AI Capstone Project #1

110550129 姚咨妙

Weblink to the dataset and code:

https://github.com/yuiolyzm/AI-Capstone/tree/main/dataset

1 Description

There's a football video game series called "EA SPORTS FC." The game has player ratings updated every year for more than 17,000 fully licensed players, and it generally reflects players' overall performance, strongness and weakness. Thus, I was wondering if it's possible to classify players' positions only based on score of different ability in this game.

In this project, I collected the detailed score from player ratings in EA SPORTS FC 25 (hereinafter referred to as FC 25) webpage along with the position of player, conducted a few experiments with several machine learning algorithms, and analyzed the result.

2 Dataset

The dataset is composed of the scores of the first 2,000 players in the player ratings, including "Overall," "Pace," "Shooting," "Passing," "Dribbling," "Defending," and "Physicality." There are 12 positions being the categories for prediction, and the number of data for different positions varied from 60 to 300. All the scores are copied from FC 25 ratings on the internet. The dataset is shared with my classmate whose student ID is 110550097.

2.1 Data Collection

I simply downloaded the HTML file of FC 25 ratings website, wrote python code to extract the position and scores of each player in the first twenty pages, which meant the 2,000 highest overall scores, and saved them to a csv file.

2.2 Data Format

The dataset is a csv file with columns "Position," "Overall," "Pace," "Shooting," "Passing," "Dribbling," "Defending," and "Physicality." The column of "Position" has 12 categories which are abbreviation composed of two to three English letters, and the other columns are integer scores.

Position includes 4 main categories, which are forward (FW), midfielder (MF), defender (DF), and goalkeeper (GK). Forward consists of strikers (ST), right wing (RW) and left wing (LW). Midfielder is composed of central attacking midfielder (CAM), central midfielder (CM), central defending midfielder (CDM), right midfielder (RM), and left midfielder (LM). Defender consists of central back fielder (CB), left back fielder (LB), right back fielder (RB). There are in total 12 kinds of positions in FC 25.

3 Methods

3.1 Logistic Regression (LR)

Logistic Regression is a simple yet effective classification algorithm that models the probability of a sample belonging to a particular class using a sigmoid function. In this experiment, it was used as a baseline model for predicting player positions based on performance attributes. Since logistic regression assumes a linear decision boundary, its performance is dependent on the separability of the dataset

3.2 Random Forest Classifier

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the most common class as the final prediction. This approach helps reduce overfitting and increases robustness compared to a single decision tree. The model was particularly effective due to its ability to capture complex relationships between features. Hyperparameter tuning, including the number of trees and tree depth, can be used to optimize performance.

3.3 Support Vector Machine (SVM)

SVM is a powerful classification algorithm that finds the optimal hyperplane to separate different classes. Since player position classification is a multi-class problem, a one-vs-rest (OvR) approach was used. Feature scaling was applied before training to ensure proper distance-based calculations.

3.4 K-Nearest Neighbors (KNN)

KNN is a non-parametric, distance-based classifier that assigns a class to a sample based on the majority vote of its K nearest neighbors. The key challenge in using KNN is selecting an appropriate K value. Cross-validation was used to determine the optimal K, and results showed that performance varies significantly depending on this choice. Additionally, the dataset was standardized before applying KNN to ensure fair distance measurements.

3.5 K-means

K-Means is an unsupervised clustering algorithm that partitions data into K clusters by minimizing the variance within each cluster. Since the number of player positions is known, we experimented with different values of K to see how well the algorithm could cluster players into meaningful groups. Inertia (sum of squared distances to the cluster centers) and silhouette score (a measure of how well-defined the clusters are) were used to determine the best K.

3.6 Feature Scaling:

Since models like SVM, KNN, and K-Means rely on distance-based calculations, all features were standardized using z-score normalization to ensure equal weighting.

3.7 Data Resampling:

The dataset was imbalanced across different player positions. To ensure fair representation, an equal number of samples per position were randomly selected before training the models.

4 Experiments

There are originally 12 classes as prediction result. However, considering the result being worse than I expected, I decided to decrease the number of classes. Since the data I collected didn't involve any information about left or right position, even the weak foot ability, it's obviously suitable to merge left wing (LW) and right wing (RW) into one, so does right midfielder (RM), left midfielder (LM), right back fielder (RB) and left back fielder (LB). This is the dataset with nine categories I used for experiments. For further comparison, a dataset with only four categories including forward, midfielder, defender, and goalkeeper is created.

There are another two balanced datasets for experiments. The one with nine kinds of positions has one hundred pieces of data for each position, and the other with four kinds of positions has two hundred pieces of data for each position.

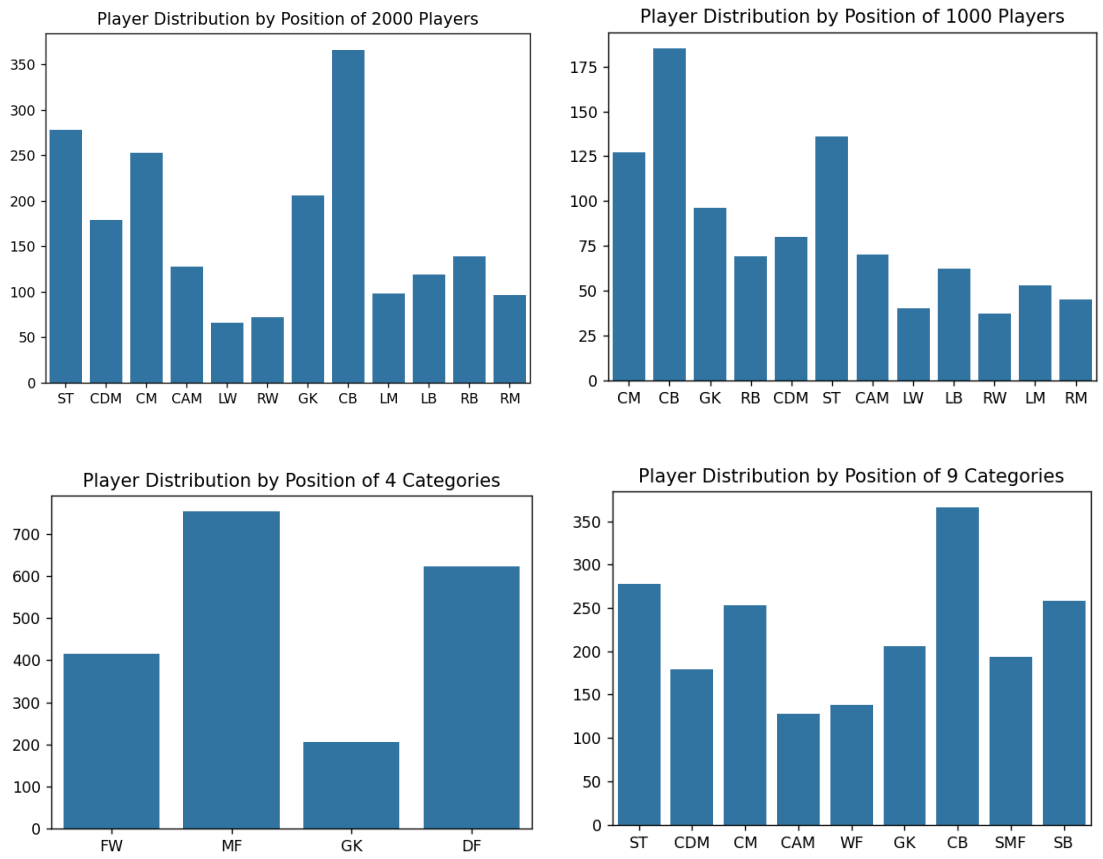

Fig.1. Player distribution of 4 imbalanced dataset

4.1 Metrics

The metrics I used to evaluate the performance of supervised learning methods are accuracy, precision, recall, F1 score and confusion matrix. For unsupervised learning method, the metrics are inertia and silhouette score.

4.2 Parameters

The experiments adopt 5-fold cross-validation. The K for KNN is set to 8, and the process for selecting is in part 5.1.

5 Analysis

5.1 Find best K in KNN

To optimize the result of KNN, I have tried different K values from one to twenty-one and visualized the performance in different metrics. The performance of K=8 is obviously the best. Thus, the optimal K in KNN is eight.

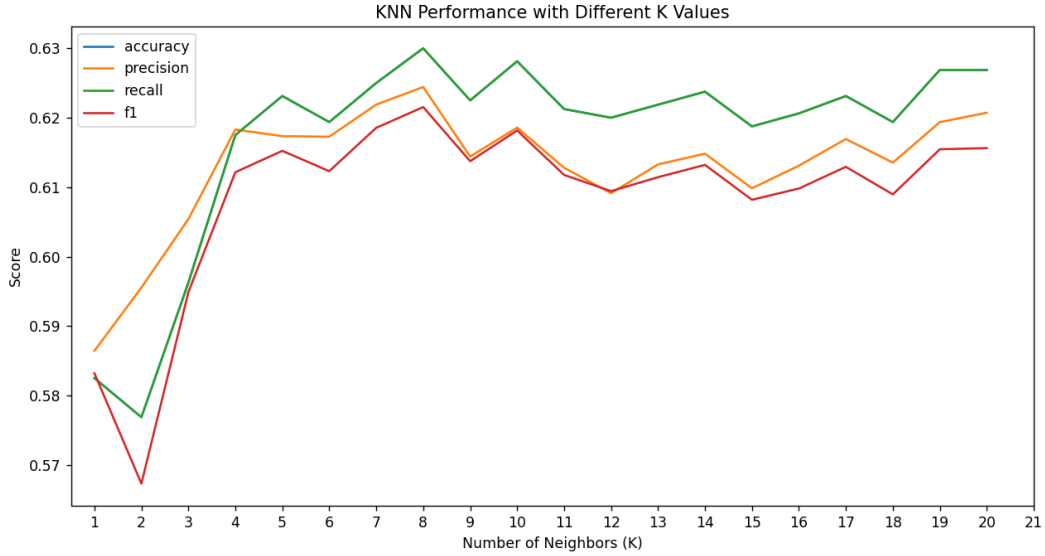

Fig.2 The performance of K in KNN from 1 to 21

5.2 Result of different methods with several datasets

5.2.1 Imbalanced data

In comparison with datasets, the dataset with only four kinds seems to be better than the others. It is a reasonable result due to the decrease in categories and the increase in the difference between categories. For datasets with nine kinds, the prediction result of balanced dataset seems to be worse than the unbalanced one, the result is even similar to datasets with 12 kinds. With similar pieces of data, it can be harder to figure out the difference of similar categories, for example CAM, CM, and CDM. For unbalanced data, the number of CAM player is more than the other two categories, it may have better result to predict the player being CAM when countering similar score distributions.

5.2.2 Number of data

The result of Data2000 seems to be worse than the result of Data1000 in most of the methods. The increase in data may not be helpful to prediction. However, the reduction of data makes the result of KNN even worse, which suggests that the number of K may need to be lower than the original one.

5.2.3 Comparison between different methods

The results of Logistic Regression (LR), Random Forest, and Support Vector Machine (SVM) seem similar in figures. In numerical results, Random Forest has the best performance with 4 kinds and 4 kinds of balanced dataset, and SVM has the best performance with the other datasets.

Fig.3 The result of LR

Fig.4 The result of Random Forest

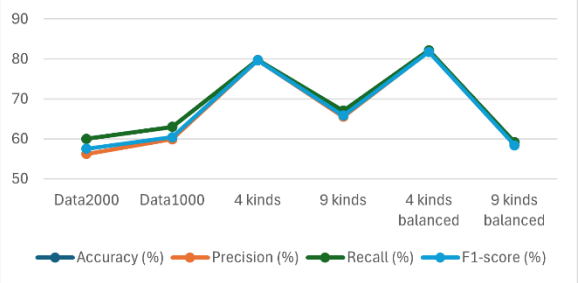

Fig.5 The result of SVM

Fig.6 The result of KNN

5.2.4 Result of K-means (unsupervised method)

Balanced datasets seem to have better results than the unbalanced datasets. For inertia, Data1000 and the two balanced datasets get the better results, and for silhouette score, 4 kinds and the two balanced datasets performs better. The inference is that the data in Data1000 is more similar thus more concentrated, with the decrease of its silhouette score. As for 4 kinds, the difference between categories may have increased. Hence, balanced datasets are more reasonable for K-means to experiment with for their overall better performance result.

K-Means	Inertia	Silhouette Score (%)
Data2000	3940.22	20.16
Data1000	1952.21	19.92
4 kinds	6622.97	23.24
9 kinds	4504.80	19.68
4 kinds balanced	2595.07	25.49
9 kinds balanced	2049.93	20.80

6 Discussion

6.1 Experiments result

To my surprise, the performance of SVM didn't exceed the other methods a lot. It seems

that logistic regression has almost as great performance as random forest and SVM.

The balanced dataset was supposed to have a better performance than the unbalanced one, and so was the dataset with more data, in my expectation. However, some of the results were quite opposite that I took a long time to come up with the possible inference.

6.2 Factors that affect the performance

Two of the most important factors must be the number of categories and dataset balance. After reducing the number of categories to a reasonable number, the performance can be improved a lot. The balance of datasets can make predictions better or worse, depending on the dataset, which is somehow confusing.

The hyperparameter can also affect performance, including the choice of K in KNN, the K in K-means and the number of trees and depth in random forest.

6.3 Experiments if available

If there's more time available, I would like to do feature engineering, dimensionality reduction, and try some deep learning methods. Investigating whether feature combinations improve classification and using PCA to reduce the number of features sounds to be helpful in investigating the dataset. DL models may have different results than the methods I used this time, which can give me profound understanding of different classification methods.

6.4 What I learned and remaining questions

Before this project, I mostly worked with existing, well-structured datasets. I had limited experience with data collection and preprocessing, which are crucial steps in building a successful machine learning model. Extracting relevant features, handling class imbalances, and selecting appropriate preprocessing techniques all have a great impact on model performance.

Additionally, analyzing different performance metrics helped me understand the trade-offs between various classification algorithms. For instance, I observed how KNN's performance is highly dependent on the choice of K, while Random Forest's ensemble approach tends to be more robust.

Appendix

Linear regression

LR	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Data2000	59.38	55.61	59.38	57.01
Data1000	60.75	57.07	60.75	57.89
4 kinds	79.44	79.43	79.44	79.42
9 kinds	66.56	65.25	66.56	65.51
4 kinds balanced	82.19	81.99	82.19	81.87
9 kinds balanced	57.92	56.91	57.92	57.13

Random Forest

Random Forest	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Data2000	57.81	54.54	57.81	55.80
Data1000	59.75	56.35	59.75	57.19
4 kinds	**80.87**	**80.88**	**80.87**	**80.86**
9 kinds	65.81	64.58	65.81	65.03
4 kinds balanced	**82.97**	**82.66**	**82.97**	**82.63**
9 kinds balanced	56.11	55.37	56.11	55.51

Support Vector Machine

SVM	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Data2000	**60.00**	**56.23**	**60.00**	**57.55**
Data1000	**63.00**	**59.92**	**63.00**	**60.40**
4 kinds	79.75	79.70	79.75	79.71
9 kinds	**67.06**	**65.51**	**67.06**	**65.81**
4 kinds balanced	82.19	81.87	82.19	81.78
9 kinds balanced	**59.17**	**58.69**	**59.17**	**58.36**

K-Nearest Neighbors

KNN	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Data2000	56.44	54.06	56.44	54.31
Data1000	55.87	53.12	55.87	53.01
4 kinds	78.94	78.82	78.94	78.72
9 kinds	63.00	62.44	63.00	62.15
4 kinds balanced	82.03	82.03	82.03	81.45
9 kinds balanced	55.28	56.53	55.28	55.03

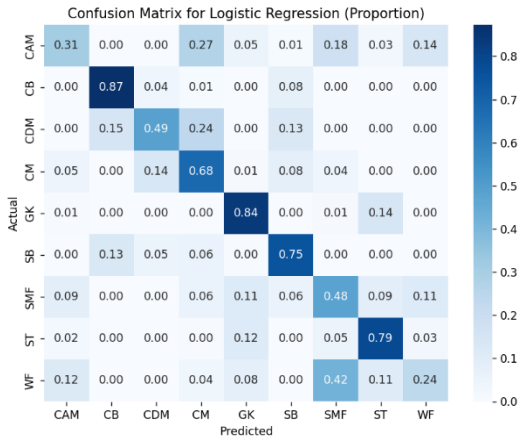

Fig. A1 Confusion matrix of 9 kinds using LR

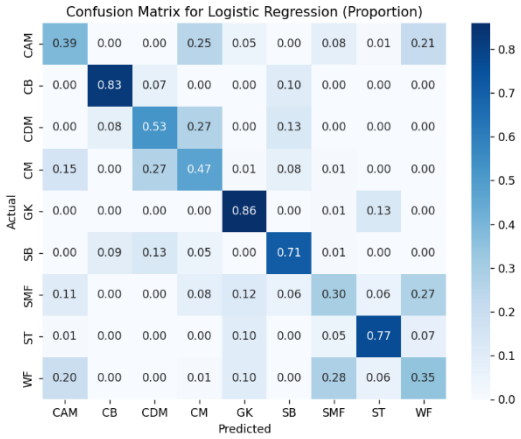

Fig. A2 Confusion matrix of 9 kinds using LR

Code

Data extraction and formulation of different datasets

```python
import pandas as pd
from bs4 import BeautifulSoup
import glob

def extract_player_data(html_file, output_csv):
    with open(html_file, "r", encoding="utf-8") as file:
        soup = BeautifulSoup(file, "html.parser")

    players = []

    # Extract player positions and stats
    positions = [pos.get_text(strip=True) for pos in soup.find_all("span",
        class_="Table_tag__vKZKn generated_utility20sm__ZX2Hf generated_utility19md__XKkU_")]
    stat_elements = soup.find_all("span", class_="Table_statCellValue__zn5Cx")
    stats = [stat.get_text(strip=True)[:2] for stat in stat_elements]

    # Assuming each player has 14 stats, split the stats into chunks
    num_stats = 14
    stats_chunks = [stats[i:i+num_stats] for i in range(0, len(stats), num_stats)]

    # Ensure data consistency
    for i in range(min(len(positions), len(stats_chunks))):
        players.append([positions[i]] + stats_chunks[i][:7])

    columns = ["Position", "Overall", "Pace", "Shooting", "Passing", "Dribbling", "Defending", "Physicality"]
    df = pd.DataFrame(players, columns=columns)
    df.to_csv(output_csv, index=False, encoding="utf-8-sig")

    print(f"Data successfully saved to {output_csv}")

def merge_csv_files(input_folder, output_csv):
    csv_files = glob.glob(f"{input_folder}/*.csv")
    dataframes = [pd.read_csv(file) for file in csv_files]

    merged_df = pd.concat(dataframes, ignore_index=True)
    merged_df = merged_df.drop_duplicates()
    merged_df.to_csv(output_csv, index=False, encoding="utf-8-sig")
    print(f"Merged CSV saved as {output_csv}")
```

```python
def revise_position(input_csv, output_csv):
    df = pd.read_csv(input_csv)
    df["Position"] = df["Position"].apply(lambda x: "FW" if x in ["ST", "LW", "RW"]
        else "MF" if x in ["CM", "CAM", "CDM", "LM", "RM"] else "DF" if x in ["CB", "LB", "RB"] else "GK")
    df.to_csv(output_csv, index=False, encoding="utf-8-sig")
    print(f"Revised data saved as {output_csv}")

def revise_9position(input_csv, output_csv):
    df = pd.read_csv(input_csv)
    df["Position"] = df["Position"].apply(lambda x: "WF" if x in ["LW", "RW"]
        else "SMF" if x in ["LM", "RM"] else "SB" if x in ["LB", "RB"] else x)
    df.to_csv(output_csv, index=False, encoding="utf-8-sig")
    print(f"Revised data saved as {output_csv}")

def get_random_players_data(input_csv, output_csv, num_samples):
    df = pd.read_csv(input_csv)
    sampled_df = df.groupby("Position").apply(lambda x: x.sample(n=num_samples,
                    random_state=42, replace=False)).reset_index(drop=True)
    sampled_df.to_csv(output_csv, index=False, encoding="utf-8-sig")
    print(f"Random sampled data saved as {output_csv}")

def get_random_total_players_data(input_csv, output_csv, num_samples):
    df = pd.read_csv(input_csv)
    sampled_df = df.sample(n=num_samples, random_state=42, replace=False)
    sampled_df.to_csv(output_csv, index=False, encoding="utf-8-sig")
    print(f"Random sampled data saved as {output_csv}")

""" extract data """
# for i in range(1, 21):
#     html_file = f"FC25players{i}.html"
#     output_csv = f"players_data{i}.csv"
#     extract_player_data(html_file, output_csv)

""" merge data """
# merge_csv_files("C:\code_sem\senior\AIC\AIC HW1\dataset\data_1-20", "data2000.csv")
```

```
""" revise position to 4 categories """
# revise_position("dataset/data2000.csv", "dataset/4categories.csv")

""" revise position to 9 categories """
# revise_9position("dataset/data2000.csv", "dataset/9categories.csv")

""" get 200 players data for each position """
# get_random_players_data("dataset/4categories.csv", "dataset/4cate_200.csv", 200)

""" get 100 players data for each position """
# get_random_players_data("dataset/9categories.csv", "dataset/9cate_100.csv", 100)

""" get random 1000 players data """
get_random_total_players_data("dataset/data2000.csv", "dataset/data1000.csv", 1000)
```

Experiments with different models

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, silhouette_score

df = pd.read_csv("dataset/data1000.csv")

# features and labels
features = ["Overall", "Pace", "Shooting", "Passing", "Dribbling", "Defending", "Physicality"]
X = df[features]
y = df["Position"]

# convert categorical labels to numerical labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel='linear'),
    "KNN": KNeighborsClassifier(n_neighbors=8)
}
```

```

# 5-fold cross-validation for each model
for name, model in models.items():
    y_pred = cross_val_predict(model, X_train, y_train, cv=5)
    accuracy = accuracy_score(y_train, y_pred)
    precision = precision_score(y_train, y_pred, average='weighted')
    recall = recall_score(y_train, y_pred, average='weighted')
    f1 = f1_score(y_train, y_pred, average='weighted')
    cm = confusion_matrix(y_train, y_pred, normalize='true')    # normalize by row

    print(f"{name} - Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-score: {f1:.4f}")

    # plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt=".2f", cmap="Blues", xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix for {name} (Proportion)")
    plt.show()
```

```
58
59  # K-Means Clustering
60  kmeans = KMeans(n_clusters=len(np.unique(y)), random_state=42, n_init=10)
61  kmeans.fit(X_scaled)
62  kmeans_inertia = kmeans.inertia_
63  kmeans_silhouette = silhouette_score(X_scaled, kmeans.labels_)
64
65  print(f"K-Means Clustering - Inertia: {kmeans_inertia:.4f}, Silhouette Score: {kmeans_silhouette:.4f}")
66
67
68  """ find the best K value for KNN """
69  # # find the best K value (KNN)
70  # k_values = range(1, 21)
71  # knn_scores = {"accuracy": [], "precision": [], "recall": [], "f1": []}
72
73  # for k in k_values:
74  #     knn = KNeighborsClassifier(n_neighbors=k)
75  #     y_pred = cross_val_predict(knn, X_train, y_train, cv=5)
76  #     knn_scores["accuracy"].append(accuracy_score(y_train, y_pred))
77  #     knn_scores["precision"].append(precision_score(y_train, y_pred, average='weighted'))
78  #     knn_scores["recall"].append(recall_score(y_train, y_pred, average='weighted'))
79  #     knn_scores["f1"].append(f1_score(y_train, y_pred, average='weighted'))
80
81  # # plot KNN's K value vs evaluation metrics
82  # plt.figure(figsize=(12, 6))
83  # for metric, scores in knn_scores.items():
84  #     plt.plot(k_values, scores, label=metric)
85  # plt.xlabel("Number of Neighbors (K)")
86  # plt.ylabel("Score")
87  # plt.title("KNN Performance with Different K Values")
88  # x_major_locator = MultipleLocator(1)
89  # ax = plt.gca()
90  # ax.xaxis.set_major_locator(x_major_locator)
91  # plt.xlim(0.5, 21)
92  # plt.legend()
93  # plt.show()
94
95  # # find the best K value
96  # best_k_knn = k_values[np.argmax(knn_scores["f1"])]
97  # print(f"Best K for KNN: {best_k_knn}")
```

Reference

1. https://www.ea.com/zh-hant/games/ea-sports-fc/ratings

2.https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

3. https://www.datacamp.com/tutorial/random-forests-classifier-python

4. https://scikit-learn.org/stable/modules/svm.html

5. https://www.geeksforgeeks.org/k-means-clustering-introduction/

6. I have used AI tools to help me with part 3 methods introduction and coding.