Project 4

EN.500.133 Bootcamp: Python, Fall 2023

January 11, 2024

1 The Problem

Inspired by Jonah Hill's character in the movie *Moneyball*, you have decided to take up sports analytics as your retirement activity, following a successful career selling widgets in the mall on Namek. Eight players are competing to become the Go Fish Champion of Namek. Figure 1 shows the draw for a single-elimination tournament that will decide the champion. Single-elimination means the loser of each matchup exits the tournament. Your task in this project is to predict the most likely winner of the entire tournament. You will make use of Arpad Elo's rating scheme (see https://en.wikipedia.org/wiki/Elo_rating_system for a description. This system is also how international chess ratings are derived and form the basis of a lot of sports prediction models, like those used by FiveThirtyEight) to pick the likely winner. Here are some major assumptions involved.

- Each player is assumed to have a certain skill level (given by a number often denoted by s). This quantity is unknown and cannot be directly measured. Below, we will learn to infer this value from historical Go Fish match data.
- We assume that a player's skill level does not change over time.
- When two players A and B with skill levels s_a and s_b, respectively, compete, the probability of player A winning is impacted by the difference in skill levels defined as $\delta = (s_a - s_b)/c$. The value c in the expression is a scale parameter: it describes how sensitive the win-loss probability is to the difference in skill levels. For our purposes, we'll select c = 100 for this entire assignment. A formula which expresses the probability that player A wins is given below:

$$P(A \quad wins) = \frac{e^{\delta}}{1 + e^{\delta}} \tag{1}$$

- Every match played has exactly one winner; the outcome can never be a tie.

Equation 1 tells you that, if two players with equal skill levels meet, we have $\delta = 0$, so they are both equally likely to win the game.

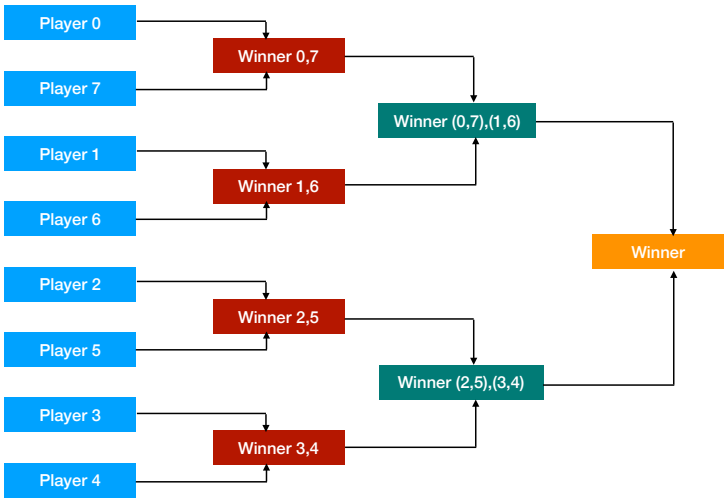

Figure 1: Tournament bracket. Only the winner of each match proceeds to the next stage.

1.1 Step One

Your first task is to obtain a rating for each player, which ideally matches their skill level. The file named `past_matches.csv` has a collection of results from prior games played between the 8 players who will be participating in the tournament. For simplicity, we use a fixed set of integer names for players: the list of player names is *always* 0,1,2,3,4,5,6,7. The file has 4 columns (excluding the index column). In each row, the data in first two columns gives the names of the two players who took part in the match. The third column gives the winner of the match, and the fourth (redundantly) gives the loser.

Task 1: You will first obtain an estimate of each player's skill level by writing a function named `calculate_ratings`, which takes as input a "past matches" filename that indicates a file in the format of `past_matches.csv` file described above. Because CSV files are more structured than generic text files, there are convenient packages for reading them. See Section 3 below for details on how to use one of these packages `Pandas` to read in CSV files. **Task 1 continued: The `calculate_ratings` function uses the data in the file to create and return a dictionary containing 8 entries where keys are the player names, and the values are the ratings for the players.** For each player i, we will denote the estimated player rating r_i, and use it as our best guess for the player's skill level s. The player ratings must be calculated as follows.

1. Initially, each player begins with a rating of 1500. That is, for each player i, set r_i equal to 1500.

2. Now, sequentially consider each match described in the input file. Each match is played between two players A (listed in column 1) and B (column 2). The probability or likelihood of A or B winning the game, respectively, is given by:

$$P(A \text{ wins}) = \frac{e^{(r_A - r_B)/c}}{1 + e^{(r_A - r_B)/c}} \quad (2)$$

$$P(B \text{ wins}) = 1 - \left(\frac{e^{(r_A - r_B)/c}}{1 + e^{(r_A - r_B)/c}} \right) \quad (3)$$

Recall that the constant c still equals 100. Below, we'll use $P(A)$ as shorthand for $P(A \text{ wins})$, the probability that A wins.

Based on the indicated outcome of the current match in the input file, update the ratings of the involved players as follows. If A does indeed win the game (as indicated by column 3 of the file), then update the player ratings as follows:

$$r_A = r_A + 5(1.0 - P(A)) \quad (4)$$

$$r_B = r_B + 5(0.0 - P(B)) \quad (5)$$

On the other hand, If A loses the game (as indicated by column 3), then update the player ratings as follows:

$$r_A = r_A + 5(0.0 - P(A)) \quad (6)$$

$$r_B = r_B + 5(1.0 - P(B)) \quad (7)$$

The constant factor 5 in the calculations above sets how greatly the rating fluctuates after each match.

Note that the player rating update is a zero-sum operation. This means that whatever rating amount player A loses (or wins), player B gains (or loses). If A wins the game, but is not expected to, meaning the original $P(A)$ was close to zero, then player A's rating is boosted heavily at the expense of player B. However if the original $P(A)$ is close to one (meaning A is expected to win), then the player ratings do not change much because of that player winning.

Once all matches in the input file have been processed, you have completed your estimate of the skill rating r_i of each player and you should return the dictionary described above.

NOTE: When your function reads data from an input file, include the code within a `try except` block, and output a suitable error message to indicate when there is a failure in reading the input file.

Task 2: Next, write a short function named `display_ratings` which takes as input a dictionary containing 8 entries where keys are the player names and the values are the ratings for the players (i.e., it takes as input the dictionary your `calculate_ratings` function returns). It should both generate to the console and save to a file named `projections.pdf` a bar graph displaying the ratings of each player. The specific format of the generated bar graph is up to you; just be sure that all 8 players are represented, and each bar is labeled. Make sure your function

both displays the graph when run interactively, and also generates the output file containing the graph. See Section 4 in this document for a description of how to use `matplotlib` or `seaborn` to generate plots and save them to files.

We have deliberately provided light details on using non-standard packages like `Pandas`, `matplotlib`, and `seaborn` as a learning exercise. Much of the skill of using Python (or any programming language) is not a matter of mastering the nitty gritty details of every function so much as getting in the habit of consulting online documentation to understand what a library's classes and functions do. The websites for the official documentation of each of these packages is provided in the relevant section below.

Additionally, towards writing fast-performing code, wherever possible you should use built-in `NumPy` or `Pandas` functions rather than writing your own code that performs the same operations. In grading your code, we may profile it to evaluate whether it is considerably slower than an optimized solution.

1.2 Step Two

Task 3: Next, write a function named `project_win_probs` that will repeatedly simulate the completed tournament based on the bracket shown in Figure 1. The function will take as input a dictionary of players named 0,1,2,3,4,5,6,7 and their ratings. That is, the function's input is an object of the form returned by your `calculate_ratings` function.

In each tournament simulation, simulate the play of all of the matches according to the player ratings supplied as input to the function.[1] Specifically, to simulate a single match between two players A and B, use the skill ratings computed from step one (r_A, r_B), and compute the expected win probability of player A as follows:

$$P(A \quad wins) = \frac{e^{\delta}}{1 + e^{\delta}} \tag{8}$$

using $\delta = r_A - r_B/c$

Then generate a random number in $[0.0, 1.0)$. If the number is less than $P(A)$, this represents a win for A; if it is not, it represents a win for B.

Repeat the simulation of the entire tournament bracket in Figure 1 (7 matches) a total of n = 100 times, to determine each player's likelihood of winning the tournament. For example, if player 0 wins the entire tournament 15 times over your 100 simulations, the likelihood of player 0 winning the tournament is 15%, or 0.15.

Once 100 simulations have been completed and the probabilities are determined, **the function should return a dictionary containing 8 entries where keys are the player names (0,1,2,3,4,5,6,7) and the values are the probabilities of the corresponding players winning the tournament. The probability values should always be in the range [0,1].**

[1] Please note that player names in the bracket do not represent tournament "seeds". It is possible, for example, that the two most highly-rated players face off in the first round.

Task 4: Next, write a short function named `display_probs` that takes as input a dictionary of the style output by your `project_win_probs` function, and generates two output files. The function does not return a value, but it will also display a plot on the screen.

The first file generated by the `display_probs` function is an output file named `probs.csv`, a comma-separated values file formatted as follows: Each line has a player name (0-7) and a corresponding probability value to win the tournament in the range [0,1]. The lines in the output file should be sorted so that the player who is most likely to win is listed at the top, the player second most likely to win is listed second, and so on.

The second file generated by the `display_probs` function is an output file named `projections_pie.pdf` containing a pie chart visually displaying your projections: each player's likelihood to win the tournament is shown as a slice of the pie. Hint: look up the documentation for the `pie` function in the `matplotlib` module. The specific format of the pie chart is up to you; just make sure the slices are labeled. In addition to being saved to a file, this pie chart must also be displayed in the console when the `display_probs` function is called.

2 Instructions for submission

- All of the required functions you write must exist in a file named `elo.py`. You are strongly encouraged to use helper functions in some way to simplify your code. Beyond this, there are no restrictions on how to design your code; you may submit additional files if your design needs them.

- Include significant comments in your code. Your code will be evaluated not only on correctness, but also on style and good programming techniques. Write docstrings for all functions and classes.

- If your solution is not working perfectly, turn it in as-is with detailed comments describing which parts are complete, and which still need work. Be sure that what you turn in runs, to make it possible to receive feedback and partial credit.

- You are strongly encouraged to adopt an incremental coding style, making small changes to your code one at a time, so that you always have a version of your program which runs, meaning you will always have something to turn in, even if it is not 100% complete. You are also reminded to exercise discipline in backing up your work. If you have trouble or need extra help, don't hesitate to attend office hours or contact a member of the course staff through Piazza.

- You are permitted to discuss the instructions with other students for clarification. However, you may not discuss any substance of your solution with anyone except course staff, whether that be algorithms, test cases, component design, or actual code.

3 Reading and writing CSV files

The most common format in which datasets or spreadsheets are typically stored is the CSV or comma separated values, file. Here is an example of a CSV file which we saved as `inventory.csv`.

```
,Model,Number
0,Civic,10
1,CRV,20
2,City,39
3,Pilot,5
4,Odyssey,4
```

We will use a data structures and analysis module called `pandas` (see https://pandas.pydata.org/) to read the CSV file. `pandas` stores data in objects called DataFrames. A DataFrame is like an excel spreadsheet where rows and columns have labels attached to them. When you read a CSV file using `pandas`, it creates a DataFrame to store the data. The methods available for the DataFrame object can be found at https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. A useful method for our purposes is `read_csv`, as shown below. The first step is to import the `pandas` module, which we alias below as `pd` for convenience.

```
1 import pandas as pd
2
3 df = pd.read_csv("inventory.csv", index_col=0)
4
5 print(df)
```

```
      Model  Number
0     Civic      10
1       CRV      20
2      City      39
3     Pilot       5
4  Odyssey        4
```

The headings shown in the first row of the output above are the column labels (Model, Number) given in the first row of the csv file. And since we specified `index_col=0` in the method call, the first column of the CSV file is being used to label rows.

Next, we demonstrate one way you can iterate through the rows of data in a DataFrame and access each element.

```
1 import pandas as pd
2
```

```
df = pd.read_csv("inventory.csv", index_col=0)

for index, row in df.iterrows():
    print(row['Model'],row['Number'])

```

```
Civic 10
CRV 20
City 39
Pilot 5
Odyssey 4
```

Writing data to a CSV file using `pandas` is also straightforward. The first step is to create a dictionary where the value associated with each key is a series, and then convert it to a data frame. Note in the example below that each series represents the data in a column of the data frame, and the associated key serves as the column header.

```
import pandas as pd

my_dict = {}

my_dict['Name'] = ['VW', 'MC', 'HN']
my_dict['Ratings'] = [2830, 2870, 2776]

# Turn my_dict into a DataFrame, then write it to CSV file
dfr = pd.DataFrame(my_dict, index = range(3))
dfr.to_csv('player_ratings.csv')

# Just for fun, read generated file back in and output the DataFrame
dfw = pd.read_csv("player_ratings.csv", index_col=0)
print(dfw)
```

```
  Name  Ratings
0   VW     2830
1   MC     2870
2   HN     2776
```

4 Plotting Data with **matplotlib** and **seaborn**

The `matplotlib` package in Python can be used to visualize data. The documentation of the API can be found at https://matplotlib.org/api/index.html. Below, we demonstrate how to create a 2D bar chart using `matplotlib`. An explanation of the code follows, and the generated plot is shown as part of Figure 2 below.

```
1  import matplotlib.pyplot as plt
2
3  plt.rc('font', family='serif')
4
5  fig = plt.figure(figsize=(6,5))
6  players = ['VW','MC','HN','LA','FC']
7  ratings = [2812, 2870, 2702, 2650, 2625]
8
9  plt.ylabel('Rating', fontsize=24)
10 plt.ylim(2000,3000)
11
12 plt.xticks(fontsize=24)
13 plt.yticks([i*200 + 2000 for i in range(6)], fontsize=24)
14
15 plt.bar(players, ratings)
16
17 plt.tight_layout()
18 plt.savefig('barplot.pdf')
19 plt.show()
```

- The first step is to import `matplotlib.pyplot`. In the code above, we use the alias `plt` to refer to it.
- Line 3 edits the fonts in the graph
- Line 5 creates an empty plot of size 6 inches by 5 inches.
- Lines 6 and 7 contain the data that we are going to plot. matplotlib can accept several data types, such as lists, `numpy` arrays, `pandas` data series, etc.
- Line 9 and 10 edit the range of the y axis and the label displayed.
- Lines 12 and 13 adjust the tick sizes and the corresponding labels in the plot.
- `plt.bar` draws a bar graph in the empty plot we created above.
- `plt.tight_layout()` adjusts the locations of the axes, and labels and packs them within the plot area.
- `plt.savefig()` can save the visualization in a variety of formats. (PNG, PDF, CSV, etc.) It takes a filename as an argument.
- `plt.show()` displays the plot

Seaborn (see https://seaborn.pydata.org/) is another package that one can use for data visualization. Plots created using Seaborn are often more appealing aesthetically than plots created using `matplotlib`. Here, we use `seaborn` to make a heatmap, but as with `matplotlib`, many other types of plots are possible. The heatmap is shown in Figure 2 below. The code to generate the heatmap follows.

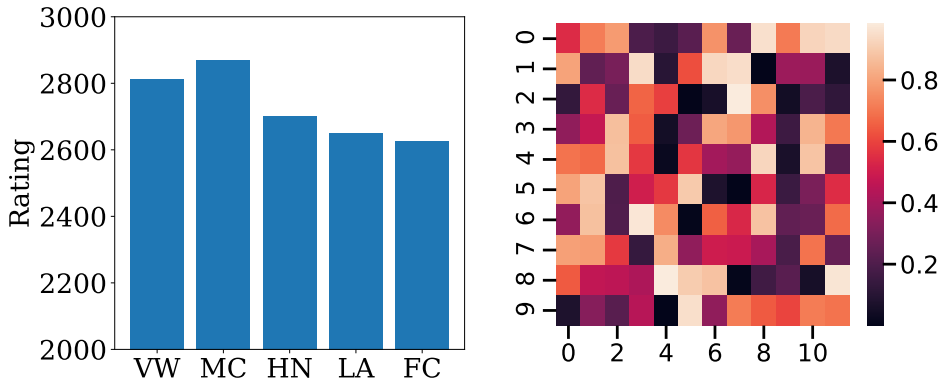

Figure 2: A sample 2D bar chart (left) and an unrelated heatmap (right).

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

sns.set_context("poster")

f = plt.figure(figsize=(6, 5))

# A 2D grid of data filled with random numbers
random_data = np.random.rand(10, 12)
print("Shape of array ",random_data.shape)

sns.heatmap(random_data)

plt.tight_layout()
plt.savefig('heatmap.pdf')
plt.show()
```