NBA Player Role Prediction

Peter Hsia, Michael Zita, Justin Zhu

phsia006@ucr.edu, mzita002@ucr.edu, jzhu184@ucr.edu

December 2nd, 2025

UC Riverside, ECE 16 Data Science for Engineering Applications

Python libraries and Tools

- pandas
- numpy
- matplotlib
- seaborn
- collections
- scikit-learn
- tensorFlow

Introduction (1-3 slides)

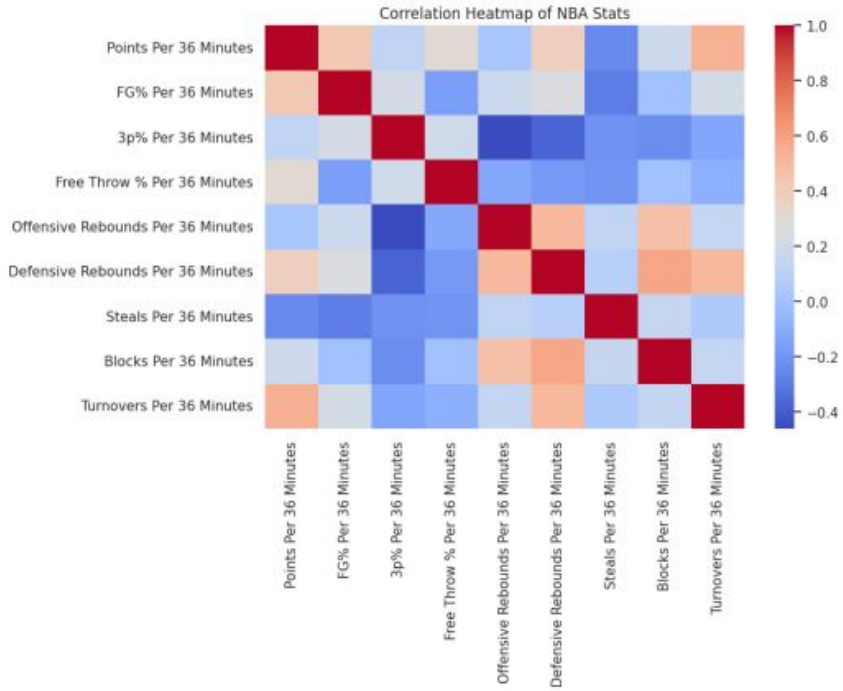
- You need to have a few ballot points about what problem you are addressing and why it matters. You can use plots (yours from observations in data) or from other sources (add the sources you use).

How are we utilizing machine learning?

-

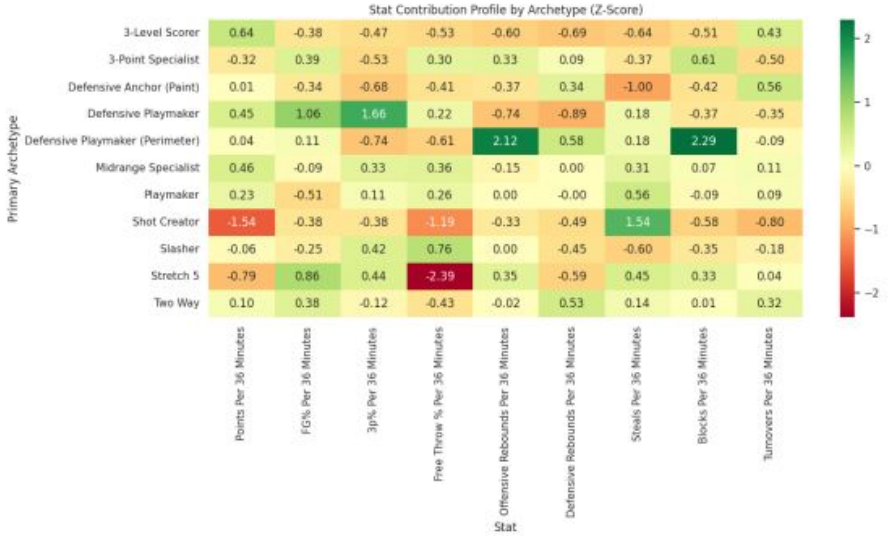
Classifying and Data Gathered

- The National Basketball Association (NBA) numbers are complex and linked together - so comparing athletes and spotting their roles is difficult.
- Our goal is to classify NBA players (including 2025 rookies) into player archetypes using per–36-minute statistics, which demonstrates a probability-style project.
- We start from 10 archetypes (Shot Creator, 3-Point Specialist, Stretch 5, etc.) and, due to limited samples taken from the top players in the league currently. The archetypes are split into 3 broader categories. (Offensive, Defensive, and Playmaker)



Why it Matters

- **Role/archetype labels summarize strengths** more clearly than raw stat tables (e.g., “rim protector” vs. “spacer”), improving interpretability for comparison.
- Correlation patterns show **which stats move together**, helping identify what features are influential for modeling archetypes.
- Real modeling constraint: **class imbalance** (especially Playmaker) can hurt classification performance, motivating broader group labels.



Data Snapshot

- Dataset: **per–36-minute NBA player stats** (includes 2025 rookies).
- Core numeric features include points/assists/steals/blocks, shooting percentages, rebounds, turnovers (per 36).
- We reduce complexity by engineering combined metrics (e.g., **Scoring Efficiency Index** and **Defensive Efficiency Index**) to better capture structure.

Data Exploration (3-6 slides)

- You need to write the resource of the data with link.
- You need to give some intuitions about what the dataset(s) you are using contain.
- You need to talk about the data preprocessing and cleaning, merging, etc. that you performed and give reasoning for each.
- You need to plot meaningful plots that explain the data.

Data source & Scope

- **Data source:** Basketball-Reference “2024–25 NBA Player Stats: Per 36 Minutes” (player per-36 table).
- **Why per-36?** Per-36 stats normalize production by playing time (stat/minutes × 36), letting you compare bench vs starters more fairly.
- **Scope:** Our report states the dataset includes **NBA players + 2025 rookies** and uses **per–36-minute statistics**.

Dataset Features

- **9 numeric features used**
  - Points, assists, steals, blocks **per 36**
  - FG%, 3P%, FT%
  - Offensive rebounds, defensive rebounds **per 36**
  - Turnovers **per 36**
- **Target labels:** 10 archetypes

Cleaning & Preprocessing

- **Coerce numeric columns:** Convert stat columns to numeric using pd.to_numeric(..., errors="coerce") to safely handle bad/missing entries.
- **Handle missing values:** Drop rows with missing key stat fields before building indices (e.g., dropna(subset=eff_features)) so normalization/index math is valid.
- **Standardization for comparisons:** Compute **z-scores** for each numeric feature to compare “above/below league average” by archetype in a common scale.
- **Label cleanup:** Strip whitespace on archetype strings (primary/secondary) to avoid duplicate categories from formatting inconsistencies.

Feature Engineering

- **Offensive (Scoring) Efficiency Index:** Min-max normalize Points + FG% + 3P% + FT%, then average to get one offense “score.”
- **Weighting primary vs secondary archetype:** Primary contributes **100%**, secondary contributes **50%** so hybrid players influence both profiles.
- **Defensive Efficiency Index:** Normalize defensive rebounds, blocks, steals, then average for one defense “score.”
- **Motivation:** Reduce complexity by combining shooting-related %’s into offense efficiency and defense stats into defense efficiency.

Machine Learning Method(s) (1-3 slides)

- You need to explain what you want to predict and why.
- You need to explain the machine learning method you are using and why you chose it with its parameters.

What We Predict

- **Prediction target:** player **role group label** from stats. We ultimately predict **Offensive vs Defensive** (binary classification: 0 = Defensive, 1 = Offensive).
- **Why not 3 classes?** When we tried **Offensive / Defensive / Playmaker**, both models were ~**50% test accuracy** because the Playmaker class had **very few labeled examples** and models biased toward the majority class.
- **Final framing:** after removing Playmaker and collapsing to **Offensive vs Defensive**, performance improves (KNN ~0.75 accuracy; neural net ~0.86 accuracy).
- **Why this matters:** with **only six engineered features**, the efficiency indices contain enough signal for simple models to separate offense vs defense roles.

Method 1: K-Nearest Neighbors (KNN)

- **Model idea:** classify a player by the **majority label among the k most similar players** in feature space (distance-based).
- **Why KNN here:** it's a strong, simple baseline for **small, tabular, low-dimensional** numerical data (no heavy training).
- Key parameters used:
  - n_neighbors = 3 (final choice)
  - **Distance metric:** default **Euclidean**
  - **Train/test split:** test_size=0.2, random_state=42, with **stratification**
- **Why k = 3:** we tuned **k = 1..15**, saw an "elbow" with best performance around **k = 3–4**, and chose **k = 3** to keep neighborhoods local while maintaining high accuracy.

Method 2: Shallow Neural Network

- **Model idea:** learn a **nonlinear decision boundary** between Offensive vs Defensive using a small feed-forward network.
- **Why choose it:** with engineered indices, a small network can model interactions between features and performed best in our tests.
- Architecture + training parameters:
  - Dense layers: **32 (ReLU) → 16 (ReLU) → softmax output**
  - Optimizer: **Adam**; Loss: **categorical cross-entropy**
  - batch_size = 8, epochs = 50, validation_split = 0.1
  - Adam default learning rate **α ≈ 0.001**; (ReLU hidden, softmax output)
- **Hyperparameter reasoning:** validation accuracy stabilized after ~**15–20 epochs**, so training to 50 epochs ensured convergence without obvious overfitting on this small dataset.

Results (3-6 slides)

- Results of your prediction based on various metrics.

Evaluation Setup & Metrics Reported

- We evaluated two models: **KNN (k = 3)** and a small **feed-forward "MLP"** network.
- We report **Accuracy, Macro Precision, Macro Recall (Sensitivity), Macro F1**, plus **Confusion Matrices**.
- Two regimes emerged:
  - **3-class** (Offensive / Defensive / Playmaker): ~**50%** test accuracy for both models (class imbalance + few Playmaker labels).
  - **Binary** (Offensive vs Defensive) after removing Playmaker: performance improves substantially.

KNN Results

- With Playmaker (3-class):
  - Accuracy **0.50**, Precision **0.39**, Recall **0.39**, F1 **0.38**.
- After removing Playmaker
  - Accuracy **0.75,** Precision **0.4889**, Recall **0.5556**, F1 **0.5185**
- Classification report highlights the weakness of the rare class (Playmaker has **0.00** precision/recall/F1 with support 1 in the shown output).

Add KNN screenshot

Neural Net Results

- With Playmaker (3-class):
  - Accuracy **0.50**, Precision **0.17**, Recall **0.33**, F1 **0.22**.
  - Notes: model **favors Offensive**, and Playmaker recall is very low due to difficulty + imbalance.
- After removing Playmaker
  - Accuracy **0.86,** Macro Precision **0.88**, Macro Recall **0.88**, Macro F1 **0.86**
- Confusion matrix (rows = true [Defensive, Offensive], cols = predicted):
  -

3	0
1	3

Side-by-Side Summary

- Table 2 (Offensive vs Defensive, before PCA):
  - **KNN (k=3):** Accuracy **≈ 0.75**, Macro Precision **≈ 0.49**, Macro Recall **≈ 0.56**, Macro F1 **≈ 0.52**
  - **Neural net:** Accuracy **≈ 0.86**, Macro Precision **≈ 0.88**, Macro Recall **≈ 0.88**, Macro F1 **≈ 0.86**
- **Takeaway:** once Playmaker is removed, the engineered efficiency features provide enough signal for both models, with the neural net strongest overall.

PCA Ablation

- After applying PCA (2 components), **both models lose performance**:
  - KNN drops to about **0.57** accuracy
  - Neural net drops to about **0.43** accuracy
- Reason given: PCA maximizes variance, which **doesn't necessarily align** with the best separation direction for Offensive vs Defensive, and with only **six engineered features**, PCA mostly discards useful information.

Conclusion and Next Steps (1-3 slides)

- Summary of your work
- Next steps and limitations

Conclusion

- Built a pipeline to **classify NBA players into archetype-based role groups** using **per–36-minute stats** (including 2025 rookies).
- Engineered compact features like **Scoring Efficiency Index** and **Defensive Efficiency Index** to reduce feature complexity while preserving signal.
- Trained and evaluated **KNN** and a small **neural network (MLP)** to predict **Offensive vs Defensive** roles, and compared results across accuracy/precision/recall/F1 + confusion matrices.

Key Results

- 3-class prediction (Offensive/Defensive/Playmaker) struggled (~**50% accuracy**) largely due to **class imbalance** and too few Playmaker examples.
- After removing Playmaker and using **binary classification**, performance improved:
  - **KNN: ~0.75 accuracy**
  - **Neural net (MLP): ~0.86 accuracy**, with only **1 error** in the shown confusion matrix.
- PCA (2 components) **reduced** accuracy for both models, suggesting PCA compression removed useful separation information for this task.

Limitations & Next Steps

- Limitations
  - **Class imbalance** (especially Playmaker) caused biased predictions and weak minority-class metrics.
  - Using a small, engineered feature set (6ish features after indices) may miss nuanced role signals (spacing, usage, defensive versatility).
- Next Steps
  - **Fix class imbalance:** get more Playmaker labels or use **class weights / oversampling**.
  - **Broaden + validate:** add more features and test across **multiple seasons**.
  - **Try stronger tabular models:** e.g., logistic regression / random forest / boosting, then compare against KNN