# DSTA

### Rating and Ranking with Markov's Method:

### The Premier League case

This is the **solution** notebook.

We will analyse Premier League results for these two interesting seasons; results have been downloaded from www.footballwebpages.co.uk:

- the 2021 - 2022 season, and
- 2022 - 2023 season.

### Import necessary Python packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

### Set Pandas and Numpy options for printing results

```
np.set_printoptions(linewidth=1000)

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.colheader_justify', 'center')
```

**Premier League winners:**

- **2021 - 2022**: Manchester City (1-point gap from Liverpool that finished second)
- **2022 - 2023**: Manchester City (5-points gap from Arsenal that finished second)

**File names**

```
# League table files
premier_league_table_2017_2018 = "./data/2017_2018_LeagueTable.csv"
premier_league_table_2018_2019 = "./data/2018_2019_LeagueTable.csv"
premier_league_table_2019_2020 = "./data/2019_2020_LeagueTable.csv"
premier_league_table_2020_2021 = "./data/2020_2021_LeagueTable.csv"
premier_league_table_2021_2022 = "./data/2021_2022_LeagueTable.csv"
premier_league_table_2022_2023 = "./data/2022_2023_LeagueTable.csv"

# Match grid files
premier_league_match_grid_2017_2018 = "./data/2017_2018_MatchGrid.csv"
premier_league_match_grid_2018_2019 = "./data/2018_2019_MatchGrid.csv"
premier_league_match_grid_2019_2020 = "./data/2017_2018_MatchGrid.csv"
premier_league_match_grid_2020_2021 = "./data/2020_2021_MatchGrid.csv"
premier_league_match_grid_2021_2022 = "./data/2021_2022_MatchGrid.csv"
premier_league_match_grid_2022_2023 = "./data/2022_2023_MatchGrid.csv"
```

**Set current working data files and next season files**

Hint: Change these variables in case you would like to rate / rank teams based on a different season and check the estimates against the actual rankings of the following season.

```
# Current (working) season
current_league_table_file = premier_league_table_2021_2022

current_match_grid_file = premier_league_match_grid_2021_2022

current_season = "2021 - 2022"

# Merged results of current season from Massey and Keener
merged_results_2021_2022 = "./data/2021_2022_MergedResults.csv"

# Next season
```

```
coming_league_table_file = premier_league_table_2022_2023

coming_match_grid_file = premier_league_match_grid_2022_2023

coming_season = "2022 - 2023"
```

### Markov's method

### For Markov we need again the match grid

```
# Each match entry is in the format ="GH-GA" (except from NaN in diagonal).
# GH are goals scored by the home team, and GA are goals scored by the away team
# Below, we read the match grid CSV and remove '=' and '"'

match_grid = (
    pd.read_csv(current_match_grid_file, dtype=str, index_col=0)
    .replace('"' , '', regex=True)
    .replace('=' , '', regex=True)
    .fillna("0-0")
    )

match_grid
```

### Create Markov's V matrix

Below is a refresher

$V_{n\times n}$ where $V_{ij}$ : Total goals conceded from team $i$ against team $j$

Here $n$ represents the number of teams in the league

### Create Markov's S matrix

Below is a refresher

$S_{n\times n}$ where $S_{ij}$ : Total goals team $i$ conceded from team $j$, divided by the total goals team $i$ conceded.

Again, $n$ here represents the number of teams in the league

### Exercise 1: Complete the code to calculate Markov's V and S matrices

**Step-by-step:**

1. Parse scores. Example: "3-2". The home team scored 3 goals and the away team 2.

Hint: Pandas applymap documentation

2. Match every team's home match with the respective away match against the same opponent.

Hint: The home match of team $i$ against $j$ is element $ij$. The respective away match is element $ji$ - row and column indexes are swapped...

```
# Parse score and get goals conceded at home
home_goals_ij = lambda score: int(score.split("-")[1])
all_home_goals_ij = match_grid.applymap(home_goals_ij)

# Parse score and get goals conceded away
# The grid is transposed to match every team's respective
# home and away matches
away_goals_ij = lambda score: int(score.split("-")[0])
all_away_goals_ij = match_grid.T.applymap(away_goals_ij)

# Sum goals conceded
V_dataframe = all_home_goals_ij + all_away_goals_ij
```

```
# row_sums: Sum of goals each team conceded
row_sums = V_dataframe.sum(axis=1)

# Create S matrix
S_dataframe = V_dataframe.div(row_sums, axis=0)
```

**Create transition and counter dictionaries**

```
# Dictionary with teams as keys and lists of probabilities as values
# Each list represents a probability of moving from current team
# to another team of the league (fair-weather fan logic)
transit_dict = S_dataframe.T.to_dict(orient = "list")

teams = S_dataframe.columns.tolist()

# Dictionary with teams as keys and number of visits as values
counter_dict = {team: 0 for team in teams}
```

**Run Markov simulation with fair-weather fan**

```
N = 100_000

# Initialize process by randomly selecting a team
curr_team = np.random.choice(teams)
counter_dict[curr_team] += 1

# Run the simulation
for i in range(N):
    probs = transit_dict[curr_team]
    curr_team = np.random.choice(teams, p = probs)
    counter_dict[curr_team] += 1

# Get the ratings
ratings = [count / (N + 1) for count in counter_dict.values()]
markov_df = (
    pd.DataFrame(ratings, index = teams, columns=["Markov_Rating"])
    .sort_values(by="Markov_Rating", ascending=False)
    )
```

**Use a MinMaxScaler to scale Markov ratings between 0 and 100 for plotting.**

Please see the relative sklearn MinMaxScaler documentation.

```
# Scale the ratings between 100 (top team) and 0 (weakest team).
# MinMaxScaler accepts a tuple (min, max) as input argument to define the range.
min_max_scaler = MinMaxScaler((0, 100))
markov_df["Markov_Scaled_Rating"] = min_max_scaler.fit_transform(
    markov_df.loc[:, "Markov_Rating"].values.reshape(-1, 1)
    )
```

**Add Markov ranking.**

```
# Add Markov ranking
markov_df["Markov_Ranking"] = np.arange(1, 21)

markov_df
```

**Add Markov results to the match grid table**

```
match_grid = match_grid.join(markov_df)
match_grid
```

**Import the league table to get actual rankings and points scored**

```
# Read the league table data - skip the first row
league_table = pd.read_csv(current_league_table_file, skiprows = 1)

league_table["Actual_Ranking"] = np.arange(1, 21)

league_table
```

**We keep only teams, actual ranking and points.**

```
required_cols = ["Unnamed: 1", "Pts", "Actual_Ranking"]

renaming = {"Unnamed: 1": "Teams", "Pts": "Points"}

# Make a copy of the league table, keeping only the necessary columns renamed
# Index is reset as the teams for the table join below
league_table = (
    league_table
    .loc[:, required_cols]
    .copy()
    .rename(columns=renaming)
    .set_index("Teams")
)

league_table
```

**Join the match grid that holds Markov ratings with the league table and the actual ratings based on team names**

```
match_grid = match_grid.join(league_table)

match_grid
```

### Keep Markov rating and ranking from the match grid

```
cols_to_keep = [
    "Markov_Rating",
    "Markov_Scaled_Rating",
    "Markov_Ranking"
    ]

# Data needed from Markov output - sort by actual ranking first
data_to_keep = (
    match_grid
    .sort_values("Actual_Ranking", ascending = True)
    .loc[:, cols_to_keep]
    .copy()
    )
```

### Import merged data with Massey and Keener results

```
# Use Teams column as index to join it later with Markov
merged_results = pd.read_csv(merged_results_2021_2022, index_col = "Teams")
```

### Merge Markov results with Massey and Keener results

```
# Merge the data
merged_results = merged_results.join(data_to_keep)

merged_results
```

### Plot Markov's scaled rating and ranking side by side with actual ranking and points scored

Documentation for matplotlib.pyplot horizontal bar plots

```
# Initialize grid of plots
figure, axis = plt.subplots(nrows = 1, ncols = 2, figsize = (12, 4), dpi = 160)

# Plot Keener scaled rating - plot 0, row 0
axis[0].barh(
    match_grid["Markov_Ranking"],
    match_grid["Markov_Scaled_Rating"],
```

```
    height = 0.6, align = 'center'
    )

# Configure y axis
axis[0].set_yticks(
    match_grid["Markov_Ranking"],
    labels = match_grid.index,
    fontsize = 7
    )

axis[0].invert_yaxis()  # labels read top-to-bottom

# X-axis and title
axis[0].tick_params(axis = "x", labelsize = 6)
axis[0].set_xlabel('Markov Scaled Rating', fontsize = 8)
axis[0].set_title(f'Season {current_season} Markov Scaled Rating', fontsize = 8)

# Plot actual ranking and point scored - plot 1, row 0
axis[1].barh(
    match_grid["Actual_Ranking"],
    match_grid["Points"],
    height = 0.6, align = 'center'
    )

# Configure y axis
axis[1].set_yticks(
    match_grid["Actual_Ranking"],
    labels = match_grid.index,
    fontsize = 7
    )
axis[1].invert_yaxis()  # labels read top-to-bottom

# X-axis and title
axis[1].tick_params(axis = "x", labelsize = 6)
axis[1].set_xlabel('Actual Points', fontsize = 8)
axis[1].set_title(f'Season {current_season} Points Scored', fontsize = 8)

# Use 'tight_layout' to avoid overlapping text
plt.tight_layout()
plt.show()
```

**Get rankings from all methods in a new table**

```
rankings = [
    "Actual_Ranking",
    "Massey_Ranking",
    "Keener_Ranking",
    "Markov_Ranking"
    ]

ranks_df = merged_results.loc[:, rankings].copy()
ranks_df
```

```
ranks_df.corr()
```

**Import the table of the subsequent season to check**

```
# Read the league table data - skip the first row
next_league_table = pd.read_csv(coming_league_table_file, skiprows = 1)

next_league_table["Actual_Ranking"] = np.arange(1, 21)

# Uncomment if you want to see the raw table
# league_table
```

**Keep necessary columns and rename them**

```
required_cols = ["Unnamed: 1", "P.2", "W.2", "D.2", "L.2", "F",
                 "A", "+/-", "Pts", "Actual_Ranking"]

renaming = {
    "Unnamed: 1": "Teams",
    "P.2": "Total_Matches_Played",
    "W.2": "Total_Wins",
    "D.2": "Total_Draws",
    "L.2": "Total_Losses",
    "F": "Goals_Scored",
    "A": "Goals_Conceded",
    "+/-": "Goal_Difference",
    "Pts": "Points"
```

```
    }

# Make a copy of the league table, keeping only the necessary columns renamed
next_league_table = (
    next_league_table
    .loc[:, required_cols]
    .copy()
    .rename(columns = renaming)
)

next_league_table
```

**Recall estimated rankings from Massey, Keener and Markov**

```
ranks_df
```