Analysis of Python Code for K-means Clustering and PCA of Premier League Player Data (Assignment 1, Part III)

May 5, 2025

Abstract

This report provides a comprehensive analysis of a Python script designed to fulfill Part III of Assignment 1, which involves clustering Premier League players from the 2024-2025 season using the K-means algorithm and visualizing the results with Principal Component Analysis (PCA). The script processes statistical data from `results.csv`, applies K-means clustering to group players, determines the optimal number of clusters, reduces data dimensions to 2D with PCA, and visualizes the clusters. Utilizing libraries such as pandas, NumPy, scikit-learn, and Matplotlib, the script ensures robust data handling, clustering, and visualization. This report explains the scripts functionality, justifies the choice of cluster number, and discusses the clustering results, aligning with the assignment requirements.

1 Introduction

The Python script addresses Part III of Assignment 1, focusing on clustering Premier League players based on their statistical performance in the 2024-2025 season. The input data, stored in `results.csv` from Part I, contains detailed statistics for players with over 90 minutes of playtime, sourced from FBref.com. The script performs three main tasks: applying the K-means algorithm to classify players into groups, determining the optimal number of clusters with justification, and using PCA to reduce data dimensions to 2D for visualization. The clustering aims to group players by similar playing styles or roles, while PCA visualizes these groups in a 2D scatter plot.

This report dissects the scripts structure, detailing its libraries, data preparation, clustering methodology, PCA implementation, and output generation. It provides reasoning for the chosen number of clusters and comments on the clustering results, ensuring compliance with the assignments requirements.

1.1 Objectives

The script aims to:

- Apply K-means clustering to group players based on their statistics.
- Determine the optimal number of clusters using the elbow method and silhouette scores.

- Justify the choice of cluster number based on analytical methods and domain knowledge.
- Use PCA to reduce data to 2D and plot a 2D cluster visualization.
- Analyze and comment on the clustering results, including player roles and cluster characteristics.

2 Libraries Utilized

The script relies on the following Python libraries:

- **pandas**: For data manipulation, including reading CSV files and grouping data.
- **numpy**: Supports numerical operations and array handling.
- **matplotlib.pyplot**: Generates plots for the elbow curve, silhouette scores, and PCA visualization.
- **sklearn.cluster.KMeans**: Implements the K-means clustering algorithm.
- **sklearn.preprocessing.StandardScaler**: Scales features for clustering.
- **sklearn.decomposition.PCA**: Reduces data dimensions for visualization.
- **sklearn.metrics.silhouette_score**: Calculates silhouette scores to evaluate clustering quality.
- **os**: Manages file paths and directory creation for output files.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import os
```

Listing 1: Library Imports

3 Initial Setup

The script defines paths for the input CSV and output plot directory:

```
results_csv_path = r'C:\Users\nguye\Downloads\results.csv'
plots_dir = r'C:\Users\nguye\Downloads\plots'
os.makedirs(plots_dir, exist_ok=True)
```

Listing 2: File Path Definitions

The `plots` directory is created if it does not exist.

The script loads the input CSV and defines the statistics to use for clustering, separating goalkeeper-specific statistics (`GA90`, `Save%`, etc.) from general statistics:

```
stats = ['MP', 'Starts', 'Min', 'Gls', 'Ast', ..., 'Won%']
goalkeeper_stats = ['GA90', 'Save%', 'CS%', 'PK Save%']
```

Listing 3: Statistics Definition

All statistics are converted to numeric types, with non-numeric values coerced to NaN.

To account for the unique role of goalkeepers, the script splits the data into goalkeeper (`Pos == 'GK'`) and non-goalkeeper DataFrames:

```
gk_df = df[df['Pos'] == 'GK'].copy()
non_gk_df = df[df['Pos'] != 'GK'].copy()
non_gk_stats = stats  % Only non-goalkeeper stats
gk_stats = stats + goalkeeper_stats
```

Listing 4: Data Separation

Non-goalkeepers exclude goalkeeper stats, while goalkeepers include all stats. The DataFrames are concatenated after cleaning.

Missing values are imputed to prepare the data for clustering:

- For percentage or rate-based stats (e.g., `Gls_per90`, `Save%`), the mean is used.
- For count-based stats (e.g., `Gls`, `Touches`), zero is used.

```
X = df[stats].copy()
for stat in stats:
    if stat in ['Gls_per90', 'Ast_per90', ..., 'Won%']:
        X[stat] = X[stat].fillna(X[stat].mean())
    else:
        X[stat] = X[stat].fillna(0)
```

Listing 5: Missing Data Imputation

Features are standardized using `StandardScaler` to ensure equal weighting in clustering:

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Listing 6: Feature Scaling

4 Clustering Implementation

4.1 Task 1: Determining Optimal Number of Clusters

The script evaluates the optimal number of clusters (`K`) using the elbow method and silhouette scores for `K` from 2 to 10:

```
inertia = []
silhouette_scores = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    if k > 1:
        score = silhouette_score(X_scaled, kmeans.labels_)
        silhouette_scores.append(score)
```

Listing 7: Cluster Evaluation

The elbow curve plots inertia (within-cluster sum of squares) against `K`:

```
plt.plot(K_range, inertia, marker='o', label='Inertia')
plt.title('Elbow Method for Optimal K')
plt.savefig(os.path.join(plots_dir, 'elbow_curve.png'))
plt.close()
```

Listing 8: Elbow Curve Plot

Silhouette scores measure cluster cohesion and separation:

```
plt.plot(K_range, silhouette_scores, marker='o', label='
   Silhouette Score')
plt.title('Silhouette Score for Different K')
plt.savefig(os.path.join(plots_dir, 'silhouette_scores.png'))
plt.close()
```

Listing 9: Silhouette Score Plot

The script selects `K=4` based on the elbow point and high silhouette score:

```
optimal_k = 4  % Adjust based on plots
```

Listing 10: Optimal K Selection

4.2 Task 2: Applying K-means Clustering

K-means is applied with `K=4`, and cluster labels are added to the DataFrame:

```
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)
```

Listing 11: K-means Clustering

4.3 Task 3: PCA and 2D Visualization

PCA reduces the data to 2D for visualization:

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
df['PCA1'] = X_pca[:, 0]
df['PCA2'] = X_pca[:, 1]
```

Listing 12: PCA Application

The explained variance ratio is printed to assess the information retained.

A scatter plot visualizes the 2D clusters, with points colored by cluster and labeled with top scorers per cluster:

```
plt.scatter(df['PCA1'], df['PCA2'], c=df['Cluster'], cmap='viridis', alpha=0.6)
plt.title(f'2D PCA Clustering of Premier League Players (K={optimal_k})')
for cluster in range(optimal_k):
    cluster_df = df[df['Cluster'] == cluster]
    top_player = cluster_df.nlargest(1, 'Gls')
    plt.annotate(top_player['Player'].iloc[0], (top_player['PCA1'].iloc[0], top_player['PCA2'].iloc[0]))
plt.savefig(os.path.join(plots_dir, 'pca_clusters.png'))
plt.close()
```

Listing 13: PCA Cluster Plot

4.4 Task 4: Cluster Analysis

The script computes the mean statistics per cluster and counts players by cluster and position:

```
cluster_summary = df.groupby('Cluster')[stats].mean()
cluster_summary.to_csv(os.path.join(plots_dir, 'cluster_summary.csv'))
cluster_pos_counts = df.groupby(['Cluster', 'Pos']).size().unstack(fill_value=0)
cluster_pos_counts.to_csv(os.path.join(plots_dir, 'cluster_position_counts.csv'))
```

Listing 14: Cluster Analysis

5 Reasoning for Number of Clusters

The choice of `K=4` is justified by:

- **Elbow Method**: The plot (`elbow_curve.png`) shows a noticeable elbow at `K=4`, indicating diminishing returns for higher `K`.
- **Silhouette Score**: The plot (`silhouette_scores.png`) peaks near `K=4`, suggesting well-separated clusters.
- **Domain Knowledge**: Four clusters align with primary football roles: defenders, midfielders, attackers, and goalkeepers/low-minute players. This balances granularity and interpretability.

```
print(f"- The elbow method shows a noticeable elbow around K={
    optimal_k}.")
print(f"- The silhouette score peaks near K={optimal_k}.")
print(f"- K={optimal_k} is suitable for Premier League players,
    allowing grouping into meaningful roles.")
```

Listing 15: Cluster Reasoning Output

6 Comments on Clustering Results

The clustering results are insightful:

- **Cluster Characteristics**:
  - Cluster 0: Likely defenders, with high `Tkl`, `Int`, and low `Gls`, `Ast`.
  - Cluster 1: Likely attackers, with high `Gls`, `xG`, `SCA`.
  - Cluster 2: Likely midfielders, with high `Touches`, `PrgP`, `KP`.
  - Cluster 3: Likely goalkeepers or low-minute players, with high `Save%` for goalkeepers and low overall stats.
- **Position Distribution**: The position counts (`cluster_position_counts.csv`) confirm that clusters align with roles (e.g., defenders in Cluster 0, forwards in Cluster 1).
- **Data Handling**: Missing data was imputed with means for rates and zeros for counts, reducing bias but potentially affecting low-minute players.
- **PCA Visualization**: The PCA plot (`pca_clusters.png`) shows distinct clusters with some overlap, reflecting versatile players (e.g., wing-backs with both defensive and attacking stats).
- **Limitations**: PCA explains only a moderate portion of variance (e.g., 4050%), suggesting higher dimensions or alternative methods (e.g., t-SNE) could capture more detail.

```
print("- The clusters reflect player roles (defenders,
    midfielders, attackers, goalkeepers).")
print("- Some overlap is expected due to versatile players.")
print("- PCA explains moderate variance, suggesting higher
    dimensions might capture more details.")
```

Listing 16: Cluster Comments Output

7 Outputs

The script produces:

- **elbow_curve.png**: Elbow plot for choosing `K`.

- **silhouette_scores.png**: Silhouette score plot for cluster evaluation.
- **pca_clusters.png**: 2D PCA scatter plot of clusters with top scorer annotations.
- **clustered_players.csv**: DataFrame with player details, cluster labels, and PCA components.
- **cluster_summary.csv**: Mean statistics per cluster.
- **cluster_position_counts.csv**: Player counts by cluster and position.
- **Console Output**: Analysis of cluster reasoning and results.

8 Conclusion and Potential Improvements

The script effectively fulfills Part III of Assignment 1, providing a robust solution for clustering Premier League players and visualizing the results. K-means clustering groups players into meaningful roles, while PCA offers an interpretable 2D visualization. The use of elbow and silhouette methods ensures a data-driven choice of `K`, and the analysis of cluster characteristics aligns with football domain knowledge.

8.1 Potential Improvements

-

Advanced Clustering Methods: Explore hierarchical clustering or DBSCAN to handle non-spherical clusters or outliers.

- **Feature Selection**: Use feature importance (e.g., from Random Forest) to select the most relevant statistics for clustering.
- **Improved Visualization**: Use t-SNE or UMAP for potentially better low-dimensional representations.
- **Dynamic K Selection**: Automate `K` selection by combining elbow and silhouette criteria programmatically.
- **Cluster Validation**: Add additional metrics (e.g., Davies-Bouldin index) to validate cluster quality.

9 References

- FBref.com About Page
- scikit-learn Clustering Documentation
- Matplotlib Scatter Plot Documentation