Winter 2023
ITC 6230 “Deep Learning”

Final Project

Option 3

Avlonitis Ektor – 216882

Table of contents

Introduction

The purpose of this project is to predict whether the Milwaukee Bucks will win or lose a game based on statistics from their roster. The project involves the use of deep learning techniques to train a model on historical data and make predictions on future games. We use data such as players' average points per game, rebounds per game, assists per game, etc., to build the model. We use techniques such as data preprocessing, feature engineering, and model selection to build the model.

We use publicly available data from various sources such as the official NBA website and third-party websites. We collect data on the Milwaukee Bucks' roster and their performance in past games. The data collected needs to be preprocessed before it can be used to train the deep learning model. We perform tasks such as data cleaning, handling missing values, scaling the data, encoding categorical variables, and feature engineering.

We use deep learning techniques such as forward neural networks (FNNs) to build the predictive model. We use Python programming language and libraries such as Tensorflow, Keras, and Pandas to build and train the model. We also use techniques such as hyperparameter tuning and regularization to optimize the model's performance.

Main Body

Data used

The nba_api library in Python was used to collect and preprocess the data for this project[1]. The code uses the library to access NBA game data and player statistics. It first finds the team ID for the Milwaukee Bucks, and then defines the seasons for which data should be retrieved.

The code then loops through each season, retrieves the game data for all games played by the Bucks in that season, and appends the data to a list. It also loops through each game and retrieves the player statistics for that game and appends the data to another list.

The code combines the game data and the Buck's player statistics for all seasons into a single dataframe, and then filters the dataframe to only include statistics for Bucks players of interest. It creates a new dataframe with game-level statistics and merges it with the original dataframe using the game ID as the key. Finally, the code saves the merged dataframe to a CSV file named 'bucks_roster_stats2.csv'.

Data processing

Data processing is a crucial step in any predictive modeling task. In our case, the data needed are selected, which includes game statistics for the two opposing teams over the last five seasons, as well as player statistics for the Milwaukee Bucks for the same period. The analysis is limited to players that are in the team's 2022/2023 roster, as we are only interested in optimizing their performances.

After retrieving the data, some of the columns are renamed and those that are not relevant for our predictive model are dropped. For instance, the columns 'SEASON_ID', 'TEAM_ID','TEAM_ABBREVIATION', 'TEAM_NAME', 'GAME_DATE', 'MATCHUP' are discarded from the game statistics, as well as 'TEAM_ID', 'TEAM_ABBREVIATION','TEAM_CITY', 'PLAYER_ID', 'NICKNAME', 'START_POSITION', and 'COMMENT' from the player statistics.

To handle the categorical column containing the players' names, a new pivot dataframe is created where each player's name becomes a column, and the corresponding statistics are the values. The multi-level column index is then flattened and new dataframe is created where each player's stats appear as separate columns, such as 'ANTETOKOUNMPO_PTS', 'ANTETOKOUNMPO_AST', 'ANTETOKOUNMPO_REB', and so on.

After obtaining the game statistics for the Milwaukee Bucks and their opposing teams, as well as the player statistics for the Bucks players for the last five seasons, these dataframes are combined into one and are saved it as a combined csv file. Some additional columns are also dropped from the combined csv, such as 'GAME_ID', 'MIN_TEAM', 'PF_TEAM', and 'PLUS_MINUS_TEAM'.

To prepare the data for our predictive model, the binary 'Win/Loss' column turns into 1s and 0s, where 1 indicates a win, and 0 a loss and the 'HOME/AWAY' column into 1s and 0s, where 1 indicates an away game and 0 indicates a home game. All the other numerical

columns are preprocessed using the preprocessing.scale method, which scales the values to a similar range, making it easier for the model to learn from them.

Finally, any missing values in the data are imputed with zeros. Through trial of different imputations like the mean, the median, the mode, the best predictions are observed when the missing values are filled with zeros. After these data processing steps, our predictive model is ready for training.

I should point out that the accuracy increased 5-10% when the opposite team statistics were added, meaning that they improve the model.

Mutual Information

Mutual information is used to rank the most relevant features in a given dataset. According to the mutual information scores, the most relevant factor for predicting the outcome of the Milwaukee Bucks games in this dataset is the number of points scored by the opposing team (PTS_OPP). This suggests that a team's defensive performance may have a stronger impact on their chances of winning than their offensive performance.

The second most important factor is the field goal percentage of the team (FG_PCT_TEAM). This indicates that a team's ability to efficiently convert their shots into points is also crucial for success.

Other important factors include the number of field goals made by the team (FGM_TEAM), the number of assists by the team (AST_TEAM), and the number of defensive rebounds by the team (DREB_TEAM). These factors are all related to a team's offensive and defensive performance, indicating that both aspects of the game are important for winning.

It's also interesting to note that some individual players' statistics appear in the list of important factors, such as George Hill's and Bobby Portis' number of three-point field goals made (FG3M_George Hill, FG3M_Bobby Portis) and Brook Lopez's number of points scored (PTS_Brook Lopez). This indicates that individual players can have a significant impact on a team's performance and success.

From the results, it is observed that offensive rebounds by Giannis Antetokounmpo have a high mutual information score, which is one of the higher scores on the list. This suggests that offensive rebounds by Giannis Antetokounmpo may be an important factor in predicting the outcome of games. Similarly, free throws made by Giannis Antetokounmpo also have a relatively high score, indicating that this may be another key factor.

However, it's important to note that the scores alone do not necessarily mean that these are the only or most important factors. The scores vary, and it is better evaluating the players that have not missed many games, since the missing values are imputed. The scores simply indicate how much mutual information each feature has with the target variable.

It's worth noting that the home/away column has a relatively lower mutual information score. This indicates that, for the Bucks, the advantage of playing at home may not be as significant.

Overall, these results provide valuable insights into the factors that contribute to winning basketball games and can be used to inform strategies for improving a team's performance.

Model – Grid Search

Firstly, a parameter grid was defined containing various hyperparameters for the model, such as the number of epochs, batch size, the dropout rate and activation functions to use. The GridSearchCV function is used from scikit-learn to search through the parameter grid and find the combination of hyperparameters that produces the highest accuracy score on the training set. The best parameters and accuracy score for the model are the following:

Best parameters: {'activation': 'sigmoid', 'batch_size': 25, 'dropout_rate': 0.2, 'epochs': 25, 'neurons': (80, 60, 40, 20), 'optimizer': 'adam'}.

Therefore, these parameters were used to build the machine learning model. A machine learning model is being built for classification. The first step involves selecting the top features from the dataset. The top 10% features are chosen and stored in a list (from 341 to 35 columns). Then, using this list a new dataset is formed with only the top 10%-features columns.

Next, the datasets are split into train and test sets using the train_test_split function from sklearn. The model architecture is defined using tf.keras.models.Sequential. It has five dense layers, which are split into the four hidden layers with sigmoid activation functions and 80, 60, 40 and 20 neurons respectively and the output layer with a softmax activation function with the number of neurons equal to the number of classes. A dropout layer with a rate of 0.2 is added after each dense layer to prevent overfitting.

The model is compiled using the Adam optimizer and categorical cross-entropy loss function. The model is trained on the training data for 25 epochs with a batch size of 25 using the fit method. After training, the model is evaluated on the test data using the evaluate method, and the test loss and accuracy are printed.

The script also plots the training and validation loss and accuracy for 100 epochs using the matplotlib.pyplot module. The plots helps in visualizing the performance of the model and identifying any overfitting issues.

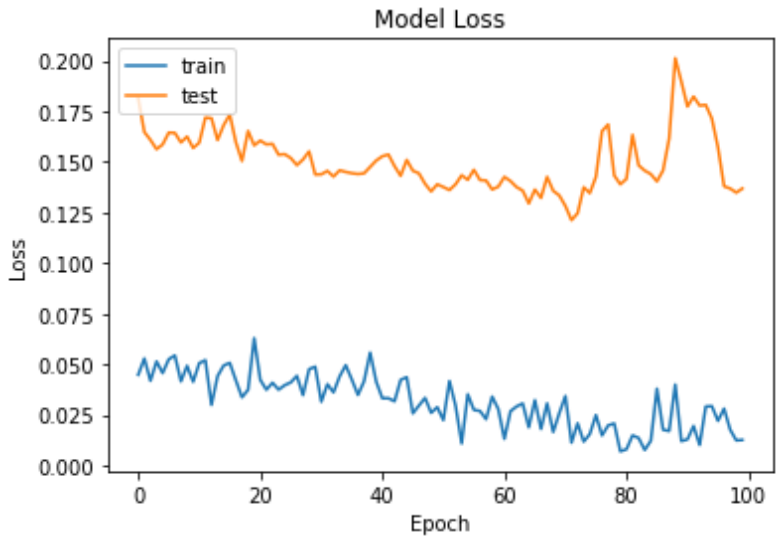

Figure 1. Training and validation loss and accuracy for each epoch.

The validation loss decreases along with the training loss, suggesting that the model is not overfitting and is generalizing well to new data.

Results

When the model.py python file is run, the outcomes for the 25 epochs are the following:

```
Epoch 1/25
16/16 [==============================] - 3s 8ms/step - loss: 0.7752 - accuracy: 0.4468
Epoch 2/25
16/16 [==============================] - 0s 6ms/step - loss: 0.7016 - accuracy: 0.5691
Epoch 3/25
16/16 [==============================] - 0s 7ms/step - loss: 0.6870 - accuracy: 0.6117
Epoch 4/25
16/16 [==============================] - 0s 8ms/step - loss: 0.6625 - accuracy: 0.6117
Epoch 5/25
16/16 [==============================] - 0s 9ms/step - loss: 0.6486 - accuracy: 0.6596
Epoch 6/25
16/16 [==============================] - 0s 9ms/step - loss: 0.6566 - accuracy: 0.6436
Epoch 7/25
16/16 [==============================] - 0s 9ms/step - loss: 0.6456 - accuracy: 0.6303
Epoch 8/25
16/16 [==============================] - 0s 9ms/step - loss: 0.6454 - accuracy: 0.6383
Epoch 9/25
16/16 [==============================] - 0s 9ms/step - loss: 0.6332 - accuracy: 0.6489
Epoch 10/25
16/16 [==============================] - 0s 8ms/step - loss: 0.5921 - accuracy: 0.6596
```

Figure 2. Training the model with 25 epochs

```
Epoch 11/25
16/16 [==============================] - 0s 8ms/step - loss: 0.5641 - accuracy: 0.7128
Epoch 12/25
16/16 [==============================] - 0s 8ms/step - loss: 0.5266 - accuracy: 0.7766
Epoch 13/25
16/16 [==============================] - 0s 9ms/step - loss: 0.4530 - accuracy: 0.8191
Epoch 14/25
16/16 [==============================] - 0s 9ms/step - loss: 0.4113 - accuracy: 0.8644
Epoch 15/25
16/16 [==============================] - 0s 9ms/step - loss: 0.3695 - accuracy: 0.9069
Epoch 16/25
16/16 [==============================] - 0s 8ms/step - loss: 0.3227 - accuracy: 0.9176
Epoch 17/25
16/16 [==============================] - 0s 8ms/step - loss: 0.2853 - accuracy: 0.9176
Epoch 18/25
16/16 [==============================] - 0s 8ms/step - loss: 0.2541 - accuracy: 0.9335
Epoch 19/25
16/16 [==============================] - 0s 8ms/step - loss: 0.2287 - accuracy: 0.9441
Epoch 20/25
16/16 [==============================] - 0s 9ms/step - loss: 0.2181 - accuracy: 0.9548
Epoch 21/25
16/16 [==============================] - 0s 9ms/step - loss: 0.2131 - accuracy: 0.9495
Epoch 22/25
16/16 [==============================] - 0s 9ms/step - loss: 0.2006 - accuracy: 0.9388
Epoch 23/25
16/16 [==============================] - 0s 8ms/step - loss: 0.1810 - accuracy: 0.9548
Epoch 24/25
16/16 [==============================] - 0s 8ms/step - loss: 0.1805 - accuracy: 0.9548
Epoch 25/25
16/16 [==============================] - 0s 8ms/step - loss: 0.1471 - accuracy: 0.9654
3/3 [==============================] - 1s 4ms/step - loss: 0.1552 - accuracy: 0.9468
```

Figure 3. Training the model with 25 epochs

During the training phase, the model's accuracy increased from 0.4468 in the first epoch to 0.9654 in the final epoch. The loss value also decreased during the training phase.

After training, the model was tested using a separate dataset, and achieved an accuracy of **0.9468** and a loss of 0.1552.

```
[test loss, test acc]= [0.15524272620677948, 0.9468085169792175]
```

A classification report is created also, that shows the precision, recall, F1-score, and support for each class, as well as their weighted average, and a confusion matrix.

	precision	recall	f1-score	support
0	1.00	0.90	0.95	20
1	0.97	1.00	0.99	74
accuracy			0.98	94
macro avg	0.99	0.95	0.97	94
weighted avg	0.98	0.98	0.98	94

Figure 4. Classification Report

```
[[18  2]
 [ 0 74]]
```

Figure 5. Confusion matrix

The model is performing quite well, achieving an overall accuracy of 98% on the test set. The precision and recall scores for both classes are also quite high, with the model achieving a precision of 100% and 97% for the negative and positive classes, respectively, and a recall of 90% and 100%, respectively. The f1-score, which is a harmonic mean of precision and recall, is also high for both classes, with the negative class achieving a score of 0.95 and the positive class achieving a score of 0.99. These results indicate that the model is doing a good job of accurately predicting both positive and negative outcomes for the dataset. However, it's important to keep in mind that these results are based on a single evaluation and may not necessarily be representative of the model's performance in all cases.

Conclusions

In order to prevent overfitting and for our model to improve, certain steps and adjustments were followed. Our model is improved when:

- The dropout rate is added.
- The top features are selected.
- The statistics of the opposing team are added.
- The home/away column is added. Whether the game was home court or away.
- The players from this year's roster are kept.
- Grid search was used to optimize the parameters.

Some limitations of the current model and potential areas for future improvement are the following. Firstly, the dataset is limited. The model is trained on data from only five seasons, which may not be sufficient to capture all the factors that influence the outcome of an NBA game. Future work could include featuring data from more seasons or other sources, such as social media or expert opinions. Also, the data were not real-time data. The model is based on historical data and does not consider real-time data, such as injuries or changes in team rosters. Future work could include combining real-time data to improve the accuracy of predictions.

Also, in the future other models can be explored, such as decision trees or rule-based models, that provide greater interpretability and precision.

Word count: 2012

Bibliography

1. nba_api. (2023). nba_api documentation. GitHub. Retrieved April 8, 2023, from https://github.com/swar/nba_api
2. National Basketball Association. (n.d.). NBA.com. Retrieved April 8, 2023, from https://www.nba.com/

Appendix

Three python files were created. After each one, I have imported a screenshot.

1.

data_collection.py

```
# BUCKS STATS

from nba_api.stats.static import teams
from nba_api.stats.endpoints import leaguegamefinder
import pandas as pd
import time
# Get the team ID for the Milwaukee Bucks
bucks_id = teams.find_teams_by_full_name('Milwaukee Bucks')[0]['id']

# Define the seasons you want to get data for
seasons = ['2018-19', '2019-20', '2020-21', '2021-22', '2022-23']

# Create an empty list to store the data for each season
data = []
# Loop through each season and retrieve the data for all Bucks games
for season in seasons:
    gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=bucks_id, season_nullable=season)
    games = gamefinder.get_data_frames()[0]
    data.append(games)
    time.sleep(1)

#win_or_lose = games['WL']

# Combine the data for all seasons into a single DataFrame
bucks_games = pd.concat(data, ignore_index=True)
```

```
for i in range(0, len(bucks_games['MATCHUP'])):
    if '@' in bucks_games['MATCHUP'][i]:
        bucks_games['MATCHUP'][i] = 'Away'
    else:
        bucks_games['MATCHUP'][i] = 'Home'


bucks_games = bucks_games.drop(['SEASON_ID', 'TEAM_ID','TEAM_ABBREVIATION',
'TEAM_NAME',
                        'GAME_DATE'
                        ], axis=1)


bucks_games = bucks_games.rename(columns={'PTS': 'PTS_TEAM', 'MIN': 'MIN_TEAM',
'FGM': 'FGM_TEAM',
                                'FGA': 'FGA_TEAM', 'FG_PCT': 'FG_PCT_TEAM', 'FG3M':
'FG3M_TEAM',
                                'FG3A': 'FG3A_TEAM', 'FG3_PCT': 'FG3_PCT_TEAM', 'FTM':
'FTM_TEAM',
                                'FTA': 'FTA_TEAM', 'FT_PCT': 'FT_PCT_TEAM', 'OREB':
'OREB_TEAM',
                                'DREB': 'DREB_TEAM', 'REB': 'REB_TEAM', 'AST':
'AST_TEAM',
                                'STL': 'STL_TEAM', 'BLK': 'BLK_TEAM', 'PF': 'PF_TEAM',
                                'PLUS_MINUS': 'PLUS_MINUS_TEAM', 'TOV':
'TOV_PCT_TEAM',
                                'MATCHUP': 'HOME/AWAY'})


# Retrieve the player statistics for each game
from nba_api.stats.endpoints import boxscoretraditionalv2
player_stats = []
opposing_team_stats = []


for game_id in bucks_games['GAME_ID']:
    stats = boxscoretraditionalv2.BoxScoreTraditionalV2(game_id=game_id)
    stats_df = stats.get_data_frames()[0]
```

```
    # Retrieve the statistics for both teams in the game
    team_stats = stats.get_data_frames()[1]

    if team_stats['TEAM_ID'][0] == bucks_id:
        opposing_team_num = 1
    else:
        opposing_team_num = 0

    # Append the stats for the opposing team to the opposing_team_stats list
    opposing_team_stats.append(team_stats.iloc[opposing_team_num].to_frame().T)
    # Append the stats for the Bucks players to the player_stats list
    player_stats.append(stats_df)

    time.sleep(2)

# Combine the player statistics for all games into a single DataFrame
bucks_stats = pd.concat(player_stats, ignore_index=True)

# Combine the opposing team statistics for all games into a single DataFrame
opposing_team_stats_df = pd.concat(opposing_team_stats, ignore_index=True)

# Filter the DataFrame to only include Bucks players
bucks_roster_stats = bucks_stats[bucks_stats['TEAM_ID'] == bucks_id]
#other_team_stats = bucks_stats[bucks_stats['TEAM_ID'] != bucks_id]

bucks_roster_stats = bucks_roster_stats.drop(['TEAM_ID',
'TEAM_ABBREVIATION','TEAM_CITY', 'PLAYER_ID',
                                'NICKNAME', 'START_POSITION', 'COMMENT'
                                ], axis=1)

# Filter the opposing team statistics DataFrame to only include the columns we want
opposing_team_stats_df = opposing_team_stats_df[['GAME_ID', 'PTS', 'FGM', 'FGA',
'FG_PCT', 'FG3M', 'FG3A',
```

```
                                'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB',
                                'AST', 'STL', 'BLK', 'TO', 'PF']]
# Rename the columns in the opposing team statistics DataFrame to match the column names in the Bucks statistics DataFrame
opposing_team_stats_df = opposing_team_stats_df.rename(columns={'PTS': 'PTS_OPP', 'FGM': 'FGM_OPP', 'FGA': 'FGA_OPP',
                                                                'FG_PCT': 'FG_PCT_OPP', 'FG3M': 'FG3M_OPP', 'FG3A': 'FG3A_OPP',
                                                                'FG3_PCT': 'FG3_PCT_OPP', 'FTM': 'FTM_OPP', 'FTA': 'FTA_OPP',
                                                                'FT_PCT': 'FT_PCT_OPP', 'OREB': 'OREB_OPP', 'DREB': 'DREB_OPP',
                                                                'REB': 'REB_OPP', 'AST': 'AST_OPP', 'STL': 'STL_OPP',
                                                                'BLK': 'BLK_OPP', 'TO': 'TOV_OPP', 'PF': 'PF_OPP'})
players_of_interest = ['Giannis Antetokounmpo', 'Khris Middleton', 'Jrue Holiday',
                       'Brook Lopez', 'Pat Connaughton', 'Bobby Portis',
                       'George Hill', 'Wesley Matthews', 'Jordan Nwora',
                       'Thanasis Antetokounmpo', 'Jevon Carter', 'Serge Ibaka',
                       'Sandro Mamukelashvili', 'Grayson Allen', 'MarJon Beauchamp',
                       'Joe Ingles', 'AJ Green']


bucks_roster_stats = bucks_roster_stats.loc[bucks_roster_stats['PLAYER_NAME'].isin(players_of_interest)]
# create a new dataframe with game-level statistics
new_df = pd.pivot(bucks_roster_stats, index=['GAME_ID'], columns='PLAYER_NAME', values=['FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A',
    'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL',
    'BLK', 'TO', 'PF', 'PTS'])


# flatten the multi-level column index
new_df.columns = ['_'.join(col).rstrip('_') for col in new_df.columns.values]


# reset the index to make game ID and team abbreviation columns
```

new_df = new_df.reset_index()

merged_df = pd.merge(bucks_games, new_df, on='GAME_ID', how='left')

merged_df2 = pd.merge(merged_df, opposing_team_stats_df, on='GAME_ID', how='left')

merged_df2.to_csv('bucks_roster_stats2.csv', index=False)

```
from nba_api.stats.static import teams
from nba_api.stats.endpoints import leaguegamefinder
import pandas as pd
import time

# Get the team ID for the Milwaukee Bucks
bucks_id = teams.find_teams_by_full_name('Milwaukee Bucks')[0]['id']

# Define the seasons you want to get data for
seasons = ['2018-19', '2019-20', '2020-21', '2021-22', '2022-23']

# Create an empty list to store the data for each season
data = []

# Loop through each season and retrieve the data for all Bucks games
for season in seasons:
    gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=bucks_id, season_nullable=season)
    games = gamefinder.get_data_frames()[0]
    data.append(games)
    time.sleep(1)


#win_or_lose = games['WL']

# Combine the data for all seasons into a single DataFrame
bucks_games = pd.concat(data, ignore_index=True)

for i in range(0, len(bucks_games['MATCHUP'])):
    if '@' in bucks_games['MATCHUP'][i]:
        bucks_games['MATCHUP'][i] = 'Away'
    else:
        bucks_games['MATCHUP'][i] = 'Home'

bucks_games = bucks_games.drop(['SEASON_ID', 'TEAM_ID','TEAM_ABBREVIATION', 'TEAM_NAME',
                                'GAME_DATE'
                                ], axis=1)

bucks_games = bucks_games.rename(columns={'PTS': 'PTS_TEAM', 'MIN': 'MIN_TEAM', 'FGM': 'FGM_TEAM',
                                          'FGA': 'FGA_TEAM', 'FG_PCT': 'FG_PCT_TEAM', 'FG3M': 'FG3M_TEAM',
                                          'FG3A': 'FG3A_TEAM', 'FG3_PCT': 'FG3_PCT_TEAM', 'FTM': 'FTM_TEAM',
                                          'FTA': 'FTA_TEAM', 'FT_PCT': 'FT_PCT_TEAM', 'OREB': 'OREB_TEAM',
                                          'DREB': 'DREB_TEAM', 'REB': 'REB_TEAM', 'AST': 'AST_TEAM',
                                          'STL': 'STL_TEAM', 'BLK': 'BLK_TEAM', 'PF': 'PF_TEAM',
                                          'PLUS_MINUS': 'PLUS_MINUS_TEAM', 'TOV': 'TOV_PCT_TEAM',
                                          'MATCHUP': 'HOME/AWAY'})

# Retrieve the player statistics for each game
from nba_api.stats.endpoints import boxscoretraditionalv2
player_stats = []
opposing_team_stats = []

for game_id in bucks_games['GAME_ID']:
    stats = boxscoretraditionalv2.BoxScoreTraditionalV2(game_id=game_id)
    stats_df = stats.get_data_frames()[0]
    # Retrieve the statistics for both teams in the game
    team_stats = stats.get_data_frames()[1]
```

```
    if team_stats['TEAM_ID'][0] == bucks_id:
        opposing_team_num = 1
    else:
        opposing_team_num = 0

    # Append the stats for the opposing team to the opposing_team_stats list
    opposing_team_stats.append(team_stats.iloc[opposing_team_num].to_frame().T)

    # Append the stats for the Bucks players to the player_stats list
    player_stats.append(stats_df)

    time.sleep(2)

# Combine the player statistics for all games into a single DataFrame
bucks_stats = pd.concat(player_stats, ignore_index=True)
# Combine the opposing team statistics for all games into a single DataFrame
opposing_team_stats_df = pd.concat(opposing_team_stats, ignore_index=True)
# Filter the DataFrame to only include Bucks players
bucks_roster_stats = bucks_stats[bucks_stats['TEAM_ID'] == bucks_id]
#other_team_stats = bucks_stats[bucks_stats['TEAM_ID'] != bucks_id]
bucks_roster_stats = bucks_roster_stats.drop(['TEAM_ID', 'TEAM_ABBREVIATION','TEAM_CITY', 'PLAYER_ID',
                                              'NICKNAME', 'START_POSITION', 'COMMENT'
                                             ], axis=1)

# Filter the opposing team statistics DataFrame to only include the columns we want
opposing_team_stats_df = opposing_team_stats_df[['GAME_ID', 'PTS', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A',
                                                 'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB',
                                                 'AST', 'STL', 'BLK', 'TO', 'PF']]
# Rename the columns in the opposing team statistics DataFrame to match the column names in the Bucks statistics DataFrame
opposing_team_stats_df = opposing_team_stats_df.rename(columns={'PTS': 'PTS_OPP', 'FGM': 'FGM_OPP', 'FGA': 'FGA_OPP',
                                                                'FG_PCT': 'FG_PCT_OPP', 'FG3M': 'FG3M_OPP', 'FG3A': 'FG3A_OPP',
                                                                'FG3_PCT': 'FG3_PCT_OPP', 'FTM': 'FTM_OPP', 'FTA': 'FTA_OPP',
                                                                'FT_PCT': 'FT_PCT_OPP', 'OREB': 'OREB_OPP', 'DREB': 'DREB_OPP',
                                                                'REB': 'REB_OPP', 'AST': 'AST_OPP', 'STL': 'STL_OPP',
                                                                'BLK': 'BLK_OPP', 'TO': 'TOV_OPP', 'PF': 'PF_OPP'})

players_of_interest = ['Giannis Antetokounmpo', 'Khris Middleton', 'Jrue Holiday',
                       'Brook Lopez', 'Pat Connaughton', 'Bobby Portis',
                       'George Hill', 'Wesley Matthews', 'Jordan Nwora',
                       'Thanasis Antetokounmpo', 'Jevon Carter', 'Serge Ibaka',
                       'Sandro Mamukelashvili', 'Grayson Allen', 'MarJon Beauchamp',
                       'Joe Ingles', 'AJ Green']

bucks_roster_stats = bucks_roster_stats.loc[bucks_roster_stats['PLAYER_NAME'].isin(players_of_interest)]

# create a new dataframe with game-level statistics
new_df = pd.pivot(bucks_roster_stats, index=['GAME_ID'], columns='PLAYER_NAME', values=['FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A',
       'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'STL',
       'BLK', 'TO', 'PF', 'PTS'])
# flatten the multi-level column index
new_df.columns = ['_'.join(col).rstrip('_') for col in new_df.columns.values]
# reset the index to make game ID and team abbreviation columns
new_df = new_df.reset_index()
merged_df = pd.merge(bucks_games, new_df, on='GAME_ID', how='left')
merged_df2 = pd.merge(merged_df, opposing_team_stats_df, on='GAME_ID', how='left')
merged_df2.to_csv('bucks_roster_stats2.csv', index=False)
```

2.

grid_search.py

Data preprocessing

from sklearn.feature_selection import mutual_info_classif

import pandas as pd

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

import tensorflow as tf

from sklearn.model_selection import GridSearchCV

from keras.wrappers.scikit_learn import KerasClassifier

import numpy as np

df = pd.read_csv("bucks_roster_stats2.csv")

```
df = df.drop(columns = ['GAME_ID', 'MIN_TEAM', 'PF_TEAM', 'PLUS_MINUS_TEAM'])

loc = df['HOME/AWAY']
loc.replace('Home', 0, inplace=True)
loc.replace('Away', 1, inplace=True)

#df.dropna(inplace=True)
y=df['WL']
y.replace('L',0,inplace=True)
y.replace('W',1,inplace=True)
Y = tf.keras.utils.to_categorical(y)

#df.fillna(df.mean(), inplace=True)
#df.fillna(df.median(), inplace=True)
#df.fillna(df.mode(), inplace=True)
df.fillna(0, inplace=True)

#x = df.iloc[:,1:]
x = df.drop(columns = ['WL'])
x = preprocessing.scale(x)

column_names = list(df.columns)
column_names = ['HOME/AWAY'] + column_names[2:]

information = mutual_info_classif(x,y)
print('Information=',information)

feature_info = {i: info for i, info in enumerate(information)}
sorted_info = sorted(feature_info.items(), key=lambda x: x[1], reverse=True)

print("Ranked features based on mutual information scores:")
for i, info in sorted_info:
```

```
    print(f"{column_names[i]}: {info}")

top_features = np.where(information >= np.percentile(information, 75), True, False)
top_features = list(top_features)

X = x[:, top_features]
print(X.shape[1])



x_train, x_test, y_train, y_test = train_test_split (X,Y, test_size=0.2, random_state=1)

N_train = len(x_train[0])
K = Y.shape[1]

# define the parameter grid
param_grid = {
    'epochs': [25, 50, 100, 150],
    'batch_size': [25, 50, 100, 150],
    'optimizer': ['adam', 'sgd'],
    'activation': ['sigmoid', 'relu', 'tanh'],
    'neurons': [(80, 60, 40, 20), (130, 100, 50, 20), (200, 150, 100, 50)],
    'dropout_rate': [0.2, 0.3, 0.4, 0.5]
}

# define the model function to be used by GridSearchCV
def create_model(neurons=(20, 40, 60, 80), activation='sigmoid',
optimizer='adam',dropout_rate=0.5):
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(neurons[0], activation=activation, input_dim=N_train),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[1], activation=activation),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[2], activation=activation),
```

```
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[3], activation=activation),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(K, activation='softmax')
    ])
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

keras_model = KerasClassifier(build_fn=create_model, verbose=1)
# create the GridSearchCV object
grid_search = GridSearchCV(estimator=keras_model,
                           param_grid=param_grid,
                           cv=3,
                           n_jobs=-1,
                           scoring='accuracy')

# perform the grid search
grid_search.fit(x_train, y_train)

# print the best parameters and score
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best accuracy: {grid_search.best_score_}")
```

```python
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
import numpy as np

df = pd.read_csv("bucks_roster_stats2.csv")
df = df.drop(columns = ['GAME_ID', 'MIN_TEAM', 'PF_TEAM', 'PLUS_MINUS_TEAM'])

loc = df['HOME/AWAY']
loc.replace('Home', 0, inplace=True)
loc.replace('Away', 1, inplace=True)

#df.dropna(inplace=True)
y=df['WL']
y.replace('L',0,inplace=True)
y.replace('W',1,inplace=True)
Y = tf.keras.utils.to_categorical(y)

#df.fillna(df.mean(), inplace=True)
#df.fillna(df.median(), inplace=True)
#df.fillna(df.mode(), inplace=True)
df.fillna(0, inplace=True)

#x = df.iloc[:,1:]
x = df.drop(columns = ['WL'])
x = preprocessing.scale(x)

column_names = list(df.columns)
column_names = ['HOME/AWAY'] + column_names[2:]

information = mutual_info_classif(x,y)
print('Information=',information)


feature_info = {i: info for i, info in enumerate(information)}

sorted_info = sorted(feature_info.items(), key=lambda x: x[1], reverse=True)

print("Ranked features based on mutual information scores:")
for i, info in sorted_info:
    print(f"{column_names[i]}: {info}")

top_features = np.where(information >= np.percentile(information, 75), True, False)
top_features = list(top_features)

X = x[:, top_features]
print(X.shape[1])
```

```
x_train, x_test, y_train, y_test = train_test_split (X,Y, test_size=0.2, random_state=1)

N_train = len(x_train[0])
K = Y.shape[1]

# define the parameter grid
param_grid = {
    'epochs': [25, 50, 100, 150],
    'batch_size': [25, 50, 100, 150],
    'optimizer': ['adam', 'sgd'],
    'activation': ['sigmoid', 'relu', 'tanh'],
    'neurons': [(80, 60, 40, 20), (130, 100, 50, 20), (200, 150, 100, 50)],
    'dropout_rate': [0.2, 0.3, 0.4, 0.5]
}

# define the model function to be used by GridSearchCV
def create_model(neurons=(20, 40, 60, 80), activation='sigmoid', optimizer='adam',dropout_rate=0.5):
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(neurons[0], activation=activation, input_dim=N_train),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[1], activation=activation),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[2], activation=activation),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(neurons[3], activation=activation),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(K, activation='softmax')
    ])
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

keras_model = KerasClassifier(build_fn=create_model, verbose=1)
# create the GridSearchCV object
grid_search = GridSearchCV(estimator=keras_model,
                           param_grid=param_grid,
                           cv=3,
                           n_jobs=-1,
                           scoring='accuracy')

# perform the grid search
grid_search.fit(x_train, y_train)

# print the best parameters and score
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best accuracy: {grid_search.best_score_}")
```

3.

model.py

Data preprocessing

from sklearn.feature_selection import mutual_info_classif

import pandas as pd

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

import tensorflow as tf

from sklearn.decomposition import PCA

import numpy as np

df = pd.read_csv("bucks_roster_stats2.csv")

df = df.drop(columns = ['GAME_ID', 'MIN_TEAM', 'PF_TEAM', 'PLUS_MINUS_TEAM'])

```
loc = df['HOME/AWAY']
loc.replace('Home', 0, inplace=True)
loc.replace('Away', 1, inplace=True)

#df.dropna(inplace=True)
y=df['WL']
y.replace('L',0,inplace=True)
y.replace('W',1,inplace=True)
Y = tf.keras.utils.to_categorical(y)

df.to_csv('processed_data2.csv', index=False)

#df.fillna(df.mean(), inplace=True)
#df.fillna(df.median(), inplace=True)
#df.fillna(df.mode(), inplace=True)
df.fillna(0, inplace=True)

#x = df.iloc[:,1:]
x = df.drop(columns = ['WL'])
x = preprocessing.scale(x)

#df.to_csv('processed_data2.csv', index=False)

'''
# Apply PCA
pca = PCA(n_components=80) # specify the number of components to keep
x = pca.fit_transform(x)
'''
```

```
# Performing mutual information
column_names = list(df.columns)
column_names = ['HOME/AWAY'] + column_names[2:]

information = mutual_info_classif(x,y)
print('Information=',information)

feature_info = {i: info for i, info in enumerate(information)}

sorted_info = sorted(feature_info.items(), key=lambda x: x[1], reverse=True)

print("Ranked features based on mutual information scores:")
for i, info in sorted_info:
   print(f"{column_names[i]}: {info}")

top_features = np.where(information >= np.percentile(information, 90), True, False)
top_features = list(top_features)

X = x[:, top_features]
print(X.shape[1])

# Split data to train and test
x_train, x_test, y_train, y_test = train_test_split (X,Y, test_size=0.2, random_state=1)

N_train = len(x_train[0])
K = Y.shape[1]
# Insert a dropout rate
dropout_rate = 0.2

# Use keras to create the model, with the best parameters from grid search
```

```
my_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(80, activation=tf.nn.sigmoid, input_dim=N_train),
    tf.keras.layers.Dropout(dropout_rate),  # Add dropout layer with rate of 0.2
    tf.keras.layers.Dense(60, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(40, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(20, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(K, activation=tf.nn.softmax)
])



my_model.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])


my_model.fit(x_train, y_train, epochs=25, batch_size=25)
res = my_model.evaluate(x_test, y_test)
my_model.save('my_model.h5')
print('[test loss, test acc]=', res)


# Train the model
history = my_model.fit(x_train, y_train, epochs=100, batch_size=25, validation_data=(x_test,
y_test))


y_pred = my_model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
# Print classification report and confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(np.argmax(y_test,axis=1), y_pred))
```

```
print(confusion_matrix(np.argmax(y_test,axis=1), y_pred))

# Evaluate the model on the test data
test_loss, test_acc = my_model.evaluate(x_test, y_test)

print('Test accuracy:', test_acc)

import matplotlib.pyplot as plt

# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```python
# Data preprocessing

from sklearn.feature_selection import mutual_info_classif
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.decomposition import PCA
import numpy as np

df = pd.read_csv("bucks_roster_stats2.csv")
df = df.drop(columns = ['GAME_ID', 'MIN_TEAM', 'PF_TEAM', 'PLUS_MINUS_TEAM'])


loc = df['HOME/AWAY']
loc.replace('Home', 0, inplace=True)
loc.replace('Away', 1, inplace=True)

#df.dropna(inplace=True)
y=df['WL']
y.replace('L',0,inplace=True)
y.replace('W',1,inplace=True)
Y = tf.keras.utils.to_categorical(y)


df.to_csv('processed_data2.csv', index=False)

#df.fillna(df.mean(), inplace=True)
#df.fillna(df.median(), inplace=True)
#df.fillna(df.mode(), inplace=True)
df.fillna(0, inplace=True)

#x = df.iloc[:,1:]
x = df.drop(columns = ['WL'])
x = preprocessing.scale(x)

#df.to_csv('processed_data2.csv', index=False)

'''
# Apply PCA
pca = PCA(n_components=80) # specify the number of components to keep
x = pca.fit_transform(x)
'''

# Performing mutual information
column_names = list(df.columns)
column_names = ['HOME/AWAY'] + column_names[2:]

information = mutual_info_classif(x,y)
print('Information=',information)


feature_info = {i: info for i, info in enumerate(information)}

sorted_info = sorted(feature_info.items(), key=lambda x: x[1], reverse=True)

print("Ranked features based on mutual information scores:")
for i, info in sorted_info:
    print(f"{column_names[i]}: {info}")

top_features = np.where(information >= np.percentile(information, 90), True, False)
top_features = list(top_features)

X = x[:, top_features]
print(X.shape[1])
```

```
# Split data to train and test
x_train, x_test, y_train, y_test = train_test_split (X,Y, test_size=0.2, random_state=1)

N_train = len(x_train[0])
K = Y.shape[1]
# Insert a dropout rate
dropout_rate = 0.2

# Use keras to create the model, with the best parameters from grid search
my_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(80, activation=tf.nn.sigmoid, input_dim=N_train),
    tf.keras.layers.Dropout(dropout_rate),  # Add dropout layer with rate of 0.2
    tf.keras.layers.Dense(60, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(40, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(20, activation=tf.nn.sigmoid),
    tf.keras.layers.Dropout(dropout_rate),
    tf.keras.layers.Dense(K, activation=tf.nn.softmax)
])

my_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

my_model.fit(x_train, y_train, epochs=25, batch_size=25)
res = my_model.evaluate(x_test, y_test)
my_model.save('my_model.h5')
print('[test loss, test acc]=', res)

# Train the model
history = my_model.fit(x_train, y_train, epochs=100, batch_size=25, validation_data=(x_test, y_test))

y_pred = my_model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
# Print classification report and confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(np.argmax(y_test,axis=1), y_pred))
print(confusion_matrix(np.argmax(y_test,axis=1), y_pred))

# Evaluate the model on the test data
test_loss, test_acc = my_model.evaluate(x_test, y_test)

print('Test accuracy:', test_acc)

import matplotlib.pyplot as plt

# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```