4F13 - COURSEWORK #2: PROBABILISTIC RANKING

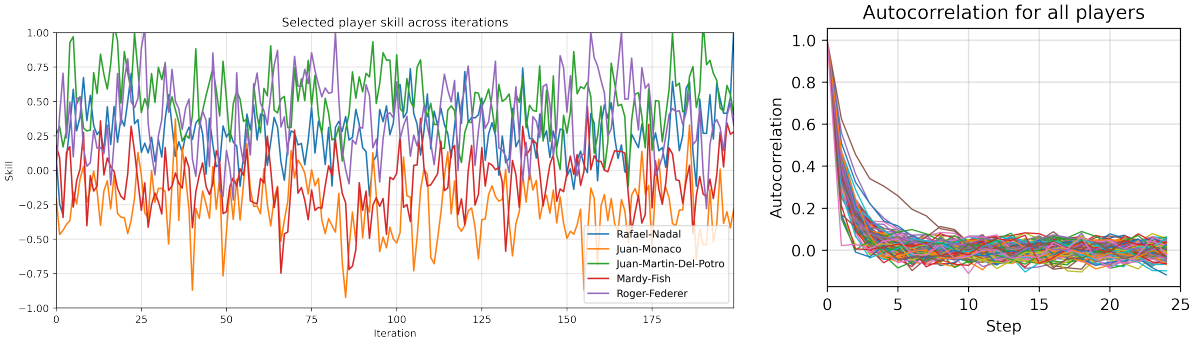
CCN: 5654F

1. Task A: Gibbs Sampling

The code in `gibbsrank.py` was completed by adding functionality to generate the sample skills given performance differences by adding line 1 of Listing 1 to the first uncompleted loop over all the players (which negates the skill difference of lost games from those won). Lines 4 onwards of the same Listing were then added to build the `iS` matrix, by adding ± 1 to each entry to match the form of the covariance matrix in the lecture notes[1].

```
1           m[p] = sum(t[np.where(G[:, 0] == p)]) - sum(
2                   t[np.where(G[:, 1] == p)]
3               )
4
5           winner = G[g, 0]
6           loser = G[g, 1]
7
8           iS[winner, winner] += 1
9           iS[loser, loser] += 1
10          iS[winner, loser] -= 1
11          iS[loser, winner] -= 1
```

Listing 1. Task A code excerpts



(a) Player Skills (b) Player Autocorrelations

Figure 1. Task A Figures

The Gibbs sampler was then run for 1100 iterations using the provided python notebook, and 4 random players skills were plotted across the first 200 iterations: Figure 1a. The data is noisy but is clearly not completely independent as each skill point stays fairly close to the previous point. The autocorrelation for each player for 25 steps is plotted in Figure 1b, which shows that an autocorrelation time of at least 10 samples is needed for the autocorrelation over the skills to be 0 for all athletes. Plotting the average population mean and variance also shows that the burn-in time (time for the chain to converge to the distribution) is lower than the 10 samples needed for the samples to become independent. Therefore the first 10 samples need to be discarded.

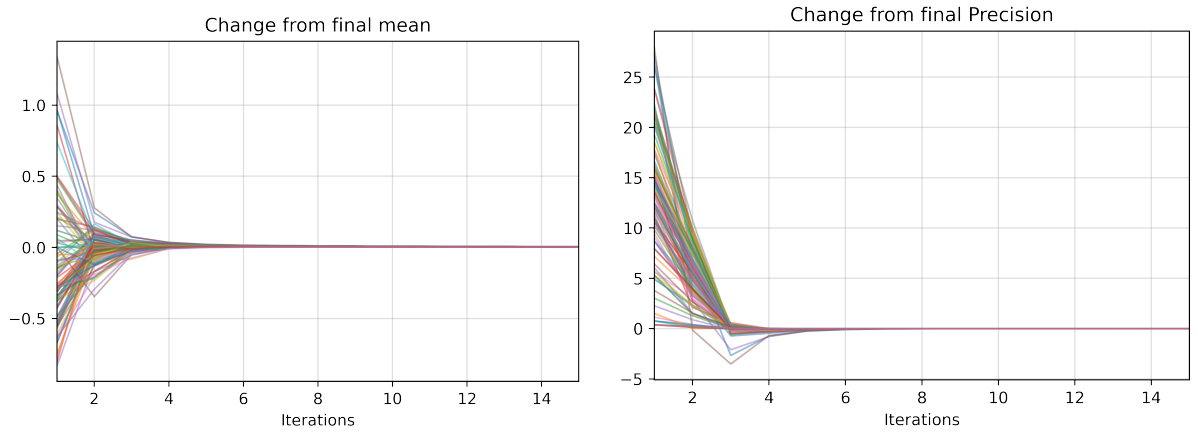
2. Task B: Message Passing

Convergence occurs when the parameters *converge* to a constant value (the true value according to the algorithm). For Gibbs Sampling this is when the the Markov chain converges to a stationary probability distribution (the joint skill distribution). For message passing/EP this

Date: Wednesday 15th November 2023.

[1]CUED 4F13 Lecture notes (course site)

occurs when a stable graph is reached, each iteration of messages passed does not affect the mean and precision of each vertex. In Section 1 we saw that the `max`(burn-in time, autocorrelation time) was ∼10 samples. A new function was first added to the `eprank.py` that returns the means and precisions at each iteration, which allowed graphs of the change in mean and precision from their final values to be produced as Figure 2. These show a more rapid convergence than the Gibbs sampling method, with the player means and precisions each reaching their final values in only ∼6 iterations, a 40% improvement over the method in Section 1.



(A) Player Mean Convergence (B) Player Precision Convergence

FIGURE 2. Task B Figures

3. Task C: Player Ranking using EP

Players i, j have skills w_i, w_j. $p(w_i > w_j)$ can be easily calculated using the definitions in the lecture notes[2] as in Equation 2. Similarly, a player wins against another in this model if the skill difference plus a noise $n \sim \mathcal{N}(0, 1)$ is greater than zero for i, or less than zero for j. Therefore the probability of i winning is given by Equation 2.

$$p(w_i > w_j) = p(w_j - w_i < 0) = \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{\lambda_i^{-1} + \lambda_j^{-1}}}\right) \tag{1}$$

$$p(w_i - w_j + n > 0) = \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{\lambda_i^{-1} + \lambda_j^{-1} + 1}}\right) \tag{2}$$

```
1         prob_has_better_skill = 1.0 - norm.cdf(
2             0, mean_differences, vars_sums**0.5
3         )
4
5         prob_win = 1.0 - norm.cdf(
6             0, mean_differences, (vars_sums + 1.0) ** 0.5
7         )
```

LISTING 2. Task C code excerpts

Lines 1 and 4 of Listing 2 are therefore used to calculate a matrix showing the probabilities of the top 4 ranked players (from the lecture notes' ranking[3]) a. having a higher skill than the others, and b. their probability of winning if playing against the other players. These lines

[2] see footnote 1
[3] see footnote 1

—	D	N	F	M
Djokovic(D)	0.500	0.940	0.909	0.985
Nadal(N)	0.060	0.500	0.427	0.767
Federer(F)	0.091	0.573	0.500	0.811
Murray(M)	0.015	0.233	0.189	0.500

TABLE 1. p(Higher Skill)

—	D	N	F	M
Djokovic(D)	0.500	0.655	0.638	0.720
Nadal(N)	0.345	0.500	0.482	0.573
Federer(F)	0.362	0.518	0.500	0.591
Murray(M)	0.280	0.427	0.409	0.500

TABLE 2. p(Win)

just demonstrate the equations from the previous paragraph, using the `scipy.stats.norm.cdf` function for Φ. The results are shown in Tables 1 and 2 respectively.

It can clearly be seen that a player with higher expected skill than another is always expected to win the match between them, and the converse. However, the probability of winning is much lower than the probability of having a higher skill due to the additional match variance term n (the performance noise). This adds to the model the fact that a player can have an "off match", and aids in training by making the model less confident in attributing skill to the winning player of each match.

4. Task D: Nadal and Federer Skill Comparison

4.1. **Approximation of Marginal Skills by Gaussians.**

5. Task E