ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ

BÀI TẬP LỚN
MÁY HỌC CƠ BẢN & ỨNG DỤNG
LỚP L01

Đề tài: Predicting the outcome soccer matches using Machine Learning

---

GVHD: ThS. VÕ TUẤN KIỆT
SV thực hiện: Đinh Bá Duy - 2010179
Lê Đức Minh Nhật - 2013994

TP. Hồ Chí Minh, Tháng 03/2024

Mục lục

Danh sách bảng

Danh sách hình vẽ

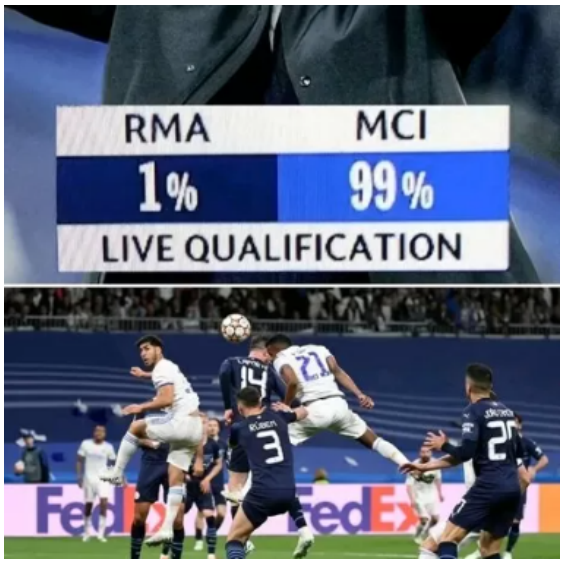
1 Tóm tắt

Đây là đề tài so sánh và đánh giá các phương pháp Machine Learning áp dụng cho việc dự đoán kết quả của các trận đấu bóng đá, thuộc dạng phân loại nhiều lớp với dữ liệu nhiễu và mất cân bằng. Để giải quyết vấn đề này, nhóm đã thực hiện việc cân đối dữ liệu, loại bỏ các thành phần dữ liệu thừa và ít ảnh hưởng đến kết quả rồi tiến hành nhiều phương pháp khác nhau với các thiết lập thông số cho các phương pháp khác nhau. Kết quả cuối cùng cho thấy, Random Forest là phương pháp cho kết quả tốt nhất với độ chính xác tổng thể 52.26% và độ chính xác cho từng dự đoán kết quả 46.9%.

2 Giới thiệu

Hiện nay, Machine Learning (Học Máy) được ứng dụng rất rộng rãi trên nhiều lĩnh vực trong đời sống của chúng ta, và một trong những ứng dụng phổ biến nhất đó chính là dùng để dự đoán kết quả. Đó có thể là kết quả của thị trường tài chính, hoặc dự đoán tiêu thụ và xu hướng khách hàng trong kinh doanh, hoặc có thể là dự đoán bất động sản như giá nhà đất, giá chung cư, ... Với sở thích của các cá nhân, nhóm đã quyết định lựa chọn đề tài "Ứng dụng của Machine Learning trong Dự đoán kết quả trận đấu bóng đá".

Trên thực tế, các BOT dự đoán kết quả bóng đá sẽ đưa ra kết quả dựa vào các thông số thu thập được tại một thời điểm, phản ánh một phần của kết quả. Tuy nhiên, các kết quả đưa ra chỉ mang tính tham khảo. Như dưới đây là ví dụ thực tế. BOT dự đoán Real Madrid chỉ có 1% chiến thắng khi trận đấu đã điểm 90 phút, nhưng kết quả cuối cùng thì Real Madrid vẫn giành kết quả chiến thắng.



Hình 1: BOT AI dự đoán tỉ lệ chiến thắng của trận đấu. Nguồn [1]

EA (Electronic Arts) là một công ty sản xuất trò chơi điện tử nổi tiếng đã áp dụng máy học để dự đoán kết quả của giải đấu World Cup từ 2010. Và điều đáng nói là đã dự đoán đúng kết quả của 4 mùa World Cup liên tiếp gần đây nhất là 2010[2], 2014[3], 2018[4] và 2022[5]. Công nghệ mà EA sử dụng để dự đoán kết quả vào World Cup 2022 được gọi là *Hypermotion 2*[6]. Hypermotion 2 sử dụng dữ liệu từ chuyển động của các cầu thủ trong các trận đấu thực tế để huấn luyện cho mô hình học máy từ đó giả lập lại các trận đấu và đưa ra kết quả của trận đấu.

3 Nội dung báo cáo

3.1 Cài đặt thư viện và mô hình phân loại

3.1.1 Cài đặt thư viện

- NumPy: Cung cấp cấu trúc mảng số học nhanh và các hàm hỗ trợ.
- pandas: Cung cấp cấu trúc DataFrame để lưu trữ dữ liệu trong bộ nhớ và làm việc với nó một cách dễ dàng và hiệu quả.
- scikit-learn: Thư viện Học máy cần thiết cho nhiều mô hình học có giám sát, trong Python.
- tensorflow: Thư viện Học máy cần thiết cho học sâu, trong Python.
- matplotlib: Thư viện vẽ biểu đồ cơ bản trong Python; hầu hết các thư viện vẽ biểu đồ Python khác được xây dựng dựa trên nó.

3.1.2 Một số mô hình phân loại nhiều lớp

- **Decision Tree**: là một thuật toán máy học được sử dụng cho các tác vụ phân loại và hồi quy. Nó là một cấu trúc cây dạng đồ thị có các nút đại diện cho các quyết định hoặc các nút lá đại diện cho các nhãn hoặc giá trị dự đoán. Ý tưởng cơ bản của Decision Tree là phân tách tập dữ liệu ban đầu thành các tập con nhỏ hơn dựa trên các quyết định dựa trên các thuộc tính của dữ liệu. Các quyết định này được xây dựng dựa trên các "câu hỏi"về các thuộc tính, và mỗi câu hỏi tạo ra một nhánh mới trong cây. Quá trình này được tiếp tục đến khi đạt được một điều kiện dừng, ví dụ như khi không còn có thêm thông tin hữu ích để phân tách hoặc khi đạt được một điều kiện dừng được định trước.
- **Random Forest**: là một thuật toán máy học dựa trên việc xây dựng một tập hợp (ensemble) các cây quyết định (decision tree). Nó kết hợp sự ngẫu nhiên trong việc xây dựng các cây quyết định để tạo ra một mô hình mạnh mẽ và ổn định. Ý tưởng cơ bản của Random Forest là xây dựng nhiều cây quyết định độc lập trên một tập dữ liệu huấn luyện được tạo ra bằng cách lấy mẫu ngẫu nhiên từ tập dữ liệu ban đầu. Mỗi cây quyết định được xây dựng bằng cách chọn ngẫu nhiên một số thuộc tính để đặt câu hỏi tại mỗi nút trong quá trình xây dựng cây. Quá trình này được lặp lại để tạo ra một tập hợp các cây quyết định. Khi tiến hành dự đoán mới, kết quả dự đoán cuối cùng được tính toán bằng cách lấy trung bình hoặc bình chọn kết quả từ các cây quyết định trong tập hợp.
- **XGBoost**: là một thuật toán máy học dựa trên kỹ thuật Gradient Boosting Decision Trees (GBDT). Nó là một trong những thuật toán được sử dụng rộng rãi và hiệu quả cao trong các cuộc thi và ứng dụng thực tế. XGBoost sử dụng mô hình cây quyết định như mô hình cơ bản và xây dựng một tập hợp các cây quyết định tuần tự. Quá trình xây dựng mô hình

diễn ra bằng cách tối ưu hóa một hàm mất mát (loss function) thông qua việc tìm kiếm các cây quyết định phù hợp. Quá trình này được thực hiện theo cách tương tự như Gradient Boosting, trong đó các cây mới được xây dựng để giảm thiểu hàm mất mát của mô hình hiện tại.

- **Support Vector Machines (SVM)**: là một thuật toán máy học phân loại và hồi quy được sử dụng rộng rãi trong lĩnh vực nhận dạng mẫu và phân loại dữ liệu. Ý tưởng cơ bản của SVM là tìm một siêu phẳng (hyperplane) trong không gian đặc trưng để phân tách các điểm dữ liệu thuộc vào các lớp khác nhau. Siêu phẳng này được tối ưu sao cho càng cách xa các điểm dữ liệu gần nhất thuộc vào các lớp khác nhau, được gọi là các vector hỗ trợ (support vectors). SVM có thể hoạt động tốt với các tập dữ liệu có đặc trưng tuyến tính hoặc phi tuyến tính thông qua sử dụng các hàm kernel để ánh xạ không gian đặc trưng ban đầu vào không gian đặc trưng nhiều chiều hơn.

- **K-Nearest Neighbors (KNN)**: là một thuật toán máy học giám sát được sử dụng trong các bài toán phân loại và hồi quy. Ý tưởng cơ bản của KNN là dự đoán nhãn hoặc giá trị của một điểm dữ liệu mới bằng cách xem xét nhãn hoặc giá trị của K điểm dữ liệu gần nhất trong tập dữ liệu huấn luyện.

- **Gaussian Naive Bayes (GNB)**: là một thuật toán học máy được sử dụng trong bài toán phân loại. Nó là một biến thể của Naive Bayes Classifier, một mô hình dựa trên nguyên tắc của định lý Bayes. Thuật toán Gaussian Naive Bayes được sử dụng trong trường hợp dữ liệu đầu vào là các biến liên tục và tuân theo phân phối Gaussian (hoặc phân phối chuẩn). Thuật toán giả định rằng các biến đầu vào độc lập và có phân phối Gaussian với mỗi lớp. Ý tưởng chính của Gaussian Naive Bayes là tính toán xác suất của một điểm dữ liệu thuộc về mỗi lớp dựa trên xác suất tiên nghiệm (prior probability) của lớp đó và xác suất của các giá trị biến đầu vào trong lớp đó. Điểm dữ liệu sẽ được phân loại vào lớp có xác suất cao nhất.

4 Tiến hành huấn luyện

4.1 Phân tích dữ liệu

Dữ liệu được sử dụng là kết quả của 11 giải bóng đá ở khu vực châu Âu, cụ thể gồm các dữ liệu từ *match*, *team*, country, *league* và *player*, được lấy từ *Kaggle*[7]. Do cơ sở dữ liệu có định dạng *SQLite*, cần sử dụng SQL để xử lý dữ liệu thành một bảng dữ liệu tổng thể. Dữ liệu tổng thể bao gồm khoảng *26.000* trận đấu từ *11* giải đấu châu Âu từ năm 2008 đến năm 2016, và mỗi trận đấu có một số thuộc tính nhất định như kết quả, đội bóng tham gia, cầu thủ tham gia và ngày thi đấu, do đó cần tiến hành liên kết bảng dữ liệu trận đấu với các bảng dữ liệu khác dựa trên các thuộc tính này để có được bảng dữ liệu tổng thể cho việc huấn luyện. Cụ thể các bước tiến hành phân tích như sau:

4.1.1 Trích xuất dữ liệu từ cơ sở dữ liệu[8]

.

```
with sqlite3.connect("dataset/database.sqlite") as con:
    matches = pd.read_sql_query("SELECT * from Match", con)
    team_attributes = pd.read_sql_query("SELECT distinct * from
   Team_Attributes", con)
    player = pd.read_sql_query("SELECT * from Player", con)
    player_attributes = pd.read_sql_query("SELECT * from
   Player_Attributes", con)
```

4.1.2 Xử lí dữ liệu từ *Match*

Xử lí các dữ liệu bị trùng lắp và định nghĩa một số hàm để trích xuất dữ liệu và nhãn

```
''' Derives a label for a given match. '''
def get_match_outcome(match):
    # Define variables
    home_goals = match['home_team_goal']
    away_goals = match['away_team_goal']

    outcome = pd.DataFrame()  # Create a new dataframe: outcome
    outcome.loc[0, 'match_api_id'] = match['match_api_id']  #
   Insert match_api_id into outcome

    # Identify match outcome
    if home_goals > away_goals:
        outcome.loc[0, 'outcome'] = "Win"
    if home_goals == away_goals:
        outcome.loc[0, 'outcome'] = "Draw"
    if home_goals < away_goals:
        outcome.loc[0, 'outcome'] = "Defeat"

```

```
    # Return outcome
    return outcome.loc[0]


''' Get the last x matches of a given team. '''
def get_last_matches(matches, date, team, x=10):
    # Filter team matches from matches
    team_matches = matches[(matches['home_team_api_id'] == team) |
    (matches['away_team_api_id'] == team)]

    # Filter x last matches from team matches
    last_matches = team_matches[team_matches.date < date].
   sort_values(by='date', ascending=False).iloc[0:x, :]

    # Return last matches
    return last_matches


''' Get the last team stats of a given team. '''
def get_last_team_stats(team_id, date, teams_stats):
    # Filter team stats
    all_team_stats = teams_stats[teams_stats['team_api_id'] ==
   team_id]

    # Filter last stats from team
    last_team_stats = all_team_stats[all_team_stats.date < date].
   sort_values(by='date', ascending=False)
    if last_team_stats.empty:
        last_team_stats = all_team_stats[all_team_stats.date >
   date].sort_values(by='date', ascending=True)

    # Return last matches
    return last_team_stats.iloc[0:1, :]


''' Get the last x matches of two given teams. '''
def get_last_matches_against_eachother(matches, date, home_team,
   away_team, x=10):
    # Find matches of both teams
    home_matches = matches[(matches['home_team_api_id'] ==
   home_team) & (matches['away_team_api_id'] == away_team)]
    away_matches = matches[(matches['home_team_api_id'] ==
   away_team) & (matches['away_team_api_id'] == home_team)]
    total_matches = pd.concat([home_matches, away_matches])

    # Get last x matches
    try:
        last_matches = total_matches[total_matches.date < date].
   sort_values(by='date', ascending=False).iloc[0:x, :]
```

```
    except:
        last_matches = total_matches[total_matches.date < date].sort_values(by='date', ascending=False).iloc[
                       0:total_matches.shape[0], :]

        # Check for error in data
        if (last_matches.shape[0] > x):
            print("Error in obtaining matches")

    # Return data
    return last_matches


''' Get the goals[home & away] of a specfic team from a set of matches. '''
def get_goals(matches, team):
    home_goals = int(matches.home_team_goal[matches.home_team_api_id == team].sum())
    away_goals = int(matches.away_team_goal[matches.away_team_api_id == team].sum())

    total_goals = home_goals + away_goals

    return total_goals


''' Get the goals[home & away] conceided of a specfic team from a set of matches. '''
def get_goals_conceided(matches, team):
    home_goals = int(matches.home_team_goal[matches.away_team_api_id == team].sum())
    away_goals = int(matches.away_team_goal[matches.home_team_api_id == team].sum())

    total_goals = home_goals + away_goals

    return total_goals


''' Get the number of wins of a specfic team from a set of matches. '''
def get_wins(matches, team):
    # Find home and away wins
    home_wins = int(matches.home_team_goal[
                        (matches.home_team_api_id == team) & (matches.home_team_goal > matches.away_team_goal)].count())
    away_wins = int(matches.away_team_goal[
                        (matches.away_team_api_id == team) & (matches.away_team_goal > matches.home_team_goal)].count())
```

```
97
98      total_wins = home_wins + away_wins
99
100     return total_wins
101
102
103 ''' Create match specific features for a given match. '''
104 def get_match_features(match, matches, teams_stats, x=10):
105     # Define variables
106     date = match.date
107     home_team = match.home_team_api_id
108     away_team = match.away_team_api_id
109
110     # Gets home and away team_stats
111     home_team_stats = get_last_team_stats(home_team, date,
    teams_stats);
112     away_team_stats = get_last_team_stats(away_team, date,
    teams_stats);
113
114     # Get last x matches of home and away team
115     matches_home_team = get_last_matches(matches, date, home_team,
     x=5)
116     matches_away_team = get_last_matches(matches, date, away_team,
     x=5)
117
118     # Get last x matches of both teams against each other
119     last_matches_against = get_last_matches_against_eachother(
    matches, date, home_team, away_team, x=3)
120
121     # Create goal variables
122     home_goals = get_goals(matches_home_team, home_team)
123     away_goals = get_goals(matches_away_team, away_team)
124     home_goals_conceided = get_goals_conceided(matches_home_team,
    home_team)
125     away_goals_conceided = get_goals_conceided(matches_away_team,
    away_team)
126
127     # Define result data frame
128     result = pd.DataFrame()
129
130     # Define ID features
131     result.loc[0, 'match_api_id'] = match.match_api_id
132     result.loc[0, 'league_id'] = match.league_id
133
134     # Create match features and team stats
135     if not home_team_stats.empty:
136         result.loc[0, 'home_team_buildUpPlaySpeed'] =
    home_team_stats['buildUpPlaySpeed'].values[0]
```

```
137         result.loc[0, 'home_team_buildUpPlayPassing'] =
    home_team_stats['buildUpPlayPassing'].values[0]
138         result.loc[0, 'home_team_chanceCreationPassing'] =
    home_team_stats['chanceCreationPassing'].values[0]
139         result.loc[0, 'home_team_chanceCreationCrossing'] =
    home_team_stats['chanceCreationCrossing'].values[0]
140         result.loc[0, 'home_team_chanceCreationShooting'] =
    home_team_stats['chanceCreationShooting'].values[0]
141         result.loc[0, 'home_team_defencePressure'] =
    home_team_stats['defencePressure'].values[0]
142         result.loc[0, 'home_team_defenceAggression'] =
    home_team_stats['defenceAggression'].values[0]
143         result.loc[0, 'home_team_defenceTeamWidth'] =
    home_team_stats['defenceTeamWidth'].values[0]
144         result.loc[0, 'home_team_avg_shots'] = home_team_stats['
    avg_shots'].values[0]
145         result.loc[0, 'home_team_avg_corners'] = home_team_stats['
    avg_corners'].values[0]
146         result.loc[0, 'home_team_avg_crosses'] = away_team_stats['
    avg_crosses'].values[0]
147
148     if (not away_team_stats.empty):
149         result.loc[0, 'away_team_buildUpPlaySpeed'] =
    away_team_stats['buildUpPlaySpeed'].values[0]
150         result.loc[0, 'away_team_buildUpPlayPassing'] =
    away_team_stats['buildUpPlayPassing'].values[0]
151         result.loc[0, 'away_team_chanceCreationPassing'] =
    away_team_stats['chanceCreationPassing'].values[0]
152         result.loc[0, 'away_team_chanceCreationCrossing'] =
    away_team_stats['chanceCreationCrossing'].values[0]
153         result.loc[0, 'away_team_chanceCreationShooting'] =
    away_team_stats['chanceCreationShooting'].values[0]
154         result.loc[0, 'away_team_defencePressure'] =
    away_team_stats['defencePressure'].values[0]
155         result.loc[0, 'away_team_defenceAggression'] =
    away_team_stats['defenceAggression'].values[0]
156         result.loc[0, 'away_team_defenceTeamWidth'] =
    away_team_stats['defenceTeamWidth'].values[0]
157         result.loc[0, 'away_team_avg_shots'] = away_team_stats['
    avg_shots'].values[0]
158         result.loc[0, 'away_team_avg_corners'] = away_team_stats['
    avg_corners'].values[0]
159         result.loc[0, 'away_team_avg_crosses'] = away_team_stats['
    avg_crosses'].values[0]
160
161     result.loc[0, 'home_team_goals_difference'] = home_goals -
    home_goals_conceided
162     result.loc[0, 'away_team_goals_difference'] = away_goals -
    away_goals_conceided
```

```
163     result.loc[0, 'games_won_home_team'] = get_wins(
    matches_home_team, home_team)
164     result.loc[0, 'games_won_away_team'] = get_wins(
    matches_away_team, away_team)
165     result.loc[0, 'games_against_won'] = get_wins(
    last_matches_against, home_team)
166     result.loc[0, 'games_against_lost'] = get_wins(
    last_matches_against, away_team)
167     result.loc[0, 'B365H'] = match.B365H
168     result.loc[0, 'B365D'] = match.B365D
169     result.loc[0, 'B365A'] = match.B365A
170 
171     # Return match features
172     return result.loc[0]
173 
174 
175 ''' Create and aggregate features and labels for all matches. '''
176 def get_features(matches, teams_stats, fifa, x=10, get_overall=
    False):
177     # Get fifa stats features
178     fifa_stats = get_overall_fifa_rankings(fifa, get_overall)
179 
180     # Get match features for all matches
181     match_stats = matches.apply(lambda i: get_match_features(i,
    matches, teams_stats, x=10), axis=1)
182 
183     # Create dummies for league ID feature
184     dummies = pd.get_dummies(match_stats['league_id']).rename(
    columns=lambda x: 'League_' + str(x))
185     match_stats = pd.concat([match_stats, dummies], axis=1)
186     match_stats.drop(['league_id'], inplace=True, axis=1)
187 
188     # Create match outcomes
189     outcomes = matches.apply(get_match_outcome, axis=1)
190 
191     # Merges features and outcomes into one frame
192     features = pd.merge(match_stats, fifa_stats, on='match_api_id'
    , how='left')
193     features = pd.merge(features, outcomes, on='match_api_id', how
    ='left')
194 
195     # Drop NA values
196     features.dropna(inplace=True)
197 
198     # Return preprocessed data
199     return features
200 
201 ''' Get overall fifa rankings from fifa data. '''
202 def get_overall_fifa_rankings(fifa, get_overall=False):
```

```
    temp_data = fifa

    # Check if only overall player stats are desired
    if get_overall == True:

        # Get overall stats
        data = temp_data.loc[:, (fifa.columns.str.contains('overall_rating'))]
        data.loc[:, 'match_api_id'] = temp_data.loc[:, 'match_api_id']
    else:

        # Get all stats except for stat date
        cols = fifa.loc[:, (fifa.columns.str.contains('date_stat'))]
        temp_data = fifa.drop(cols.columns, axis=1)
        data = temp_data

    # Return data
    return data
```

Có tổng cộng 25979 kết quả trận đấu. Điều này có nghĩa là ta đang phân tích số lượng trận đấu này từ cơ sở dữ liệu. Ta có thể bắt đầu bằng cách xem xét dữ liệu của người dự đoán. Ta có thể thấy rằng số lượng dữ liệu trận đấu của người dự đoán khác nhau cho mỗi người dự đoán. Ta bắt đầu bằng cách chọn người dự đoán có số lượng dữ liệu dự đoán nhiều nhất có sẵn.

```
viable_matches = matches.sample(n=5000)
b365 = viable_matches.dropna(subset=['B365H', 'B365D', 'B365A'], inplace=False)
b365.drop(['BWH', 'BWD', 'BWA',
           'IWH', 'IWD', 'IWA',
           'LBH', 'LBD', 'LBA',
           'PSH', 'PSD', 'PSA',
           'WHH', 'WHD', 'WHA',
           'SJH', 'SJD', 'SJA',
           'VCH', 'VCD', 'VCA',
           'GBH', 'GBD', 'GBA',
           'BSH', 'BSD', 'BSA'], inplace = True, axis = 1)

bw = viable_matches.dropna(subset=['BWH', 'BWD', 'BWA'],inplace=False)
bw.drop(['B365H', 'B365D', 'B365A',
           'IWH', 'IWD', 'IWA',
           'LBH', 'LBD', 'LBA',
           'PSH', 'PSD', 'PSA',
           'WHH', 'WHD', 'WHA',
           'SJH', 'SJD', 'SJA',
           'VCH', 'VCD', 'VCA',
```

```
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

iw = viable_matches.dropna(subset=['IWH', 'IWD', 'IWA'],inplace=
    False)
iw.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

lb = viable_matches.dropna(subset=['LBH', 'LBD', 'LBA'],inplace=
    False)
lb.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

ps = viable_matches.dropna(subset=['PSH', 'PSD', 'PSA'],inplace=
    False)
ps.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

wh = viable_matches.dropna(subset=['WHH', 'WHD', 'WHA'],inplace=
    False)
wh.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA',
```

```
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

sj = viable_matches.dropna(subset=['SJH', 'SJD', 'SJA'],inplace=
    False)
sj.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

vc = viable_matches.dropna(subset=['VCH', 'VCD', 'VCA'],inplace=
    False)
vc.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'GBH', 'GBD', 'GBA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

gb = viable_matches.dropna(subset=['GBH', 'GBD', 'GBA'],inplace=
    False)
gb.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'BSH', 'BSD', 'BSA'], inplace=True, axis = 1)

bs = viable_matches.dropna(subset=['BSH', 'BSD', 'BSA'],inplace=
    False)
bs.drop(['B365H', 'B365D', 'B365A',
            'BWH', 'BWD', 'BWA',
            'IWH', 'IWD', 'IWA',
            'LBH', 'LBD', 'LBA',
            'PSH', 'PSD', 'PSA',
            'WHH', 'WHD', 'WHA',
            'SJH', 'SJD', 'SJA',
            'VCH', 'VCD', 'VCA',
            'GBH', 'GBD', 'GBA'], inplace=True, axis = 1)
```

```
111
112 lis = [b365, bw, iw, lb, ps, wh, sj, vc, gb, bs]
113
114 viable_matches = max(lis, key = lambda datframe: datframe.shape
    [0])
```

Phân tích mô tả của dữ liệu, ta thấy rằng người dự đoán liên quan đến Bet 365 có thông tin sẵn nhiều nhất và, do đó, ta quyết định chọn nó làm đầu vào cho các mô hình của ta.

Ta cũng cần xem xét rằng một số trận đấu có thể không có trong các thuộc tính mà ta cần. Trong trường hợp đó, ta cần loại bỏ bất kỳ trận đấu nào không chứa bất kỳ thông tin thống kê nào, vì phương pháp điền giá trị trung bình sẽ không hoạt động trong trường hợp này.

Ta cũng cần loại bỏ một số hàng không chứa bất kỳ thông tin nào về vị trí của các cầu thủ cho một số trận đấu.

```
1 teams_stats = team_attributes
2 viable_matches = viable_matches.dropna(inplace=False)
3
4 home_teams = viable_matches['home_team_api_id'].isin(teams_stats['
    team_api_id'].tolist())
5 away_teams = viable_matches['away_team_api_id'].isin(teams_stats['
    team_api_id'].tolist())
6 viable_matches = viable_matches[home_teams & away_teams]
```

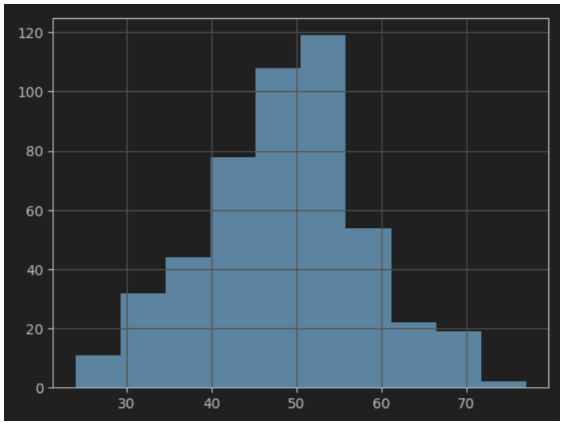
4.1.3 Xử lí dữ liệu về *thuộc tính của đội*

Thuộc tính buildUpPlayDribbling

Xem xét mô tả thuộc tính của đội, chúng ta có thể thấy rằng có rất nhiều giá trị bị thiếu trong cột buildUpPlayDribbling, trong khi tất cả các cột khác dường như có đúng số lượng dòng. Điều này có nghĩa là có rất nhiều giá trị 'Nan' trong cột này.

Loại bỏ các dòng đó không phải là giải pháp lý tưởng. Có vẻ như dữ liệu thiếu trong cột này là có hệ thống - tất cả các giá trị thiếu đều nằm trong cùng một cột - lỗi này có khả năng làm sai lệch phân tích của chúng ta.

Một cách để xử lý các giá trị thiếu là điền trung bình. Nếu chúng ta biết rằng các giá trị cho phép đo lường nằm trong một phạm vi nhất định, chúng ta có thể điền vào các giá trị trống bằng giá trị trung bình của phép đo đó.



Hình 2: Phân bố dữ liệu của thuộc tính *buildUpPlayDribbling*

Hầu hết các giá trị của buildUpPlayDribbling nằm trong khoảng 45 - 55, vì vậy chúng ta hãy điền các mục nhập này bằng tốc độ buildUpPlaySpeed trung bình được đo.

```
build_up_play_drib_avg = teams_stats['buildUpPlayDribbling'].mean
    ()
teams_stats.loc[(teams_stats['buildUpPlayDribbling'].isnull()), '
    buildUpPlayDribbling'] = build_up_play_drib_avg
```

Sau khi thực hiện việc điền giá trị trung bình cho *buildUpPlayDribbling*, chúng ta có thể thấy không còn các giá trị thiếu nữa cho *buildUpPlayDribbling*. Sau đó, chúng tôi quyết định chỉ chọn dữ liệu liên tục, tức là chỉ chọn các cột chứa các giá trị số mà chúng tôi sẽ cung cấp cho đầu vào của các mô hình học có giám sát.

```
teams_stats.drop(['buildUpPlaySpeedClass', '
    buildUpPlayDribblingClass', 'buildUpPlayPassingClass',
     'buildUpPlayPositioningClass', 'chanceCreationPassingClass', '
    chanceCreationCrossingClass','chanceCreationShootingClass','
    chanceCreationPositioningClass','defencePressureClass', '
    defenceAggressionClass','defenceTeamWidthClass','
    defenceDefenderLineClass'], inplace = True, axis = 1)
```

Thuộc tính avg_shots

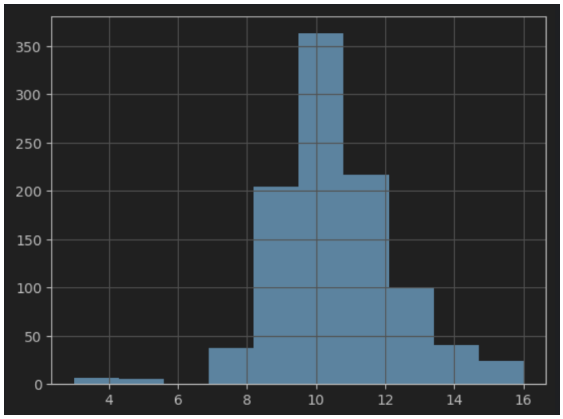
Chúng ta sẽ bắt đầu bằng cách thêm trung bình số lượng cú sút của mỗi đội. Số lượng cú sút bao gồm tổng số cú sút trúng và cú sút không trúng. Sau khi gộp tất cả thông tin vào team_stats, chúng ta phải phân tích lại dữ liệu.

```
shots_off = pd.read_csv("dataset/shotoff_detail.csv")
shots_on = pd.read_csv("dataset/shoton_detail.csv")
shots = pd.concat([shots_off[['match_id', 'team']], shots_on[['
    match_id', 'team']]])

total_shots = shots["team"].value_counts()
```

```
total_matches = shots.drop_duplicates(['match_id', 'team'])["team"
    ].value_counts()

for index, n_shots in total_shots.items():
    n_matches = total_matches[index]
    avg_shots = n_shots / n_matches
    teams_stats.loc[teams_stats['team_api_id'] == index, '
    avg_shots'] = avg_shots
```



Hình 3: Phân bố dữ liệu của thuộc tính *avg_ shots*

Chúng ta có thể thấy rằng hầu hết các giá trị avg_shots nằm trong phạm vi 7 - 14, vì vậy hãy điền các giá trị này bằng giá trị trung bình được đo đạc của avg_shots.

```
shots_avg_team_avg = teams_stats['avg_shots'].mean()
# mean imputation
teams_stats.loc[(teams_stats['avg_shots'].isnull()), 'avg_shots']
    = shots_avg_team_avg
```

Thuộc tính possession

Về thực tế, tỉ lệ kiểm soát bóng thể hiện rõ được thế trận của trận đấu, tuy nhiên, không phải yếu tố chính để đánh giá kết quả. Tỉ lệ kiểm soát bóng thay đổi liên tục xuyên suốt trận, tuỳ thuộc vào chiến thuật của đội bóng. Cụ thể một đội bóng kiểm soát bóng nhiều hơn nhưng cơ hội thành thấp hoặc không có cơ hội ghi bàn thì khả năng để có chiến thắng thấp hơn. Do đó, chúng tôi quyết định loại bỏ thuộc tính này, mặc dù thuộc tính này có ý nghĩa rất quan trọng.

Thuộc tính corners

Chúng tôi sẽ thêm một tính năng nữa, đó là số lượng phạt góc, cũng là một chỉ số quan trọng về sự kiểm soát thế trận trong một trận đấu bóng đá.

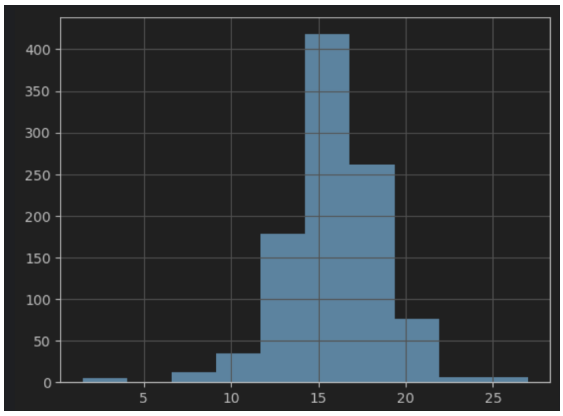
```
corners_data = pd.read_csv("dataset/corner_detail.csv")
corners = corners_data[['match_id', 'team']]

total_corners = corners["team"].value_counts()
```

```
total_matches = corners.drop_duplicates(['match_id', 'team'])["
    team"].value_counts()

for index, n_corners in total_shots.items():
    n_matches = total_matches[index]
    avg_corners = n_corners / n_matches
    teams_stats.loc[teams_stats['team_api_id'] == index, '
    avg_corners'] = avg_corners
```

Sau khi hợp nhất tất cả dữ liệu về phạt góc đã được cung cấp, chúng ta nhận thấy có một số giá trị bị thiếu. Trong dữ liệu được cung cấp sẵn, có rất nhiều giá trị NaN trên cột avg_corners. Điều này thể hiện các đội không có dữ liệu về phạt góc trong tập dữ liệu này. Thay vì loại bỏ những dòng đó và cung cấp ít dữ liệu hơn cho các mô hình, chúng ta cần phải thực hiện lại việc điền giá trị trung bình và xử lý những giá trị này.



Hình 4: Phân bố dữ liệu của thuộc tính *avg_corners*

Chúng ta có thể thấy rằng hầu hết các giá trị avg_corners nằm trong khoảng 8.5 - 12, vì vậy sẽ điền các mục này bằng giá trị trung bình của avg_corners đã đo được.

```
crosses_avg_team_avg = teams_stats['avg_crosses'].mean()

teams_stats.loc[(teams_stats['avg_crosses'].isnull()), '
    avg_crosses'] = crosses_avg_team_avg
```

Thuộc tính crosses

Thuộc tính cuối cùng cần được thêm vào là dữ liệu về các đường căng ngang. Thông thường, đội bóng tạo ra nhiều cơ hội ghi bàn hơn sẽ có nhiều đường căng ngang hơn vì nó tạo ra nhiều cơ hội ghi bàn hơn trong suốt một trận đấu. Sau khi hợp nhất tất cả dữ liệu, chúng ta cần kiểm tra xem có các dòng thiếu trên cột mới được thêm vào không.

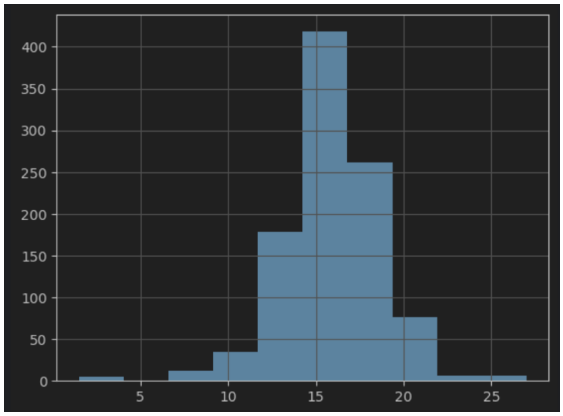
```
crosses_data = pd.read_csv("dataset/cross_detail.csv")

crosses = crosses_data[['match_id', 'team']]
```

```
total_crosses = crosses["team"].value_counts()
total_matches = crosses.drop_duplicates(['match_id', 'team'])["
    team"].value_counts()

for index, n_crosses in total_crosses.items():
    n_matches = total_matches[index]
    avg_crosses = n_crosses / n_matches
    teams_stats.loc[teams_stats['team_api_id'] == index, '
    avg_crosses'] = avg_crosses
```

Tương tự, có rất nhiều giá trị NaN trên cột avg_crosses. Điều này thể hiện cho các đội không có dữ liệu về các đường căng ngang trong tập dữ liệu này. Thay vì loại bỏ những dòng đó và cung cấp ít dữ liệu hơn cho các mô hình, chúng ta cần phải thực hiện lại việc điền giá trị trung bình và xử lý những giá trị này.



Hình 5: Phân bố dữ liệu của thuộc tính *avg_ crosses*

Có thể thấy rằng hầu hết các giá trị avg_crosses nằm trong khoảng 12.5 - 17.5, vì vậy sẽ điền các mục này bằng giá trị trung bình của avg_crosses đã đo được.

```
crosses_avg_team_avg = teams_stats['avg_crosses'].mean()

teams_stats.loc[(teams_stats['avg_crosses'].isnull()), '
    avg_crosses'] = crosses_avg_team_avg
```

4.1.4 Xử lí dữ liệu của *FIFA*

Chúng ta sẽ thu thập dữ liệu FIFA liên quan đến điểm tổng thể của các đội. Điều này sẽ tạo ra một số cột bao gồm điểm tổng thể của các cầu thủ thuộc về một đội. Như vậy, chúng ta có thể dễ dàng so sánh giá trị của mỗi đội hơn.

```
def get_fifa_stats(match, player_stats):
    ''' Aggregates fifa stats for a given match. '''

```

```
    #Define variables
    match_id =  match.match_api_id
    date = match['date']
    players = ['home_player_1', 'home_player_2', 'home_player_3', "home_player_4", "home_player_5",
               "home_player_6", "home_player_7", "home_player_8", "home_player_9", "home_player_10",
               "home_player_11", "away_player_1", "away_player_2", "away_player_3", "away_player_4",
               "away_player_5", "away_player_6", "away_player_7", "away_player_8", "away_player_9",
               "away_player_10", "away_player_11"]
    player_stats_new = pd.DataFrame()
    names = []

    #Loop through all players
    for player in players:

        #Get player ID
        player_id = match[player]

        #Get player stats
        stats = player_stats[player_stats.player_api_id == player_id]

        #Identify current stats
        current_stats = stats[stats.date < date].sort_values(by = 'date', ascending = False)[:1]

        if np.isnan(player_id) == True:
            overall_rating = pd.Series(0)
        else:
            current_stats.reset_index(inplace = True, drop = True)
            overall_rating = pd.Series(current_stats.loc[0, "overall_rating"])

        #Rename stat
        name = "{}_overall_rating".format(player)
        names.append(name)

        #Aggregate stats
        player_stats_new = pd.concat([player_stats_new, overall_rating], axis = 1)

    player_stats_new.columns = names
    player_stats_new['match_api_id'] = match_id

    player_stats_new.reset_index(inplace = True, drop = True)

```

```
    #Return player stats
    return player_stats_new.iloc[0]


def get_fifa_data(matches, player_stats, path = None, data_exists = False):
    ''' Gets fifa data for all matches. '''

    #Check if fifa data already exists
    if data_exists == True:
        fifa_data = pd.read_pickle(path)

    else:
        print("Collecting fifa data for each match...")
        start = time()

        #Apply get_fifa_stats for each match
        fifa_data = matches.apply(lambda x :get_fifa_stats(x, player_stats), axis = 1)

        end = time()
        print("Fifa data collected in {:.1f} minutes".format((end - start)/60))

    #Return fifa_data
    return fifa_data
```

```
fifa_data = get_fifa_data(viable_matches, player_attributes, None, data_exists = False)
```

4.1.5 Kết hợp các đặc điểm đầu vào

Trong trường hợp này, chúng ta cần kết hợp tất cả các đặc điểm, chọn đầu vào mà chúng ta sẽ truyền vào các mô hình và loại bỏ cột liên quan đến nhãn kết quả. Để cải thiện các chỉ số tổng thể của các mô hình học có giám sát, chúng ta cần chuẩn hóa các đặc điểm của mình trước khi huấn luyện các mô hình.

```
viables = get_features(viable_matches, teams_stats, fifa_data, 10, False)
inputs = viables.drop('match_api_id', axis=1)
outcomes = inputs.loc[:, 'outcome']
# all features except outcomes
features = inputs.drop('outcome', axis=1)
features.iloc[:,:] = Normalizer(norm='l1').fit_transform(features)
```

4.2 Tiến hành phân loại & Đánh giá kết quả

Cross-validation là một phương pháp được sử dụng để đánh giá hiệu suất của mô hình máy học và ước lượng khả năng tổng quát hóa của nó trên dữ liệu mới. Khi huấn luyện một mô hình máy học, chúng ta muốn đánh giá khả năng dự đoán của mô hình trên dữ liệu mà nó chưa từng thấy trước đó, để đảm bảo rằng mô hình không chỉ học thuộc lòng dữ liệu đào tạo mà có khả năng tổng quát hóa.

Vì tập dữ liệu sử dụng tương đối nhỏ cũng như mong muốn tận dụng tối đa dữ liệu huấn luyện có sẵn, nhóm quyết định sử dụng **K-fold Cross-validation**, trong đó dữ liệu được chia thành k tập con và mô hình được đánh giá trên tất cả các tập con này.

4.2.1 Một số hàm khác để hỗ trợ

Khi sử dụng phương pháp *Gradient Boosting*, chúng ta cần sử dụng hàm *convert_xgb_labels* để chuyển đổi cái giá trị ngõ ra trong tập huấn luyện thành các giá trị số nguyên. Sau đó để dự đoán dùng hàm *convert_xgb_y_predict* để chuyển đổi các giá trị ngõ ra từ dạng số nguyên thành các label ở dạng plain text

```
def convert_xgb_labels(labels):
    xgb_labels = labels.copy()
    for i in range(len(xgb_labels)):
        if xgb_labels[i] == 'Win':
            xgb_labels[i] = 0
        elif xgb_labels[i] == 'Draw':
            xgb_labels[i] = 1
        else:
            xgb_labels[i] = 2
    return xgb_labels
```

```
def convert_xgb_y_predict(y_predict):
    mapping_dict = {0: 'Win', 1: 'Draw', 2: 'Defeat'}
    y_predict = np.array([mapping_dict[value] for value in
   y_predict], dtype=object)
    return y_predict
```

4.2.2 Huấn luyện và đánh giá mô hình

4.2.3 Một số hàm hỗ trợ cho huấn luyện mô hình

Hàm *train_model* sẽ chia tập huấn luyện theo phương pháp KFold, sau đó training trên tập huấn luyện để ra được thông số của mô hình dự đoán. Sau khi mô hình được huấn luyện, tập kiểm tra sẽ được đưa vào để cho ra kết quả ngõ ra của tập kiểm tra

Sau khi nhận được kết quả ngõ ra của tập kiểm tra, chúng ta sẽ tiến hành đánh giá kết quả huấn luyện thông qua các giá trị *Accuracy*, *Recall*, *Precision* và *F1 score* từ việc so sánh hai tập là *ngõ ra dữ đoán* và *ngõ ra thực tế* của tập kiểm tra.

```python
def train_predict(clf, data, outcomes):
    print("Training a {} using a training set size of {}. . .".format(clf.__class__.__name__, len(data)))
    if clf.__class__.__name__ == 'XGBClassifier':
        xgb_outcomes = convert_xgb_labels(outcomes)
        y_predict = train_model(clf, data, xgb_outcomes)
        predict_metrics(outcomes, convert_xgb_y_predict(y_predict))
    else:
        y_predict = train_model(clf, data, outcomes)
        predict_metrics(outcomes, y_predict)
```

```python
def train_model(clf, data, labels):
    kf = KFold(n_splits=5)
    predictions = []
    for train, test in kf.split(data):
        X_train, X_test = data[data.index.isin(train)], data[data.index.isin(test)]
        y_train, y_test = labels[data.index.isin(train)], labels[data.index.isin(test)]
        clf.fit(X_train, y_train)
        predictions.append(clf.predict(X_test))

    y_predict = predictions[0]
    y_predict = np.append(y_predict, predictions[1], axis=0)
    y_predict = np.append(y_predict, predictions[2], axis=0)
    y_predict = np.append(y_predict, predictions[3], axis=0)
    y_predict = np.append(y_predict, predictions[4], axis=0)

    return y_predict
```

```python
def predict_metrics(y_test, y_predict):
    ls = ['Win', 'Draw', 'Defeat']
    from sklearn import metrics
    # cm = metrics.confusion_matrix(y_test, y_predict, ls)
    # disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=ls)
    # disp.plot(include_values=True, values_format='d')
    # plt.show()

    print(metrics.classification_report(y_test, y_predict, target_names=ls))
    print("Accuracy: ", metrics.accuracy_score(y_test, y_predict))
    print("Recall: ", metrics.recall_score(y_test, y_predict, average='macro'))
    print("Precision: ", metrics.precision_score(y_test, y_predict, average='macro', zero_division=0))
    print("F1 Score: ", metrics.f1_score(y_test, y_predict, average='macro'))
    print("-----------------------------------")
```

4.2.4 Tiến hành huấn luyện

Mô hình phân loại K-Nearest Neighbors

Mô hình phân loại K-Nearest Neighbors với số k láng giềng được khởi tạo là 90. Mô hình sẽ xem xét 90 láng giềng gần nhất để quyết định lớp của một điểm dữ liệu mới.

```
clf = KNeighborsClassifier(n_neighbors=90)
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.52	0.27	0.36	726
Draw	0.51	0.03	0.06	629
Defeat	0.51	0.92	0.66	1158
accuracy	-		0.51	2513
macro avg	0.51	0.41	0.36	2513
weighted avg	0.51	0.51	0.42	2535

Bảng 1: Đánh giá hiệu suất của mô hình phân loại KNN

Kết quả đánh giá

Accuracy: 0.5109430959013131

Recall: 0.4083588173804571

Precision: 0.5131985712541695

F1 score: 0.35772019895120993

Đánh giá mô hình KNN

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) đạt 51%, cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Tỉ lệ dự đoán chính xác cao cho trường hợp dự đoán Thua (Defeat) đạt 91%, thể hiện mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- F1-score cao cho trường hợp dự đoán thua (Defeat) đạt 0.66, cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw) có tỉ lệ 0.01, cho thấy mô hình gặp khó khăn trong việc xác định các trận hòa.
- Recall thấp cho dự đoán Hòa (Draw) và Thắng (Win) lần lượt là 0.01 và 0.3, thể hiện mô hình bỏ sót nhiều trận hòa và thắng trong dự đoán.

- F1-score thấp cho dự đoán Hòa (Draw) là 0.03, cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

Mô hình phân loại Decision Tree

random_state: Đây là tham số để thiết lập trạng thái ngẫu nhiên cho quá trình xây dựng cây quyết định. Bằng cách đặt giá trị này là 0, chúng ta cố định trạng thái ngẫu nhiên, điều này giúp việc tái tạo mô hình trở nên dễ dàng hơn.

criterion: Đây là tham số để chọn phương pháp đo lường chất lượng của việc phân chia trong cây quyết định. Tham số *Entropy* được sử dụng ở đây để đo lường mức độ không chắc chắn (uncertainty) trong dữ liệu.

splitter: Đây là tham số để chỉ định phương pháp chọn nút chia trong quá trình xây dựng cây. *random* được sử dụng ở đây để chọn một cách ngẫu nhiên nút chia, điều này có thể giúp mô hình tránh tình trạng quá mức đào tạo (overfitting) trong một số trường hợp.

max_depth: Đây là tham số để giới hạn độ sâu của cây quyết định. Ở đây, cây được giới hạn để có độ sâu tối đa là **5**, điều này giúp tránh quá mức đào tạo và làm cho cây quyết định trở nên dễ diễn giải hơn.

```
clf = DecisionTreeClassifier(random_state=0, criterion='entropy',
    splitter='random', max_depth=5)
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.50	0.40	0.44	726
Draw	0.28	0.01	0.02	629
Defeat	0.53	0.86	0.65	1163
accuracy	-	-	0.52	2513
macro avg	0.43	0.43	0.37	2513
weighted avg	0.46	0.52	0.44	2513

Bảng 2: Đánh giá hiệu suất của mô hình phân loại Decision Tree

Kết quả đánh giá

Accuracy: 0.5173099880620772

Recall: 0.4259888211565916

Precision: 0.4334725009998747

F1 score: 0.37438081721197386

Đánh giá mô hình Decision Tree

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) cao: 52% cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Chính xác cao cho dự đoán Thua (Defeat): 86% cho thấy mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- Điểm F1 cao cho dự đoán Thua (Defeat): 0.65 cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw): 0.01 cho thấy mô hình gặp khó khăn trong việc xác định các trận hòa.
- Recall thấp cho dự đoán Hòa (Draw) và Thắng (Win): 0.01 và 0.40 cho thấy mô hình bỏ sót nhiều trận hòa và thắng trong dự đoán.
- F1-score thấp cho dự đoán Hòa (Draw): 0.02 cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

Mô hình phân loại Random Forest

n_ estimators: Đây là tham số để thiết lập số lượng cây quyết định được sử dụng trong mô hình. Ở đây, chúng ta đặt số lượng cây là **100**.

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.50	0.45	0.48	726
Draw	0.25	0.06	0.10	629
Defeat	0.55	0.81	0.66	1158
accuracy	-	-	0.52	2513
macro avg	0.43	0.44	0.41	2513
weighted avg	0.46	0.52	0.47	2513

Bảng 3: Đánh giá hiệu suất của mô hình phân loại Random Forest

Kết quả đánh giá

Accuracy: 0.5216872264226025

Recall: 0.44309795650850203

Precision: 0.43450150500510687

F1 score: 0.41113073683574397

Đánh giá mô hình Random Forest

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) cao: 52% cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Chính xác cao cho dự đoán Thua (Defeat): 81% cho thấy mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- F1-score cao cho dự đoán Thua (Defeat): 0.66 cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.
- Hiệu suất dự đoán tốt hơn mô hình Decision Tree: So sánh với mô hình Decision Tree, mô hình Random Forest có accuracy, recall, và F1-score cao hơn cho cả Win và Defeat.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw): 0.25 cho thấy mô hình gặp khó khăn trong việc xác định các trận hòa.
- Recall thấp cho dự đoán Hòa (Draw): 0.06 cho thấy mô hình bỏ sót nhiều trận hòa trong dự đoán.
- F1-score thấp cho dự đoán Hòa (Draw): 0.10 cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

Mô hình phân loại Naive Bayes

var_ smoothing: Đây là tham số để điều chỉnh sự mượt mà của phân phối Gaussian. Tham số này được sử dụng để tránh việc chia cho 0 khi tính toán phân phối. Bằng cách đặt giá trị này là **1.1**, chúng ta thêm một lượng rất nhỏ vào phương sai của các đặc trưng, giúp tránh được việc chia cho 0 và tăng tính ổn định của mô hình.

```
clf = GaussianNB(var_smoothing=1.1)
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.47	0.50	0.48	726
Draw	0.33	0.05	0.09	629
Defeat	0.55	0.79	0.65	1158
accuracy	-		0.52	2513
macro avg	0.45	0.45	0.41	2513
weighted avg	0.47	0.52	0.46	2513

Bảng 4: Đánh giá hiệu suất của mô hình phân loại Naive Bayes

Kết quả đánh giá

Accuracy: 0.5196975726223637

Recall: 0.44600014736613885

Precision: 0.45117279653324777

F1 score: 0.4091482160001297

Đánh giá mô hình Naive Bayes

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) cao: 52% cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Chính xác cao cho dự đoán Thua (Defeat): 79% cho thấy mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- F1-score cao cho dự đoán Thua (Defeat): 0.65 cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw): 0.33 cho thấy mô hình gặp khó khăn trong việc xác định các trận hòa.
- Độ nhớ (recall) thấp cho dự đoán Hòa (Draw): 0.05 cho thấy mô hình bỏ sót nhiều trận hòa trong dự đoán.
- F1-score thấp cho dự đoán Hòa (Draw): 0.09 cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

Mô hình phân loại Gradient Boosting

max_depth: Đây là tham số để thiết lập độ sâu tối đa của các cây quyết định trong ensemble. Ở đây, chúng ta đặt độ sâu tối đa là **3** để tránh quá mức đào tạo và giảm độ phức tạp của mô hình.

learning_rate: Đây là tham số để điều chỉnh tốc độ học của mô hình. Giá trị này ảnh hưởng đến tốc độ học của mô hình trong quá trình huấn luyện. Một giá trị nhỏ hơn sẽ làm cho mô hình học chậm hơn nhưng có thể cải thiện tính tổng quát của mô hình.

n_estimators: Đây là tham số để thiết lập số lượng cây trong ensemble. Ở đây, chúng ta đặt số lượng cây là **100**, tức là sẽ có 100 cây quyết định được sử dụng trong ensemble để phân loại dữ liệu.

```
clf = XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators
    =100)
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.49	0.45	0.47	726
Draw	0.29	0.07	0.11	629
Defeat	0.55	0.80	0.66	1158
accuracy	-	-	0.52	2513
macro avg	0.44	0.44	0.41	2513
weighted avg	0.47	0.52	0.47	2513

Bảng 5: Đánh giá hiệu suất của mô hình phân loại Gradient Boosting

Kết quả đánh giá

Accuracy: 0.5189017111022682

Recall: 0.4421221494037432

Precision: 0.4415736264224635

F1 score: 0.41136897703362774

Đánh giá mô hình Gradient Boosting

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) cao: 52% cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Chính xác cao cho dự đoán Thua (Defeat): 80% cho thấy mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- F1-score cao cho dự đoán Thua (Defeat): 0.66 cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw): 0.29 cho thấy mô hình gặp khó khăn trong việc xác định các trận hòa.
- Độ nhớ (recall) thấp cho dự đoán Hòa (Draw): 0.07 cho thấy mô hình bỏ sót nhiều trận hòa trong dự đoán.
- F1-score thấp cho dự đoán Hòa (Draw): 0.11 cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

Mô hình phân loại Support Vector Machines

kernel: chọn kernel là *poly*, nghĩa là dữ liệu sẽ được biến đổi thông qua một hàm đa thức để tạo ra các đặc trưng mới để tạo ra đường biên phân chia phức tạp hơn.

coef0: là một tham số trong hàm đa thức, được sử dụng để cân chỉnh sự ảnh hưởng của các bậc cao của đa thức. Giá trị này có thể được điều chỉnh để kiểm soát mức độ phi tuyến tính của mô hình.

```
clf = SVC(coef0=5, kernel='poly')
train_predict(clf, features, outcomes)
```

	precision	recall	f1-score	support
Win	0.48	0.44	0.46	726
Draw	0.33	0.00	0.00	629
Defeat	0.54	0.86	0.66	1158
accuracy	-	-	0.52	2513
macro avg	0.45	0.43	0.37	2513
weighted avg	0.47	0.52	0.44	2513

Bảng 6: Đánh giá hiệu suất của mô hình phân loại SVM

Kết quả đánh giá

Accuracy: 0.5220851571826503

Recall: 0.4327156708775395

Precision: 0.44984407671786

F1 score: 0.3742415613205623

Đánh giá mô hình SVM

Điểm mạnh:

- Độ chính xác tổng thể (accuracy) cao: 52% cho thấy mô hình có khả năng dự đoán chính xác kết quả trận đấu ở mức trung bình khá.
- Chính xác cao cho dự đoán Thua (Defeat): 86% cho thấy mô hình có khả năng nhận diện các trận thua một cách hiệu quả.
- F1-score cao cho dự đoán Thua (Defeat): 0.66 cho thấy mô hình có sự cân bằng tốt giữa độ chính xác và độ nhạy cho việc dự đoán Thua.

Điểm yếu:

- Độ chính xác thấp cho dự đoán Hòa (Draw): 0.00 cho thấy mô hình hoàn toàn không dự đoán đúng bất kỳ trận hòa nào.
- Độ nhớ (recall) thấp cho dự đoán Hòa (Draw): 0.00 cho thấy mô hình bỏ sót tất cả các trận hòa trong dự đoán.
- F1-score thấp cho dự đoán Hòa (Draw): 0.00 cho thấy mô hình thiếu cân bằng giữa độ chính xác và độ nhạy cho việc dự đoán Hòa.

4.2.5 Nhận xét

Điểm chung về các mô hình huấn luyện

Cả 6 mô hình đều có độ chính xác tổng thể (accuracy) ở mức trung bình khá (khoảng 52%).

Các mô hình đều gặp khó khăn trong việc dự đoán các trận hòa, với độ chính xác cụ thể, độ hồi phục (recall), và điểm F1 score thấp.

Khả năng dự đoán Thua (Defeat) của các mô hình đều tốt, với độ chính xác cao và điểm F1 score cao.

Điểm riêng biệt của từng mô hình

K- Nearest Neighbors: Hiệu quả tốt cho dự đoán Thắng và Thua, nhưng gặp khó khăn trong việc dự đoán Draw.

Decision Tree: Hiệu quả tốt cho Thắng và Thua, nhưng gặp khó khăn trong việc dự đoán Draw.

Random Forest: Hiệu quả tốt cho Thắng và Thua, hiệu suất dự đoán tốt hơn Decision Tree, nhưng gặp khó khăn trong việc dự đoán Hòa.

Naive Bayes: Hiệu quả tốt cho Thắng và Thua, hiệu suất dự đoán tốt hơn Decision Tree, nhưng gặp khó khăn trong việc dự đoán Hòa.

Gradient Boosting: Hiệu quả tốt cho Thắng và Thua, hiệu suất dự đoán tốt hơn Decision Tree và Naive Bayes, nhưng gặp khó khăn trong việc dự đoán Hòa.

Support Vector Machines: Hiệu quả tốt cho Win và Defeat, điểm F1 cao nhất cho Defeat, nhưng hoàn toàn không thể dự đoán Draw.

Đề xuất cho việc lựa chọn mô hình để dự đoán

Việc lựa chọn mô hình phù hợp phụ thuộc vào mục tiêu cụ thể và yêu cầu của việc dự đoán:

- Nếu mục tiêu chính là dự đoán chính xác các trận thua (Defeat): Nên chọn SVM, Gradient Boosting, hoặc Naive Bayes.
- Nếu mục tiêu là dự đoán chính xác cả Win và Defeat: Nên chọn Random Forest hoặc Gradient Boosting.
- Nếu cần cân bằng giữa độ chính xác và đơn giản: Nên chọn KNN hoặc Decision Tree.

5 Tổng kết

Sau khi phân tích dữ liệu và tiến hành huấn luyện, kết quả đạt được của từng phương pháp khá tốt với độ chính xác tổng thể (accuracy) đều 51% (đặc biệt hữu ích trong việc cá cược khi chúng ta chỉ cần số trận dự đoán đúng nhiều hơn số trận dự đoán thua). Tuy nhiên, dữ liệu sử dụng cho việc nghiên cứu đề tài chứa nhiễu (các dữ liệu không cần thiết) và không cân bằng, do đó, nhóm còn xét đến các thông số như độ hồi phục (recall), độ chính xác (precision) và "F1 score".

Đối với độ chính xác tổng thể (accuracy), Support Vector Machines là phương pháp cho kết quả tốt nhất với tỷ lệ chính xác tổng thể 52.21%. Tuy nhiên, việc dự đoán kết quả bóng đá thường tập trung vào độ chính xác cụ thể cho lựa chọn đội thắng (hoặc hòa, thua), do đó chỉ số độ chính xác cụ thể (precision) được đề cao hơn cả, và đối với tiêu chí này, K - Nearest Neighbors là phương pháp cho kết quả tốt nhất với 51.32%. Ngoài ra, K – Nearest Neighbors cũng cho độ chính xác tổng thể dự đoán 51.09%, là kết quả tương đối tốt. Cân nhắc các chỉ số trên cùng với độ hồi phục (recall) và "F1 score", K – Nearest Neighbors là phương pháp cho kết quả tốt nhất để dự đoán kết quả trận đấu bóng đá.

Mặc dù kết quả thu được khá tốt, tuy nhiên đề tài nghiên cứu vẫn còn cần phải cải thiện hơn. Một số hướng phát triển mà nhóm đề xuất như thực hiện huấn luyện với các phương pháp khác như Neural Networks, Deep Neural Networks khi tập dữ liệu khá nhiều tiêu chí kết hợp, hoặc là thu thập thêm nhiều tiêu chí quan trọng hơn như là Huấn luận viên đội bóng, sơ đồ đội hình, danh tiếng cầu thủ, thời tiết, ... để bổ sung cho việc huấn luyện Học máy và dự đoán.

Bên cạnh đó, qua quá trình huấn luyện tập dữ liệu, nhóm cũng đã có được cái nhìn cụ thể về điểm mạnh, điểm yếu của từng phương pháp Học máy và đưa ra những đề xuất chọn lọc phương pháp phù hợp cho việc dự đoán kết quả bóng đá, tùy thuộc vào mục tiêu, tiêu chí hay yêu cầu cụ thể nào đó.

Tóm lại, việc dự đoán kết quả cho các trận đấu bóng đá thực sự là một việc rất khó, và kết quả thực tế nhiều khi lại rất bất ngờ. Do đó, trong quy mô môn học cũng như theo sở thích cá nhân, nhóm đã rất nỗ lực trong việc phân tích dữ liệu và xây dựng các mô hình dự đoán kết quả bóng đá với nhiều phương pháp khác nhau. Dù cho còn nhiều hạn chế về số lượng tiêu chí đặt ra cho thống kê và dự đoán cũng như số lượng phương pháp sử dụng, kết quả nhóm đạt được vẫn tương đối tốt. Cùng với các định hướng phát triển cho đề tài mà nhóm đã đặt ra, rất mong kết quả cho các nghiên cứu sau ngày càng chính xác hơn với thực tế.

6 Tài liệu tham khảo

Tài liệu

[1] All Football App, 09/05/2022, *Real Madrid vs Man City: Champions League start time, odds, prediction*, truy cập từ https://m.allfootballapp.com/news/Serie-A/Real-Madrid-vs-Man-City-Champions-League-start-time-odds-prediction/3093111

[2] Electronic Arts, 03/06/2010, *EA SPORTS Predicts Spain Will Win the 2010 FIFA World Cup*, truy cập từ https://www.ea.com/news/ea-sports-predicts-spain-win-the-2010-fifa-world-cup

[3] Electronic Arts, 04/06/2014, *HISTORY TO BE MADE AT THE 2014 FIFA WORLD CUP*, truy cập từ https://www.ea.com/games/fifa/news/ea-sports-2014-fifa-world-cup-prediction

[4] Electronic Arts, 29/05/2018, *EA SPORTS PREDICTS FRANCE TO WIN 2018 FIFA WORLD CUP™*, truy cập từ https://www.ea.com/games/fifa/news/ea-sports-predicts-world-cup-fifa-18

[5] Electronic Arts, 08/11/2022, *EA SPORTS™ PREDICTS ARGENTINA TO WIN THE FIFA WORLD CUP 2022™*, truy cập từ https://www.ea.com/games/fifa/fifa-23/news/ea-sports-fifa-world-cup-22-prediction

[6] Electronic Arts, *HYPERMOTION2 TECHNOLOGY ULTRA-REALISTIC GAMEPLAY*, truy cập từ https://www.ea.com/games/fifa/fifa-23/hypermotion2

[7] kaggle, 2017, *European Soccer Database*, truy cập từ https://www.kaggle.com/datasets/hugomathien/soccer/data

[8] Data Carpentry, 18/05/2023, *Accessing data stored in SQLite using Python and Pandas*, truy cập từ https://datacarpentry.org/python-ecology-lesson/instructor/09-working-with-sql.html#accessing-data-stored-in-sqlite-using-python-and-pandas