted of the statlines of the top 3 players of each team in their last 4 games. The binary classifier NN performed the best in predicting game outcomes, achieving 59.8% accuracy on the test set. The exponential score predictor achieved 59.1% accuracy and reproduced the score distribution of the test set quite well, indicating that it may be effective in predicting the over/under for NBA games. The softmax score predictor only achieved 57.6% accuracy but it managed to reproduce the margin of victory distribution remarkably well. Ultimately, a mismatch between the train set and the dev/test sets as well as an overall lack of data likely led to the unremarkable performance of the three models.

Motivation and Related Work

Of all the areas to which machine learning has been applied during its rise to ubiquity, one of the most popular has been predictive modeling for sporting events[1]. Though it may seem frivolous, the recent legalization of sports betting in 2018 has made correctly predicting the outcomes of sporting events quite lucrative[2]. For my CS 230 project, I decided to take a sport that I love and know very well, NBA basketball, and try to create a deep learning model that can outperform prior models, the best of which lie in the 60-70% range[3][4][5][6] in predicting game outcomes, which is comparable to human experts[6][7]. These approaches place an emphasis on team-level metrics that are averaged over whole seasons or several games. Prior deep learning approaches, particularly that of ref. [6] where the authors utilize a fusion of four NN classifiers via a Bayesian network, have seen accuracies in excess of 70% using team-level metrics. While this out-performs the experts, it doesn't allow for the prediction of final scores which gives access to several key betting metrics. This study also used a dataset of only 620 games, so the model is not likely to generalize well as the stlye of NBA play has been evolving so rapidly as of late. This project aims to address these issues with a feature design focused on recent performance of individual players.

Predicting the outcome of sporting events in general can be considerably difficult, especially in sports with a low number of scoring opportunities. Basketball is a unique sport in that there are many scoring opportunities throughout each game, reducing the influence of randomness on game outcomes. In this work, several NN architectures are trained to predict winners via both binary classification and final scores, with the latter allowing for the additional prediction of point spreads, over/unders, and other popular sports bets/metrics. Choosing final scores as the objective is ambitious, as the randomness mentioned before has an opportunity to manifest itself in predicting something with such a high variance, but the values generated can still be effective in predicting the common metrics above.

Dataset and Features

The base dataset for this work is simply the complete statline for every player in each game from the 2012-2013 NBA season to the 2017-2018 season. I have chosen a 80/10/10 split for train/dev/test, which results in dataset sizes of about ∼5800/725/725 games depending on which feature parameters are chosen, as I will explain shortly. In order to ensure that the model cannot 'see the future' (i.e. that the model is trying to predict the outcomes of games that occurred after games in the training set) and also that the test and dev sets are pulled from the same distribution, the train set is simply the first 80% chunk of the games by chronological order and the test/dev set examples are sampled uniformly from the remaining 20%. Histograms for individual team score, combined scoring (relevant for over/under bets), and point spread for the train/dev/test sets are included below in Figure 1. We can see already that the train set has a slightly different distribution

than the test/dev sets, most evident in the per game scoring where the mean of the distribution is shifted upward ∼10 points in the dev/test sets. This is likely due to the recent change in pace and style of play in the NBA à la the Golden State Warriors post-2016 where teams take and make much more 3 pointers than before and thus have much higher scoring games. The distributions of the point spreads (margin of victory) seem to be fairly consistent, however. We will discuss the potential impact of these factors later.

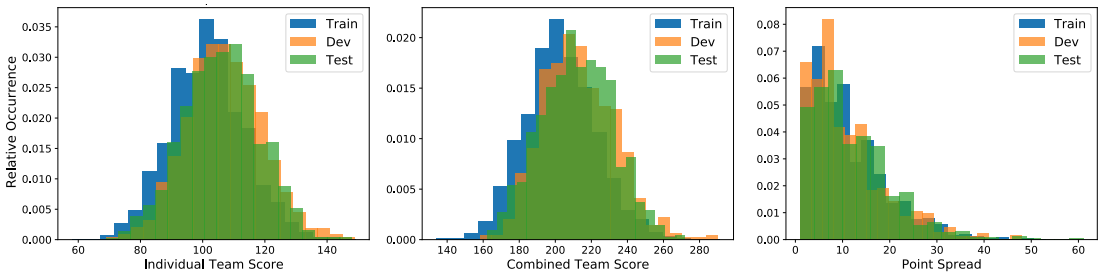

Figure 5: *Histograms for individual team scores (left), combined scores (middle) and point spreads (right) from the NBA game corpus used in this project.*

The main aspect of my approach that differs from previous attempts at final score prediction or even just game outcomes in the past is feature engineering. Up until now, all of the prior attempts at predicting NBA game outcomes that I found used team-level metrics (total points, rebounds, assists, etc.) averaged over entire seasons[8][9][10][11]. While this does lead to prediction accuracies of game outcomes on par with the experts, I felt that this data is just too simplistic to accurately predict the outcomes of games, and particularly their scores. For example, what if one player will not be playing due to injury, or is just returning from one? What if one player is having an incredible hot streak (many NBA fans will remember 'Linsanity', for example)? What if the team itself is in a slump? These are the edge cases that I aimed to capture with my feature design.

My feature/label set up is as follows: Each feature vector contains the statlines of the top n_p players (ranked by scoring) that are on the roster for the game in question for each team in their personal previous n_g games. This means that if a player is just returning from an injury, we pull from the last n_g games they played before being injured. For example, when you consider the performance in the previous 5 games of the top 8 players and omit no stats (17 total), you have feature vectors of length 1360. The player statlines for each team are stacked home team first, so as to encode the home-court advantage, and then standardized by subtracting the mean and dividing by the standard deviation. The labels are simply the final scores of each team with the order corresponding to the order in which the statlines are stacked. If a binary classifier is being trained, the labels are converted to 1's and 0's where 1 corresponds to home team win. An example of an un-normalized/standardized feature vector for $n_g = 1$ and $n_p = 2$ is found below, where THP is top home player, 2HP is second-ranked home player, and TAP is top away player, and the stats you see are the statlines for the players in question from the last game that they played:

$$\underbrace{[\ \underbrace{111}_{\text{home score}}\ ,\ \underbrace{102}_{\text{away score}}\ ]}_{\text{label}}\ \underbrace{[\ \underbrace{29}_{\text{THP PTS}}\ ,\ \underbrace{8}_{\text{THP REB}}\ ,\ \underbrace{5}_{\text{THP AST}}\ ,\ \ldots\ \underbrace{17}_{\text{2HP PTS}}\ ,\ \underbrace{3}_{\text{2HP REB}}\ ,\ \ldots\ \underbrace{26}_{\text{TAP PTS}}\ ,\ \underbrace{4}_{\text{TAP REB}}\ ,\ \ldots]}_{\text{feature vector}}$$

The code to produce these datasets from player box scores stored in csv files can be found at http://github.com/JesseRodriguez/CS230_229_FinalProject. The Box Score data must be in the same format as the Kaggle dataset found at http://www.kaggle.com/pablote/nba-enhanced-stats.

The Refridgerator class (found in RNBA.py in the repo), as I've called it (this is a reference to the late, great NBA commentator Chick Hearn), takes these box scores and produces a dataset for different values of n_g and n_p specified by the user. Below, we sweep these values to search for an optimal combination to maximize predictive power of the model.

Methods

In this work, three distinct Neural Network architectures are explored. The first and most simple architecture is a simple binary classifier fully-connected network with 3 hidden layers and a sigmoid output layer that utilizes the binary cross entropy loss. The second architecture is designed to predict final scores and is forked; containing 6 fully connected layers with a fork after layer 3 and an output layer with an exponential activation for each fork. This model utilizes the summed mean squared error loss of each of the forked outputs. The final architecture contains 9 fully connected layers with a fork after layer 6. The output layer is a softmax classifier where the classes are the possible final scores of each team. In our dataset, the highest score observed is 149 and the lowest is 58, leading to two output layers of size 92 (one for each fork). This model uses the categorical cross entropy loss. A diagram of each model along with the number of nodes in each layer and expressions for their output layer activations and losses is included below.

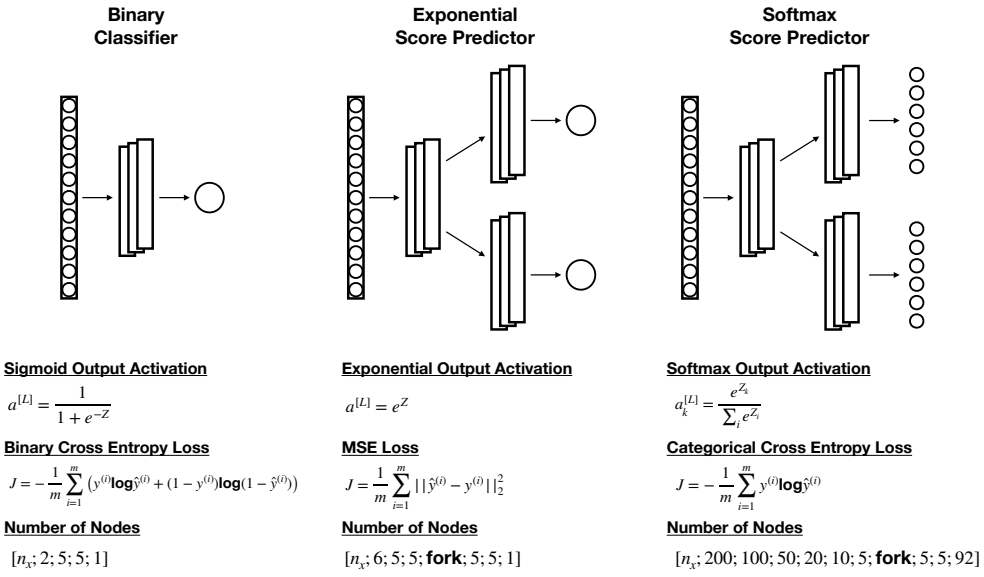

Figure 6: *NN architectures for the three model types explored in this work.*

Each model uses ReLU activations in the hidden layers. The weight kernels are initialized via Xavier Initialization and the biases are initialized at 0. The models are optimized via the Adam algorithm[12] with a minibatch size of 128, a learning rate of 0.001, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Regularization methods experimented with include L2 and L1. Due to the small size of the dataset, HPC resources were not required and the models were trained on a personal laptop with Keras[13].

Results and Discussion

Design Considerations/Hyperparameters

The first observation we need to make here is how large our dataset is in relation to the size of our feature vectors. For example, if $n_g = n_p = 10$ (yielding feature vectors of length 1700) and the first hidden layer of the neural network is just 4 units, we already have more parameters than training examples, so the model will almost certainly overfit if there is just one output. Thus, the number of nodes in each layer of each model was chosen according to the train set accuracy. Once

the game outcome prediction accuracy for the train set corresponding to $n_g = n_p = 10$ approached 90%, I ceased modifying the NN architecture. Ideally, the architecture would be optimized for each n_g/n_p pair, but that type of investigation proved to be beyond the time constraints of this project. The learning rate, Adam parameters, and number of training epochs were similarly set via an optometrist-like algorithm. Since the novelty in this project lies in the feature design, the hyperparameter tuning effort was invested in the hyperparameters associated with the features, leaving further tuning of other hyperparameters to future work.

Feature Development

To see which feature design would lead to a model with the greatest predictive power, I swept over all combinations of n_g and n_p in the range of 1 to 10 and trained each model architecture arrived at above. The results of this sweep are found below in Figure 3 where the models are evaluated according to whether or not they predicted the outcomes of the games correctly.

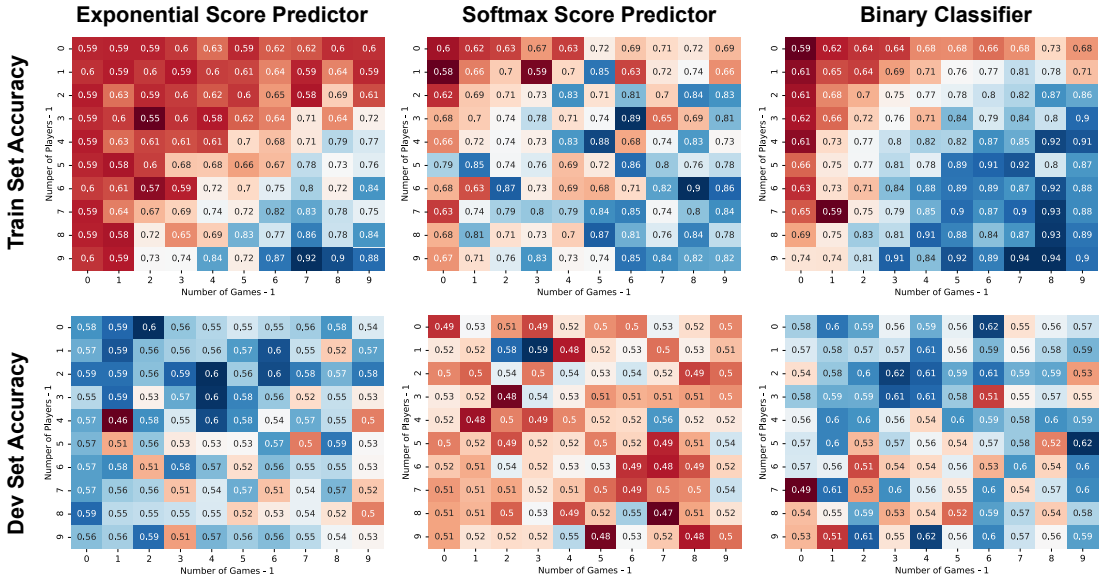

Figure 7: *Dev set performance of the 3 NN models for each n_g, n_p pair.*

We can see immediately that of the two score predictor models, the exponential score predictor is far superior, where the softmax score predictor for most combinations of n_p and n_g performs a bit better than a coin flip, occasinally approaching 60% accuracy on the dev set. The binary classifier model consistently performs in excess of 60% on the dev set, reaching the realm of human experts in performance. It is clear through evaluation of these values, however, that all three architectures have bias and variance problems, and tend to overfit to the training set for large values of n_g and n_p. Adding L2 or L1 regularization to all or a subset of the model layers does not seem to address the variance problem. I also tried modifying the complexity of the models manually to try and decrease variance in the lower right regions of these heatmaps, but ultimately the performance for other combos of n_g and n_p would suffer and the accuracy values reached would not be as high as we observe here. It is possible that adding dropout could improve model performance. Ultimately, however, it may just be a fundamental mistmatch of the train set to the dev set that causes the problems we see here. As was mentioned before, unfortunately the NBA saw quite the revolution in style of play right about at the time of the train/dev split, so any model trained on this corpus is somewhat destined to be biased.

Test Set Performance

Based on the results above, for the exponential score predictor, I tested the $n_g = 4$, $n_p = 3$ dataset; for the softmax score predictor, I tested the $n_g = 4$, $n_p = 2$ dataset; and for the binary classifier, I chose the $n_g = 4$, $n_p = 3$ dataset. Accuracies on the test set are found below:

Model	n_g,n_p pair	Train Set Accuracy	Dev Set Accuracy	Test Set Accuracy
Exponential	(4,3)	62%	60%	59.1%
Softmax	(4,2)	59%	59%	57.6%
Binary	(4,3)	75%	62%	59.8%

Thus we have some slight overfitting to the dev set, but overall we have comparable performance on the test set. In addition, we take a look at the distributions of the scores predicted by the exponential and softmax models as compared to the true test set labels.

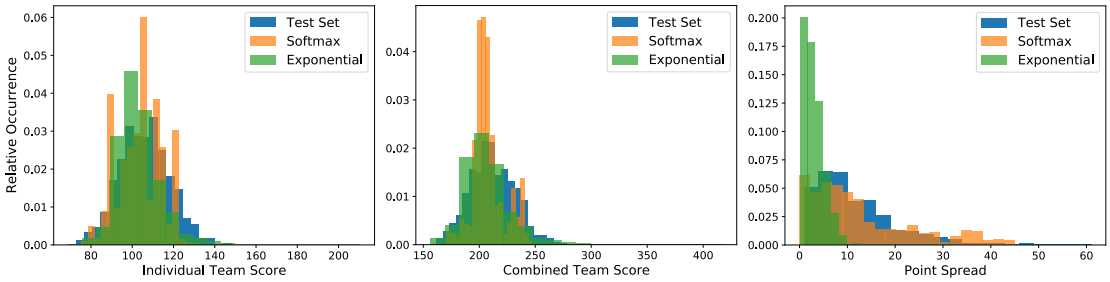

Figure 8: *Histograms for individual team scores (left), combined scores (middle) and point spreads (right) from the true test set values and the predictions from each model.*

Interestingly, the predicted score distributions produced by the exponential model look much better than the sigmoid (despite some evidence of overfitting to the train set as the mean is shifted down about 10 points as we see above), but the point spread distribution produced by the sigmoid is much more representative. This suggests that the sigmoid architecture may be more conducive to predicting point spreads while the exponential model is more suited to predicting game outcomes and over/unders, though a more rigorous investigation of these metrics is needed to say for sure.

Conclusion and Future Work

The most predictive feature design consisted of the statlines of the top 3 players of each team in their last 4 games. The binary classifier NN performed the best in predicting game outcomes, nearly achieving a performance on the test set on par with human experts. The exponential score predictor reproduced the score distribution of the test set quite well, indicating that it may be effective in predicting the over/under for NBA games. The softmax score predictor performed relatively poorly on the previous two tasks but managed to reproduce the margin of victory distribution remarkably well. Ultimately, a mismatch between the train set and the dev/test sets as well as an overall lack of data likely led to the unremarkable performance of the three models.

In the future, a number of immediate next steps along with a couple long-term steps are apparent. First, tuning of the hyperparameters associated with the optimization algorithm would likely lead to stronger results. Second, omission of some stats (like 2 pointers made and attempted, for example) along with the inclusion of other stats that might be quite predictive and weren't present in this dataset (such as +/- or unique player IDs), would likely improve performance by shrinking the size of the feature vectors and making the information present more potent. Ultimately, however, the main issue that needs to be addressed is the lack of training data and the mismatch between the train and dev/test sets. To tackle both of these problems, it may be effective to artificially generate data via the new (and quite sophisticated) NBA basketball video games.

These games simulate entire seasons of the NBA remarkably well, and using them to generate an immense dataset to train on may lead to very high prediction accuracy. Overall though, I believe the work presented here is a good first step toward considering a new dataset paradigm in NBA basketball prediction, and it is primed to be built upon.