Analysis of Python Code for Estimating Premier League Player Transfer Values (Assignment 1, Part IV)

May 5, 2025

Abstract

This report provides a detailed analysis of a Python script designed to fulfill Part IV of Assignment 1, which involves collecting player transfer values for the 2024-2025 Premier League season and proposing a method to estimate these values. The script processes statistical data from `results.csv`, matches players with hard-coded transfer values due to inaccessible web scraping, and uses an XGBoost regression model to predict market values. Utilizing libraries such as pandas, NumPy, scikit-learn, XGBoost, and fuzzywuzzy, the script ensures robust data handling, fuzzy matching, and model evaluation. This report explains the scripts functionality, justifies the feature selection and modeling approach, and discusses the results and limitations, aligning with the assignment requirements.

1 Introduction

The Python script addresses Part IV of Assignment 1, focusing on collecting transfer values for Premier League players with over 900 minutes of playtime in the 2024-2025 season and proposing a method to estimate these values. The input data, stored in `results.csv` from Part I, contains detailed player statistics from FBref.com. Due to the inability to access FootballTransfers.com, the script uses hard-coded transfer values for 50 players. It employs fuzzy matching to align player names and trains an XGBoost regression model to predict market values based on selected features. The script generates predictions, evaluates model performance, and visualizes results.

This report dissects the scripts structure, detailing its libraries, data processing, fuzzy matching, feature selection, modeling approach, and output generation. It justifies the chosen features and model, discusses limitations, and suggests improvements, ensuring compliance with the assignments requirements.

1.1 Objectives

The script aims to:

- Collect transfer values for players with over 900 minutes, using hard-coded data as a fallback.
- Match player names between `results.csv` and transfer data using fuzzy matching.

- Propose a method for estimating player values with XGBoost regression.
- Justify feature selection based on statistical relevance to market value.
- Evaluate model performance and visualize actual vs. predicted values and feature importance.
- Save results to specified files and generate a report summary.

2 Libraries Utilized

The script relies on the following Python libraries:

- **pandas**: For data manipulation, including reading CSV files and merging datasets.
- **numpy**: Supports numerical operations and array handling.
- **fuzzywuzzy**: Performs fuzzy string matching to align player names.
- **xgboost**: Implements the XGBoost regression model for value prediction.
- **sklearn.model_selection**: Provides tools for train-test splitting, grid search, and cross-validation.
- **sklearn.preprocessing**: Scales features for modeling.
- **sklearn.metrics**: Calculates mean absolute error (MAE) and Rš for model evaluation.
- **matplotlib.pyplot**: Generates plots for actual vs. predicted values and feature importance.
- **os**: Manages file paths and directory permissions.
- **unicodedata**: Normalizes player names by removing diacritics.
- **stat**: Sets directory permissions.

```
import pandas as pd
import numpy as np
from fuzzywuzzy import fuzz
import xgboost as xgb
from sklearn.model_selection import train_test_split,
   GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import os
import unicodedata
import stat
```

Listing 1: Library Imports

3 Initial Setup

The script defines paths for input and output files and includes version checking for debugging:

```
SCRIPT_VERSION = "2025-05-09-v9"
results_csv_path = r'C:\Users\nguye\Downloads\results.csv'
transfer_values_csv_path = r'C:\Users\nguye\Downloads\
   transfer_values.csv'
model_results_csv_path = r'C:\Users\nguye\Downloads\
   player_value_predictions.csv'
plots_dir = r'C:\Users\nguye\Downloads\plots'
```

Listing 2: File Path and Version Definitions

The `plots` directory is created with appropriate permissions.

The script loads `results.csv`, filters for players with over 900 minutes, and removes duplicates:

```
df = pd.read_csv(results_csv_path)
df['Min'] = pd.to_numeric(df['Min'], errors='coerce')
df = df[df['Min'] > 900]
df = df.drop_duplicates(subset=['Player'], keep='first')
```

Listing 3: Data Loading and Filtering

4 Name Normalization and Matching

Player names are normalized to handle variations (e.g., diacritics, aliases):

```
def normalize_name(name):
    name = ''.join(c for c in unicodedata.normalize('NFD', name)
       if unicodedata.category(c) != 'Mn')
    parts = [part.strip('.').strip() for part in name.lower().
       split()]
    parts = [part for part in parts if part not in ['de', '
       andrade']]
    name = ' '.join(parts)
    alias_map = {'rodri': 'rodrigo hernandez', 'j grealish': '
       jack grealish', ...}
    return alias_map.get(name, name)
```

Listing 4: Name Normalization

Fuzzy matching aligns hard-coded transfer values with players in `results.csv` using a two-pass approach (thresholds of 80 and 75):

```
for _, row in manual_df.iterrows():
    normalized_name = row['Normalized_Player']
    value = row['Transfer_Value']
    best_match = None
    best_score = 0
    threshold = 80
```

```
    for target_player, target_normalized in zip(players,
       normalized_players):
        score = fuzz.ratio(normalized_name, target_normalized)
        if score > best_score and score >= threshold:
            best_score = score
            best_match = target_player
    if not best_match:
        threshold = 75
        ...
```

Listing 5: Fuzzy Matching

Unmatched players are saved to `unmatched_players.txt` for debugging.

5 Transfer Value Collection

Due to the inability to access FootballTransfers.com, the script uses hard-coded transfer values for 50 players:

```
transfer_data = [
    {"Player": "Aaron Wan-Bissaka", "Transfer_Value": 20},
    {"Player": "Erling Haaland", "Transfer_Value": 150},
    ...
]
manual_df = pd.DataFrame(transfer_data)
```

Listing 6: Hard-Coded Transfer Data

The data is saved to `transfer_values.csv` after matching.

6 Player Value Estimation

6.1 Feature Selection

The script selects features relevant to market value:

- **Numeric Features**: `Age`, `Min`, `Gls`, `Ast`, `xG`, `xAG`, `PrgC_standard`, `PrgP_standard`, `PrgR_standard`, `Tkl`, `Int`, `Blocks`, `Touches`, `Carries`, `Rec`, `Won%`, `Save%`, `CS%`, `Gls_Ast` (derived).
- **Categorical Features**: `Pos`, `Squad` (one-hot encoded).

```
features = ['Age', 'Min', 'Gls', 'Ast', 'xG', 'xAG', ..., '
   Gls_Ast']
categorical_features = ['Pos', 'Squad']
```

Listing 7: Feature Definition

Justification:

- `Age`: Younger players often have higher market values due to potential.
- `Min`: Indicates playing time and importance to the team.

- `Gls`, `Ast`, `xG`, `xAG`: Reflect offensive contributions and expected impact.
- `PrgC`, `PrgP`, `PrgR`: Capture progressive play, valued in modern football.
- `Tkl`, `Int`, `Blocks`: Indicate defensive contributions.
- `Touches`, `Carries`, `Rec`: Reflect involvement in play.
- `Won%`, `Save%`, `CS%`: Capture aerial and goalkeeping performance.
- `Gls_Ast`: Combines goal and assist contributions.
- `Pos`, `Squad`: Account for role-specific and team-specific value differences.

6.2 Model Selection

The script uses an XGBoost regression model due to its ability to handle non-linear relationships, feature interactions, and small datasets:

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
    random_state=42, early_stopping_rounds=10)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0]
}
grid_search = GridSearchCV(xgb_model, param_grid, cv=5, scoring='
    neg_mean_absolute_error')
```

Listing 8: XGBoost Model

Justification:

- XGBoost is robust to outliers and missing data, suitable for the small dataset.
- Grid search optimizes hyperparameters to improve performance.
- Early stopping prevents overfitting.
- Cross-validation ensures reliable performance estimates.

6.3 Model Training and Evaluation

The data is split into training (80%) and test (20%) sets, with features scaled:

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
    test_size=0.2, random_state=42)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Listing 9: Train-Test Split and Scaling

The model is evaluated using MAE, Rš, and cross-validation:

```
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)
cv_scores = cross_val_score(xgb_model_no_early, X_scaled, y, cv
   =5, scoring='neg_mean_absolute_error')
```

Listing 10: Model Evaluation

6.4 Visualization

The script generates two plots:

- **Actual vs. Predicted**: A scatter plot comparing actual and predicted market values (`actual_vs_predicted.png`).
- **Feature Importance**: A bar plot of the top 10 features (`feature_importance.png`).

```
plt.barh(importance_df['Feature'][:10], importance_df['Importance
   '][:10])
plt.title('Top 10 Features Influencing Market Value')
plt.savefig(os.path.join(plots_dir, 'feature_importance.png'))
```

Listing 11: Feature Importance Plot

7 Outputs

The script produces:

- **transfer_values.csv**: Player statistics with matched transfer values.
- **player_value_predictions.csv**: Player data with actual and predicted market values.
- **unmatched_players.txt**: List of unmatched players (e.g., Rodri, Jack Grealish).
- **actual_vs_predicted.png**: Scatter plot of actual vs. predicted values.
- **feature_importance.png**: Bar plot of top feature importances.
- **report_summary.md**: Summary of data collection, feature selection, model performance, and limitations.
- **Console Output**: Debugging information, model metrics, and top features.

8 Limitations

The script notes several limitations:

- **Small Dataset**: Only 50 players were matched (out of 307), limiting model reliability (low Rš, high MAE).

- **Hard-Coded Data**: Reliance on manual transfer values due to inaccessible web scraping.
- **Unmatched Players**: Five players (e.g., Rodri, Jack Grealish) were unmatched, possibly due to low minutes or name mismatches.
- **Model Performance**: Low Rš and high MAE indicate poor predictive power, likely due to the small dataset.

9 Conclusion and Potential Improvements

The script fulfills Part IV of Assignment 1 by collecting transfer values (via hard-coded data), matching players with fuzzy logic, and estimating market values using XGBoost. The feature selection captures key performance metrics, and the model provides insights despite limitations. The outputs align with the assignments requirements, with clear visualizations and a detailed report.

9.1 Potential Improvements

- **Web Scraping**: Implement robust scraping for FootballTransfers.com to expand the dataset.
- **Enhanced Matching**: Use additional identifiers (e.g., team, position) to improve fuzzy matching accuracy.
- **Feature Engineering**: Add features like contract length, international caps, or recent form.
- **Model Alternatives**: Test other models (e.g., Random Forest, Neural Networks) for comparison.
- **Data Augmentation**: Incorporate historical transfer data to increase sample size.

10 References

- FBref.com About Page
- XGBoost Documentation
- scikit-learn Model Evaluation Documentation
- Matplotlib Scatter Plot Documentation