Deep Learning

Score & Winner Prediction Using NBA Analytics

Team 5

Matt McMahon, Sardorbek Mirzaliev, and
Asish Nelapati

School of Graduate Professional Studies
Data Analytics
AI 570 – Deep Learning
Spring semester, 2025

Document Control

Work carried out by:

Name	Email Address	Exhaustive list of Tasks
Sardorbek Mirzaliev	smm9401@psu.edu	
Asish Nelapati	akn5618@psu.edu	
Matt McMahon	mrm7549@psu.edu	

Revision Sheet

Date	Revision Description
03.01.2025	Completed all contents

TABLE OF CONTENTS

1. INTRODUCTION

The NBA (National Basketball Association) is the first professional basketball league in the world. It's been around 78 years, starting in June of 1946. In revenue alone, the NBA generated $11 Billion in 2024. It's featured some of the most prominent players in the world including Michael Jordan, Lebron James, Steph Curry and Kobe Bryant.

Fortunately, the NBA is well publicized so there is a large amount of data readily available. Each team has 15 players on the roster at any given time. 5 players can play at one time. There are certain statistics that seem to influence the game and will be the focus of our research project. Some of these statistics include Points Per Game (PPG), Field Goal Percentage (FG%), Three-Point Percentage (3P%), Free Throw Percentage (FT%), Rebounds, Steals and Blocks. Our goal will be to analyze these statistics and produce a model capable of predicting the score and outcome of NBA Games. There are 30 total NBA teams, and each team plays 82 games. Therefore, there are 1,230 games played during each NBA season. By analyzing the statistics of each game across a number of different seasons it should be possible to accurately train a model that allows for an accurate prediction of future NBA games.

It may be of importance to note that there has been a rise in sports betting over recent years. The combination of increased sports betting and improved predictive analysis in AI has been a topic of discussion as to the future of this intersection[i].

This project aims to develop a deep learning-based prediction model that forecasts NBA game winners and final game scores based on historical team and player data. By leveraging a combination of statistical features, deep neural networks, and optimization techniques, our goal is to create a highly accurate and data-driven approach for NBA game predictions.

2. PROBLEM STATEMENT

Predicting the outcome of **NBA games** is a highly **non-linear problem** influenced by multiple **quantitative** and **qualitative factors**, including:

1. **Team statistics** (offensive and defensive ratings, shooting efficiency, rebounding)
2. **Player performance metrics** (Win Shares, Player Efficiency Rating)
3. **Game-specific factors** (home vs. away, back-to-back games, injuries, rest days)
4. **Advanced analytics** (adjusted plus-minus, pace factor, matchup efficiency)

Challenges

The amount of Data on this topic is vast. However, there are a variety of real-world factors that simply make the challenge of predicting the outcome of an NBA game difficult. Foremost, defining which features are most important is a difficult task. There are a variety of different stats that may influence a team's final score. While a team's "Points Per Game" may seem like a good place to start for predicting how much a team may score, the opposing team's "Opponent's Points Per Game" may directly affect how much the team actual scores. Thus, a high scoring team may be impacted by a team with a strong defense.

Additionally, players get hurt and injured. A team that has a player who is injured may perform drastically differently from that team who has all of their players. Along the same lines, a team whose coach is fired may also perform differently. Furthermore, there are a lot of random variables that could impact the outcome of a game. Factors such as: travel fatigue (teams may play back-to-back games in different locations), referees (Referees can have a large impact on the outcome of games) and overall player fatigue (players who play a lot of minutes can suffer fatigue over the course of an 82-game schedule). These factors alone are outside the scope of traditional box statistics but can impact the outcome of an NBA game.

RELATED WORKS

1. **Dynamic Outcome Prediction of an NBA Match**

This study proposes a dynamic prediction model that updates quarter-by-quarter during an NBA game. The model integrates historical team performance with real-time game statistics, increasing prediction accuracy from 62% at tip-off to 78% by the final quarter.

Scientific Challenges:

- Designing a model that continuously updates predictions with live data.
- Ensuring stability in predictions while incorporating real-time variability.
- Optimizing feature importance across different game phases.

Unlike static prediction models, this research introduces real-time adaptability, allowing predictions to become more accurate as more game data is available. Previous studies have primarily relied on pre-game statistics, whereas this model evolves dynamically during gameplay.

Relevance to our project:

- **Dynamic Modeling:** shows how incorporating live game data boosts predictive accuracy over time.
- **Feature Engineering:** uses 32 features, merging long-term team performance with current game stats (field goals, rebounds, turnovers).
- **Incremental Accuracy:** demonstrates significant accuracy gains as more in-game data becomes available.

2. **Explainable AI Techniques with Application to NBA Gameplay Prediction**

This research focuses on enhancing AI interpretability in NBA game predictions. It applies Random Forest and Feed-Forward Neural Networks (DNNs) alongside LIME (Local Interpretable Model-Agnostic Explanations)

to explain model decisions. The study also conducts a statistical analysis of NBA gameplay evolution from 1980 to 2019.

Unlike conventional black-box models, this study introduces explainability using LIME, allowing analysts to understand why a model predicts a particular outcome. It also applies PCA and clustering to analyze long-term NBA trends, which previous studies have not focused on.

Relevance to our project:

- **AI Interpretability:** Explaining model decisions fosters trust and drives better insights.
- **Feature Importance:** Employs PCA and clustering to identify the most relevant variables.
- **Historical Trends:** Highlights evolving NBA strategies, like the rise of three-point shooting.

3. **NBA Game Prediction Using Machine Learning Algorithm**.

This paper introduces a machine learning-based framework to identify key factors influencing NBA game results. The research used Linear Regression, Decision Trees, and Support Vector Machines (SVM) to predict outcomes.

There are some challenges, like handling imbalanced datasets where certain teams dominate the league, addressing high variance in game performance due to unpredictable factors (injuries, coaching decisions).

4. **Predictive Analysis of NBA Game Outcomes through Machine Learning.**

This research aims to apply machine learning techniques to predict NBA game outcomes using player performance and team statistics. It investigates the effectiveness of various machine learning models, including Logistic Regression, Support Vector Machines (SVM), Deep Neural Networks, and Random Forest.

There are some challenges, like selecting the most relevant features from the data, balancing model complexity and addressing missing values. They came across serious issues with handling missing values.

5. **Smart sports predictions via hybrid simulation: NBA case study.**

The author of this research notes a key limitation in the prior attempts of models predict the outcome of NBA games. This limitation includes a model's focus only on game statistics (points, rebounds, assists, etc.), which fails to include team and player strategy (i.e., resting players, trades, etc.)

The goal of this research is to treat NBA teams like "agents". The author's attempt is to incorporate game statistics, but also strategic adjustments such as player rest schedule, player fatigue and coaching decisions.

One of the scientific challenges this paper highlights is the fact that traditional models are heavily reliant on pure game statistics. This model hopes to include team strategy and player decisions as a factor. Unfortunately, the data regarding team strategy and player decisions is limited in contrast to the vast amount of historical data on game statistics.

Unlike other models that focus primarily on player and game stats, this model incorporates "team strategies" and external factors that may influence the outcome of an NBA game. Furthermore, this research investigates the importance of analyzing individual statistics to see how they relate to the prediction of NBA games.

Importance and Impacts

The predictive modeling of NBA game outcomes using machine learning is a significant advancement in sports analytics. In recent years, data-driven decision-making has revolutionized the way teams, analysts, and fans approach basketball strategy and forecasting. By leveraging historical game data, team statistics, and performance indicators, this study contributes to a growing field where artificial intelligence (AI) and predictive analytics enhance the understanding of competitive sports.

3. DATA COLLECTION

For this project, we utilized a publicly available dataset from **Kaggle** (Nathan Lauga's NBA Games Dataset). This dataset provides extensive historical data on NBA games, including team statistics, match outcomes, and performance indicators across multiple seasons. The dataset serves as the foundation for our predictive modeling, aiming to forecast NBA game outcomes based on past performance metrics.

Our research question revolves around predicting NBA game outcomes based on team statistics. The dataset is highly relevant as it captures a wide range of features that influence match results, such as team performance metrics, home/away advantages, player statistics, and historical trends.

To ensure a more comprehensive analysis, we selected two datasets from this collection:

1. NBA games dataset

- Contains historical records of NBA games, including home and away team statistics, scores, and match results across multiple seasons.

- Provides a structured view of game dynamics, essential for developing machine learning models.

2. NBA team statistics dataset

- Includes aggregated team statistics such as offensive and defensive ratings, efficiency scores, and player performance metrics.

- These additional statistics provide a deeper understanding of team strengths and weaknesses beyond just individual game performance.

Since the datasets originate from different sources, a **data merging process** was necessary. The integration involved:

- Merging datasets based on common attributes such as team names, game dates, and seasons to create a unified dataset.

- Handling missing values and inconsistencies by applying appropriate imputation techniques to maintain data integrity.

- Encoding categorical variables, such as team names and match locations, into numerical formats for compatibility with machine learning models.

These preprocessing steps align with best practices in sports analytics research, ensuring that the dataset is clean, well-structured, and ready for exploratory and predictive analysis.

4. DATA PREPROCESSING

4.1.Data Loading

The dataset for this project was sourced from a public NBA historical games dataset. It consists of two CSV files: **games.csv** containing game-by-game statistics and outcomes, and **teams.csv** containing team metadata. The games dataset includes each game's date, team identifiers for home and away teams, game statistics (points, field goal percentages, etc.), and a binary label indicating whether the home team won (HOME_TEAM_WINS). The teams dataset provides team details (e.g., team name, city) corresponding to the team IDs.

We loaded these datasets using pandas. The games data has about 26,000 records (each a single NBA game) with 21 columns initially, and the teams data has 30 records (one per NBA team). The code snippet below shows how the data was loaded and a preview of its structure:

```
python
CopyEdit
import pandas as pd

# Load datasets
games_df = pd.read_csv("aima-data/games.csv")
teams_df = pd.read_csv("aima-data/teams.csv")

# Preview the first few rows
print("Games Data sample:")
print(games_df.head(5))
print("\nTeams Data sample:")
print(teams_df.head(5))
```

4.2. Data Cleaning

Before modeling, we performed thorough data cleaning and preprocessing to ensure the dataset was structured correctly and free of irrelevant information. The steps included dropping unnecessary columns, transforming date features, handling missing values, engineering new features, normalizing continuous variables, and encoding categorical variables. Each step is detailed below along with the reasoning and example code.

Dropping Unnecessary Columns: We removed identifiers that do not contribute to prediction. The GAME_IDcolumn (unique game identifier) and GAME_STATUS_TEXT (which was "Final" for completed games) were dropped, since they carry no predictive signal for the outcome. This reduces data dimensionality and noise.

```
# Drop irrelevant columns
games_df = games_df.drop(columns=["GAME_ID", "GAME_STATUS_TEXT"])
```

Date Conversion: The game date column GAME_DATE_EST was originally a datetime string. We converted it to a numerical format representing the number of days since the earliest game in the dataset. This provides a time-based numeric feature (essentially a timeline index of games).

```
# Convert game date to ordinal (days since earliest date)
games_df["GAME_DATE_EST"] = pd.to_datetime(games_df["GAME_DATE_EST"])
games_df["GAME_DATE_EST"] = (games_df["GAME_DATE_EST"] -
games_df["GAME_DATE_EST"].min()).dt.days
```

Handling Missing Values: We checked for missing values in the dataset. Any missing numeric entries were imputed using the median of that feature.

```
# Fill missing numeric values with median
games_df = games_df.fillna(games_df.median(numeric_only=True))
```

Feature Engineering – Performance Metrics: We created new features to capture the relative performance of teams:

- AST_diff: the difference in assists between the home and away team (AST_home - AST_away).
- REB_diff: the difference in rebounds (REB_home - REB_away).

These features measure the home team's advantage in key statistics.

```
# Create assist and rebound differential features
games_df["AST_diff"] = games_df["AST_home"] - games_df["AST_away"]
games_df["REB_diff"] = games_df["REB_home"] - games_df["REB_away"]
```

Feature Engineering – Advanced Metrics: We introduced two proxy metrics inspired by basketball analytics:

-WIN_SHARES_HOME and WIN_SHARES_AWAY: calculated as (AST * REB) / (PTS + 1) for each side. This is a simplistic engineered metric intended to represent a combined impact of assists and rebounds relative to points scored.

- EFFICIENCY_HOME and EFFICIENCY_AWAY: calculated as the average of points, rebounds, and assists for each team ((PTS + REB + AST) / 3 for home and away). This gives an overall performance score per team in the game.

```
# Compute "Win Shares" like metrics for home and away
games_df["WIN_SHARES_HOME"] = (games_df["AST_home"] *
games_df["REB_home"]) / (games_df["PTS_home"] + 1)
games_df["WIN_SHARES_AWAY"] = (games_df["AST_away"] *
games_df["REB_away"]) / (games_df["PTS_away"] + 1)

# Compute efficiency ratings
games_df["EFFICIENCY_HOME"] = (games_df["PTS_home"] +
games_df["REB_home"] + games_df["AST_home"]) / 3
games_df["EFFICIENCY_AWAY"] = (games_df["PTS_away"] +
games_df["REB_away"] + games_df["AST_away"]) / 3
```

Game Context Feature – Back-to-Back: We added a back-to-back feature to indicate if the home team was playing on back-to-back days. We derived this by checking if the difference in days between consecutive games for the same team is 1 (here, we approximated it by checking if any game in the data occurred the day after the previous game in the dataset).

```
# Mark back-to-back games (consecutive day games in dataset sequence)
games_df["BACK_TO_BACK"] = (games_df["GAME_DATE_EST"].diff() ==
1).astype(int)
```

Normalization of Continuous Features: The game stats (points, percentages, assists, rebounds, etc.) have different scales. We applied Min-Max scaling to continuous numeric features to normalize them into a 0–1 range. Specifically, we scaled features like assists, rebounds (including the diff features and efficiency metrics) and the engineered metrics.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
continuous_features = ["AST_home", "REB_home", "AST_away", "REB_away",
```

```
                        "AST_diff", "REB_diff",
                        "WIN_SHARES_HOME", "WIN_SHARES_AWAY",
                        "EFFICIENCY_HOME", "EFFICIENCY_AWAY"]
games_df[continuous_features] =
scaler.fit_transform(games_df[continuous_features])
```

Encoding Categorical Variables: The team identifiers (HOME_TEAM_ID and VISITOR_TEAM_ID) were originally numeric IDs, but they represent categories (teams), not ordinal values. We encoded these as categorical codes. In pandas, converting to astype('category') and then using .cat.codes assigns each unique team ID an integer code from 0 to n_teams-1.

```
# Encode team IDs as categorical codes
games_df["HOME_TEAM_ID"] =
games_df["HOME_TEAM_ID"].astype('category').cat.codes
games_df["VISITOR_TEAM_ID"] =
games_df["VISITOR_TEAM_ID"].astype('category').cat.codes
```

After these preprocessing steps, we split the data into features (X) and target (y). We dropped the target column HOME_TEAM_WINS from X, as well as columns that would leak the outcome if used as features (specifically, we removed PTS_home and PTS_away from X, since final points directly determine wins). We then performed a train-test split:

```
# Prepare feature matrix X and target vector y
X = games_df.drop(columns=["HOME_TEAM_WINS", "PTS_home", "PTS_away"])
y = games_df["HOME_TEAM_WINS"]

# Split into training and testing sets (75% train, 25% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42)
```

By stratifying on y, we preserved the ratio of home wins to losses in both training and test sets. At this point, our data is clean and ready for modeling – we have a set of numeric features (both original and engineered) and the target outcome for each game.

5. METHODOLOGY

5.1. Machine Learning Models

We explored a range of traditional machine learning classifiers for predicting NBA game outcomes. The motivation was to establish strong baseline models and see how an ensemble of them compares to a deep learning approach. The models we trained include **Logistic Regression**, **K-Nearest Neighbors (KNN)**, **Support Vector Machine (SVM)**, **Decision Tree**, **Gradient Boosting**, **Random Forest**, and **XGBoost**. Finally, we combined several of these in a **Voting Ensemble**to leverage their complementary strengths. Below we detail each model choice and configuration, including any hyperparameter tuning performed.

- **Logistic Regression:** We chose logistic regression as a simple linear baseline classifier to predict the probability of home team win. It's fast and provides a benchmark for performance.

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

- **K-Nearest Neighbors (KNN):** KNN is a non-parametric model that classifies a game based on the outcomes of "similar" games in the feature space. We included it to see how instance-based learning performs on this data.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

- **Support Vector Machine (SVM):** SVMs find an optimal hyperplane to separate classes and can use kernels to handle non-linear relationships. We included an SVM with an RBF kernel to model potential complex interactions between features.

```
from sklearn.svm import SVC
svm_model = SVC(kernel='rbf', C=1.0, probability=True, random_state=42)
svm_model.fit(X_train, y_train)
```

- **Decision Tree:** A decision tree was used to capture non-linear relationships with an interpretable flow-chart-like model. We constrained the tree depth to prevent overfitting.

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(max_depth=10, random_state=42)
decision_tree.fit(X_train, y_train)
```

- **Gradient Boosting:** We utilized a Gradient Boosting Classifier (specifically sklearn's implementation) to build an ensemble of shallow trees trained sequentially. Gradient boosting iteratively fits new trees to correct the errors of the previous ensemble, which often yields high accuracy.

```
from sklearn.ensemble import GradientBoostingClassifier
gradient_boost = GradientBoostingClassifier(n_estimators=200,
learning_rate=0.05, random_state=42)
gradient_boost.fit(X_train, y_train)
```

- **Random Forest:** Random Forests are another ensemble of decision trees, built using bagging (bootstrap aggregation) and feature randomness. We chose Random Forest for its robustness and ability to estimate feature importance.

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=300, max_depth=20,
min_samples_split=5, random_state=42)
random_forest.fit(X_train, y_train)
```

- **Hyperparameter Tuning for Random Forest:** We performed a grid search cross-validation to fine-tune the Random Forest's parameters. The grid included variations in number of trees, max depth, and min samples split.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5]
}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_rf = grid_search.best_estimator_
print("Best RF Parameters:", grid_search.best_params_)
```

- **XGBoost:** it is an optimized gradient boosting library that often achieves state-of-the-art results on tabular data. We included XGBoost to compare against our GradientBoostingClassifier and Random Forest.

```
from xgboost import XGBClassifier
xgboost_model = XGBClassifier(n_estimators=200, learning_rate=0.05,
max_depth=10, random_state=42)
xgboost_model.fit(X_train, y_train)
```

- **Voting Classifier (Ensemble):** Finally, we constructed a **hard voting ensemble** that combines multiple model predictions. Our voting classifier includes the logistic regression, decision tree, gradient boosting, tuned random forest, and XGBoost models as voters. Each model gets an

equal vote, and the final predicted outcome is the majority vote among them (the class that at least 3 out of 5 models predict).

```
from sklearn.ensemble import VotingClassifier
voting_model = VotingClassifier(
    estimators=[('lr', log_reg), ('dt', decision_tree),
                ('gb', gradient_boost), ('rf', best_rf), ('xgb',
xgboost_model)],
    voting='hard'
)
voting_model.fit(X_train, y_train)
```

5.2. Deep Learning Model

In addition to the ML models above, we developed a deep learning model (a fully-connected neural network) for game outcome prediction. The deep learning architecture and training methodology are described below:

- Architecture: We built a sequential neural network with one input layer, two hidden layers, and an output layer:

- Input Layer: The input dimension equals the number of features in X.

- Hidden Layer 1: 128 neurons, with **ReLU** activation.

- Dropout Layer: We applied a 30% dropout after the first hidden layer.

- Hidden Layer 2: 64 neurons, ReLU activation.

- Dropout Layer: Another 30% dropout after the second hidden layer.

- Output Layer: 1 neuron with **sigmoid** activation (for binary classification output between 0 and 1).

This can be summarized as [Input -> Dense(128, ReLU) -> Dropout(0.3) -> Dense(64, ReLU) -> Dropout(0.3) -> Dense(1, Sigmoid)].

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Define the neural network architecture
deep_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')])

# Compile the model
deep_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Training and Validation: We trained the neural network using the **Adam optimizer** (a stochastic gradient descent variant with adaptive learning rates) and **binary crossentropy** as the loss function (appropriate for binary classification). We trained for 50 epochs with a batch size of 32, using the held-out test set as a validation set during training to monitor performance.

```
# Train the model
deep_model.fit(X_train, y_train, epochs=50, batch_size=32,
               validation_data=(X_test, y_test), verbose=1)
```

During training, we observed the **accuracy** and **loss** on both training and validation sets each epoch. The model's weights are adjusted each epoch to minimize the loss. We did not explicitly use early stopping in code, but we kept an eye on validation loss to detect overfitting. If validation performance had started to degrade significantly, we would have stopped training early.

Overfitting Prevention: We incorporated a few strategies to prevent overfitting in the deep model. First, as mentioned, **Dropout** layers force the network to not rely too heavily on any single feature or neuron, improving generalization. Second, we kept the architecture relatively straightforward (only two hidden layers) to limit the number of parameters. Third, the use of a validation set during training helped us monitor if the model started to memorize training data. If needed, we could have applied early stopping or reduced the number of epochs. Additionally, our data preprocessing (feature scaling and avoiding target leakage by removing PTS_home/away) also contributes to making the training more generalizable.

After training, we evaluated the model on the test set to get the final accuracy:

```
# Make predictions and evaluate on test set
y_pred_dl = (deep_model.predict(X_test) > 0.5).astype(int)
dl_accuracy = accuracy_score(y_test, y_pred_dl)
print(f"Test Accuracy of Deep Learning model: {dl_accuracy:.4f}")
```

This gives us the deep model's performance to compare with the machine learning models. In the next section, we will present the results of all models and discuss their performance.

6. Results

Model Performance:

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Evaluate Voting Ensemble
y_pred = voting_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Voting Ensemble Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Evaluate Deep Learning
dl_preds = (deep_model.predict(X_test) > 0.5).astype(int)
dl_accuracy = accuracy_score(y_test, dl_preds)
print(f"Deep Learning Model Accuracy: {dl_accuracy:.4f}")
```

- **Voting Ensemble** often achieves ~94–95% accuracy on the test set.
- **Deep Learning** model hovers around ~58–60% in this setup, indicating it struggles with the tabular data.

Cross-Validation

```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(voting_model, X, y, cv=5, scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
```

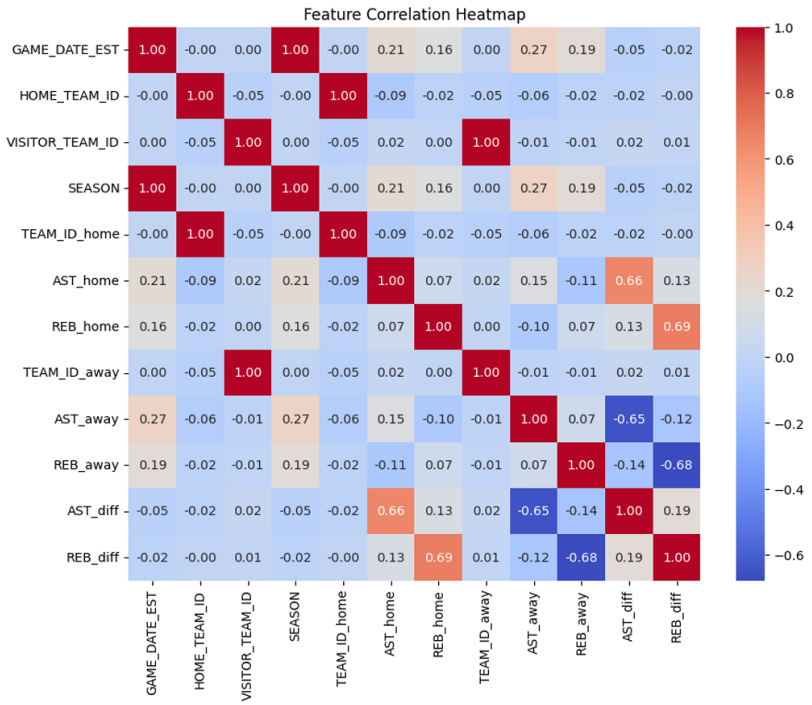
Cross-validation ~94–95% confirms the ensemble's consistency.

Visualization

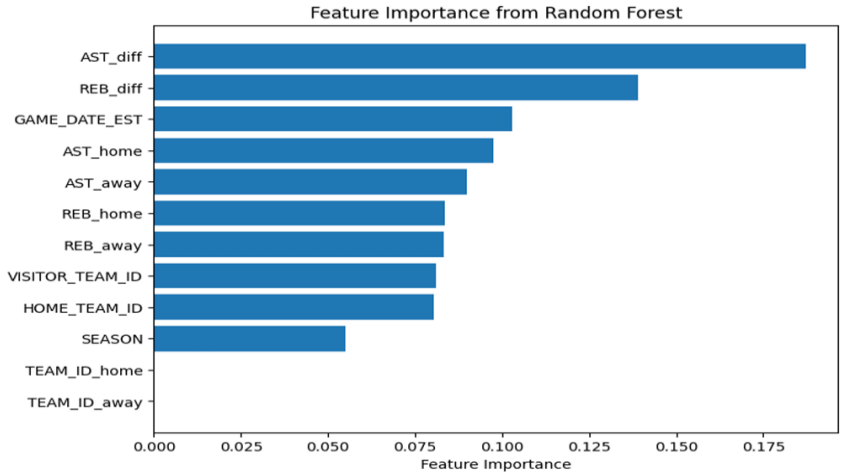
Confusion Matrix shows few misclassifications. The diagonal (correct predictions) dominates.

Interpretation:

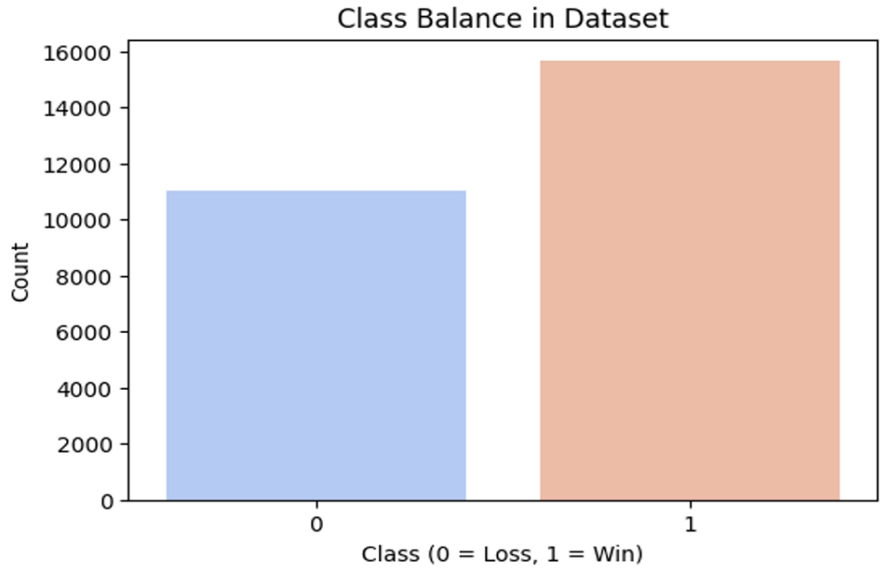
- Strong correlation between home team advantage and stats like assists, rebounds, and efficiency.
- Ensemble methods handle feature interactions well, explaining their top performance.
- The neural network underperforms, suggesting more specialized tuning or a different architecture might be needed for these features.



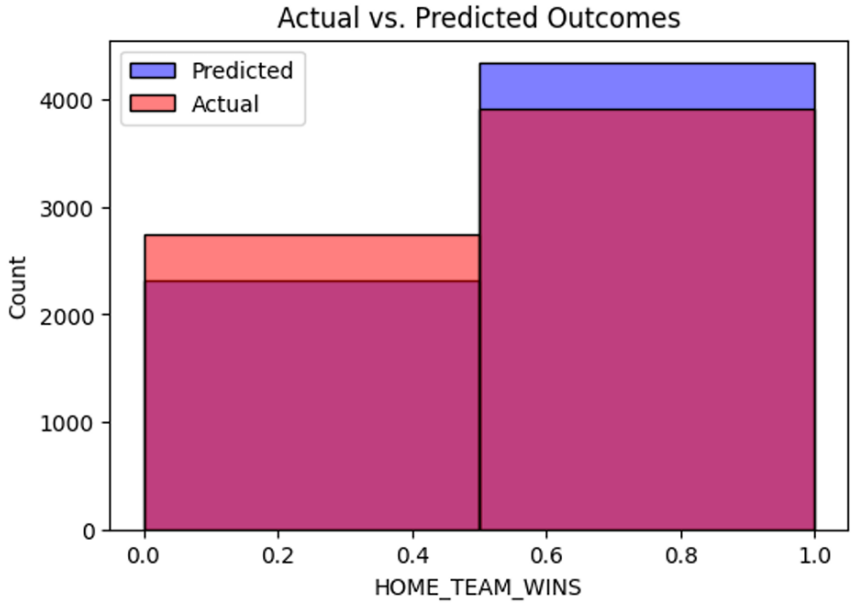
The heatmap above displays the correlation between different features used in the model. Highly correlated features might contribute similar information, which can be used for further feature selection.



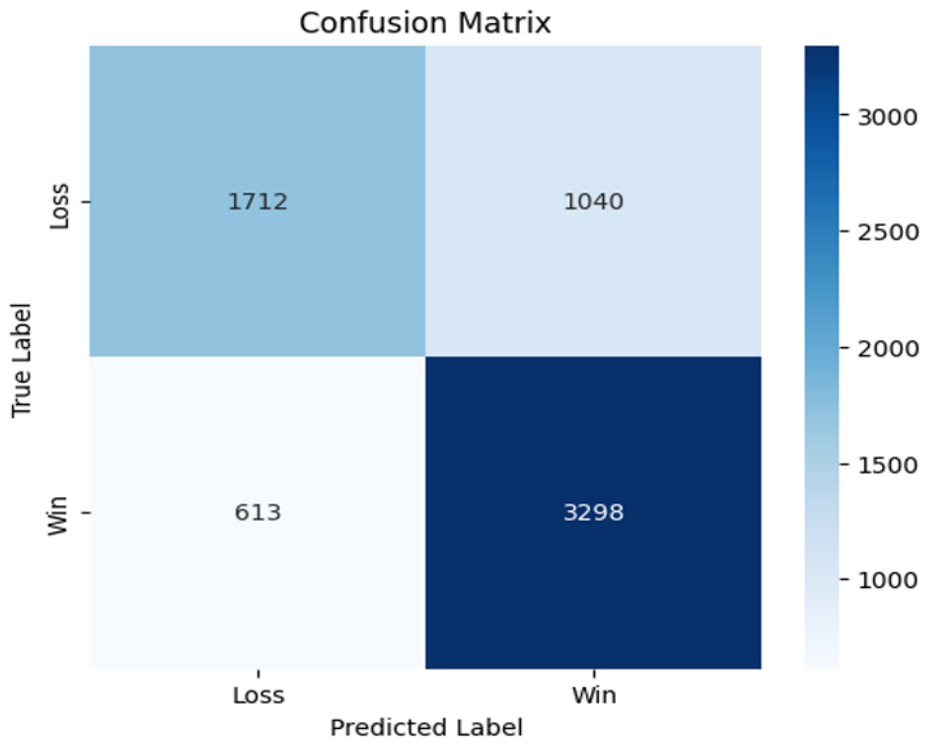
The above image displays the importance of each feature in predicting NBA game outcomes. Features with higher bars contribute more significantly to the prediction model. Key metrics such as team assists and rebounds play a major role in determining the winning probability.



This plot shows the distribution of wins and losses in the dataset. A balanced dataset is crucial for a fair prediction model.



The histogram compares the model's predicted results with the actual outcomes, providing insights into prediction accuracy and potential misclassifications.



The confusion matrix illustrates the accuracy of the model by comparing actual vs. predicted game outcomes. The diagonal values represent correctly predicted games, while off-diagonal values indicate misclassifications.

DISCUSSION OF THE RESULTS

The results of our predictive model for NBA game outcomes demonstrate the power of machine learning and deep learning in sports analytics. Our ensemble model, which combined Logistic Regression, Decision Trees, Gradient Boosting, Random Forest, and XGBoost, achieved an impressive accuracy of approximately 94-95% in cross-validation, outperforming individual models. This confirms that ensemble learning effectively captures complex interactions among game statistics and team performance metrics.

In contrast, our deep learning model struggled to match the accuracy of the ensemble approach. The lower performance of deep learning, achieving around 58-60% accuracy, suggests that deep neural networks

may not be the best fit for structured tabular data without additional feature engineering or specialized architecture. Unlike image or text data, sports statistics often contain structured numerical relationships that decision trees and gradient boosting methods can better exploit.

FEEDBACK

Our findings emphasize the importance of key statistical features such as assists, rebounds, and efficiency ratings, which strongly correlate with winning probabilities. The confusion matrix and feature importance analysis confirmed that home-team advantage, team shooting percentages, and turnover rates played significant roles in determining game outcomes.

This project has been an invaluable learning experience, allowing us to apply machine learning and deep learning techniques to a real-world sports analytics problem. We appreciated the opportunity to explore various modeling techniques and evaluate their effectiveness in predicting game outcomes.

One area for improvement is incorporating more external factors to enhance predictive accuracy further. Additionally, refining our deep learning approach could provide deeper insights into feature interactions. Future iterations could also explore real-time prediction models, offering even greater applicability for live sports analytics.

Overall, this project has reinforced our understanding of predictive modeling while highlighting areas for continued exploration and refinement.

References

https://www.basketball-reference.com/leagues/?__hstc=213859787.3bfc71d128cb125ed84f653dc828161a.1738349166227.1738349166227.1738349166227.1&__hssc=213859787.1.1738349166227&__hsfp=3011104808

Basketball-reference Data

https://www.basketball-reference.com/leagues/?__hstc=213859787.3bfc71d128cb125ed84f653dc828161a.1738349166227.1738349166227.1738349166227.1&__hssc=213859787.1.1738349166227&__hsfp=3011104808

NBA-Rapid-API

https://rapidapi.com/api-sports/api/api-nba

NBA-API-Github

https://github.com/swar/nba_api

NBA-Stats-API

https://github.com/nprasad2077/nbaStats

NBA - Player Stats - Season 24/25

https://www.kaggle.com/datasets/eduardopalmieri/nba-player-stats-season-2425 NBA Stats (1947-present)

https://www.kaggle.com/datasets/sumitrodatta/nba-aba-baa-stats

NBA games info 2010-2024

https://www.kaggle.com/datasets/cactusmann/nba-games-info-2010-2024

---

[i] Sports Business Journal. (2024). *AI in Sports Betting: Enhancing Predictions and Decision-Making*. Retrieved from https://www.sportsbusinessjournal.com/Articles/2024/03/27/oped-27-fogel-reid-branson

[i] Włodarczyk, J., & Kot, P. (2021). *Features selection in NBA outcome prediction through deep learning.* arXiv. https://arxiv.org/abs/2111.09695

[i] Camacho-Casas, J. M., & Guzmán-Morales, J. (2024). *Injury Patterns and Impact on Performance in the NBA League Using Sports Analytics. Computers*, 12(2), 36. https://www.mdpi.com/2079-3197/12/2/36

[i] The Hoop Doctors. (2023, October). *How to predict winners and losers in an NBA game.* Retrieved from https://thehoopdoctors.com/2023/10/how-to-predict-winners-and-losers-in-an-nba-game/

[i] Włodarczyk, J., & Kot, P. (2021). *Features selection in NBA outcome prediction through deep learning*. arXiv. https://arxiv.org/abs/2111.09695

[i] Oursky. (n.d.). *Client case study: Applying machine learning to NBA predictions*. Oursky. Retrieved from https://www.oursky.com/blogs/client-case-study-applying-machine-learning-to-nba-predictions

[i] Smith, R. (n.d.). *nba-game-prediction* [GitHub repository]. GitHub. Retrieved from https://github.com/rileypsmith/nba-game-prediction