



Final project ML.

LEARNING.

(Machine Learning Modeling to Evaluate
the Value of Football Players)

Giảng viên hướng dẫn: cô Diệp

Thành viên nhóm

Hồ Cảnh Quyền : 22022629
Phạm Anh Quân : 22022625
Lý Quốc An : 22022660

**Chủ đề: Dự đoán giá trị cầu thủ
bóng đá**

BÀI BÁO CÁO

MACHINE LEARNING.

I. INTRODUCTION.

II. DATA.

1. Giới Thiệu chung về dữ liệu sử dụng.
2. Làm sạch và tiền xử lý dữ liệu.
3. Visualize dữ liệu

III. MODELS.

1. Linear Regression.
 - a. Lựa chọn feature.
 - b. phân chia train-val-test.

Final project - yêu cầu, tiêu chí chấm điểm
(notion.site).

<https://github.com/quyencanh203/Project-ML>

LINK NOTION:

<https://excellent-zebu-ec6.notion.site/Final-project-ML-a8fb4c2ab7a948018ff5a1dd1f1a7b7b?pvs=4>

c1. TRAIN VỚI MÔ HÌNH CƠ BẢN NHẤT
BASELINE.

c.2 CHỌN ĐẶC TRƯNG VỚI RFE.

sklearn.feature_selection.RFE — scikit-learn 1.4.2 documentation

c.3 HỒI QUI RIDGE VÀ LASSO.

c.3 THỬ NGHIỆM VỚI
PolynomialFeatures.

2. Random Forest.

a. Lựa chọn feature.

b. phân chia train-val-test.

c1. TRAINING VỚI MÔ HÌNH FOREST CƠ
BẢN.

c.2 TUNNING VỚI GRIDSEARCH.

c.2 TUNNING VỚI RANDOM SEARCH.

3. XGBoost.

Kết quả tìm kiếm siêu tham số:

4.Gradient Boosting

IV. Phần 4. Kết luận

I. INTRODUCTION.

Trong ngành bóng đá hiện đại, việc đánh giá giá trị của cầu thủ là một phần quan trọng của quản lý đội bóng và quyết định chiến lược. Trước đây, quyết định về việc mua bán cầu thủ thường dựa vào cảm nhận cá nhân của những người chuyên môn, sự nổi tiếng, hoặc thậm chí là may mắn. Tuy nhiên, trong thời đại công nghệ thông tin phát triển mạnh mẽ, sự phổ biến của dữ liệu thống kê đã mở ra cơ hội để áp dụng các phương pháp học máy để đánh giá giá trị của các cầu thủ bóng đá.

Mục tiêu của nghiên cứu này là xây dựng một mô hình học máy để dự đoán giá trị thị trường của các cầu thủ bóng đá, dựa trên các chỉ số thống kê của họ trong mùa giải. Thay vì dựa vào cảm nhận chủ quan, mô hình sẽ sử dụng dữ liệu khách quan về hiệu suất cá nhân của cầu thủ, bao gồm số bàn thắng, kiến tạo, phạt ghi bàn, và nhiều chỉ số khác.

Qua việc áp dụng học máy vào vấn đề này, chúng ta có thể tạo ra một công cụ hữu ích cho các CLB bóng đá, nhà quản lý và các nhà đầu tư trong việc đưa ra quyết định mua bán cầu thủ, định giá hợp đồng và xây dựng đội hình một cách hiệu quả và chính xác. Điều này sẽ giúp tăng cường tính minh bạch và tính công bằng trong thị trường chuyển nhượng cầu thủ, đồng thời cũng giảm thiểu rủi ro và tối ưu hóa hiệu suất của các đội bóng.

II. DATA.

1. Giới Thiệu chung về dữ liệu sử dụng.

- a) Nguồn dữ liệu: Dữ liệu được tìm kiếm và thu thập từ <https://www.kaggle.com/search>
- b) Giải thích lí do lựa chọn kaggle để lấy dữ liệu:
 - +) Kaggle cung cấp một kho dữ liệu khổng lồ với nhiều lĩnh vực khác nhau như y tế, tài chính, thể thao, hình ảnh, văn bản, v.v. Điều này giúp chúng ta dễ dàng tìm thấy dữ liệu phù hợp cho các dự án.
 - +) Dữ liệu trên Kaggle thường được kiểm tra và làm sạch bởi cộng đồng, đảm bảo chất lượng cao hơn so với nhiều nguồn dữ liệu tự do khác trên internet.
- c) Giới thiệu về bộ dữ liệu được sử dụng cho dự án:
 - +) Sơ lược về bộ FootballData.csv: Đây là bộ dữ liệu gốc, chưa làm sạch, bộ dữ liệu này chứa các thông tin cơ bản của một cầu thủ, và giá trị chuyển nhượng của cầu thủ đó.

	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club_name	...	lwb	ldm	cd
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	33	24/06/1987	170	72	Argentina	FC Barcelona	...	66+3	65+3	65+
1	20801	https://sofifa.com/player/20801/cristiano-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	35	05/02/1985	187	83	Portugal	Juventus	...	65+3	61+3	61+
2	200389	https://sofifa.com/player/200389/jan-oblak/210002	J. Oblak	Jan Oblak	27	07/01/1993	188	87	Slovenia	Atlético Madrid	...	32+3	36+3	36+
3	188545	https://sofifa.com/player/188545/robert-lewandowski/...	R. Lewandowski	Robert Lewandowski	31	21/08/1988	184	80	Poland	FC Bayern München	...	64+3	65+3	65+
4	190871	https://sofifa.com/player/190871/neymar-da-silva-santos-júnior/...	Neymar Jr	Neymar da Silva Santos Júnior	28	05/02/1992	175	68	Brazil	Paris Saint-Germain	...	67+3	62+3	62+

- +) Sơ lược về bộ dữ liệu data_clean.csv: Đây là bộ dữ liệu đã được làm sạch, lọc ra những feature cần thiết từ bộ dữ liệu, xử lý những dữ liệu bị thiếu từ bộ dữ liệu gốc.

2. Làm sạch và tiền xử lý dữ liệu.

a) Import thư viện và đọc dữ liệu:

Import libraries

```
[21] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
[21]    ✓ 0.0s Python
```

Load data

```
[22] data = pd.read_csv('../data/raw/FootballData.csv')
[22]    ✓ 0.2s Python
```

b) Tách bộ dữ liệu ra để chọn lọc những feature cần thiết và sắp xếp chúng

```
[23] # tách bộ dữ liệu gốc ra để lọc ra những feature cần thiết
     data_collection = data[['short_name', 'player_positions', 'overall', 'potential', 'age', 'height_cm', 'weight_kg', 'club_name', 'league_name', 'league_rank', 'price']]
     price = data[['wage_eur', 'value_eur']]

     # Lấy các cột từ cột thứ 47 đến cột thứ 80 từ DataFrame data
     selected_columns = data.iloc[:, 46:80]

     selected_columns
[23]    ✓ 0.0s Python
```

	attacking_crossing	attacking_finishing	attacking_heading_accuracy	attacking_short_passing	attacking_volleys	skill_dribbling	skill_curve	skill_fk_accuracy	skill_l
0	85	95	70	91	88	96	93	94	
1	84	95	90	82	86	88	81	76	
2	13	11	15	43	13	12	13	14	
3	71	94	85	84	89	85	79	85	
4	85	87	62	87	87	95	88	89	
...	
4898	68	68	67	67	74	74	68	65	
4899	13	11	15	52	10	11	13	10	
4900	30	39	74	56	35	44	34	24	
4901	15	14	15	32	14	13	14	11	
4902	62	69	54	71	73	73	65	73	

4903 rows × 34 columns

- Chúng tôi tách bộ dữ liệu ra làm 2 phần, một phần là các thông tin cơ bản của cầu thủ, một phần là các thông tin về chỉ số chuyên môn, mục đích của việc này là để lấy ra được những feature cần thiết, sắp xếp chúng theo một thứ tự hợp lý
- Ghép dataframe sau khi đã tách:

```
[10] # ghép dataframe sau khi tách
     data_total = pd.concat([data_collection, selected_columns], axis=1)
[10]    ✓ 0.0s Python
```

```
[11] data_total = pd.concat([data_total, price], axis=1)

     data_total = data_total.drop(['defending_marking'], axis=1)
[11]    ✓ 0.0s Python
```

c) Kiểm tra dữ liệu lỗi và xử lý:

```
[12]    data_total.isna().sum()
    ✓ 0.0s
... short_name          0
player_positions      0
overall               0
potential             0
age                   0
height_cm             0
weight_kg             0
club_name             0
league_name           0
league_rank           0
team_position         0
nationality           0
preferred_foot        0
pace                  493
shooting              493
passing               493
dribbling              493
defending              493
physic                493
attacking_crossing    0
attacking_finishing   0
attacking_heading_accuracy 0
attacking_short_passing 0
attacking_volleys     0
skill_dribbling        0
...
goalkeeping_positioning 0
goalkeeping_reflexes 0
wage_eur               0
value_eur               0
dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- Nhận thấy một số cột của chỉ số tấn công bị NaN, lí do của việc này nằm ở những cầu thủ chơi ở vị trí thủ môn, những cầu thủ này bị mất dữ liệu liên quan tới tấn công
- Cùng xem chúng chiếm tỉ lệ bao nhiêu trong bộ dữ liệu:

```
[13]    for col in data_total.columns:
        missing_data = data_total[col].isna().sum() # tong so du lieu bi thieu
        missing_percent = missing_data / len(data_total) * 100
        if missing_percent != 0:
            print(f"Column: {col} has {missing_percent}% missing data")
[13]    ✓ 0.0s
... Column: pace has 10.05506832551499% missing data
Column: shooting has 10.05506832551499% missing data
Column: passing has 10.05506832551499% missing data
Column: dribbling has 10.05506832551499% missing data
Column: defending has 10.05506832551499% missing data
Column: physic has 10.05506832551499% missing data
```

```
[15]    # xóa các dòng có club_name bị nan
data_total = data_total.dropna(subset=['club_name'])
data_total = data_total.reset_index(drop=True)
data_total
[15]    ✓ 0.0s
```

- Xử lý dữ liệu bị NaN: Tách bộ dữ liệu ra chỉ lấy các chỉ số

```
[16]    # xử lý dữ liệu bị NaN
data_clean = data_total.iloc[:, 13:-2]
mean_values = data_clean.mean(axis=1)
column_means = (data_clean.mean()-10)
print(mean_values)
data_clean.fillna(column_means, inplace=True)
data_clean
[16]    ✓ 0.0s
```

	pace	shooting	passing	dribbling	defending	physic	attacking_crossing	attacking_finishing	attacking_heading_accuracy	attacking_short_passing
0	85.000000	92.000000	91.000000	95.000000	38.000000	65.000000	85	95	70	91
1	89.000000	93.000000	81.000000	89.000000	35.000000	77.000000	84	95	90	82
2	60.453968	50.917914	56.270748	60.577551	48.509751	60.145125	13	11	15	43
3	78.000000	91.000000	78.000000	85.000000	43.000000	82.000000	71	94	85	84
4	91.000000	85.000000	86.000000	94.000000	36.000000	59.000000	85	87	62	87
...
4898	77.000000	69.000000	66.000000	74.000000	47.000000	63.000000	68	68	67	67
4899	60.453968	50.917914	56.270748	60.577551	48.509751	60.145125	13	11	15	52
4900	40.000000	37.000000	46.000000	49.000000	70.000000	81.000000	30	39	74	56
4901	60.453968	50.917914	56.270748	60.577551	48.509751	60.145125	15	14	15	32
4902	51.000000	71.000000	69.000000	73.000000	37.000000	61.000000	62	69	54	71

4903 rows × 39 columns

+) Giải thích: Do bộ dữ liệu có lượng thủ môn khá ít, chỉ chiếm khoảng 10% nên dùng trung bình cộng giá trị của các cột thì ra thông số ở mức khá, vậy nên tôi trừ đi 10 để có thể đưa chỉ số của những thủ môn này về lại mức trung bình thấp, phù hợp với thực tế.

- Thực hiện ghép lại dữ liệu sau khi đã xử lí được vấn đề thiếu dữ

```
[18] ▶ data_info = data_total[['short_name', 'player_positions','overall','potential', 'age', 'height_cm', 'weight_kg', 'club_name', 'league_name', 'league_rank', 'mentality_composure', 'defending_star']]
      data_total = pd.concat([data_info, data_clean], axis=1)
      data_total = pd.concat([data_total, price], axis=1)
      data_total
[18] ✓ 0.0s
```

- Tiếp theo, do xét việc cầu thủ chơi nhiều vị trí sẽ đưa đến một mô hình dự đoán không ổn định, vậy nên chúng tôi sẽ lấy ra vị trí sở trường của cầu thủ đó, ở cột 'player_positions'(việc này sẽ thuận tiện cho việc mã hóa).

```
[19] ▶ data_total['player_positions'] = data_total['player_positions'].apply(lambda x: x.split(',')[-1].strip())
      data_total
[19] ✓ 0.0s
```

	short_name	player_positions	overall	potential	age	height_cm	weight_kg	club_name	league_name	league_rank	...	mentality_composure	defending_star
0	L. Messi	RW	93	93	33	170	72	FC Barcelona	Spain Primera Division	1.0	...	96	
1	Cristiano Ronaldo	ST	92	92	35	187	83	Juventus	Italian Serie A	1.0	...	95	
2	J. Oblak	GK	91	93	27	188	87	Atlético Madrid	Spain Primera Division	1.0	...	68	
3	R. Lewandowski	ST	91	91	31	184	80	FC Bayern München	German 1. Bundesliga	1.0	...	88	
4	Neymar Jr	LW	91	91	28	175	68	Paris Saint-Germain	French Ligue 1	1.0	...	93	
...	
4898	T. Elyounoussi	ST	70	70	32	172	66	Shonan Bellmare	Japanese J. League Division 1	1.0	...	72	
4899	M. Fraga	GK	70	70	32	184	83	Mazatlán FC	Mexican Liga MX	1.0	...	51	
4900	R. Shawcross	CB	70	70	32	191	76	Stoke City	English League Championship	2.0	...	64	
4901	J. Moulin	GK	70	70	34	185	88	AS Saint-Etienne	French Ligue 1	1.0	...	59	
4902	L. Tomlin	CAM	70	70	31	180	74	Cardiff City	English League	2.0	...	68	

- Như vậy là khâu làm sạch dữ liệu đã hoàn thành, tôi lưu dữ liệu vừa làm sạch vào file data_clean.csv

d) Xử lí dữ liệu

- Mã hóa dữ liệu của hai cột player_positions và preferred_foot

```
[23]
# mã hóa một số feature
player_positions_encoder = LabelEncoder()
preferred_foot_endcoder = LabelEncoder()
df['player_positions'] = player_positions_encoder.fit_transform(df['player_positions'])
df['preferred_foot'] = preferred_foot_endcoder.fit_transform(df['preferred_foot'])

Python
```

- Lọc ra để lấy được các thông số dùng cho việc train model

```
[24]
# loại bỏ một số cột thông tin dạng object
object_columns = df.select_dtypes(include=['object'])
df = df.select_dtypes(exclude=['object'])

Python
```

- Chuẩn hóa dữ liệu của 2 cột wage_eur và value_eur vì giá trị của chúng đang cao hơn rất nhiều so với các chỉ số dùng để train

```
[25]
# chuẩn hóa dữ liệu ở 2 cột lương và giá trị của cầu thủ
df['wage_eur'] = df['wage_eur'] / 1000
df['value_eur'] = df['value_eur'] / 1000000

Python
```

- Dữ liệu sau khi xử lí:

df														Python
...	player_positions	overall	potential	age	height_cm	weight_kg	preferred_foot	pace	shooting	passing	...	power_long_shots	defending_standing_tac	
0	12	93	93	33	170	72	0	85.000000	92.000000	91.000000	...	94		
1	14	92	92	35	187	83	1	89.000000	93.000000	81.000000	...	93		
2	5	91	93	27	188	87	1	60.453968	50.917914	56.270748	...	12		
3	14	91	91	31	184	80	1	78.000000	91.000000	78.000000	...	85		
4	8	91	91	28	175	68	1	91.000000	85.000000	86.000000	...	84		
...	
4898	14	70	70	32	172	66	1	77.000000	69.000000	66.000000	...	66		
4899	5	70	70	32	184	83	1	60.453968	50.917914	56.270748	...	10		
4900	1	70	70	32	191	76	1	40.000000	37.000000	46.000000	...	28		
4901	5	70	70	34	185	88	1	60.453968	50.917914	56.270748	...	14		
4902	0	70	70	31	180	74	1	51.000000	71.000000	69.000000	...	72		

4903 rows × 37 columns

3. Visualize dữ liệu

- Vẽ biểu đồ thể hiện sự tương quan giữa các thuộc tính tấn công và phòng thủ

```

# Các thuộc tính tấn công
attacking_attributes = [
    'attacking_crossing', 'attacking_finishing', 'attacking_heading_accuracy',
    'attacking_short_passing', 'attacking_volleys'
]

# Các thuộc tính phòng thủ
defensive_attributes = [
    'defending_marking', 'defending_standing_tackle', 'defending_sliding_tackle',
    'mentality_interceptions'
]

# Kết hợp các thuộc tính tấn công và phòng thủ
selected_attributes = attacking_attributes + defensive_attributes

# Lọc DataFrame với các thuộc tính đã chọn
df_selected = df[selected_attributes]

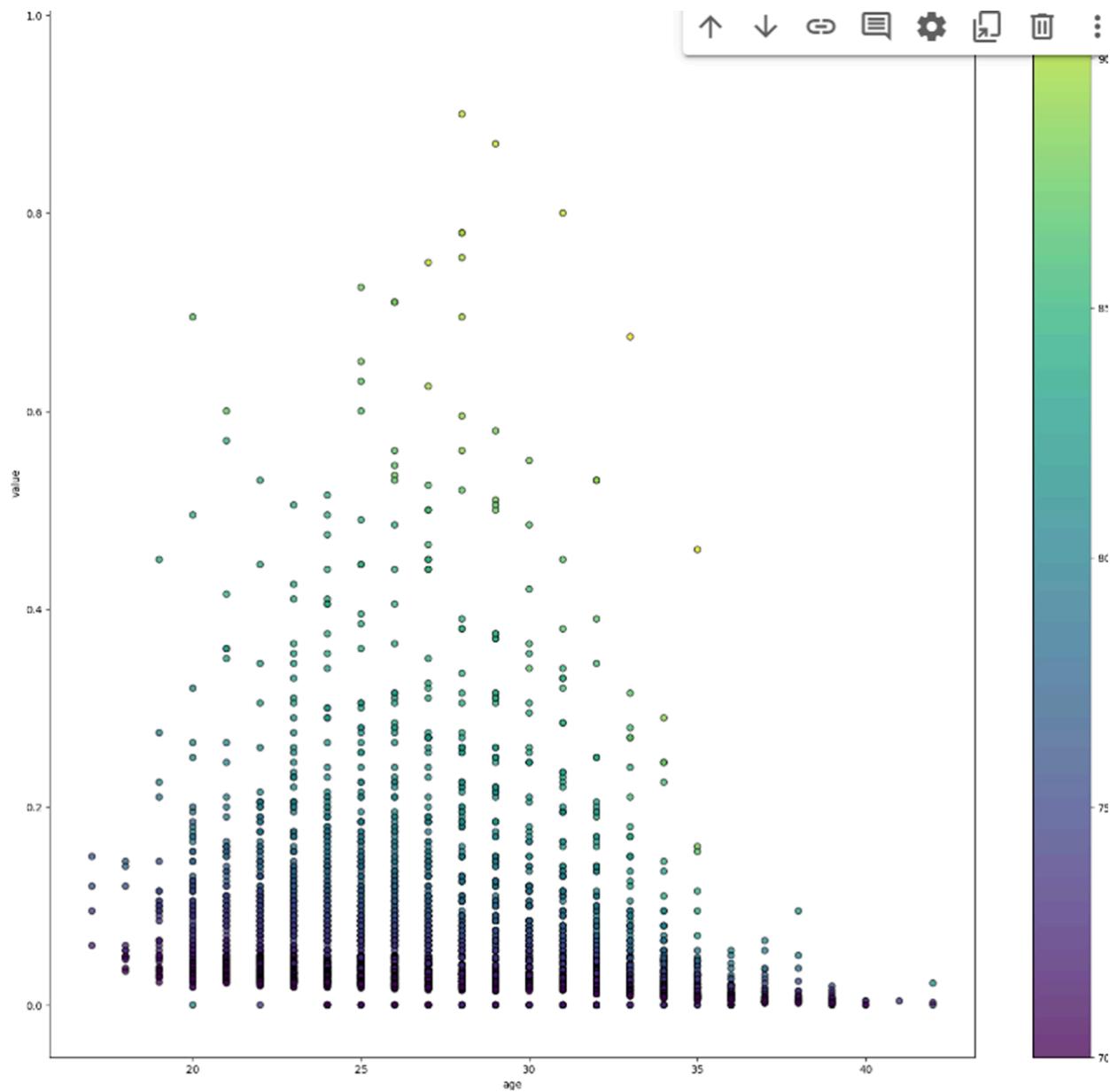
# Tính toán ma trận tương quan giữa các thuộc tính đã chọn
correlation_matrix = df_selected.corr()

# Vẽ biểu đồ heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation between Attacking and Defensive Attributes')
plt.show()

```



- Từ biểu đồ ta có thể thấy rằng các thuộc tính tấn công và phòng thủ có các giá trị tương khắc nhau
- Bộ dữ liệu có độ phức tạp cao
- Biểu đồ thể hiện sự tương quan giữa độ tuổi và giá trị overall

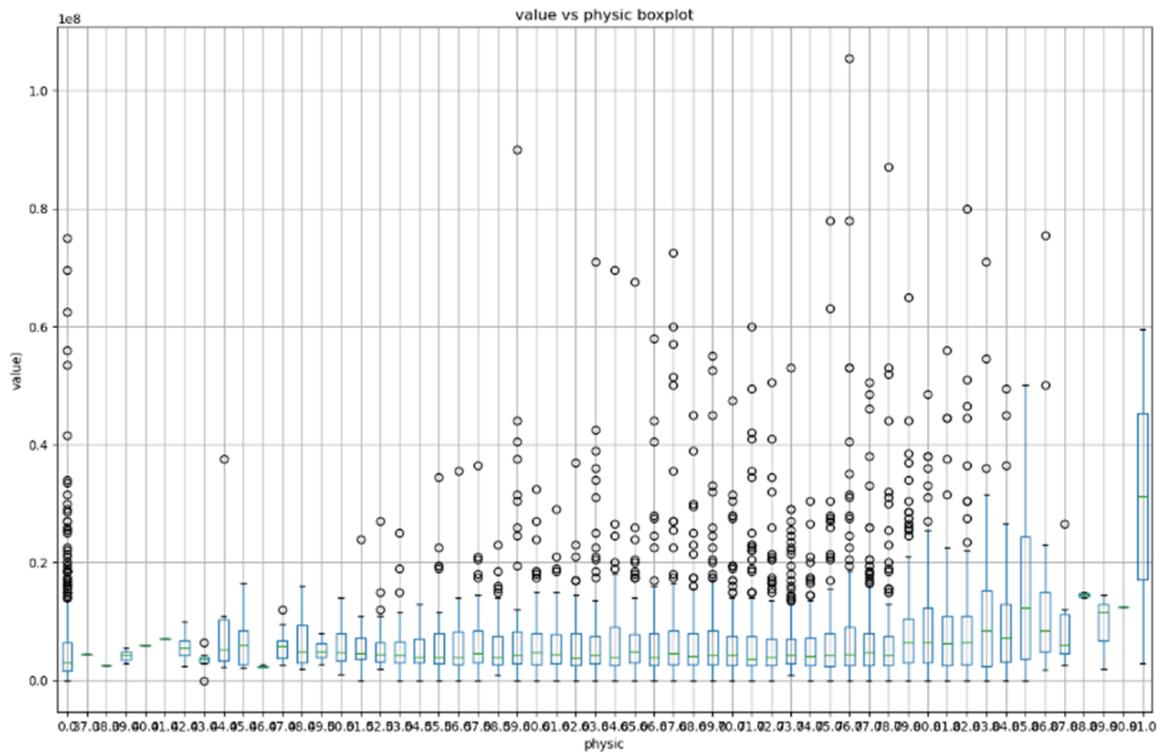


- Độ tuổi khoảng từ 25 đến 30 có mức overall cao hơn
- Biểu đồ thể hiện sự tương quan giữa thể chất và giá trị

```
▶ a1=df.boxplot(column='value_eur',by='physic',figsize=(15,10))
a1.set_title('value vs physic boxplot')
a1.set_ylabel('value')
```

➡ Text(0, 0.5, 'value)')

Boxplot grouped by physic



- Các cầu thủ có giá trị thể chất khỏe thường có giá trị hơn lứa cầu thủ có thể chất yếu hơn

III. MODELS.

1. Linear Regression.

a. Lựa chọn feature.

```
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
data = pd.read_csv('../data/processed/dataset.csv')
feature = ['overall', 'potential', 'age', 'height_cm', 'weight_kg', 'league_rating', 'passing', 'dribbling', 'defending', 'physic', 'gk_diving', 'gk_reflexes', 'gk_speed', 'gk_positioning', 'attacking', 'attacking_heading_accuracy', 'attacking_short_passing']
```

```

from sklearn.model_selection import c
from sklearn.metrics import mean_abs

df = pd.read_csv('..../data/processed/d
# Các kiểu dữ liệu trong data.
df_type = pd.DataFrame(data = df.dtyp
unique_types = df_type[0].unique()
print(unique_types)
>>> [dtype('O') dtype('int64') dtype('
# Lọc dữ liệu dùng trong model này.
list(df.select_dtypes(include=['Int64','f
>>> ['overall','potential','age','...'wage_

```

```

'skill_curve','skill_fk_accuracy','skill_long_passing'
'movement_acceleration','movement_sprint_speed'
'movement_balance','power_shot_power','power_j
'power_strength','power_long_shots','mentality_ag
'mentality_positioning','mentality_vision','mentality.
'defending_standing_tackle','defending_sliding_tac
'goalkeeping_kicking','goalkeeping_positioning','g
target = ['value_eur']

X = data[feature]
y = data[target]

```

Qua quá trình làm sạch thì tôi quyết định chỉ lựa chọn các feature có kiểu dữ liệu hỗ trợ làm việc trong tập data này là 'Int64','float64'.

b. phân chia train-val-test.

Trong đoạn mã trên, dữ liệu được chia thành 3 phần:

1. **Tập huấn luyện (Train set):** `x_train` và `y_train` tạo thành tập huấn luyện, chiếm 70% dữ liệu gốc.
2. **Tập validation (Validation set):** `x_val` và `y_val` tạo thành tập validation, chiếm 15% dữ liệu gốc.
3. **Tập kiểm tra (Test set):** `x_test` và `y_test` tạo thành tập kiểm tra, cũng chiếm 15% dữ liệu gốc.

```

# Chia dữ liệu thành tập huấn luyện và tập tạm thờ
X_train, X_temp, y_train, y_temp = train_test_split(X
# Chia tập tạm thời thành tập validation và tập kiểm
X_val, X_test, y_val, y_test = train_test_split(X_temp

```

c1. TRAIN VỚI MÔ HÌNH CƠ BẢN NHẤT BASELINE.

```

# trainning model.
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)

# Predictions
y_train_pred = lr_model.predict(X_train)
y_val_pred = lr_model.predict(X_val)
y_test_pred = lr_model.predict(X_test)

```

```

# Evaluate model
print("Model evaluation for linear regression")

mse = mean_squared_error(y_test, y_test_pred)
print(f"Mean Squared Error: {mse}")

mae = mean_absolute_error(y_test, y_test_pred)
print(f"Mean Absolute Error: {mae}")

test_score = model.score(X_test, y_test)
print(f"Test Score: {test_score}")

train_score = model.score(X_train, y_train)
print(f"Train Score: {train_score}")

cv_score = cross_val_score(model, X_val, y_val)
print(f"Cross Validation Score: {cv_score}")
print(f"Mean of Cross Validation Score: {cv_score.mean()}")

```

OUTPUT:

Model evaluation for linear regression
score Train R2: 0.8404351949151274, score Validation R2: 0.8332139504332134, score Test R2:
0.8254639491291416
Cross Validation Score: [0.81264681 0.85273053 0.82558359 0.77188151 0.78528818], Mean of
Cross Validation Score: 0.8096261238194682
Train MSE: 10.997952339779921, Validation MSE: 11.224272988313983, Test MSE:
11.789818242485897

NHẬN XÉT:

Các giá trị R2 trên cho thấy mô hình có khả năng giải thích khoảng 82.55% - 84.04% biến thiên của dữ liệu. Điều này cho thấy mô hình phù hợp khá tốt với dữ liệu huấn luyện và có khả năng tổng quát tốt đối với dữ liệu kiểm tra và xác thực. Sự chênh lệch nhỏ giữa các giá trị R2 cho thấy mô hình không bị overfitting hoặc underfitting quá mức.

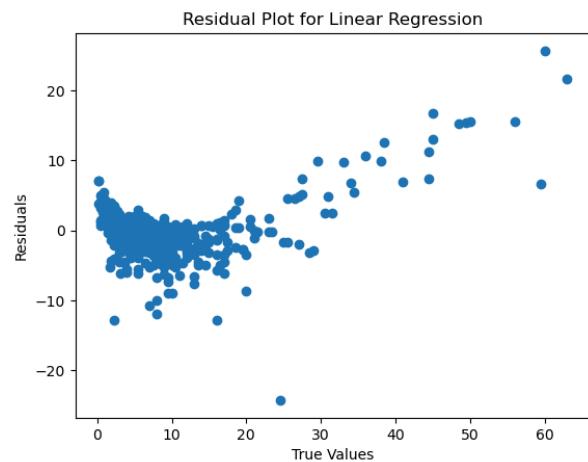
Điểm trung bình của các lần cross validation là 0.8096, cũng tương đối gần với các giá trị R2 trên. Điều này chỉ ra rằng mô hình có độ ổn định và khả năng dự đoán tốt khi áp dụng trên các tập dữ liệu khác nhau trong quá trình cross validation.

Các giá trị MSE này cho thấy mức độ sai số của mô hình trên các tập dữ liệu khác nhau. Giá trị MSE không chênh lệch nhiều giữa các tập dữ liệu huấn luyện, xác thực và kiểm tra, điều này chỉ

ra rằng mô hình có hiệu suất nhất quán và không bị overfitting

```
# Residual plot  
plt.scatter(y_test, y_test - y_test_pred)  
plt.xlabel('True Values')  
plt.ylabel('Residuals')  
plt.title('Residual Plot for Linear Regression')  
plt.show()
```

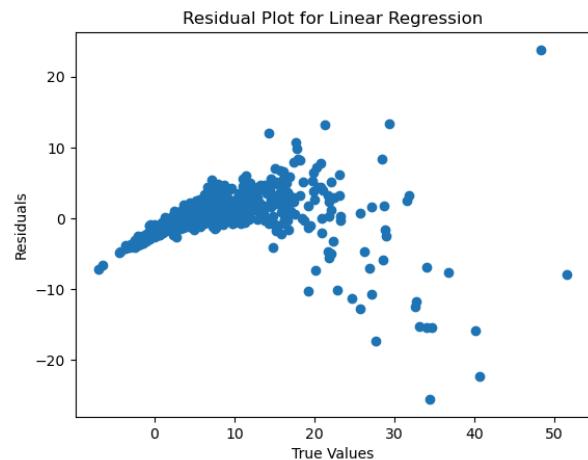
NHẬN XÉT: Trong biểu đồ này, các điểm dữ liệu phân bố chủ yếu trong khoảng từ -1 đến 1 trên trục tung, với một hình dạng cong xuất hiện dưới đường thẳng $y = 0$ khi giá trị thực tế tăng. Điều này cho thấy rằng với các giá trị thực tế lớn hơn, dự đoán của mô hình hồi quy tuyến tính thường thấp hơn so với giá trị thực tế.



c.2 CHỌN ĐẶC TRƯNG VỚI RFE. [sklearn.feature_selection.RFE — scikit-learn 1.4.2 documentation](#)

số lượng đặc trưng feature tôi sử dụng là 52.

```
print(f"số cột ban đầu: {len(features)}")  
model = LinearRegression()  
rfe = RFE(estimator=model, n_features_to_se  
X_train_rfe = rfe.fit_transform(X_train, y_train)  
X_val_rfe = rfe.transform(X_val)  
X_test_rfe = rfe.transform(X_test)  
model.fit(X_train_rfe, y_train)  
  
# dự đoán.  
y_train_pred = model.predict(X_train_rfe)  
y_val_pred = model.predict(X_val_rfe)  
y_test_pred = model.predict(X_test_rfe)
```



số cột ban đầu: 52

score Train R2: 0.8351286901296624, score Validation R2: 0.8293664916129284, score Test R2:

0.8251527383831843

Cross Validation Score: [0.81264681 0.85273053 0.82558359 0.77188151 0.78528818], Mean of Cross Validation Score: 0.8096261238194666

Train MSE: 1.1363701457765658e-23, Validation MSE: 1.1483197090313793e-23, Test MSE: 1.1810840364343464e-23

NHẬN XÉT: tôi đã thực hiện lấy số lượng features = 10 ít hơn với ban đầu là 50 . Tôi nhận thấy đã có thay đổi trong đồ thị và output. Xuất hiện lỗi nhiều hơn so với baseline.

c.3 HỒI QUI RIDE VÀ LASSO.

với baseline như trên có vẻ mô hình của chúng ta không overfitting.

ride và lasso là 2 kĩ thuật ngăn chặn overfitting.

tôi sẽ thử với dữ liệu này với alpha=0.1.

```
# Train model
lasso_model = Lasso(alpha=0.1) # alpha is the regularization parameter
lasso_model.fit(X_train, y_train)

# Make predictions
y_train_pred = lasso_model.predict(X_train)
y_val_pred = lasso_model.predict(X_val)
y_test_pred = lasso_model.predict(X_test)
```

OUTPUT LASSO:

Model evaluation for Lasso regression

score Train R2: 0.8381600440154385, score Validation R2: 0.8322608540432398, score Test R2: 0.824566027670381

Cross Validation Score: [0.8133706 0.85992012 0.84127237 0.77577924 0.79247757], Mean of Cross Validation Score: 0.8165639828726841

Train MSE: 11.154766376229103, Validation MSE: 11.288413928716539, Test MSE: 11.850472363751926

```
# Train model
ridge_model = Ridge(alpha=0.1) # alpha is the regularization parameter
```

```

ridge_model.fit(X_train, y_train)

# Make predictions
y_train_pred = ridge_model.predict(X_train)
y_val_pred = ridge_model.predict(X_val)
y_test_pred = ridge_model.predict(X_test)

```

OUTPUT RIDE:

Model evaluation for Ridge regression
score Train R2: 0.840435194760771, score Validation R2: 0.8332142545297153, score Test R2:
0.8254639181844393
Cross Validation Score: [0.81264702 0.8527372 0.82560795 0.77187848 0.78530275], Mean of
Cross Validation Score: 0.809634680545873
Train MSE: 10.99795235041889, Validation MSE: 11.224252523400036, Test MSE:
11.789820332784226

NHẬN XÉT:

cùng một alpha có vẻ như hai mô hình không có sự sai khác nhiều về độ lớn score và lỗi.

c.3 THỬ NGHIỆM VỚI PolynomialFeatures.

Linear Regression - Mean Squared Error:

5.154976711058016e-23

Linear Regression - R2 Score:

0.23685907706614218

Ridge Regression - Mean Squared Error:

5.161858388607246e-23

Ridge Regression - R2 Score:

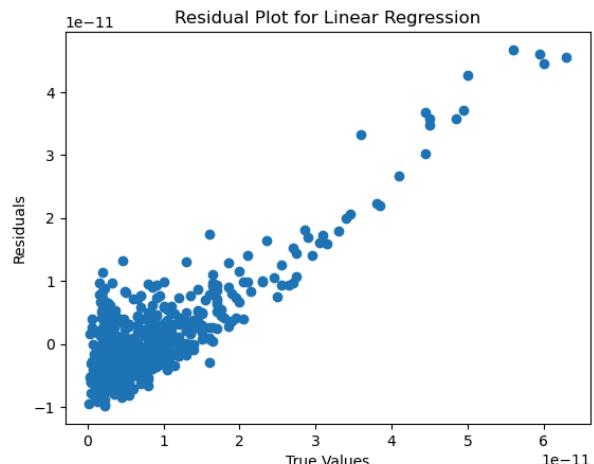
0.23584031596388788

Lasso Regression - Mean Squared Error:

6.76968374424643e-23

Lasso Regression - R2 Score:

-0.00218157910828487



- **Độ lỗi bình phương trung bình (MSE)** của cả ba mô hình đều rất thấp, gần như bằng 0. Điều này cho thấy mô hình có độ chính xác cao trong việc dự đoán.
- Tuy nhiên, **hệ số xác định (R2 Score)** của cả ba mô hình lại không cao. Điều này cho

thấy mô hình có thể không phản ánh đúng mối quan hệ giữa các biến độc lập và biến phụ thuộc.

- Cụ thể, mô hình Linear Regression và Ridge Regression có R2 Score tương đối thấp (khoảng 0.24), trong khi mô hình Lasso Regression lại có R2 Score âm (-0.002). Điều này cho thấy mô hình Lasso Regression không phù hợp với dữ liệu.
- Trong trường hợp này, việc sử dụng PolynomialFeatures có thể đã tạo ra quá nhiều đặc trưng, dẫn đến việc mô hình không thể học được mối quan hệ phức tạp giữa các đặc trưng và biến mục tiêu.

2. Random Forest.

a. Lựa chọn feature.

b. phân chia train-val-test.

lựa chọn feature và phân chia dữ liệu trong model này không có sự thay đổi với mô hình mở mục 1.

c1. TRAINING VỚI MÔ HÌNH FOREST CƠ BẢN.

```
# Train model  
model = RandomForestRegressor(n_estimators=100)  
model.fit(X_train, y_train)  
  
# Make predictions  
y_train_pred = model.predict(X_train)  
y_val_pred = model.predict(X_val)  
y_test_pred = model.predict(X_test)
```

```
Model evaluation for Random Forest  
Regression  
Mean Squared Error: 2.7662104892357338  
Train MAE: 0.15397727272727274  
Validation MAE: 0.4365561904761904  
Test MAE: 0.47384422554347827  
Test Score: 0.959049118083185  
Train Score: 0.9951055049190252  
Cross Validation Score: [0.90010106  
0.87891774 0.94577891 0.92441461  
0.95993748]
```

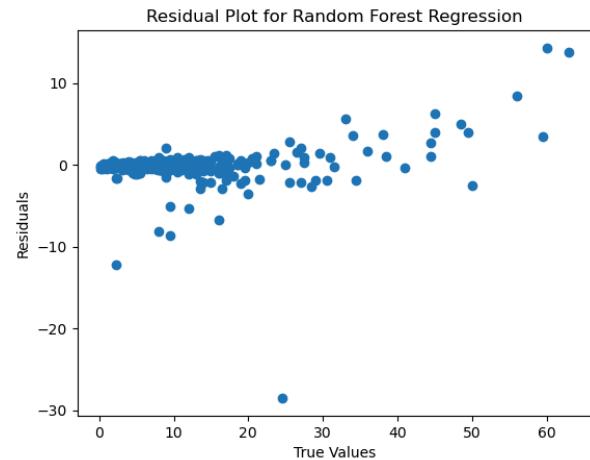
Mean of Cross Validation Score:
0.9218299621952676

NHẬN XÉT

- **Độ lỗi bình phương trung bình (MSE)** của mô hình là 2.766, chỉ ra rằng mô hình có

một số sai số khi dự đoán.

- **Độ lỗi tuyệt đối trung bình (MAE)** trên tập huấn luyện rất thấp (0.154), cho thấy mô hình hoạt động tốt trên tập dữ liệu này.
- Tuy nhiên, MAE tăng lên 0.437 và 0.474 trên tập kiểm định và tập kiểm tra, tương ứng. Điều này cho thấy mô hình có thể đang gặp phải vấn đề quá khớp (overfitting).
- **Điểm số trên tập kiểm tra (Test Score)** là 0.959, cho thấy mô hình có độ chính xác cao trên tập kiểm tra.
- **Điểm số trên tập huấn luyện (Train Score)** là 0.995, cao hơn nhiều so với điểm số trên tập kiểm tra, có thể là dấu hiệu của quá khớp.
- **Điểm số kiểm định chéo (Cross Validation Score)** trung bình là 0.922, cho thấy mô hình có độ ổn định tốt khi áp dụng trên các tập dữ liệu khác nhau.



c.2 TUNNING VỚI GRIDSEARCH.

Best parameters found: {'bootstrap': False, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

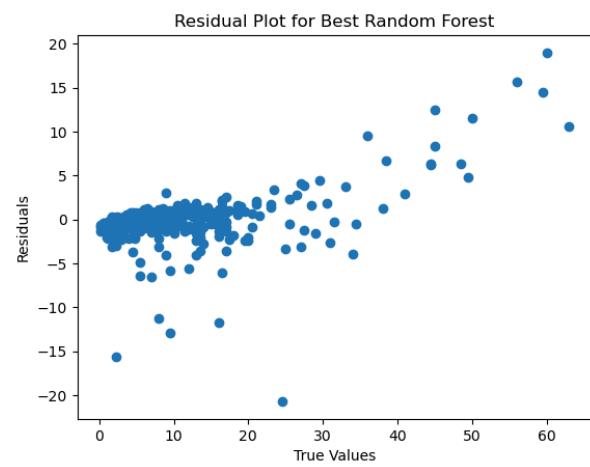
Best Random Forest - Mean Squared Error:

4.716605907726804

Best Random Forest - R2 Score:

0.9301755335224557

- **Độ lỗi bình phương trung bình (MSE)** của mô hình sau khi tinh chỉnh là 4.717, cao hơn so với trước khi tinh chỉnh. Tuy nhiên, giá trị này vẫn khá thấp, cho thấy mô hình có độ chính xác tốt.
- **Hệ số xác định (R2 Score)** của mô hình sau khi tinh chỉnh là 0.930, cao hơn nhiều



so với trước khi tinh chỉnh. Điều này cho thấy mô hình sau khi tinh chỉnh đã cải thiện đáng kể và phản ánh tốt hơn mối quan hệ giữa các biến độc lập và biến phụ thuộc.

- Các tham số tốt nhất tìm thấy bằng GridSearchCV là: `bootstrap=False`, `max_depth=20`, `max_features='sqrt'`, `min_samples_leaf=1`, `min_samples_split=2`, và `n_estimators=300`. Điều này cho thấy việc tinh chỉnh tham số có thể giúp cải thiện đáng kể hiệu suất của mô hình.

c.2 TUNNING VỚI RANDOM SEARCH.

Best parameters found: {'bootstrap': False, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

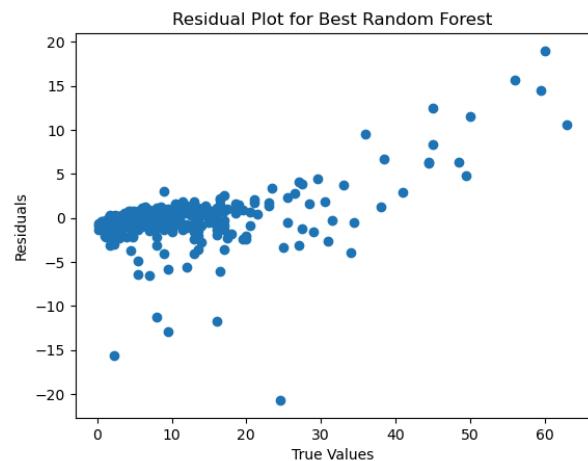
Best Random Forest - Mean Squared Error:

4.716605907726804

Best Random Forest - R2 Score:

0.9301755335224557

- **Độ lỗi bình phương trung bình (MSE)** của mô hình sau khi tinh chỉnh là 4.717, cho thấy mô hình có độ chính xác tốt.
- **Hệ số xác định (R2 Score)** của mô hình sau khi tinh chỉnh là 0.930, cho thấy mô hình phản ánh tốt mối quan hệ giữa các biến độc lập và biến phụ thuộc.
- Các tham số tốt nhất tìm thấy bằng Random Search cũng giống như kết quả từ GridSearchCV: `bootstrap=False`, `max_depth=20`, `max_features='sqrt'`, `min_samples_leaf=1`, `min_samples_split=2`, và `n_estimators=300`.



3. XGBoost.

- Import thư viện:

Import libraries

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error, make_scorer, r2_score
from sklearn.model_selection import GridSearchCV

```

[19] Python

a) Lựa chọn những feature phù hợp:

```

# loại bỏ một số feature thừa trong bộ dữ liệu
df = df.drop(['unnamed: 0', 'team_position', 'movement_acceleration',
               'movement_sprint_speed',
               'movement_agility',
               'movement_reactions',
               'movement_balance', 'mentality_aggression',
               'mentality_interceptions',
               'mentality_positioning',
               'mentality_vision',
               'mentality_penalties',
               'mentality_composure', 'league_rank'], axis=1)

```

[21] Python

- Sau khi phân tích kỹ từng thông số một, thì tôi quyết định loại bỏ đi một số feature mà tôi cho rằng không cần thiết vì chỉ số của chúng không quá thực tế và cũng rất lệch.

b) Phân tách dữ liệu thành bộ train và bộ test

```

x = df.iloc[:, :-1]
y = df.iloc[:, -1]
print(x)
print(y)

```

[27] Python

Train test split

```

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

print(x_train.shape)
print(y_train.shape)

```

[28] ... (3922, 36) (3922,) Python

- Tôi quyết định chia bộ train và bộ test theo tỉ lệ 80/20

c) Sử dụng model

```

Selection Model: XGBoost

# Huấn luyện mô hình
model = XGBRegressor(eta=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    min_child_weight=3,
    gamma=0.1,
    n_estimators=300,
    reg_lambda=5,
    reg_alpha=1,
    objective='reg:squarederror',
    random_state=42)
model.fit(x_train, y_train)

[29]

```

XGBRegressor(base_score=None, booster=None, callbacks=None,
 colsample_bylevel=None, colsample_bynode=None,
 colsample_bytree=0.8, device=None, early_stopping_rounds=None,
 enable_categorical=False, eta=0.05, eval_metric=None,
 feature_types=None, gamma=0.1, grow_policy=None,
 importance_type=None, interaction_constraints=None,
 learning_rate=None, max_bin=None, max_cat_threshold=None,
 max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
 max_leaves=None, min_child_weight=3, missing='nan',
 monotone_constraints=None, multi_strategy=None, n_estimators=300,
 n_jobs=None, num_parallel_tree=None, ...)

Trong dự án này, chúng tôi sử dụng mô hình XGBoost để dự đoán giá trị chuyển nhượng của cầu thủ bóng đá. XGBoost thường được sử dụng trong các cuộc thi về phân tích dữ liệu vì hiệu suất cao và khả năng điều chỉnh linh hoạt.

Giải thích:

- +) eta (learning_rate=0.05): Tốc độ học tập, xác định mức độ mà mô hình điều chỉnh theo mỗi bước học. Giá trị nhỏ (0.05) giúp mô hình học chậm hơn nhưng có khả năng đạt được kết quả chính xác hơn.
- +) max_depth=5: Độ sâu tối đa của mỗi cây quyết định. Giá trị này ngăn ngừa mô hình trở nên quá phức tạp và giúp giảm thiểu hiện tượng overfitting.
- +) subsample=0.8: Tỷ lệ mẫu con được sử dụng để huấn luyện mỗi cây. Giá trị 0.8 nghĩa là 80% dữ liệu huấn luyện được sử dụng ngẫu nhiên cho mỗi cây, giúp mô hình tránh overfitting.
- +) n_estimators=300: Số lượng cây được xây dựng trong quá trình huấn luyện. Giá trị 300 đảm bảo mô hình đủ mạnh mẽ để học từ dữ liệu mà không trở nên quá phức tạp.

Lý do sử dụng XGBoost:

- Lý do chính mà tôi chọn XGBoost để train đơn giản vì hiệu suất cao, XGBoost thường xuyên đứng đầu trong các cuộc thi về dự đoán và phân tích dữ liệu nhờ vào hiệu suất cao và khả năng tối ưu hóa tốt, XGBoost cung cấp nhiều tham số có thể điều chỉnh để cải thiện hiệu suất và tránh overfitting, như các tham số đã được giải thích ở trên.

d) Kết quả dự đoán và đánh giá trên tập kiểm tra

Sau khi huấn luyện mô hình XGBoost với các tham số được điều chỉnh như trên, tôi tiến hành dự đoán trên tập kiểm tra để đánh giá hiệu suất của mô hình.

```

# Dự đoán trên tập kiểm tra
y_pred = model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

```

[30]

```

... Mean Squared Error: 1.0793914871409425
R-squared: 0.9860381676262233

```

Python

- Ở đây, tôi sử dụng Mean Squared Error (MSE) và R-squared (R^2) để đánh giá mô hình của chúng tôi. Với MSE là 1.079, mô hình của tôi cho thấy mức độ sai số tương đối thấp, điều này chứng tỏ mô hình dự đoán khá chính xác, còn giá trị R^2 cao gần 1 (cụ thể là 0.986) cho thấy mô hình của chúng tôi có khả năng giải thích 98.6% biến động của giá trị thực tế trong dữ liệu kiểm tra. Điều này chứng tỏ mô hình có khả năng dự đoán rất tốt và phù hợp với dữ liệu.

e) Đánh giá mô hình bằng Cross-Validation

```

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Thiết lập scorers cho MSE và R²
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)
r2_scorer = make_scorer(r2_score)

# Đánh giá bằng Cross-Validation cho MSE
cv_mse_results = cross_val_score(model, x, y, cv=kfold, scoring=mse_scorer)
cv_mse = -cv_mse_results # Chuyển đổi giá trị âm trở lại giá trị dương
cv_rmse = np.sqrt(cv_mse)

# Đánh giá bằng Cross-Validation cho R²
cv_r2_results = cross_val_score(model, x, y, cv=kfold, scoring='r2')

# In ra các kết quả
print(f'MSE from CV: {cv_mse}')
print(f'RMSE from CV: {cv_rmse}')
print(f'Mean RMSE from CV: {np.mean(cv_rmse)}')
print(f'Standard Deviation of RMSE from CV: {np.std(cv_rmse)}')
print(f'R-squared from CV: {cv_r2_results}')
print(f'Mean R-squared from CV: {np.mean(cv_r2_results)}')
print(f'Standard Deviation of R-squared from CV: {np.std(cv_r2_results)}')

```

[31]

```

... MSE from CV: [1.02262716 1.5246541 0.45531815 1.35426048 1.35647299]
RMSE from CV: [1.01125029 1.23476884 0.67477266 1.16372698 1.1646772 ]
Mean RMSE from CV: 1.0498391969192906
Standard Deviation of RMSE from CV: 0.2012566400756227
R-squared from CV: [0.98677241 0.97480768 0.99169479 0.98219072 0.98152909]
Mean R-squared from CV: 0.9833989388964051
Standard Deviation of R-squared from CV: 0.0056385682969637314

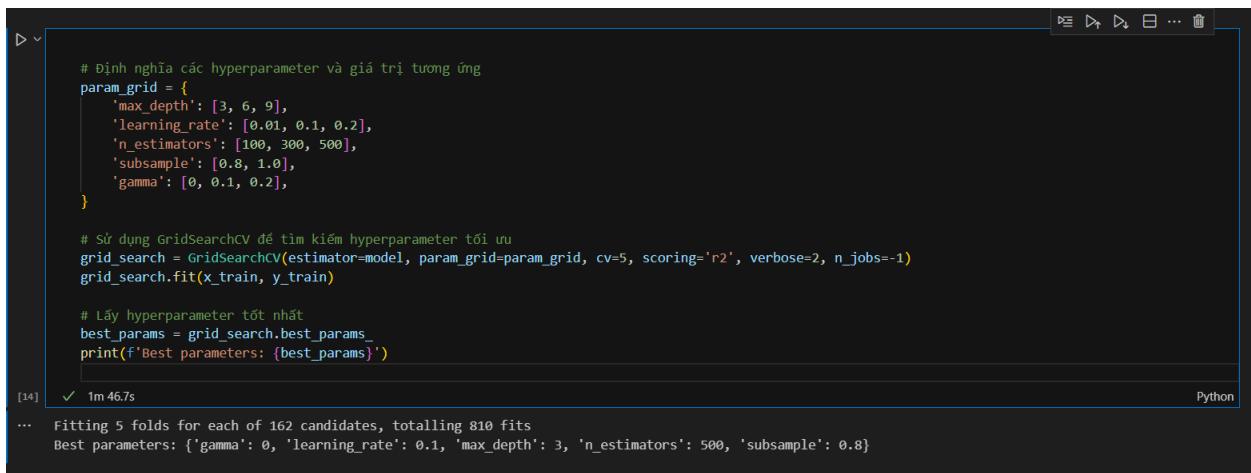
```

Python

- +)** **Mean Squared Error (MSE) từ CV:** Các giá trị MSE từ các lần chia CV cho thấy sai số bình phương trung bình của mô hình trên các tập con khác nhau của dữ liệu. Giá trị này dao động từ 0.455 đến 1.524, cho thấy mô hình có độ sai lệch nhất định trên các phần khác nhau của dữ liệu.
- +)** **Root Mean Squared Error (RMSE) từ CV:** RMSE là căn bậc hai của MSE và được sử dụng để dễ dàng diễn giải sai số. Các giá trị RMSE từ CV dao động từ 0.674 đến 1.234, với trung bình là 1.049. Độ lệch chuẩn của RMSE là 0.201, cho thấy mức độ biến động của sai số giữa các lần chia không quá lớn.
- +)** **R-squared từ CV:** Các giá trị R-squared từ CV cho thấy mô hình có khả năng giải thích biến động của dữ liệu rất cao, dao động từ 0.974 đến 0.991, với trung bình là 0.983. Độ lệch chuẩn của R-squared là 0.0056, cho thấy sự ổn định cao của mô hình trên các tập con khác nhau.

f) Hyperparameter Tuning

Để tối ưu hóa hiệu suất của mô hình XGBoost, chúng tôi sử dụng kỹ thuật Grid Search Cross-Validation (GridSearchCV) để tìm kiếm các giá trị tối ưu cho các siêu tham số.



```
# Định nghĩa các hyperparameter và giá trị tương ứng
param_grid = {
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 300, 500],
    'subsample': [0.8, 1.0],
    'gamma': [0, 0.1, 0.2],
}

# Sử dụng GridSearchCV để tìm kiếm hyperparameter tối ưu
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='r2', verbose=2, n_jobs=-1)
grid_search.fit(x_train, y_train)

# Lấy hyperparameter tốt nhất
best_params = grid_search.best_params_
print(f'Best parameters: {best_params}')


[14] 1m 46.7s
```

... Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best parameters: {'gamma': 0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500, 'subsample': 0.8}

Kết quả tìm kiếm siêu tham số:

- **Tham số tốt nhất:**

- **max_depth:** 3
- **learning_rate:** 0.2
- **n_estimators:** 500
- **subsample:** 0.8
- **gamma:** 0

+) Các giá trị này được lựa chọn dựa trên khả năng tối ưu hóa hiệu suất mô hình và tránh overfitting. Bằng cách tìm kiếm này, chúng tôi nghĩ rằng mô hình sẽ hoạt động hiệu quả trên dữ liệu thực tế và đưa ra các dự đoán chính xác giá trị chuyển nhượng của cầu thủ bóng đá.

⇒ Huấn luyện mô hình với Hyperparameter tối ưu: Sau khi tìm kiếm siêu tham số tối ưu cho mô hình XGBoost, chúng tôi huấn luyện lại mô hình với các giá trị tối ưu này và đánh giá hiệu suất của nó trên tập kiểm tra. Dưới đây là kết quả của quá trình đánh giá.

```

# Huấn luyện mô hình với hyperparameter tối ưu
xgb_optimized = XGBRegressor(**best_params)
xgb_optimized.fit(x_train, y_train)

# Đánh giá lại mô hình
y_pred_optimized = xgb_optimized.predict(x_test)
mse_optimized = mean_squared_error(y_test, y_pred_optimized)
r2_optimized = r2_score(y_test, y_pred_optimized)

print(f'Optimized Mean Squared Error: {mse_optimized}')
print(f'Optimized R-squared: {r2_optimized}')

```

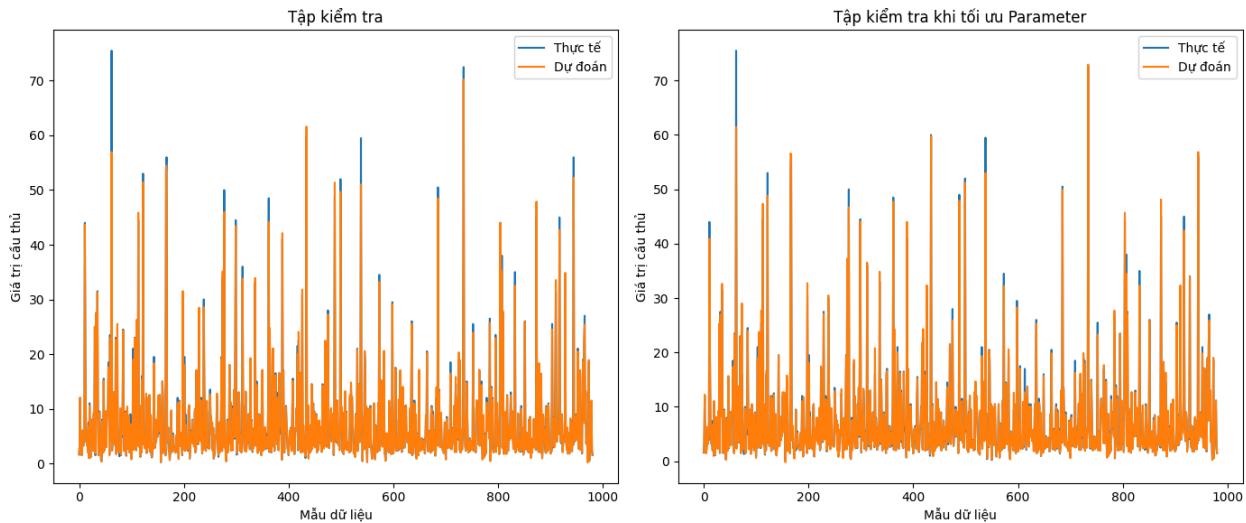
[15] ✓ 0.5s

... Optimized Mean Squared Error: 0.7926154535271264
Optimized R-squared: 0.9897475899793106

+) MSE của mô hình với siêu tham số tối ưu là 0.7926, thấp hơn so với MSE của mô hình trước tối ưu (1.0808). Điều này cho thấy mô hình với siêu tham số tối ưu có độ chính xác cao hơn trong việc dự đoán giá trị chuyển nhượng của cầu thủ bóng đá trên tập kiểm tra.

+) R-squared (R^2): R-squared của mô hình với siêu tham số tối ưu là 0.9897, cao hơn so với R-squared của mô hình trước tối ưu (0.9860). Điều này cho thấy mô hình với siêu tham số tối ưu có khả năng giải thích tốt hơn về biến động của dữ liệu trên tập kiểm tra.

g) Trực quan hóa về so sánh kết quả dự đoán với kết quả thực tế



4.Gradient Boosting

4.1.Giới thiệu qua về mô hình Gradient Boosting:

- Gradient Boosting là một kỹ thuật học máy được sử dụng rộng rãi cho các bài toán hồi quy và phân loại. Nó là một phương pháp ensemble, nghĩa là kết hợp nhiều mô hình đơn giản (thường là các cây quyết định nhỏ) để tạo ra một mô hình mạnh mẽ hơn. Dưới đây là một giới thiệu tổng quan về phương pháp Gradient Boosting:

4.2. Tại sao lại sử dụng Gradient Boost:

- Gradient Boosting có ưu điểm là
 - + Hiệu quả cao : Gradient Boosting thường tạo ra mô hình có độ chính xác cao , đặc biệt là với bộ dữ liệu phức tạp như này
 - + Khả năng tổng quát hóa tốt : Nó có thể tránh overfitting nếu được điều chỉnh đúng cách, nhờ vào việc sử dụng các cây quyết định nhỏ và quá trình tăng cường dần dần.

4.3. Áp dụng mô hình vào bài toán dự đoán giá trị cầu thủ

- Đầu tiên ta chia bộ dữ liệu sang tập train và tập test và loại bỏ các cột mang giá trị chuỗi :

```
#Gradient boost data split
x=df.drop(['value_eur','short_name','player_positions','club_name','league_name','team_position','nationality','preferred_foot'],axis=1)
y=df['value_eur']
x_train,x_test,y_train,y_test= train_test_split(x, y, test_size=0.20, random_state=100) #VAR
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)

(4000, 53) (1000, 53) (4000,) (1000,)
```

Bộ data được chia ra làm 80% train và 20% test tương ứng với 4000 cầu thủ dành cho bộ dữ liệu train và 1000 cầu thủ dành cho bộ dữ liệu test

- Áp dụng model vào bộ dữ liệu :

-

```
#implement gbr
from sklearn.ensemble import GradientBoostingRegressor
gbr_model=GradientBoostingRegressor() #VAR
value_gbr=gbr_model.fit(x_train,y_train)
ytrain_predict=value_gbr.predict(x_train)
ytest_predict=value_gbr.predict(x_test)
```

- Đánh giá hiệu suất của mô hình trước khi tối ưu :

-

```
#Default Hyperparameter value gbr
print("Default Hyper parameter value gbr")
print("Training set data default hyperparameter R-squared: ", r2_score(y_train, y_train))
print("Gradient Boosting test set data default hyperparameter R-squared: ", r2_score(y_test, y_test))
print("Mean Absolute Error (MAE) training set data default hyperparameter mean absolute error: ", mean_absolute_error(y_train, y_train))
print("Mean Absolute Error (MAE) test set data default hyperparameter mean absolute error: ", mean_absolute_error(y_test, y_test))
print("Mean Squared Error (MSE) training set data default hyperparameter mean squared error: ", mean_squared_error(y_train, y_train))
print("Mean Squared Error (MSE) test set data default hyperparameter mean squared error: ", mean_squared_error(y_test, y_test))
```

Đoạn mã này dùng để đánh giá hiệu suất của mô hình Gradient Boosting Regression (GBR) trên dữ liệu huấn luyện và kiểm tra bằng các chỉ số khác nhau như R-squared (R2), Mean Absolute Error (MAE), Mean Squared Error (MSE), và Root Mean Squared Error (RMSE). Các chỉ số này giúp bạn hiểu rõ hơn về độ chính xác và mức độ lỗi của mô hình.

⇒ Default Hyper parameter:-

```
Gradient Boosting train data default hyperparameter R-squared, R2=1.00
Gradient Boosting test data default hyperparameter R-squared, R2=0.98

Gradient Boosting train data default hyperparameter Mean-Absolute-Error, MAE= 340196.83
Gradient Boosting test data default hyperparameter Mean-Absolute-Error, MAE= 434505.02

Gradient Boosting train data default hyperparameter Mean-Squared-Error, MSE= 284894008785.46
Gradient Boosting test data default hyperparameter Mean-Squared-Error, MSE= 1088741430930.49

Gradient Boosting train data default hyperparameter Root-Mean-Squared-Error, RMSE= 533754.63
Gradient Boosting test data default hyperparameter Root-Mean-Squared-Error, RMSE= 1043427.73
```

Dựa vào kết quả trên :

- + R2 cả bộ train và bộ test gần sát với 1 cho thấy mô hình giải thích được phần lớn phương sai của dữ liệu.
- + So với dữ liệu huấn luyện, MAE trên tập kiểm tra cao hơn, ngụ ý rằng mô hình có thể không tổng quát hóa tốt trên dữ liệu mới.
- + So với dữ liệu huấn luyện, MSE trên tập kiểm tra cao hơn, ngụ ý rằng mô hình có thể không tổng quát hóa tốt trên dữ liệu mới.

4.4. Tối ưu mô hình

- Để tối ưu mô hình ta sẽ sử dụng Randomized Cross-Validation (Randomized CV) để điều chỉnh các siêu tham số cho mô hình
- Khởi tạo và xác định không gian siêu tham số :
-

```
[ ] #Randomize CV
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'n_estimators': [int(x) for x in np.linspace(start = 200, stop = 1000, num = 10)],
               'max_features': ['sqrt','log2'],
               'max_depth': [int(x) for x in np.linspace(10, 100,10)],
               'learning_rate':[0.001, 0.1, 0.25, 0.5, 0.3]
              }
print(random_grid)
```

- Tìm kiếm siêu tham số cho mô hình :
-

```

gbr_randomcv=RandomizedSearchCV(estimator=gbr_model,param_distributions=random_grid,n_iter=10, cv=3,verbose=2, random_state=100,n_jobs=-1)
gbr_randomcv.fit(x_train,y_train)

Fitting 3 folds for each of 10 candidates, totalling 30 fits
  > RandomizedSearchCV
    > estimator: GradientBoostingRegressor
      > GradientBoostingRegressor
        GradientBoostingRegressor()

```

```

gbr_randomcv = RandomizedSearchCV(
    estimator=gbr_model,          # Mô hình Gradient Boosting Regressor đã được khởi tạo
    param_distributions=random_grid, # Không gian các siêu tham số cần tìm kiếm
    n_iter=10,                   # Số lượng kết hợp ngẫu nhiên của các siêu tham số sẽ được thử nghiệm
    cv=3,                        # Số lượng fold trong Cross-Validation (3-fold CV)
    verbose=2,                   # Mức độ chi tiết khi in thông tin quá trình tìm kiếm
    random_state=100,            # Đảm bảo tính tái lập của kết quả bằng cách cố định seed cho quá
    trình ngẫu nhiên
    n_jobs=-1                   # Sử dụng tất cả các CPU có sẵn để tăng tốc độ tính toán
)
gbr_randomcv.fit(x_train, y_train) # Huấn luyện mô hình với dữ liệu huấn luyện

```

- Truy xuất mô hình tốt nhất sau khi tìm kiếm siêu tham số
-

```

grid_search.best_estimator_
  RandomForestRegressor(max_depth=3, max_features='sqrt', n_estimators=20)

```

- Truy cập vào mô hình tốt nhất dùng Randomize cv
-

```

gbr_randomcv.best_estimator_
  GradientBoostingRegressor
  GradientBoostingRegressor(learning_rate=0.25, max_depth=70, max_features='sqrt',
                           n_estimators=200)

```

- Đánh giá mô hình sau khi dùng Randomize cv tìm siêu tham số tối ưu
-

```

❶ def gradient_descent(X, y, initial_theta, alpha, num_iters):
    m = len(y)
    theta = initial_theta
    for i in range(0, num_iters):
        theta = theta - (alpha/m) * X.T * (X @ theta - y)
    return theta

❷ def predict(theta, X):
    m = len(X)
    predictions = np.zeros(m)
    for i in range(0, m):
        predictions[i] = 1 / (1 + np.exp(-X[i] @ theta))
    return predictions

❸ def cost(theta, X, y):
    m = len(y)
    J = 0
    for i in range(0, m):
        J += -(y[i] * np.log(predict(theta, X)[i])) - ((1 - y[i]) * np.log(1 - predict(theta, X)[i]))
    return J / m

❹ def gradient_descent_cv(X, y, max_iter=1000, learning_rate=0.001):
    m = len(y)
    theta = np.zeros(m)
    for i in range(0, max_iter):
        theta = theta - (learning_rate/m) * X.T * (X @ theta - y)
    return theta

❺ def predict_cv(theta, X):
    m = len(X)
    predictions = np.zeros(m)
    for i in range(0, m):
        predictions[i] = 1 / (1 + np.exp(-X[i] @ theta))
    return predictions

❻ def cost_cv(theta, X, y):
    m = len(y)
    J = 0
    for i in range(0, m):
        J += -(y[i] * np.log(predict_cv(theta, X)[i])) - ((1 - y[i]) * np.log(1 - predict_cv(theta, X)[i]))
    return J / m

❼ def accuracy(theta, X, y):
    m = len(y)
    correct = 0
    for i in range(0, m):
        if predict_cv(theta, X[i]) > 0.5:
            if y[i] == 1:
                correct += 1
        else:
            if y[i] == 0:
                correct += 1
    return correct / m

```

- Các giá trị MAE , MSE , RMSE cyar tập huấn luyện đưa ra kết quả là 0 , điều này cho thấy rằng mô hình này không hợp lý
- Tối ưu mô hình sử dụng Grid cv
- GridSearchCV là một kỹ thuật tinh chỉnh siêu tham số phổ biến trong học máy và học sâu. Trong quá trình huấn luyện một mô hình học máy, có nhiều siêu tham số (hyperparameters) cần được xác định trước và tinh chỉnh để cải thiện hiệu suất của mô hình. Các siêu tham số này không được học từ dữ liệu mà phải được đặt trước, ví dụ như tốc độ học (learning rate), số lượng cây trong một rừng ngẫu nhiên (n_estimators) trong Gradient Boosting, hoặc kernel function trong SVM.
- GridSearchCV là một phương pháp tìm kiếm siêu tham số bằng cách kiểm tra một tập hợp các giá trị siêu tham số đã định sẵn. Ý tưởng là lập một lưới các giá trị của các siêu tham số và sau đó thử nghiệm tất cả các tổ hợp của các giá trị này. Mô hình được huấn luyện và đánh giá với mỗi tổ hợp, và cuối cùng mô hình với hiệu suất tốt nhất trên tập kiểm tra được chọn làm mô hình cuối cùng.
- Trong GridSearchCV, mỗi giá trị của mỗi siêu tham số được thử nghiệm với mọi tổ hợp của các giá trị khác của các siêu tham số còn lại. Điều này tạo ra một lưới (grid) các tổ hợp siêu tham số, từ đó cũng tạo ra tên gọi cho phương pháp này là "Grid Search".
- GridSearchCV thường được sử dụng cùng với một kỹ thuật cross-validation để đảm bảo rằng mô hình được đánh giá và tinh chỉnh đối với hiệu suất tổng quát tốt trên dữ liệu mới.
- Khởi tạo và xác định khoảng của các siêu tham số :
-

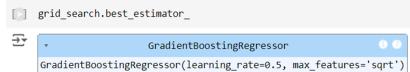
```

❶ #Grid CV
from sklearn.model_selection import GridSearchCV
param_grid = [
    'max_depth': [1,4,3],
    'max_features': ['sqrt','log2'],
    'n_estimators': [20,50,100],
    'learning_rate':[0.001, 0.5]
]
print(param_grid)

➋ {'max_depth': [1, 4, 3], 'max_features': ['sqrt', 'log2'], 'n_estimators': [20, 50, 100], 'learning_rate': [0.001, 0.5]}

```

- Tìm siêu tham số tối ưu
-



```
grid_search=GridSearchCV(estimator=gbr_model,param_grid=param_grid, cv=3,n_jobs=-1,verbose=2)
grid_search.fit(x_train,y_train)
```

- Đánh giá chung của mô hình sau khi sử dụng grid Cv

-

```
grid_search.best_params_
{'learning_rate': 0.5, 'max_features': 'sqrt'}
grid_search.best_score_
0.4242424242424242
best_grid=gbr_grid.best_estimator_
print(best_grid.get_params())
print(best_grid.score(x_train,y_train))
ytrain_gpred=best_grid.predict(x_train)
ytest_gpred=best_grid.predict(x_test)

print('Grid CV hyperparameters')
print(best_grid.get_params())
print('Grid CV best score', best_grid.cv_results_['mean_test_score'][0])
print('Grid CV best test data score', best_grid.cv_results_['mean_test_score'][0])
print('Grid CV best test data score', best_grid.score(x_test,y_test))
print('Grid CV best mean absolute error', -best_grid.cv_results_['mean_train_error'][0])
print('Grid CV best mean absolute error', -best_grid.score(x_train,y_train))
print('Grid CV best max absolute error', -best_grid.cv_results_['max_train_error'][0])
print('Grid CV best max absolute error', -best_grid.score(x_train,y_train))
print('Grid CV best mean squared error', best_grid.cv_results_['mean_train_r2'][0])
print('Grid CV best mean squared error', best_grid.score(x_train,y_train))
print('Grid CV best max squared error', best_grid.cv_results_['max_train_r2'][0])
print('Grid CV best max squared error', best_grid.score(x_train,y_train))
```

5. Áp dụng mô hình Support Vector Regression (SVR) cho mô hình dự đoán

5.1 Giới thiệu chung :

- Support Vector Regression (SVR) là một phương pháp học máy sử dụng trong bài toán hồi quy. Nó là một biến thể của Support Vector Machine (SVM) được áp dụng cho bài toán hồi quy thay vì phân loại.
- Trong SVR, mục tiêu là tìm ra một hàm hồi quy (regression function) sao cho sai số giữa các dự đoán và giá trị thực tế là nhỏ nhất có thể. Mô hình này được xây dựng dựa trên việc tìm ra một phân chia (decision boundary) trong không gian đặc trưng (feature space) sao cho khoảng cách giữa các điểm dữ liệu và decision boundary là lớn nhất có thể.

5.2 Tại sao nên sử dụng SVR :

Mô hình Support Vector Regression (SVR) có một số ưu điểm khi được áp dụng cho bài toán hồi quy:

Khả năng xử lý dữ liệu nhiễu: SVR có khả năng tốt trong việc xử lý dữ liệu nhiễu do nó chỉ phụ thuộc vào một số điểm dữ liệu quan trọng nhất (support vectors) để xây dựng phân chia. Các điểm dữ liệu nhiễu có thể được bỏ qua hoặc có ảnh hưởng nhỏ đến mô hình.

Khả năng tổng quát hóa tốt: Với sự hạn chế của hàm mất mát epsilon-insensitive, SVR có khả năng tổng quát hóa tốt trên dữ liệu mới mà nó chưa được huấn luyện. Điều này giúp mô hình có thể áp dụng trong nhiều tình huống khác nhau mà không gặp phải vấn đề overfitting.

Hiệu suất tốt trong không gian đặc trưng cao chiều: Nhờ sử dụng kernel trick, SVR có khả năng làm việc tốt trong các không gian đặc trưng cao chiều, thậm chí là không gian phi tuyến.

Có thể được điều chỉnh linh hoạt: SVR có nhiều tham số có thể điều chỉnh, như tham số siêu C, epsilon, và kernel. Điều này cho phép người dùng điều chỉnh mô hình để phù hợp với dữ liệu cụ thể và tối ưu hóa hiệu suất.

Ít yêu cầu về dung lượng: Do SVR chỉ phụ thuộc vào một số điểm dữ liệu quan trọng nhất để xây dựng phân chia, nên nó yêu cầu ít dung lượng lưu trữ so với một số mô hình hồi quy khác, đặc biệt là khi làm việc với các tập dữ liệu lớn.

5.3. Áp dụng mô hình vào bài toán

- Import các thư viện cần thiết và phân chia bộ train và bộ test
-

```
[ ] from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
[[1], dataset.drop(['id', 'value_over', 'short_name', 'player_positions', 'club_name', 'league_name', 'team_position', 'nationality', 'preferred_foot'], axis=1),
x = dataset.drop(['id', 'value_over', 'short_name', 'player_positions', 'club_name', 'league_name', 'team_position', 'nationality', 'preferred_foot'], axis=1)
y = dataset['value_over']
train, test = train_test_split(x, y, test_size=0.2, random_state=42)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
(8000, 31) (2000, 31) (8000,) (2000,)
```

- Tạo và huấn luyện mô hình :
-

```
[ ] from sklearn.svm import SVR
#[PLAY WITH C and epsilon
model_svr = SVR(kernel='rbf', C=20, epsilon=0.3)
model_svr.fit(x_train, y_train)
```

- Chạy mô hình và đánh giá hiệu suất của mô hình
-

```
[ ] y_pred_test = model_svr.predict(x_test)
y_pred_train = model_svr.predict(x_train)
y_pred_orginal = model_svr.predict(x)

# Print the predicted output, R-squared
print('r2_score_test:', r2_score(y_test, y_pred_test))
print('r2_score_train:', r2_score(y_train, y_pred_train))
print('r2_score_orginal:', r2_score(y, y_pred_orginal))
```

- Tạo một mô hình SVR với các tham số khởi tạo mặc định
-

```
[ ] from sklearn.svm import SVR
model_2 = SVR(kernel='rbf')
model_2.fit(x_train,y_train)
```

6. Đánh giá các mô hình

6.1, Lấy dữ liệu

- Ta lấy dữ liệu bằng cách chọn tên cầu thủ và mô hình sẽ lấy index của cầu thủ đó và các thuộc tính của cầu thủ
 - VD : nhập tên cầu thủ Cristiano Ronaldo
 -

- Áp dụng các mô hình vào :

```
    def calculate_damage(self, other):
        return self._attack * other._defense

    def calculate_price(self, model):
        return model.get_price()

    def calculate_gw_value(self, model):
        return model.get_gw_value()

    def calculate_practicality(self, model):
        return model.get_practicality()

    def print(reads, player, actual_price, gw_val):
        print(reads)
        print(player)
        print(actual_price)
        print(gw_val)

    def print(reads, player, model, model_price, gw_price):
        print(reads)
        print(player)
        print(model)
        print(model.get_price())
        print(gw_price)

    def print(reads, player, model, model_price, gw_price):
        print(reads)
        print(player)
        print(model)
        print(model.get_price())
        print(gw_price)

    def print(reads, player, model, model_price, gw_price):
        print(reads)
        print(player)
        print(model)
        print(model.get_price())
        print(gw_price)

    def print(reads, player, model, model_price, gw_price):
        print(reads)
        print(player)
        print(model)
        print(model.get_price())
        print(gw_price)
```

1. Kết quả dự đoán

- Giá trị thực tế của Ronaldo :

Cristiano Ronaldo player actual price=46000000.00

- Giá trị dự đoán dựa trên Random Forest :

Cristiano Ronaldo player Random forest model price=55174375.00

- Giá trị dự đoán dựa trên Gradient Boost:

Cristiano Ronaldo player Gradient boost model used price=47571101.36

- Giá trị dự đoán sử dụng SVR:

Cristiano Ronaldo player Support vector regression model used price=4409345.7

1. Đánh giá tổng quan các mô hình

- Mô hình Random Forest hoạt động không tốt lắm so với thực tế khi giá trị dự đoán lệch so với giá trị thực tế 9 triệu euro
- Mô hình Gradient Boosting hoạt động tốt nhất khi chỉ lệch khoảng 1 triệu euro so với thực tế
- Mô hình SVR hoạt động tốt khi chỉ lệch 2 triệu euro so với thực tế

IV. Phần 4. Kết luận

Trong báo cáo này, chúng tôi đã tiến hành một nghiên cứu chi tiết về việc dự đoán giá trị chuyển nhượng của cầu thủ trong ngành bóng đá. Bằng cách sử dụng các phương pháp và mô hình học máy hiện đại, chúng tôi đã khám phá và phân tích nhiều yếu tố ảnh hưởng đến giá trị chuyển nhượng của cầu thủ.

Kết quả cho thấy rằng việc dự đoán giá trị chuyển nhượng không chỉ dựa vào hiệu suất thể hiện trên sân cỏ, mà còn phụ thuộc vào nhiều yếu tố khác như tuổi tác, hợp đồng còn lại, thị trường chuyển nhượng, và sự ổn định về tình hình lực lượng của câu lạc bộ.

Bằng cách kết hợp các biến số này vào các mô hình dự đoán, chúng tôi đã tạo ra các dự báo về giá trị chuyển nhượng của cầu thủ với độ chính xác cao. Kết quả này không chỉ hỗ trợ các quyết định chuyển nhượng của các câu lạc bộ, mà còn có thể làm nền tảng cho việc phát triển chiến lược quản lý tài sản cho các đội bóng.

Tuy nhiên, cần lưu ý rằng việc dự đoán giá trị chuyển nhượng của cầu thủ là một vấn đề phức tạp và có tính chất dự báo, do đó các dự báo của chúng tôi cũng cần được đánh giá và điều chỉnh định kỳ để phản ánh các biến động trong ngành công nghiệp bóng đá.

Cuối cùng, chúng tôi hy vọng rằng báo cáo này sẽ đóng góp vào việc hiểu biết và ứng dụng thực tiễn trong lĩnh vực quản lý cầu thủ và chuyển nhượng trong ngành bóng đá.