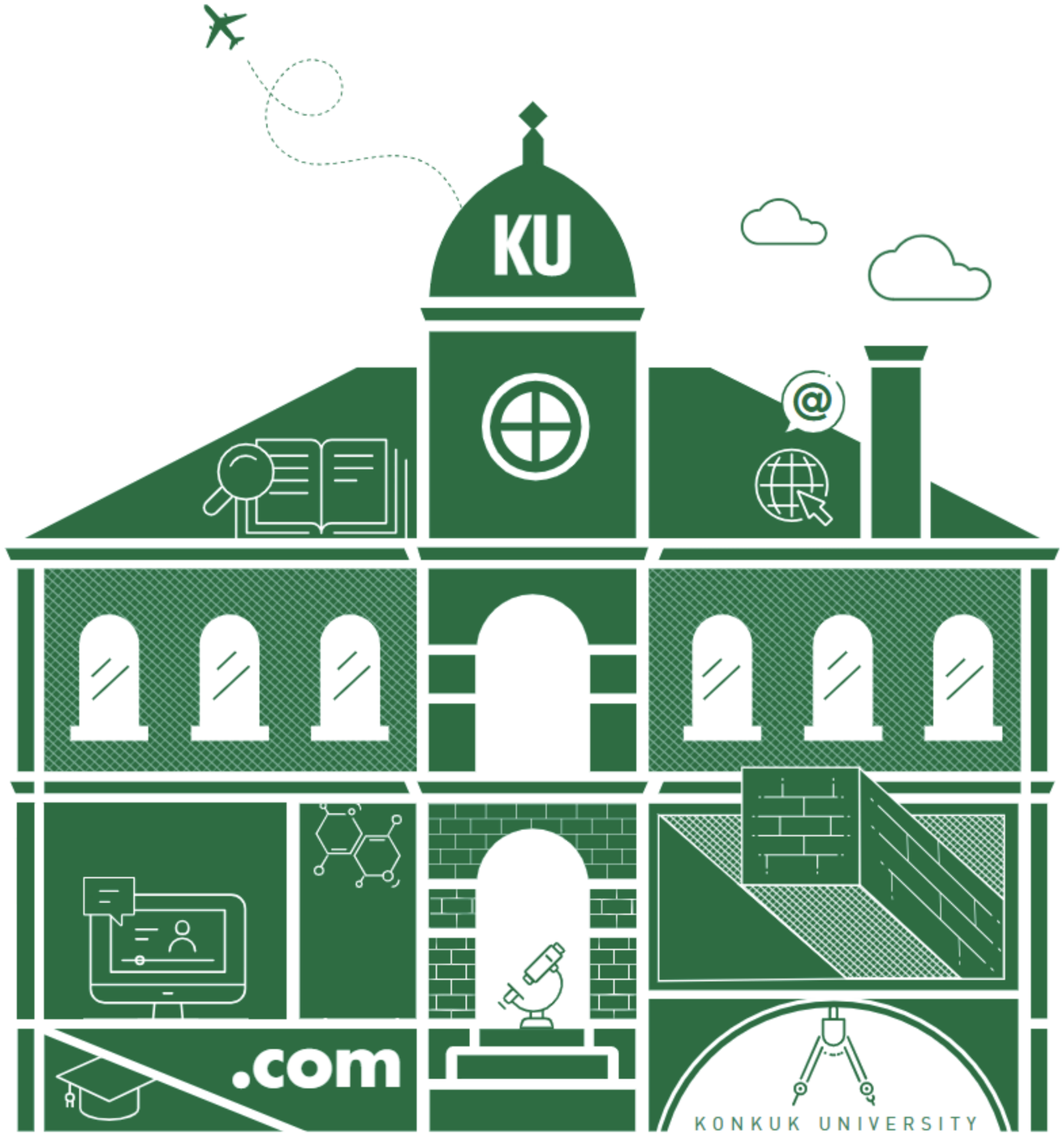


KUGGLE 2차 프로젝트

11기 배지원
11기 지명근
12기 박슬옹
12기 황천조
12기 천시원

K
U

CONTENTS

KONKUK

데이터 소개

1. 데이터 소개

database.sqlite (313.09 MB)

About this file

Suggest Edits

This file does not have a description yet.

Table	Total Rows	Total Columns
Country	11	2
League	11	3
Match	25979	115
Player	11060	7
Player_Attributes	183978	42
Team	299	5
Team_Attributes	1458	25

- 25,000경기 이상의 경기 데이터
- 10,000명 이상의 선수 데이터
- 11개 유럽 국가의 주요 리그 챔피언십 정보
- 2008~2016 시즌 데이터 포함
- 선수 및 팀의 속성(능력치)은 EA Sports의 FIFA 비디오 게임 시리즈에서 주간 업데이트 정보를 기반으로 수집됨
- 팀 라인업 및 스쿼드 포메이션(선수별 X, Y 좌표 포함)
- 10,000경기 이상의 상세 경기 이벤트(득점 유형, 점유율, 코너킥, 크로스, 파울, 카드 등) 기록
- FIFA에서 추출한 팀 속성 데이터가 포함
- 7개의 테이블, 199개의 컬럼

데이터 전처리

2. 데이터 전처리 – 경기 결과 컬럼 생성

```python
# Match 테이블 불러오기
# 변수가 100개가 넘어서 모델링에 사용할 변수만 선택
query = """
SELECT
    match_api_id, date, season, stage,
    home_team_api_id, away_team_api_id,
    home_team_goal, away_team_goal,
    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
FROM Match
WHERE home_player_1 IS NOT NULL AND away_player_1 IS NOT NULL
"""
match_df = pd.read_sql(query, conn)
```

- Match 테이블에서 모델링에 사용할 주요 변수(컬럼)만 선택
- 선수 정보가 없는 경기(결측치)는 제외

```python
# 경기 결과를 Win, Defeat, Draw로 반환하는 함수 생성
def get_result(row):
    if row['home_team_goal'] > row['away_team_goal']:
        return 'Win'
    elif row['home_team_goal'] < row['away_team_goal']:
        return 'Defeat'
    else:
        return 'Draw'

match_df['result'] = match_df.apply(get_result, axis=1)

```

- 각 경기의 결과를 'Win', 'Defeat', 'Draw’로 분류
- 홈팀과 어웨이팀의 득점을 비교하여 결과를 새로운 컬럼으로 추가 -> 타깃 변수

2. 팀 평균 능력치 계산

```python
player_attrs = pd.read_sql("""
    SELECT player_api_id, date, overall_rating, potential, stamina
    FROM Player_Attributes
    WHERE overall_rating IS NOT NULL
""", conn)

# 날짜 포맷 정리
player_attrs['date'] = pd.to_datetime(player_attrs['date'])
match_df['date'] = pd.to_datetime(match_df['date'])

# 선수별 최신 능력치만 가져오기
latest_attrs = (
    player_attrs.sort_values(['player_api_id', 'date'])
    .drop_duplicates(subset=['player_api_id'], keep='last')
    .set_index('player_api_id')
)[['overall_rating', 'potential', 'stamina']]
```

```python
home_players = [f'home_player_{i}' for i in range(1, 12)]
away_players = [f'away_player_{i}' for i in range(1, 12)]

# 홈팀 선수 능력치
for col in home_players:
    match_df = match_df.merge(
        latest_attrs[['overall_rating']].rename(columns={'overall_rating': f'ovr_{col}'}),
        left_on=col,
        right_index=True,
        how='left'
    )

# 어웨이팀 선수 능력치
for col in away_players:
    match_df = match_df.merge(
        latest_attrs[['overall_rating']].rename(columns={'overall_rating': f'ovr_{col}'}),
        left_on=col,
        right_index=True,
        how='left'
    )
```

```python
# 각 선수의 최신 능력치 불러온 후, 각 경기의 홈/어웨이 팀 선수 11명의 능력치를 병합 (평균)
match_df['home_avg_rating'] = match_df[[f'ovr_home_player_{i}' for i in range(1, 12)]].mean(axis=1)
match_df['away_avg_rating'] = match_df[[f'ovr_away_player_{i}' for i in range(1, 12)]].mean(axis=1)
```

- 각 경기의 홈팀, 어웨이팀 11명 선수별로 최신 능력치(평점)를 병합, 각 선수의 능력치를 경기 데이터에 추가
- 각 경기의 홈팀, 어웨이팀 선수 11명의 평균 평점을 계산하여 새로운 컬럼으로 저장

2. 팀 전략 데이터 병합 및 전처리

```sql
team_attrs = pd.read_sql("""
    SELECT *
    FROM Team_Attributes
    WHERE buildUpPlaySpeed IS NOT NULL
""", conn)

# 날짜 처리
team_attrs['date'] = pd.to_datetime(team_attrs['date'])
match_df['date'] = pd.to_datetime(match_df['date'])
```

```sql
# 전략에서 사용할 변수만 선택
cols_to_use = [
    'team_api_id', 'date',
    'buildUpPlaySpeed', 'chanceCreationPassing', 'defencePressure'
]
team_attrs_sub = team_attrs[cols_to_use].copy()
team_attrs_sub['date'] = pd.to_datetime(team_attrs_sub['date'])

# 홈팀 전략 병합
match_with_home = get_team_attributes(match_df, team_attrs_sub, team_col_prefix='home')

# 어웨이팀 전략 병합
match_with_all = get_team_attributes(match_with_home, team_attrs_sub, team_col_prefix='away')
```

```sql
def get_latest_team_attr(team_id, match_date, team_attrs):
    """
    특정 팀의 경기일 이전 전략 정보 중 가장 최근 값을 반환
    """
    past_attrs = team_attrs[
        (team_attrs['team_api_id'] == team_id) &
        (team_attrs['date'] <= match_date)
    ]
    if not past_attrs.empty:
        return past_attrs.sort_values('date').iloc[-1]
    else:
        return pd.Series()

def get_team_attributes(match_df, team_attrs, team_col_prefix):
    """
    홈팀 또는 어웨이팀 전략을 match_df에 붙임
    """
    features = []

    for _, row in match_df.iterrows():
        team_id = row[f"{team_col_prefix}_team_api_id"]
        match_date = row['date']
        team_attr = get_latest_team_attr(team_id, match_date, team_attrs)
        features.append(team_attr)

    features_df = pd.DataFrame(features)
    features_df = features_df.add_prefix(f"{team_col_prefix}_")

    return pd.concat([match_df.reset_index(drop=True), features_df.reset_index(drop=True)], axis=1)

```

- 팀 전략 정보(빌드업 속도, 패스, 수비 압박 등)를 불러옴
- 날짜를 기준으로, 각 경기 시점에 가장 최근의 팀 전략 정보를 병합

2. 데이터 전처리 – 결측치 제거

```python
# 모델에 사용할 변수만 추린 후 데이터 프레임 생성
selected_columns = [
    'home_avg_rating', 'away_avg_rating',
    'home_buildUpPlaySpeed', 'away_buildUpPlaySpeed',
    'home_chanceCreationPassing', 'away_chanceCreationPassing',
    'home_defencePressure', 'away_defencePressure',
    'result'
]

final_df = match_with_all[selected_columns].copy()
```

- 모델에 사용할 주요 변수만 추려서 최종 데이터프레임을 만듬
- 결측치 -> 제거

```python
# 결측치 확인
print(final_df.isnull().sum())
```

```
home_avg_rating              0
away_avg_rating              0
home_buildUpPlaySpeed     4977
away_buildUpPlaySpeed     4962
home_chanceCreationPassing    4977
away_chanceCreationPassing    4962
home_defencePressure         4977
away_defencePressure         4962
result                       0
dtype: int64
```

```python
final_df = final_df.dropna()
print(final_df.isnull().sum())
```

```
home_avg_rating              0
away_avg_rating              0
home_buildUpPlaySpeed        0
away_buildUpPlaySpeed        0
home_chanceCreationPassing    0
away_chanceCreationPassing    0
home_defencePressure         0
away_defencePressure         0
result                       0
dtype: int64
```

Cat boost 모델링 & 하이퍼 파라미터 튜닝

4. 모델링

적절한 모델 탐색

1.LightGBM - 특징 : 대용량 데이터에서 빠르고 효율적으로 동작하는 트리 기반 부스팅 알고리즘, leaf-wise 트리 성장, GOSS, EFB 등으로 높은 예측 성능과 학습 속도를 제공. 범주형 변수와 결측치를 자동 처리하지만, 작은 데이터에서는 과적합에 주의

2. XGBoost - 특징 : 부스팅 알고리즘을 사용하여 변수 중요도를 시각화하고 해석하기 유리 - 장점 : 변수 상호작용이 있는 복잡한 데이터에도 강함.

3. CatBoost - 특징 : 범주형 변수(categorical feature) 처리가 강력한 것이 특징 별도의 복잡한 전처리 없이 범주형 변수를 자동으로 처리하고, 데이터 누수와 오버피팅을 효과적으로 방지하는 알고리즘(Ordered Target Encoding, Ordered Boosting)을 갖춤

Cat boost 모델링 & 하이퍼 파라미터 튜닝

```python
# 라이브러리 불러오기
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Draw vs NotDraw (이진 분류용 데이터 준비)

# 타겟: 'Draw'이면 1, 아니면 0
binary_y = (final_df['result'] == 'Draw').astype(int)

# 피처에서 결과 변수 제거
X_binary = final_df.drop(columns=['result']).values

# 학습/테스트 셋 분할
X_binary_train, X_binary_test, y_binary_train, y_binary_test = train_test_split(
    X_binary, binary_y, test_size=0.2, stratify=binary_y, random_state=42
)
```

```python
# CatBoost 이진 분류기 학습 (Draw 여부 판단)
from catboost import CatBoostClassifier

draw_clf = CatBoostClassifier(
    iterations=300,
    depth=6,
    learning_rate=0.1,
    loss_function='Logloss',
    eval_metric='F1',
    verbose=100,
    random_seed=42
)

draw_clf.fit(X_binary_train, y_binary_train)
```

Cat boost 모델링 & 하이퍼 파라미터 튜닝

```python
# 다중 분류용 데이터 준비

# 클래스 인코딩 (Win, Draw, Defeat → 숫자)
le = LabelEncoder()
y_multi = le.fit_transform(final_df['result'])

# 학습/테스트 셋 분할
X_train, X_test, y_train, y_test = train_test_split(
    X_binary, y_multi, test_size=0.2, stratify=y_multi, random_state=42
)

# CatBoost 다중 분류기 학습 (전체 경기 결과 예측)
cat_model = CatBoostClassifier(
    iterations=300,
    depth=6,
    learning_rate=0.1,
    loss_function='MultiClass',
    eval_metric='TotalF1',
    verbose=100,
    random_seed=42
)

cat_model.fit(X_train, y_train)
```

```python
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Draw 이진 분류기 확률 예측
draw_probs = draw_clf.predict_proba(X_test)[:, 1]

# 다중 클래스 예측 결과
multi_probs = cat_model.predict_proba(X_test)
y_pred_multi = np.argmax(multi_probs, axis=1)

# 조합: Draw 확률이 일정 기준이상이면 무조건 Draw로 수정
# 코드의 0.28 부분 수치를 조정하면서 하이퍼파라미터 튜닝 진행하였습니다!
y_combined = y_pred_multi.copy()
y_combined[draw_probs > 0.28] = 1  # 1 = Draw
```

K4

성능평가

4. 성능평가

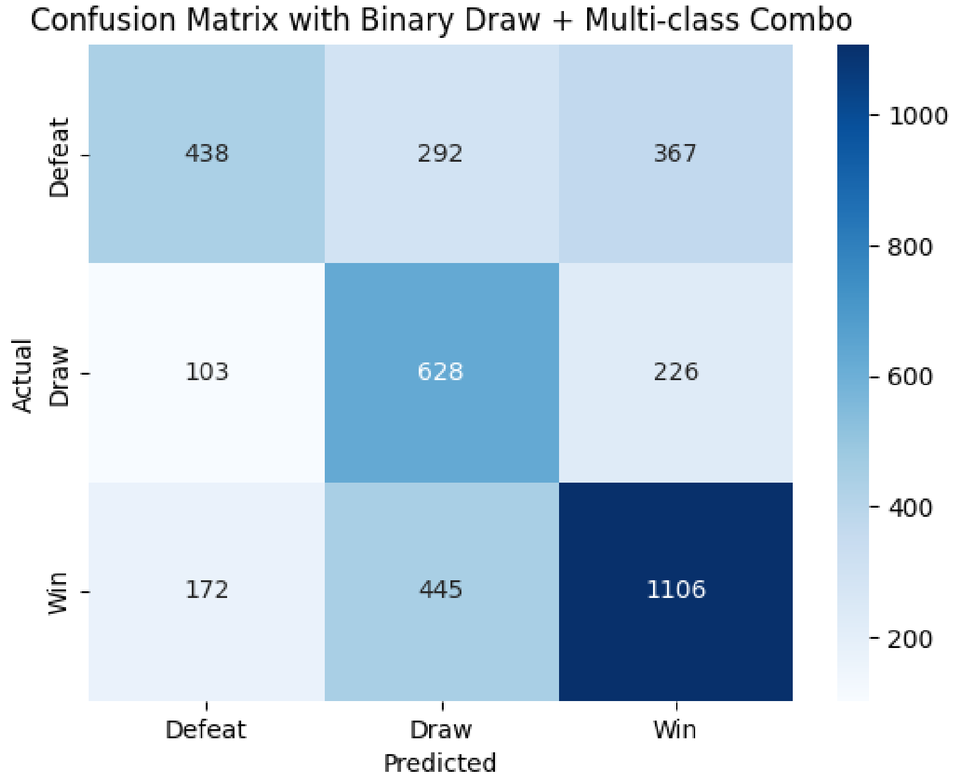
```python
# 평가 및 시각화
print("Classification Report (Combined):")
print(classification_report(y_test, y_combined, target_names=le.classes_))

# 혼동 행렬 시각화
cm = confusion_matrix(y_test, y_combined)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix with Binary Draw + Multi-class Combo')
plt.show()
```

```
Classification Report (Combined):
              precision    recall  f1-score   support

      Defeat       0.61      0.40      0.48      1097
        Draw       0.46      0.66      0.54       957
         Win       0.65      0.64      0.65      1723

    accuracy                           0.58      3777
   macro avg       0.58      0.57      0.56      3777
weighted avg       0.59      0.58      0.57      3777
```



- 델의 전체 정확도는 58%로, 3개 클래스를 모두 예측하는 다중 분류 문제에서 무작위(33%)보다는 훨씬 높지만, 완벽하진 않음
- 모델이 Win(승) 예측에는 강점이 있으나, Defeat(패)와 Draw(무승부) 간 혼동이 존재
- 특히 무승부 예측에서 precision이 낮으므로, 무승부를 더 정확하게 구분할 수 있는 추가 변수나 모델 개선이 필요

지금까지 발표를 들어주셔서

감사합니다