Predicting an Optimal NBA Fantasy Lineup

Eunice Park
Advised by Ethan Meyers

A senior project submitted in partial fulfillment of the requirements of
the Bachelor of Science Degree in Statistics & Data Science

Department of Statistics & Data Science
24 Hillhouse Avenue
New Haven, CT 06511

Predicting an Optimal NBA Fantasy Lineup

Eunice Park

Abstract

In basketball, statistical analysis has always been a crucial element for assessing player performance, ranging from basic measures such as field goal shooting percentage to more comprehensive metrics for player efficiency. However, despite the availability of numerous player statistics, there is a lack of systematic use of these measures for predicting individual player performance. This type of analysis is often carried out qualitatively by amateur fans and fantasy sports enthusiasts on a daily basis. In fantasy basketball, users choose a lineup of eight players from a specified set of games while staying under a $50,000 salary cap. These users compete against each other by earning fantasy points, which are calculated based on actual game statistics. The primary objective of this project is to create a DraftKings NBA fantasy score predictor. Utilizing historical game data obtained from Basketball-Reference.com and DraftKings, a LSTM model is implemented to predict a player's total fantasy points on a given day. Using these predictions and historical data, an optimization algorithm is created to generate a lineup of 8 players with the highest expected fantasy points.

1 Introduction

In May 2018, the Supreme Court ruled that individual states had the right to legalize sports betting, which overturned a longstanding 1992 law. [1] Effectively, sports betting became legalized, and over these past few years, sports betting has been increasingly popularized in the United States. Through the assistance of online betting platforms, over one-in-five U.S. adults engage in sports betting, and such popularity is expected to increase as more states legalize commercial sports betting. [2] By 2025, the cumulative value of the American sports betting market is expected to soar to $8 billion. [3] As sports betting is becoming more accessible, the availability of data and advancement of technology have driven new opportunities to utilize machine learning algorithms to predict optimal sports betting strategies.

In particular, the fantasy sports betting space is a prime candidate to apply machine learning principles. Because of the predictive nature of selecting a high-scoring team and the large volume of data publicly available from previous sports seasons, fantasy sports provide unique opportunities for potential betting profits with the application of machine learning. As opposed to general game bets of an over/under calculation based on team-based statistics, fantasy bets integrate individualized player data in addition to

conventional game statistics, which can generate more accurate predictions of player performance. [4]

This project will examine the applications of machine learning to DraftKings NBA Daily Fantasy Sports (DFS).

DraftKings is a dominant player in the DFS space. [5] DraftKings NBA DFS allows users to draft a team lineup on a daily basis, picking players from any team playing on a particular game night. The team lineup comprises eight players, including a Point Guard, Shooting Guard, Small Forward, Power Forward, Center, general Guard (point or shooting), general Forward (small or power), and a Util (any of the five positions). Each player in the lineup has an associated salary cost, typically ranging from $3,000 to $11,000 in DraftKings currency, and users have a $50,000 salary constraint to draft their team lineup. Players earn DraftKings fantasy points based on their performance in 9 various categories. These categories are as follows: points (PTS), 3 points made (3P), total rebounds (TRB), assists (AST), steals (STL), blocks (BLK), turnovers (TOV), double doubles (DD), and triple doubles (TD). Player i's total fantasy points (yi) is defined as follows:

$$y_i = PTS_i + 0.5\ 3P_i + 1.25TRB_i + 1.25AST_i + 2STL_i + 2BLK_i - 0.5TOV + 1.5DD + 3TD$$

There is a limit of 1 DD and 1 TD for each player, so that players who achieve a DD or TD are boosted in their total fantasy points with the addition of $+1.5$(DD) and $+3$(TD). [6]

DraftKings allows users to draft a fantasy NBA lineup and have that team be entered into a pool of other teams. If that team's total number of points is higher than that of a large percentage of its competitors, the creator of that team receives a multiple of the buy-in as payout, with the potential to win much more if their team is in the top fraction of scorers in the pool. DFS allows one to curate a personalized team, and such personalization is suited for a machine learning model for fantasy value projection which can directly predict a player's fantasy score from a set of player features. In practice, a new fantasy lineup can be created for each slate of games scheduled for a particular day, following constraints of player positions and a salary cap of $50,000 on the draft, with better players costing more.

In fantasy sports betting, the crux of the problem is in trying to determine and predict fantasy scores for all players such that the most efficient lineup can be constructed in terms of fantasy value per salary unit. These defined limitations of a salary cap and player positions create an opportunity for a machine learning model to best predict which individual players can construct the most efficiently generating team. An optimal lineup will generate the highest number of fantasy points while staying within the salary and position constraints.

2 Related Work

Most related work does not center on fantasy basketball, but rather around team-based basketball game performance. Nevertheless, the paper "A Starting Point for Navigating the World of Daily Fantasy Basketball" (2017) provides a useful outline for daily fantasy basketball performance. [7] This paper developed a system for predicting the optimal DFS lineup. First, a Bayesian random effects model is implemented to predict NBA players' daily performance using individual player statistics across the 2013-2014 NBA season. The Bayesian model intakes a player's performance across their last 10 games to generate future player performance predictions. Next, using these predictions, a series of valid team lineups are constructed under DFS salary and position constraints. Finally, the optimal team lineup is chosen based on Permutation based and K-nearest neighbors approaches which identify the valid lineup with the highest total fantasy points. These optimal lineups are tested in their efficacy through simulating daily competitions over the course of the 2015-2016 season, which generated a net profit of thousands of dollars.

This project draws upon the paper "A Starting Point for Navigating the World of Daily Fantasy Basketball" (2017) for inspiration. Similar to the paper, this project first implements a model to predict NBA players' daily performance. (see 4.2) Using these predictions, this project constructs an optimal lineup with position and salary constraints (see 4.5). Though this project was unable to simulate all daily competitions over the course of a basketball season due to a time constraint, the optimal lineup generated by this project's algorithm is compared with historical optimal lineups for given game dates.

3 Dataset

3.1 Data

I obtained player box-score statistics data from Kaggle, Basketball-Reference.com, and RotoGuru.

The Kaggle dataset, titled "NBA Players and Team Data" web scrapes player statistics for all NBA seasons, ranging from 1949-50 to 2022-23, from Basketball-Reference.com. [8] The dataset has the following columns ['Season', 'Game_ID', 'Player_name', 'Team', 'Game_date', 'Matchup', 'WL', 'MIN', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'Plus_minus']. It contains a full history of how every player in the NBA, denoted by 'Player_name', performed for each of their games, denoted by 'Game_ID' and 'Game_date'. Various player statistics are detailed by 'MIN' (minutes played) and 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL', 'BLK', 'TOV', 'PF', and 'PTS'.

‘MATCHUP’ denotes the game matchup and the team results of the game are denoted by ‘WL’ and ‘Plus_minus’ for each player.

Because the Kaggle dataset did not include player positions, I needed to webscrape further data from Basketball-Reference.com. [9] The webscraped Basketball-Reference.com data includes player positions for each NBA player across NBA seasons. Player positions are important in constructing the LSTM and lineup function, as total fantasy points are variable based on player position and lineups are constructed with a player position constraint (see 4.2, 4.3, 4.5, 4.6). Furthermore, the Kaggle dataset did not have updated 2022-2023 player statistics, which resulted in having to additionally webscrape Basketball-Reference.com player statistics for the complete 2022-2023 season.

Now that all player statistics are retrieved for historical game performance, the last missing piece for lineup construction is salary. On any given game date, DraftKings publishes a salary for each player eligible to be selected for the NBA DFS lineup. These player salaries are based on their previous game performances, such that salaries typically range from $3,000 to $11,000, with better performing players commanding higher salaries. These player salaries can change on a daily basis, such that it is necessary to collect historical data of player salaries throughout each day/s of interest. Although a current optimal lineup can be constructed through simply retrieving the published current player salaries on DraftKings, DraftKings does not publish, nor have an API to retrieve, past player DFS salaries of past game dates. Past player salaries of past game dates are important to test the accuracy of the results of lineup constructions for past dates. (see 4.2) Past salaries will be used, alongside a cutoff date, to determine an optimal NBA DFS lineup for a particular game during a specific game_date (cutoff date). This optimal NBA DFS lineup will be compared with what is the best NBA DFS lineup for that specific game_date based on how players actually played that day (total_fantasy_points players accumulated during games after the fact). Past DraftKings NBA DFS salaries for the 2020-2021 season were obtained through web scraping RotoGuru. This data includes a dataframe of player name, game date, various player statistics, and player’s draft king salaries for each given game date. [10]

3.2 Data Preprocessing

First, I combined the Kaggle and Basketball-Reference.com data to make a more complete dataframe with the following information ['Season', 'Game_ID', 'Player_name', ‘Pos’, 'Team', 'Game_date', 'Matchup', 'WL', 'MIN', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'Plus_minus'], such that this dataframe included player statistics for each player

('Player_name), their positions ('Pos'), and team ('Team') for each individual game they played ('Game_date').

I grouped this dataframe by NBA season, utilizing the NBA season dates of OCT 18, 2022 - APR 9, 2023, OCT 19, 2021 - APR 10, 2022, DEC 22, 2020 - MAY 16, 2021, and OCT 22, 2019 - AUG 14, 2020 to generate 4 filtered data frames of the 2022-2023, 2021-2022, 2020-2021, and 2019-2020 NBA season. These 4 filtered data frames are represented in my code full_boxscore_data as filtered_df_1, filtered_df_2, filtered_df_3, filtered_df_4, respectively. (Appendix 7.3)

Next, I referenced which player statistics are important in the DraftKings DFS NBA total fantasy points function to ensure that the data frame had these player statistics denoted as columns. As referenced in 1, a player's (i) total fantasy points (yi) is defined as follows:
$yi = 1(PTSi) + 0.5(3Pi) + 1.25(TRBi) + 1.25(ASTi) + 2(STLi) + 2(BLKi) - 0.5(TOVi) + 1.5(DD) + 3(TD)$. DraftKings DFS NBA total fantasy points are calculated as a function of total points (PTS), 3 points made (3P), total rebounds (TRB), assists (AST), steals (STL), blocks (BLK), turnovers (TOV), double doubles (DD), and triple doubles (TD). In our dataframe, we have information for total 'PTS' (pts), 'FG3M' (3p), 'REB' (trb), 'AST' (ast), 'STL' (stl), 'BLK' (blk), and 'TOV' (tov). However, there needs to be two additional columns added which denote a player's DD and TD. A DD is defined as obtaining 10 or more in two categories of either points, rebounds, assists, steals, or blocks. (PTS, REB, AST, STL, BLK). A TD is similarly defined for 3 or more categories.

```
df = filtered1_df

df['DD'] = ((df['PTS'] >= 10) + (df['REB'] >= 10) + (df['AST'] >= 10) +
            (df['STL'] >= 10) + (df['BLK'] >= 10)) >= 2
df['DD'] = df['DD'].map({True: 'yes', False: 'no'})

df['TD'] = ((df['PTS'] >= 10) + (df['REB'] >= 10) + (df['AST'] >= 10) +
            (df['STL'] >= 10) + (df['BLK'] >= 10)) >= 3
df['TD'] = df['TD'].map({True: 'yes', False: 'no'})

df.to_csv('filtered1_df.csv', index=False)
filtered1_df = df
```

As implemented in full_boxscore_data, the code above adds two columns 'DD' and 'TD' to the dataframe, which indicates whether the player achieved a 'DD' or 'TD' during a particular game. These two new columns were added to each of the 4 filtered datasets for the NBA seasons.

Now with this dataframe of complete information of player statistics, an additional column 'total_fantasy_points' was added to each of the 4 filtered datasets. 'Total_fantasy_points' denotes how many fantasy points, in accordance with the DraftKings DFS NBA total fantasy points function, a player would have received for a particular game based on their historical performance.

```python
def calculate_total_fantasy_points(lineup):
    total_points = 0

    for player in lineup:
        points = player['points']
        made_three_pt = player['made_three_pt']
        rebounds = player['rebounds']
        assists = player['assists']
        steals = player['steals']
        blocks = player['blocks']
        turnovers = player['turnovers']
        double_double = player['double_double']
        triple_double = player['triple_double']

        total_points += points
        total_points += made_three_pt * 0.5
        total_points += rebounds * 1.25
        total_points += assists * 1.5
        total_points += steals * 2
        total_points += blocks * 2
        total_points -= turnovers * 0.5
        if double_double:
            total_points += 1.5
        if triple_double:
            total_points += 3

    return total_points
```

These preprocessing steps yielded 4 clean dataframes: filtered_1_df (2022-2023 NBA season), filtered_2_df (2021-2022 NBA season), filtered_3_df (2020-2021 NBA season), and filtered_4_df (2022-2023 NBA season). Each of these dfs had all player statistics and total fantasy points for each player's historical game performance, with each row representing each game a NBA player has played within a particular season.

4 Implementation and Results

4.1 LSTM Overview

Now that I have the appropriate dfs of complete information of player statistics, I would like to predict a player's total fantasy points using this player statistic information. I chose to implement a Long Short-Term Memory (LSTM) model to make this prediction of total fantasy points for NBA players based on their past performance.

LSTMs are a type of artificial neural network that is well-suited for analyzing sequential data and thus is a perfect candidate for predicting time-series information. They are recognized for their ability to overcome the vanishing gradient issue that often occurs in recurrent neural networks. Like the human brain, an LSTM is capable of retaining essential sequential information and storing it for a period. [11] LSTMs contain a hidden layer of memory blocks, each of which consists of a recurrent memory cell, an input/output gate, and a forget-gate, which are trainable. These memory blocks can learn to preserve or discard information. [12] In the sports domain, LSTMs have been applied to game outcome prediction [13], player efficiency ratings [13], activity recognition [14], shot trajectory prediction [15], offensive play classification[16], and more. As opposed to a simple Feed

Forward network which would make a prediction based on a single game, a LSTM model looks at a sequence of games to generate a player's total fantasy points prediction. Additionally, a player's total fantasy points prediction scores over a specific time period may be codependent as a player may be in certain shape or performance to achieve similar results across successive games. A LSTM model is well suited to learn these dependencies to generate an accurate prediction.

4.2 LSTM Implementation

The LSTM model is defined as follows:

```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, dropout_prob):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout_prob)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).requires_grad_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.dropout(out[:, -1, :])
        out = self.fc(out)
        return out
```

The LSTM model includes dropout layers to prevent from overfitting, and is trained with the following hyperparameters. Initially, the LSTM model was trained on a hidden_size = 64 and num_layers = 2. However, these hyperparameters of an increase in hidden_size = 128 and num_layers = 3 were the best parameters for the LSTM model (as calculated by the lowest loss and highest accuracy).

```
# parameters
hidden_size = 128
num_layers = 3
output_size = 1
dropout_prob = 0.5
```

Additional details for the LSTM model are found below. The LSTM model uses a learning rate scheduler to adjust the learning rate during training. Instead of a Mean Squared Error (MSE) loss function, it uses a Mean Absolute Error (MAE) loss function which is more appropriate for regression problems.

```
# model, loss function, optimizer
model = LSTMModel(input_features, hidden_size, num_layers, output_size, dropout_prob)
criterion = nn.L1Loss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# learning rate scheduler
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5,
                                                       verbose=True)

# training loop
num_epochs = 100
batch_size = 8
train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
```

With each epoch, train loss, test loss, and test accuracy is printed and appended to their corresponding lists. Test accuracy is defined by this regression function below, which denotes a predicted result as accurate if a player's predicted total fantasy points for a particular game is within a 30% threshold of their actual total fantasy points for the game.

```
def regression_accuracy(y_true, y_pred, threshold=0.3):
    assert y_true.shape == y_pred.shape, "y_true and y_pred must have the same shape"
    within_threshold = torch.abs(y_true - y_pred) <= (threshold * y_true)
    accuracy = torch.mean(within_threshold.type(torch.float32))
    return accuracy.item()
```

The input for the LSTM model initially takes the cleaned dataframe from 3.2. It then creates a new dataframe that only includes the relevant columns needed for analysis.

```
df = df[['PLAYER_NAME', 'POS', 'Team', 'GAME_DATE', 'MIN', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT',
       'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL', 'BLK',
       'TOV', 'PF', 'PTS', 'PLUS_MINUS',
       'total_fantasy_points']]
```

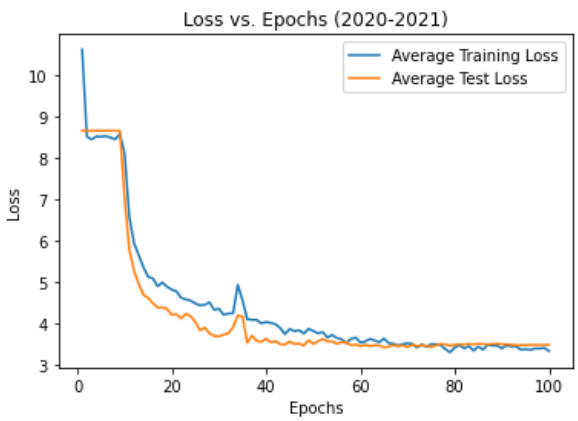
All relevant numerical features of the dataset are scaled using a Standard Scaler, and the relevant categorical features of the dataset (player position and player team) are encoded using a LabelEncoder which transforms this categorical data into numerical data which can be used by the LSTM model.

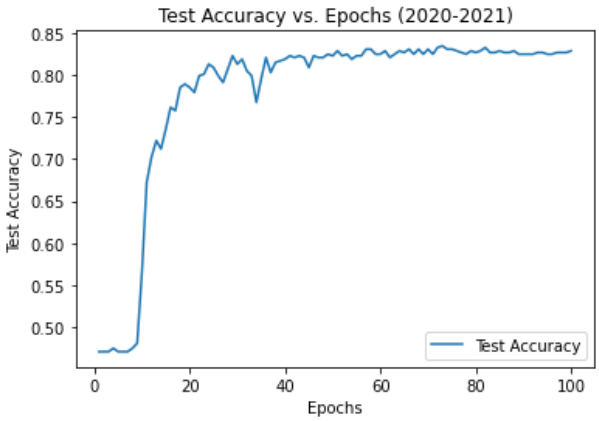
Next, a sequence of games for each player is created, considering a fixed number (50) of past games. Considering that each team plays 82 games in a typical NBA season, with each player playing 54 games on average, a sequence length of 50 is appropriate for the LSTM model. [17] Various sequence lengths of 5, 10, 15, 20, 25, 60, and 80, were experimented on the LSTM model, and a sequence length of 50 was shown to achieve the lowest loss and highest accuracy. For predictions during the early stages of an NBA season, dataframes that include the past season's results can be merged to make predictions.

Using these parameters and inputs, the LSTM model is trained.

4.3 LSTM Results

The LSTM results for the 2020-2021 NBA season are provided below. LSTM results for the rest of the NBA seasons analyzed (2019-2020, 2021-2022, 2022-2023) are provided in Appendix 7.1.





As evidenced in the Loss vs. Epochs and Test Accuracy vs. Epochs graphs, the LSTM model trains well throughout the 100 epochs with decreasing average training and test loss and increasing test accuracy. The LSTM model achieves a final test accuracy nearing 85% in predicting players' total fantasy points. The LSTM models for the rest of NBA seasons analyzed achieve similar results, with decreasing low loss and nearing around 80% accuracy. Each LSTM model took around 4 hours to complete.

Using the trained LSTM model, I can now predict the total fantasy points for each player. A function 'predict_next_game_scores' is constructed to predict a player's total fantasy points for their next upcoming game. 'predict_next_game_scores' creates a sequence of games for each player based on their latest games to return the predicted fantasy points for each player's next game. This function 'predict_next_game_scores' can be modified to include a cutoff date. Including this cutoff date will create a sequence of games for each player based on their latest games before a cutoff date to return the predicted fantasy points for each player's next game after the cutoff date. This cutoff date option will prove useful for validating lineups in 4.6, such that a predicted optimal NBA DFS lineup can be compared to the historical best lineup for that game date.

With this LSTM model, I can save various dfs. First, for each NBA season, I can save a df titled 'filtered_prediction' which is composed of all player names for the NBA season and their predicted total fantasy points for their next upcoming game. Furthermore, depending on which lineup results I wish to validate for in 4.6, I can save additional dfs with various cutoff dates. For example, in 4.6, I generated an optimal lineup for the NBA DFS game for 1/16/2021 of the 2020-2021 NBA season. In order to construct this optimal lineup, I referenced my previous saved dataframe 'filtered3_prediction_cutoff_1_16.csv' which contains players' predicted total fantasy points for their upcoming game on 1/16/2021. In practice, for current DraftKings NBA DFS contests, only the saved dataframe with 'filtered_prediction' with players and their predicted total fantasy points for their next upcoming game is useful. However, additional dfs with various cutoff dates are useful in

constructing optimal lineups that can be compared to historical best lineups for the game date.

4.4 Lineup Overview

A DraftKings NBA DFS lineup has two constraints: player positions and salary. First, the lineup of 8 players: PG, SG, SF, PF, C, G, F, UTIL must have a unique player for each position. Second, the total sum of the 8 players' salaries must be <= the salary cap of $50,000.

In order to construct the optimal lineup, three data frames will be used as inputs. The first dataframe, previously detailed in 3.1, contains historical DraftKings DFS salaries for the 2020-2021 season for each player. The second dataframe, filtered_df, contains all player statistics and total fantasy points for each player's historical game performance, with each row representing each game a NBA player has played for a certain position within a particular season. The third dataframe, 'filtered_prediction' contains players' predicted total fantasy points for their next upcoming game or their game on a specified cutoff date. These three dataframes will be merged so that the final merged_df includes information of all player names, their positions, their DFS salaries, and their player statistics for all games or all games within a specified cutoff date.

For the purposes of generating an optimal lineup that can be validated, I initially set the second dataframe as filtered_3_df which contains all player statistics and total fantasy points for each player's historical game performance, with each row representing each game a NBA player has played for a certain position within the 2020-2021 NBA season. I also set the third dataframe as 'filtered3_prediction_cutoff_1_16.csv' which contains players' predicted total fantasy points for their upcoming game on 1/16/2021. The final merged_df included information of NBA player names, their positions, their DFS salaries, and their player statistics for 2020-2021 games up till the cutoff date of 1/16/2021. This contains all the necessary information to generate an optimal NBA DFS lineup for 1/16/2021. Note that different iterations of this final merged_df were constructed, to validate additional optimal lineups for different NBA DFS dates. The outputs of these iterations were similar in accuracy to 4.6, and can be found in Appendix 7.2.

Finding an optimal lineup is computationally intensive. For example, for the NBA DFS game on 1/16/2021 alone, 140 million valid lineups can be generated that satisfy the lineup constraints. A simplistic method to solve the lineup optimization problem would be to generate all valid lineups and pick the valid lineup with the greatest total predicted fantasy points score. However, due to the extremely large number of valid lineups

generated for each NBA DFS game, as well as the fact that DFS users will benefit most from an efficient solution to generate an optimal lineup on their personal computers, an alternate solution from brute force calculation is preferable.

Therefore, it is necessary to utilize a dynamic programming-based approach to solve the problem of selecting an optimal fantasy basketball lineup. A dynamic programming-based approach, whose specific implementation is detailed in 4.5, systematically explores the solution space, guarantees an optimal solution, and can efficiently solve the problem with memoization. Memoization is an optimization technique used to store the results of expensive function calls and return the cached result when the same inputs occur again. It helps eliminate the overhead of redundant calculations by reusing previously computed results. Dynamic programming is an approach which efficiently solves a class of problems that have overlapping subproblems and optimal substructure properties. [18] In this case, a dynamic programming-based approach is ideal to solve the problem of selecting an optimal fantasy basketball lineup, as many valid lineups have overlapping unique players in the same or different player positions. Through dynamic programming, iterations of lineup costs and remaining player positions that have been calculated before return the saved memoized result.

Dynamic programming is preferred to other algorithms such as a greedy algorithm or genetic algorithm.

Greedy Algorithm vs Dynamic Programming

A greedy algorithm makes locally optimal choices at each stage, hoping to find a globally optimal solution. However, in the lineup optimization problem, a greedy approach might not always result in the best overall lineup because the globally optimal solution may require a combination of players that are not necessarily the highest scoring or cheapest individually. The greedy algorithm can get stuck in a suboptimal solution since it does not consider the interactions between player selections and player position constraints. On the other hand, the dynamic programming-based algorithm considers all possible combinations of players and positions, guaranteeing an optimal solution.

Genetic Algorithm vs Dynamic Programming

Genetic algorithms are inspired by the process of natural selection and work by evolving a population of candidate solutions over several generations. While genetic algorithms can be efficient in finding good solutions for some optimization problems, they might not be as well-suited for the lineup optimization problem due to several reasons, including but not limited to:

- Convergence Speed: Genetic algorithms usually require a large number of iterations to converge to an optimal solution. They are often used for complex problems where

the solution space is not well-understood. For the lineup optimization problem, the dynamic programming approach can find the optimal solution more quickly and deterministically by systematically exploring the solution space and memoizing partial solutions.

- Precision: Genetic algorithms work based on stochastic processes like mutation and crossover, which may not guarantee an exact optimal solution. They often find near-optimal solutions, which might not be sufficient for problems like lineup optimization where a precise optimal solution is desired. In contrast, the dynamic programming approach guarantees an optimal solution.

While a greedy algorithm or genetic algorithm could potentially be applied to the lineup optimization problem, they may not be as effective or efficient as the dynamic programming-based approach [4.5] in finding the optimal solution. The dynamic programming-based approach [4.5] guarantees the optimal lineup and efficiently solves the lineup optimization problem with memoization.

4.5 Lineup Implementation

This algorithm is a dynamic programming-based approach to solve the problem of selecting an optimal fantasy basketball lineup. It maximizes the predicted fantasy score while adhering to the salary constraint of $50,000 and player position constraint of the specific lineup structure (PG, SG, SF, PF, C, G, F, UTIL). The goal is to determine the best player selection for each player position that collectively yields the highest predicted score of total fantasy points, without exceeding the budget of $50,000.

The algorithm follows these steps:

1. It defines the player positions and organizes the player data into tuples containing their name, cost, predicted score, and valid player positions..

```
# Convert the dataframe to a list of tuples
players = merged_df_2.reset_index()[['PLAYER_NAME', 'salary', 'PRED_SCORE', 'POS']].apply(tuple, axis=1).tolist()
```

2. It sorts the list of players by their predicted total fantasy points score in descending order.
3. It initializes a dictionary for memoization to store calculated maximum scores for the current lineup.
4. It defines a recursive function, maximize_score, which takes the remaining positions, the current lineup cost, and the list of available players as input.

```
def maximize_score(remaining_roles, current_cost, players, memo, lineup):
    if not remaining_roles:
        return 0, lineup

    if current_cost > 50000:
        return float('-inf'), lineup

    if not players:
        return float('-inf'), lineup

    state = (tuple(remaining_roles), current_cost)
    if state in memo:
        return memo[state]

    player = players[0]
    name, cost, predicted_score, valid_roles = player

    remaining_players = players[1:]
```

```
    remaining_players = players[1:]

    # Calculate score by selecting the current player
    select_player_score, selected_lineup = float('-inf'), lineup
    if current_cost + cost <= 50000:
        for role in set(remaining_roles) & set(valid_roles):
            new_remaining_roles = list(remaining_roles)
            new_remaining_roles.remove(role)
            new_lineup = lineup.copy()
            new_lineup[role] = name
            score, lineup_result = maximize_score(new_remaining_roles, current_cost + cost,
                                                  remaining_players, memo, new_lineup)
            if predicted_score + score > select_player_score:
                select_player_score = score + predicted_score
                selected_lineup = lineup_result

    # Calculate score by not selecting the current player
    skip_player_score, skip_lineup = maximize_score(remaining_roles, current_cost, remaining_players,
                                                    memo, lineup)

    # Store the maximum score in the memoization cache
    if select_player_score > skip_player_score:
        memo[state] = (select_player_score, selected_lineup)
    else:
        memo[state] = (skip_player_score, skip_lineup)

    return memo[state]
```

Inside the maximize_score function, the following actions take place:

a. For invalid results: If there are no remaining positions, the function returns 0. If the current lineup cost exceeds the budget, a negative infinity score is returned as the current selection is invalid. If there are no more players available, a negative infinity score is returned.
d. For memoized results: If the current state (lineup cost and remaining positions) has been calculated before, the memoized result is returned.
e. For all other results: It initializes two variables to store the scores obtained by selecting and not selecting the current player. If the current player is eligible for any remaining positions, the score obtained by selecting them is calculated, updating the cost and remaining positions. The maximize_score function is called recursively. The score obtained by not selecting the current player is calculated by calling the maximize_score function without updating the cost and remaining positions. The maximum of the two scores is stored in the memoization cache and returned.

5. For the final step, the algorithm takes the final_merged_df (see 4.4.) as an input to generate an optimal lineup for a given NBA DFS date with the maximum total predicted fantasy score and a total cost that is <= $50,000.

In this algorithm, memoization is used to cache the maximum scores for different states, where a state is represented by the tuple of remaining player positions and the current salary cost. This is an optimal algorithm to solve the problem because it efficiently explores all possible combinations of players and player positions while avoiding duplicate calculations. By utilizing memoization, the algorithm can quickly determine the best lineup by remembering previously computed optimal scores for certain states. As a result, it can efficiently determine the lineup with the maximum predicted fantasy score while meeting the $50,000 salary constraint and player positions requirements.

To analyze the optimal lineup generated by this algorithm, Step 1 in the algorithm can be revised so that the optimal lineup of maximum predicted total fantasy points can be compared against the optimal lineup of historical total fantasy points.

```
# Convert the dataframe to a list of tuples
players = merged_df_2.reset_index()[['PLAYER_NAME', 'salary', 'total_fantasy_points', 'POS']].
apply(tuple, axis=1).tolist()
```

Revising the players list of tuples as 'total_fantasy_points' instead of 'PRED_SCORE' will generate an optimal lineup of historical total fantasy points, so that the optimal lineup is constructed after the game date based on how players ended up performing in their games with their actual total_fantasy_points. This is the ideal lineup, based on information of player performance after the game date. The optimal lineup with the maximum predicted fantasy score generated by this algorithm should be as close as possible to the ideal lineup.

4.6 Lineup Results

The NBA DFS 5/1/2021 optimal lineups are provided below. Additional NBA DFS optimal lineups for additional game dates are provided in Appendix 7.2. Each of these lineups, provided by a dynamic programming algorithm, took around 20 seconds to generate.

1/16/2021 NBA DFS Predicted Optimal Lineup (based on predicted total fantasy points)

```
Maximum Predicted Fantasy Score: 244.1683464050293
Optimal Lineup: {'PG': 'T.J. McConnell', 'SG': 'Josh Richardson', 'SF': 'Justin Holiday', 'PF': 'JaMychal Green',
'C': 'Rudy Gobert', 'G': 'Paul George', 'F': 'Justise Winslow', 'UTIL': 'Russell Westbrook'}
Total Cost of Optimal Lineup: 50000.0
```

1/16/2021 NBA DFS Best Optimal Lineup (based on historical actual total fantasy points)

```
Maximum Predicted Fantasy Score: 277.5
Optimal Lineup: {'PG': 'Cole Anthony', 'SG': 'Frank Jackson', 'SF': 'Saddiq Bey', 'PF': 'Kenyon Martin Jr.', 'C': 'Ka
rl-Anthony Towns', 'G': 'Ricky Rubio', 'F': 'Cody Martin', 'UTIL': 'Russell Westbrook'}
Total Cost of Optimal Lineup: 50000.0
```

These optimal lineup fantasy scores, generated by a dynamic programming algorithm, differ in around 30 total fantasy points. Though this number may initially seem significant, as evidenced by DraftKings NBA DFS contest results in Fantasy Cruncher [19], the minimum total fantasy score vs. the optimal (1st place) total fantasy score generally differs in 40-100 points for each NBA DFS contest. If a lineup reaches a minimum total fantasy score, the user is eligible to profit off this lineup DFS entry. Thus, within the NBA DFS for 5/1/2021, the difference of 30 between our optimal predicted lineup vs. ideal lineup falls well within the range for a minimum total fantasy score to guarantee a profit in this optimal predicted lineup construction. As seen in Appendix 7.2, additional NBA DFS optimal lineups for additional game dates exhibit a similar window between the optimal vs. ideal lineup fantasy scores. Thus, the dynamic programming algorithm generates an optimal lineup that is likely to guarantee a profit for each DFS contest.

5 Conclusion / Future Work

This project predicts an optimal fantasy lineup for DraftKings NBA DFS contest on any given game date. With historical game data obtained from Basketball-Reference.com, DraftKings, and RotoGuru, a LSTM model is constructed to predict a player's total fantasy points. With these predictions and historical data, an optimization algorithm using a dynamic programming-based approach generates a lineup of 8 players with the maximum number of predicted fantasy points.

Future work can be conducted through live testing on current DraftKings NBA DFS contests. Additionally, additional data can be gathered to further validate historical DraftKings NBA DFS contest scores. For example, Swish Analytics provides historical DraftKings NBA DFS salaries for a particular date. [19] Although these DFS salaries cannot be web scraped in bulk over a time period, these DFS salaries are still very useful in validating optimal lineups with particular game cutoff dates. Optimal lineups can further be compared with Fantasy Cruncher, which provides historical DFS contest results to see if generated optimal lineups are historically profitable. [20] Furthermore, although this project focused on constructing one optimal fantasy lineup, future work can explore the benefit of submitting multiple fantasy lineups for DraftKings NBA DFS contest, which allows a user to submit up to 150 unique lineups. [21] Similar to a portfolio optimization problem that balances risk to maximize reward, the construction of a fantasy lineup portfolio is an interesting future problem to explore.

The problem of constructing an optimal fantasy lineup is ripe for exploration. Utilizing the results of this project, a model to predict a player's total fantasy points and an algorithm that optimizes a valid lineup can be fine-tuned to increase accuracy.

6 References

[1] New York Times. (2018, May 14). Supreme Court Ruling Favors Sports Betting. https://www.nytimes.com/2018/05/14/us/politics/supreme-court-sports-betting-new-jersey.html

[2] Pew Research Center. (2022, September 14). https://www.pewresearch.org/fact-tank/2022/09/14/as-more-states-legalize-the-practice-19-of-u-s-adults-say-they-have-bet-money-on-sports-in-the-past-year/

[3] MarketWatch. (2019, November 4). https://www.marketwatch.com/story/firms-say-sports-betting-market-to-reach-8-billion-by-2025-2019-11-04#:~:text=NEW%20YORK%20(AP)%20%E2%80%94%20Investors,the%20U.S.%20within%20five%20years.

[4] Wang, J., Liu, Y., & Li, J. (2020). Predicting NBA Game Outcomes with Machine Learning. http://cs230.stanford.edu/projects_fall_2020/reports/55766293.pdf

[5] Casino.org. (2021). DraftKings DFS Dominance Is A Plus For The Stock, Says Analyst. https://www.casino.org/news/draftkings-dfs-dominance-is-a-plus-for-the-stock-says-analyst/

[6] DraftKings. (n.d.). NBA Rules. https://www.draftkings.com/help/rules/nba?wpsrc=Organic%20Search&wpaffn=Google&wpkw=https%3A%2F%2Fwww.draftkings.com%2Fhelp%2Frules%2Fnba&wpcn=help&wpscn=rules%2Fnba

[7] Stern, H. S. (2017). Sports Betting in America: An Empirical Inquiry. The American Statistician, 72(2), 139-147. doi: 10.1080/00031305.2017.1401559

[8] Lauton, L. (2021). NBA Players and Team Data. Kaggle. https://www.kaggle.com/datasets/loganlauton/nba-players-and-team-data?resource=download

[9] Basketball Reference. (2023). Steven Adams Game Log 2023. https://www.basketball-reference.com/players/a/adamsst01/gamelog/2023#all_game_log_summary

[10] Minton, J. (2021). NBA Daily Homie Data (DHD) Notes. RotoGuru. http://rotoguru1.com/hoop/nba-dhd-2021-notes.txt

[11] Vicent, J.F., Moreno, E., and Gil, D. (2020). Is the Future of Basketball Being Influenced by Predictive Data Analysis. SSRN Electronic Journal. doi: 10.2139/ssrn.4308292

[12] Sherstinsky, A. (2020) Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404:132306.

[13] Zhang, Q., Zhang, X., Hu, H., Li, C., Lin Y., and Ma, R. (2022) Sports match prediction model for training and exercise using attention-based lstm network. Digital Communications and Networks, 8(4):508–515.

[14] Chen J., Dinesh J.R., and Parathasarathy P. (2021) Lstm with bio inspired algorithm for action recognition in sports videos. Image and Vision Computing, 112:104214

[15] Zhao, Y., Yang, R., Chevalier, G., Shah, R., Romijnders, R. (2018) Applying deep bidirectional LSTM and mixture density network for basketball trajectory prediction. Optik, 158:266–272

[16] Wang, K.C., Zemel, R. (2016) Classifying NBA offensive plays using neural networks. In Proceedings of MIT Sloan Sports Analytics Conference, volume 4

[17] TrueHoop. (2020). NBA stars missed an average of 28 games in the 2019-20 season. https://www.truehoop.com/p/nba-stars-missed-an-average-of-28

[18] Programiz. (n.d.). Dynamic Programming. https://www.programiz.com/dsa/dynamic-programming

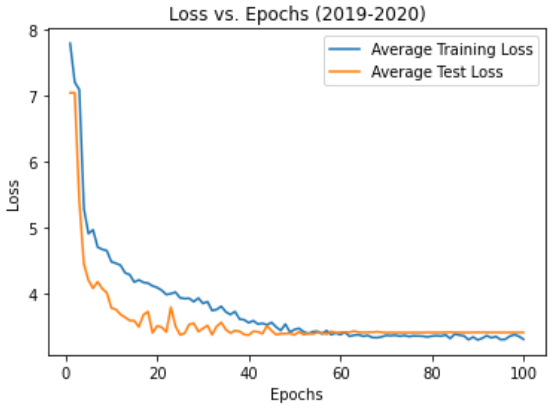
[19] Swish Analytics. (n.d.) https://swishanalytics.com/optimus/nba/fanduel-draftkings-live-scoring

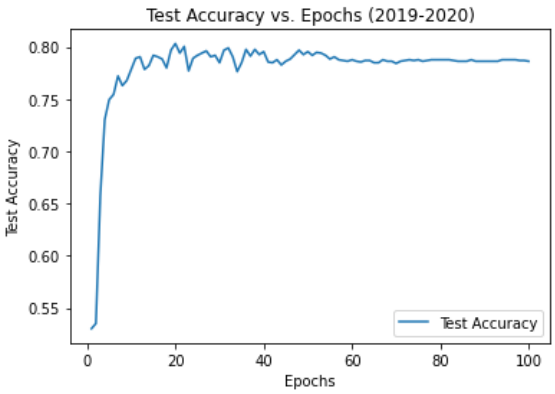
[20] Fantasy Cruncher. (n.d.) https://www.fantasycruncher.com/contest-links/NBA

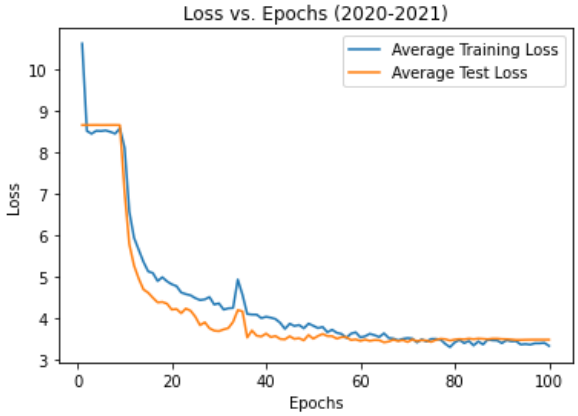
[21] DK Playbook. (2020) https://dknation.draftkings.com/nfl/2020/6/26/21309178/how-many-lineups-should-i-play

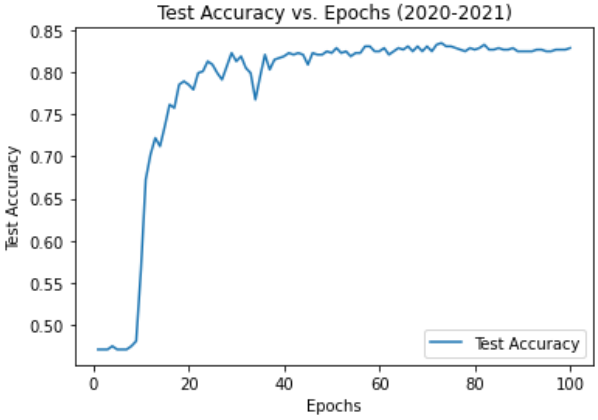
7 Appendix

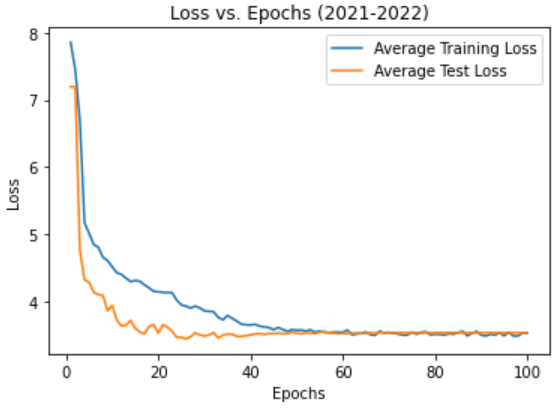
7.1 LSTM results for all seasons

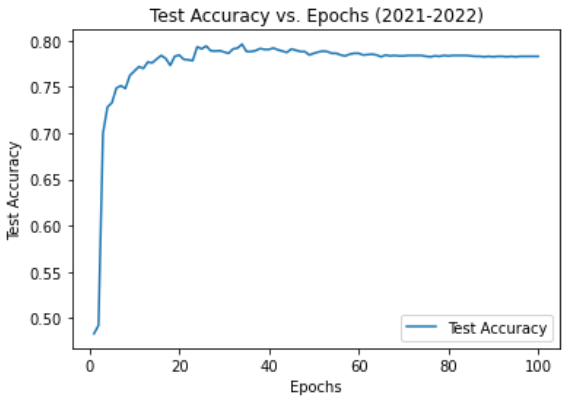


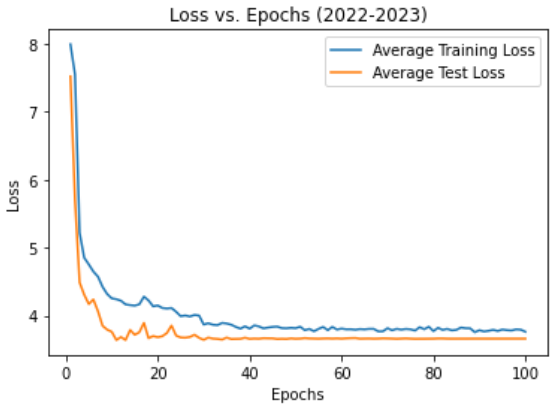


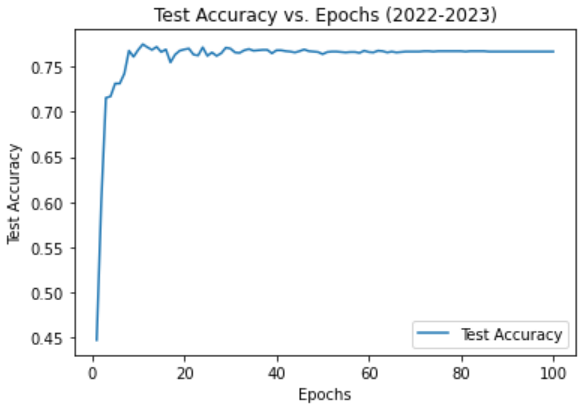












7.2 Optimal Lineups

5/1/2021 NBA DFS Predicted Optimal Lineup

```
Maximum Predicted Fantasy Score: 244.1683464050293
Optimal Lineup: {'PG': 'T.J. McConnell', 'SG': 'Josh Richardson', 'SF': 'Justin Holiday', 'PF': 'JaMychal Green',
'C': 'Rudy Gobert', 'G': 'Paul George', 'F': 'Justise Winslow', 'UTIL': 'Russell Westbrook'}
Total Cost of Optimal Lineup: 50000.0
```

5/1/2021 NBA DFS Best Optimal Lineup (based on historical actual total fantasy points)

```
Maximum Predicted Fantasy Score: 277.5
Optimal Lineup: {'PG': 'Cole Anthony', 'SG': 'Frank Jackson', 'SF': 'Saddiq Bey', 'PF': 'Kenyon Martin Jr.', 'C': 'Ka
rl-Anthony Towns', 'G': 'Ricky Rubio', 'F': 'Cody Martin', 'UTIL': 'Russell Westbrook'}
Total Cost of Optimal Lineup: 50000.0
```

1/16/2021 NBA DFS Predicted Optimal Lineup

```
Maximum Predicted Fantasy Score: 221.48938179016113
Optimal Lineup: {'PG': 'Dejounte Murray', 'SG': 'Max Strus', 'SF': 'Yuta Watanabe', 'PF': 'Ke
vin Durant', 'C': 'Bam Adebayo', 'G': 'Kyle Lowry', 'F': 'Matisse Thybulle', 'UTIL': "Devont
e' Graham"}
Total Cost of Optimal Lineup: 49500.0
```

1/16/2021 NBA DFS Best Optimal Lineup (based on historical actual total fantasy points)

```
Maximum Predicted Fantasy Score: 259.25
Optimal Lineup: {'PG': 'Ricky Rubio', 'SG': 'Frank Jackson', 'SF': 'Tim Hardaway Jr.', 'PF':
'JaMychal Green', 'C': 'Karl-Anthony Towns', 'G': 'Russell Westbrook', 'F': 'Justin Holiday',
'UTIL': 'Khem Birch'}
Total Cost of Optimal Lineup: 47500.0
```

7.3 Full Code

https://github.com/yp0328/sportbet