The Impact of LeBron James

Arwin Rahmatpanah & David Daniels

PHOTO: STACY REVERE/GETTY IMAGES

Problem Motivation

Motivating Question

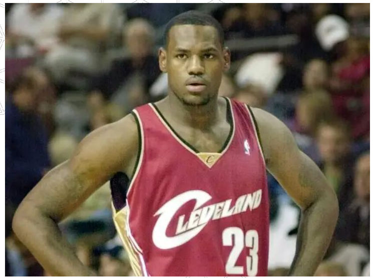
- Is it possible to quantify one player's impact on the outcome of a basketball game?

Case Study – LeBron James

- All-time leading scorer in the NBA
- 20-year career (currently still active)
- Won the NBA Championship 4 times with 3 different teams
- 4-time recipient of the Most Valuable Player award

Goal of Analysis

- How accurately can we predict the outcome of a game based on LeBron's individual performance?



Berkeley
UNIVERSITY OF CALIFORNIA

Dataset Description

Dataset Source:

- Game log stats from Basketball Reference
- 1,421 records

Key Variables:

- Points Scored
- Field Goal Percent
- Minutes Played
- Plus/Minus
- Game Score

2003-04 Regular Season Share & Export ▾ Glossary

Rk	G	Date	Age	Tm		Opp		GS	MP	FG	FGA	FG%	3P	3PA	3P%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	GmSc	+/-
1	1	2003-10-29	18-303	CLE	@	SAC	L (-14)	1	42:50	12	20	.600	0	2	.000	1	3	.333	2	4	6	9	4	0	2	3	25	24.7	-9
2	2	2003-10-30	18-304	CLE	@	PHO	L (-9)	1	40:21	8	17	.471	1	5	.200	4	7	.571	2	10	12	8	1	0	7	1	21	14.7	-3

Berkeley
UNIVERSITY OF CALIFORNIA

Data Cleaning & EDA

- Every game in LeBron's Career (2003 – 2023)
- Our goal: Clean up numerical variables for ML models, and one-hot encode the few categorical variables
  - Created 'win_or_loss' column for output variable
  - Parsed 'age', 'minutes_per_game', etc. into decimal format
  - Added year_num column to group by season
  - Changed all column types to fit the relevant ML model
  - Replaced NaN values with 0 – as we saw fit
  - Applied one-hot-encodings to categorical variables

Berkeley
UNIVERSITY OF CALIFORNIA

Solution & Approach

Training, Validation, and Test Split

- 70/15/15 chronological split
- Time-series data so chronological intuitively makes sense

Binary Classification Problem

1. Logistic Regression (No Tensorflow & Tensorflow)
2. Decision Tree
3. Random Forest
4. Neural Network

Logistic Regression (non-tf)

Validation Accuracy

```
y_val_pred = log_reg.predict(X_val_scaled)

# Calculate accuracy
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation accuracy: {val_accuracy}")

# Print classification report
print(classification_report(y_val, y_val_pred))
```

```
Validation accuracy: 0.8504672897196262
              precision    recall  f1-score   support

           L       0.84      0.76      0.80        83
           W       0.86      0.91      0.88       131

    accuracy                           0.85       214
   macro avg       0.85      0.83      0.84       214
weighted avg       0.85      0.85      0.85       214
```

Test Accuracy

```
# Make predictions on the test set
y_test_pred = log_reg.predict(X_test_scaled)

# Calculate the accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test accuracy:", test_accuracy)

# Print the classification report
print(classification_report(y_test, y_test_pred))
```

```
Test accuracy: 0.7840375586854460
              precision    recall  f1-score   support

           L       0.76      0.67      0.72        86
           W       0.80      0.86      0.83       127

    accuracy                           0.78       213
   macro avg       0.78      0.77      0.77       213
weighted avg       0.78      0.78      0.78       213
```

Logistic Regression (Tensorflow)

```
Epoch 30/40
32/32 - 0s - loss: 0.5085 - accuracy: 0.8280 - val_loss: 0.6168 - val_accuracy: 0.8318 - 50ms/epoch - 2ms/step
Epoch 31/40
32/32 - 0s - loss: 0.4852 - accuracy: 0.8400 - val_loss: 0.6041 - val_accuracy: 0.8037 - 62ms/epoch - 2ms/step
Epoch 32/40
32/32 - 0s - loss: 0.4417 - accuracy: 0.8370 - val_loss: 0.5852 - val_accuracy: 0.8131 - 53ms/epoch - 2ms/step
Epoch 33/40
32/32 - 0s - loss: 0.4170 - accuracy: 0.8521 - val_loss: 0.5797 - val_accuracy: 0.8131 - 48ms/epoch - 1ms/step
Epoch 34/40
32/32 - 0s - loss: 0.4261 - accuracy: 0.8541 - val_loss: 0.5688 - val_accuracy: 0.8318 - 47ms/epoch - 1ms/step
Epoch 35/40
32/32 - 0s - loss: 0.4064 - accuracy: 0.8481 - val_loss: 0.5776 - val_accuracy: 0.8131 - 47ms/epoch - 1ms/step
Epoch 36/40
32/32 - 0s - loss: 0.3980 - accuracy: 0.8602 - val_loss: 0.5942 - val_accuracy: 0.8271 - 48ms/epoch - 1ms/step
Epoch 37/40
32/32 - 0s - loss: 0.4098 - accuracy: 0.8571 - val_loss: 0.5525 - val_accuracy: 0.8411 - 48ms/epoch - 2ms/step
Epoch 38/40
32/32 - 0s - loss: 0.3770 - accuracy: 0.8632 - val_loss: 0.5976 - val_accuracy: 0.8131 - 48ms/epoch - 2ms/step
Epoch 39/40
32/32 - 0s - loss: 0.5141 - accuracy: 0.8421 - val_loss: 0.5351 - val_accuracy: 0.8411 - 49ms/epoch - 2ms/step
Epoch 40/40
32/32 - 0s - loss: 0.3701 - accuracy: 0.8672 - val_loss: 0.5296 - val_accuracy: 0.8411 - 48ms/epoch - 1ms/step
7/7 [==============================] - 0s 1000us/step - loss: 0.6901 - accuracy: 0.8122
Test accuracy (Logistic Regression with TensorFlow): 0.8122065663337708
```

Feature Importance (LR)

```python
# Get coefficients from the logistic regression model
log_reg_coefficients = log_reg.coef_[0]

# Create a DataFrame with the feature names and their coefficients
coefficients_df = pd.DataFrame(
    {"feature": x_train.columns, "coefficient": log_reg_coefficients}
)

# Sort the DataFrame by the absolute values of the coefficients in descending order
coefficients_df_sorted = coefficients_df.reindex(
    coefficients_df.coefficient.abs().sort_values(ascending=False).index
)

# Display the sorted DataFrame
print(coefficients_df_sorted)
```

```
        feature  coefficient
20   minus_plus     3.919856
6      threeatt    -0.582375
1            mp    -0.450482
30      opp_CHI    -0.425536
59      opp_UTA    -0.406391
..          ...          ...
26      opp_ATL     0.007534
42      opp_MIA    -0.005041
5         three     0.003616
24     team_LAL     0.000000
4           fgp     0.000000

[64 rows x 2 columns]
```

Berkeley
UNIVERSITY OF CALIFORNIA

Decision Tree

Validation Accuracy

```
# Set the seed for reproducibility
seed = 42

# Create the Decision Tree model with the seed
dtree = DecisionTreeClassifier(max_depth=1, random_state=seed)

# Train the model on the training set
dtree.fit(X_train_scaled, y_train)

# Make predictions on the validation set
y_val_pred_dtree = dtree.predict(X_val_scaled)

# Calculate the accuracy
val_accuracy_dtree = accuracy_score(y_val, y_val_pred_dtree)
print("Validation accuracy (Decision Tree):", val_accuracy_dtree)

# Print the classification report
print(classification_report(y_val, y_val_pred_dtree))
```

```
Validation accuracy (Decision Tree): 0.8738317757009346
              precision    recall  f1-score   support

           L       0.84      0.83      0.84        83
           W       0.89      0.90      0.90       131

    accuracy                           0.87       214
   macro avg       0.87      0.87      0.87       214
weighted avg       0.87      0.87      0.87       214
```

Test Accuracy

```
# Make predictions on the test set
y_test_pred_dtree = dtree.predict(X_test_scaled)

# Calculate the accuracy
test_accuracy_dtree = accuracy_score(y_test, y_test_pred_dtree)
print("Test accuracy (Decision Tree):", test_accuracy_dtree)

# Print the classification report
print(classification_report(y_test, y_test_pred_dtree))
```

```
Test accuracy (Decision Tree): 0.8169014084507042
              precision    recall  f1-score   support

           L       0.81      0.72      0.76        86
           W       0.82      0.88      0.85       127

    accuracy                           0.82       213
   macro avg       0.81      0.80      0.81       213
weighted avg       0.82      0.82      0.81       213
```

Feature Importance (DT)

```
# Get feature importances from the Decision Tree model
feature_importances = dtree.feature_importances_

# Create a DataFrame with the feature names and their importance scores
feature_importance_df = pd.DataFrame(
    {"feature": x_train.columns, "importance": feature_importances}
)

# Sort the DataFrame by importance scores in descending order
feature_importance_df_sorted = feature_importance_df.sort_values(
    by="importance", ascending=False
)

# Display the sorted DataFrame
print(feature_importance_df_sorted)
```

```
       feature  importance
20  minus_plus         1.0
0         game         0.0
33     opp_DAL         0.0
35     opp_DET         0.0
36     opp_GSW         0.0
..         ...         ...
27     opp_BOS         0.0
28     opp_BRK         0.0
29     opp_CHA         0.0
30     opp_CHI         0.0
63         day         0.0

[64 rows x 2 columns]
```

Berkeley
UNIVERSITY OF CALIFORNIA

Random Forest

Validation Accuracy

```python
# Set the seed for reproducibility
seed = 42

# Create the Random Forest model with the seed
rf = RandomForestClassifier(min_samples_split=4, n_estimators=100, random_state=seed)

# Train the model on the training set
rf.fit(X_train_scaled, y_train.values.ravel())

# Make predictions on the validation set
y_val_pred_rf = rf.predict(X_val_scaled)

# Calculate the accuracy
val_accuracy_rf = accuracy_score(y_val, y_val_pred_rf)
print("Validation accuracy (Random Forest):", val_accuracy_rf)

# Print the classification report
print(classification_report(y_val, y_val_pred_rf))
```

Validation accuracy (Random Forest): 0.8785046728971962

	precision	recall	f1-score	support
L	0.90	0.77	0.83	83
W	0.87	0.95	0.91	131
accuracy			0.88	214
macro avg	0.88	0.86	0.87	214
weighted avg	0.88	0.88	0.88	214

Test Accuracy

```python
# Make predictions on the test set
y_test_pred_rf = rf.predict(X_test_scaled)

# Calculate the accuracy
test_accuracy_rf = accuracy_score(y_test, y_test_pred_rf)
print("Test accuracy (Random Forest):", test_accuracy_rf)

# Print the classification report
print(classification_report(y_test, y_test_pred_rf))
```

Test accuracy (Random Forest): 0.8262910798122066

	precision	recall	f1-score	support
L	0.86	0.69	0.76	86
W	0.81	0.92	0.86	127
accuracy			0.83	213
macro avg	0.83	0.80	0.81	213
weighted avg	0.83	0.83	0.82	213

Berkeley
UNIVERSITY OF CALIFORNIA

Feature Importance (RF)

```
# Get the feature importances
importances = rf.feature_importances_

# Get the column names from the input dataset
feature_names = x_train.columns

# Create a dictionary of feature names and their importances
feature_importances = dict(zip(feature_names, importances))

# Sort the dictionary by importance score in descending order
sorted_feature_importances = sorted(feature_importances.items(), key=lambda x: x[1], reverse=True)

# Print the sorted feature importances
for feature, importance in sorted_feature_importances:
    print(f"{feature}: {importance}")
```

```
minus_plus: 0.4129913200597449
game_score: 0.058358479027977396
mp: 0.03851677041489027
fga: 0.030672452032309224
ast: 0.02857400407156052
pts: 0.027114384877615406
decimal_age: 0.025401084611770167
year: 0.025291838964842724
game: 0.025246148275027438
ftp: 0.024204507543174315
tov: 0.02211705224597309
fg: 0.022095775205857423
drb: 0.02094001271410278
trb: 0.020259088934534607
threeatt: 0.020159656687059307
day: 0.01993866430177415
```

Berkeley
UNIVERSITY OF CALIFORNIA

Neural Network (Tensorflow)

Validation Accuracy

```
Epoch 1/15
50/50 - 1s - loss: 0.7165 - accuracy: 0.6247 - val_loss: 0.6142 - val_accuracy: 0.6168 - 874ms/epoch - 17ms/step
Epoch 2/15
50/50 - 0s - loss: 0.6329 - accuracy: 0.6549 - val_loss: 0.5980 - val_accuracy: 0.6215 - 111ms/epoch - 2ms/step
Epoch 3/15
50/50 - 0s - loss: 0.5982 - accuracy: 0.6982 - val_loss: 0.5646 - val_accuracy: 0.6636 - 100ms/epoch - 2ms/step
Epoch 4/15
50/50 - 0s - loss: 0.5486 - accuracy: 0.7294 - val_loss: 0.5178 - val_accuracy: 0.6869 - 91ms/epoch - 2ms/step
Epoch 5/15
50/50 - 0s - loss: 0.5231 - accuracy: 0.7425 - val_loss: 0.4751 - val_accuracy: 0.7523 - 106ms/epoch - 2ms/step
Epoch 6/15
50/50 - 0s - loss: 0.4711 - accuracy: 0.7847 - val_loss: 0.4735 - val_accuracy: 0.7290 - 97ms/epoch - 2ms/step
Epoch 7/15
50/50 - 0s - loss: 0.3867 - accuracy: 0.8189 - val_loss: 0.4505 - val_accuracy: 0.7710 - 91ms/epoch - 2ms/step
Epoch 8/15
50/50 - 0s - loss: 0.4045 - accuracy: 0.8129 - val_loss: 0.4389 - val_accuracy: 0.7991 - 89ms/epoch - 2ms/step
Epoch 9/15
50/50 - 0s - loss: 0.3642 - accuracy: 0.8310 - val_loss: 0.4230 - val_accuracy: 0.8178 - 95ms/epoch - 2ms/step
Epoch 10/15
50/50 - 0s - loss: 0.3377 - accuracy: 0.8682 - val_loss: 0.4081 - val_accuracy: 0.8224 - 108ms/epoch - 2ms/step
Epoch 11/15
50/50 - 0s - loss: 0.2956 - accuracy: 0.8853 - val_loss: 0.4439 - val_accuracy: 0.8318 - 97ms/epoch - 2ms/step
Epoch 12/15
50/50 - 0s - loss: 0.2941 - accuracy: 0.8823 - val_loss: 0.4343 - val_accuracy: 0.8458 - 101ms/epoch - 2ms/step
Epoch 13/15
50/50 - 0s - loss: 0.2820 - accuracy: 0.8873 - val_loss: 0.4293 - val_accuracy: 0.8505 - 102ms/epoch - 2ms/step
Validation accuracy (Neural Network): 0.8504672646522522
```

Test Accuracy

```
# Scale the test set
x_test_scaled = scaler.transform(x_test)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test_scaled, y_test_encoded, verbose=2)
print("Test accuracy (Neural Network):", test_accuracy)
```

```
7/7 - 0s - loss: 0.5326 - accuracy: 0.7793 - 22ms/epoch - 3ms/step
Test accuracy (Neural Network): 0.7793427109718323
```

NN's model complex relationships!

Results & Takeaways

Our Models

- **Random Forest:** Best performance (Val Acc: ~0.87, Test Acc: ~0.82)
- **Decision Tree, Logistic Regression (tf):** Similar performance, slightly below
- **Neural Network:** Lower performance

Key Findings

- **Minus_plus** (point differential with LeBron on court) is the most critical predictor
- **Game_score** (player-specific efficiency rating) is the second most important predictor

Notable Insights

- **Individual stats** (points, assists, rebounds, etc.) are not significant in predicting game outcomes
- **LeBron's impact** on the game is more important than his raw stats
- **Team performance improves** with LeBron on the court, increasing win probability

Conclusion

- **Key Finding:** minus_plus – The Ultimate Indicator
- **Takeaway:** LeBron's on-court presence drives team success

Berkeley
UNIVERSITY OF CALIFORNIA

Potential Ethical, Legal, or Personal Concerns

- Not many ethical, legal, or personal concerns
- Unlikely to generalize to other players
- Possible collinearity between minus__plus and game outcome

Berkeley
UNIVERSITY OF CALIFORNIA

Thank you

We are now open to questions!