Uppsala University

Department of Information Technology

1RT705: Project assignment
Advanced Probabilistic Machine Learning

Aleix Nieto Juscafresa
aleixnieto@gmail.com

Abstract

This report presents the documentation of the group project conducted in the Advanced Probabilistic Machine Learning course. The project focuses on implementing a Bayesian method rooted in the TrueSkill[1] ranking system to assess the skill levels of participants in competitive settings. The output of the project is provided as detailed responses to 10 key questions that guide our exploration. This includes the modeling of the scenario into a TrueSkill Bayesian framework, its representation using a Bayesian network, and the application of various techniques such as Gibbs sampling, assumed density filtering, factor graphing, and message passing. Additionally, the project delves into an open-ended extension where the Bayesian approach is applied to an external dataset, showcasing its adaptability and relevance in real-world scenarios. The report culminates in a comprehensive discussion of the findings, offering valuable insights and potential avenues for further research in the realm of probabilistic machine learning.

Introduction

The TrueSkill ranking system is a real-world implementation of probabilistic machine learning methods to estimate the skill levels of players in a competing environment. It was developed by Microsoft, who used it to assess players' skill in the Halo game console title Herbrich et al. [2006]. It performs similarly to the Elo rating system and might be considered as an extension of it as it provides a more flexible probabilistic approach. The Trueskill model is distinctive in that it can evaluate a player's skill and quantify the degree of uncertainty in that evaluation.

In this project, we will first implement the Trueskill model, followed by its application to a dataset consisting of the Italian 2018/2019 Serie A league results. As this model lacks an analytically solvable solution, we will make use of many statistical methods, including Bayesian inference, Gibbs sampling, assumed density filtering, moment matching, factor graphs, and message passing algorithms. These methods will be used to iteratively update player skills and make predictions for futures. The Trueskill model will then be put to the test using an additional dataset. This dataset consists of all the matches played in League of Legends Champions Korea (LCK) during the spring and summer of 2023. In the final part of the project, we will try to improve the model by adding features to handle draws and variability in player skills, addressing the issue of players getting stuck at specific ranks.

[1]Trademark Microsoft research, used with permission for non-commercial projects https://www.microsoft.com/en-us/research/project/trueskill-ranking-system.

Q1 - Modeling

To formulate the TrueSkill Bayesian model for one match between two players, we will define the joint distribution of all the random variables involved in the model. The model consists of four random variables:

- s_1 and s_2; Gaussian random variables representing the skills of the two players. We denote these as $s_1 \sim \mathcal{N}(s_1; \mu_1, \sigma_1^2)$ and $s_2 \sim \mathcal{N}(s_2; \mu_2, \sigma_2^2)$.
- t: Gaussian random variable representing the outcome of the match. This variable follows a Gaussian distribution, $t \sim \mathcal{N}(t; s_1 - s_2, \beta^2)$.
- y: Discrete random variable representing the result of the game. Since there is no possibility of a draw, y can take two values: $y = +1$ if Player 1 wins, and $y = -1$ if Player 2 wins.

The joint distribution of these random variables can be represented as:

$$p(s_1, s_2, t, y) = p(y|t)p(t|s_1, s_2)p(s_1)p(s_2),$$

where

- $p(s_1) = \mathcal{N}(s_1; \mu_1, \sigma_1^2)$
- $p(s_2) = \mathcal{N}(s_2; \mu_2, \sigma_2^2)$
- $p(t|s_1, s_2) = \mathcal{N}(t; \mu_{t|s}, \sigma_{t|s}^2) = \mathcal{N}(t; s_1 - s_2, \beta^2)$
- $p(y|t) = \text{sign}(t)$

To complete the model, we would have to specify values for the five hyperparameters: μ_1, μ_2, σ_1^2, σ_2^2, and β^2. We will denote $s = [s_1, s_2]^\top$ and although it is a vector we will omit bold notation.

Q2 - Bayesian Network

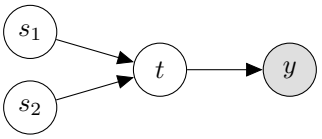

Figure 1: Bayesian network of the model from Q.1. The node y is grey as it has been observed.

We have $A = \{s_1, s_2\}$, $B = \{y\}$, and $C = \{t\}$, then

$$A \perp\!\!\!\perp B \,|\, C,$$

because t is *head-to-tail* on all paths between A and B (the node in the middle t has been observed). Another set of independence is $A = \{s_1\}$, $B = \{s_2\}$, and $C = \{\emptyset\}$ (unobserved *head-to-head*).

We are in a head-to-tail nodes case ($s1$ or $s2$ is the head and y is the tail). Note that conditional independence is automatically ensured through d-separation, as observing t, blocks the path from y to s_1 and s_2. This is not the case of Figure 1 where we represent the model from Q.1, not the case where t is observed. Starting from $p(s_1, y, t)$, marginalizing out t gives $p(s_1, y, t) = p(s_1, y|t)p(t)$. On the other hand using the factorization of the joining probability distribution, together with Bayes' theorem on $p(t|s_1)$ gives $p(s_1, y, t) = p(s_1|t)p(y|t)p(t)$. Equating these two terms we obtain the desired conditional independence. Mathematically:

$$p(s_1, y|t) = \frac{p(s_1, y, t)}{p(t)} = \frac{p(s_1)p(t|s_1)p(y|t)}{p(t)} = \frac{p(y|t)p(s_1|t)p(t)}{p(t)} = p(y|t)p(s_1|t).$$

This proves that $s_1 \perp\!\!\!\perp y, t$. For s_2, the procedure is exactly the same.

By expressing s_1 and s_2 in compact form as $s = [s_1, s_2]^\top$ and following the same procedure we can prove that s_1 and s_2 are jointly conditionally independent of y given t.

Q3 - Computing with the model

$\boxed{p(s_1, s_2|t, y)}$

We can denote the Gaussian random variables s_1 and s_2 as a multivariate Gaussian random variable $s \sim \mathcal{N}(s; \boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$, where

$$s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \boldsymbol{\mu}_s = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \boldsymbol{\Sigma}_s = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}.$$

Then, by using that s_1 and s_2 are conditional independent of y given t as we proved in Q2:

$$p(s_1, s_2|t, y) = p(s|t, y) = \frac{p(s, t, y)}{p(t, y)} = \frac{p(s, y|t)p(t)}{p(y|t)p(t)} = \frac{\cancel{p(y|t)}p(s|t)\cancel{p(t)}}{\cancel{p(y|t)}\cancel{p(t)}} = p(s|t).$$

From the *Corollary 1 (Affine transformation – Conditioning)* in Wahlström [2022]:

$$p(s) = \mathcal{N}(s; \boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s),$$
$$p(t|s) = \mathcal{N}(t; s_1 - s_2, \beta^2) = \mathcal{N}(t; \boldsymbol{A}s + b, \beta^2),$$

where $\boldsymbol{A} = [1 \quad -1]$ and $b = 0$. The vector $\boldsymbol{A}$ follows from $\boldsymbol{A}\boldsymbol{s} = [1 \quad -1] \cdot [s_1 \quad s_2]^\top = s_1 - s_2$. Then the conditional distribution s given t is

$$p(s|t) = \mathcal{N}(s; \boldsymbol{\mu}_{s|t}, \boldsymbol{\Sigma}_{s|t}),$$

where

$$\boldsymbol{\mu}_{s|t} = \boldsymbol{\Sigma}_{s|t}\left(\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\mu}_s + \boldsymbol{A}^\top(\beta^2)^{-1}(t - b)\right) = \Sigma_{s|t}\left(\begin{bmatrix} \frac{\mu_1}{\sigma_1^2} \\ \frac{\mu_2}{\sigma_2^2} \end{bmatrix} + \frac{t}{\beta^2}\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} \frac{\mu_1\beta^2 + \mu_1\sigma_2^2 + \mu_2\sigma_1^2 + \sigma_1^2 t}{\beta^2 + \sigma_1^2 + \sigma_2^2} \\ \frac{\mu_1\sigma_2^2 + \mu_2\beta^2 + \mu_2\sigma_1^2 - \sigma_2^2 t}{\beta^2 + \sigma_1^2 + \sigma_2^2} \end{bmatrix},$$

$$\boldsymbol{\Sigma}_{s|t} = \left(\boldsymbol{\Sigma}_s^{-1} + \boldsymbol{A}^\top(\beta^2)^{-1}\boldsymbol{A}\right)^{-1} = \left(\begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}^{-1} + \frac{1}{\beta^2}\begin{bmatrix} 1 \\ -1 \end{bmatrix}[1 \quad -1]\right)^{-1} = \begin{bmatrix} \frac{\sigma_1^2(\beta^2 + \sigma_2^2)}{\beta^2 + \sigma_1^2 + \sigma_2^2} & \frac{\sigma_1^2\sigma_2^2}{\beta^2 + \sigma_1^2 + \sigma_2^2} \\ \frac{\sigma_1^2\sigma_2^2}{\beta^2 + \sigma_1^2 + \sigma_2^2} & \frac{\sigma_2^2(\beta^2 + \sigma_1^2)}{\beta^2 + \sigma_1^2 + \sigma_2^2} \end{bmatrix}.$$

$\boxed{p(t|s_1, s_2, y)}$

The full conditional distribution of the outcome is given by:

$$p(t|s_1, s_2, y) = \frac{p(s_1, s_2, t, y)}{p(s_1, s_2, y)} = \frac{p(s_1)p(s_2)p(t|s_1, s_2)p(y|t)}{p(s_1, s_2, y)}.$$

By disregarding the terms which are independent of t,

$$p(t|s_1, s_2, y) \propto p(y|t)p(t|s_1, s_2).$$

The second factor is Gaussian while the first one is nonzero only when y and t have the same sign. This is precisely a *Truncated Gaussian* with parameters $\mu = s_1 - s_2$, $\sigma^2 = \beta^2$, a and b. We distinguish the two cases $y = 1$ and $y = -1$:

$$y = 1 : a = 0, b = \infty,$$
$$y = -1 : a = -\infty, b = 0.$$

Formalizing this, we have that

$$p(t|s_1, s_2, y) = \begin{cases} p(t|s_1, s_2) & \text{if sign}(t) = y \\ 0 & \text{otherwise} \end{cases}$$

So $t|s_1, s_2$ is a truncated Gaussian, on the positive real axis if $y > 0$, on the negative real axis otherwise.

$p(y = 1)$

By definition, $p(y = 1)$ is equivalent to $p(t > 0)$. To find $p(t)$, *Corollary 2 (Affine transformation – Marginalization)* in Wahlström [2022] can be used for marginalization. Then, given $p(s)$ and $p(t|s)$:

$$p(t) = \mathcal{N}(t; \mu_t, \sigma_t^2),$$

where

$$\mu_t = \boldsymbol{A}\boldsymbol{\mu}_s + b = \mu_1 - \mu_2,$$
$$\sigma_t^2 = \sigma_{t|s}^2 + \boldsymbol{A}\boldsymbol{\Sigma}_s\boldsymbol{A}^\top = \sigma_1^2 + \sigma_2^2 + \beta^2.$$

The random variable t follows a normal distribution:

$$t \sim \mathcal{N}(t, \mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2 + \beta^2).$$

This means that t is normally distributed with mean $\mu_1 - \mu_2$ and variance $\sigma_1^2 + \sigma_2^2 + \beta^2$. The goal is to compute $p(t > 0)$, i.e., the probability that Player 1's performance exceeds Player 2's performance.

Now, $p(t > 0)$ can be calculated using the cumulative distribution function (CDF) of a normal distribution:

$$\begin{aligned} p(y = 1) = p(t > 0) = 1 - p(t < 0) &= 1 - \int_{-\infty}^{0} \mathcal{N}(t; \mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2 + \beta^2)dt \\ &= 1 - \Phi(\frac{\mu_2 - \mu_1}{\sqrt{\sigma_1^2 + \sigma_2^2 + \beta^2}}) \\ &= \Phi(\frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2 + \beta^2}}), \end{aligned}$$

where we used $\Phi(-x) = 1 - \Phi(x)$.

Also, note that we convert t into a standard normal variable as Φ is the CDF of the standard normal distribution. Standardization is done by subtracting the mean and dividing by the standard deviation:

$$z = \frac{t - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2 + \sigma_2^2 + \beta^2}}.$$

In this case, we are interested in $p(t > 0)$. So, instead of the generic t, we substitute 0 for t because we want the probability that the performance difference t is greater than zero.

Thus, the standardized value is:

$$z = \frac{0 - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2 + \sigma_2^2 + \beta^2}} = \frac{\mu_2 - \mu_1}{\sqrt{\sigma_1^2 + \sigma_2^2 + \beta^2}}.$$

This gives us the z-score, which represents how many standard deviations away the point $t = 0$ is from the mean $\mu_1 - \mu_2$.

Q4 - A first Gibbs sampler

We will implement a method based on Gibbs sampling to compute the posterior distribution of the skills s_1 and s_2 given the result of one match y.

We initialize the distributions of s_1, s_2 and $t|s_1, s_2$ with the following hyperparameters: $\mu_1 = \mu_2 = 25$, $\sigma_1^2 = \sigma_2^2 = (25/3)^2$, and $\beta^2 = (25/6)^2$. The motivation for this particular selection can be found in Herbrich et al. [2006].

To sample from $p(s_1, s_2|y)$, we alternate between two conditional distributions: $p(s_1, s_2|t, y) = p(s_1, s_2|t)$ and $p(t|s_1, s_2, y)$. At each iteration, given the current value of t (t is initialized to some value greater than 0 to represent that the first player has won the game), we first sample s_1 and s_2 from $p(s_1, s_2|t)$, which is a multivariate Gaussian with the mean and covariance computed based on the current skill estimates. Next, we sample t from $p(t|s_1, s_2, y)$, which is a truncated normal distribution, where the truncation depends on the game outcome y. These steps are repeated to generate samples of s_1 and s_2, gradually building an empirical approximation to $p(s_1, s_2|y)$.

In Figure 2 we observe the values of s_1 and s_2 after 10 000 samples of the posterior distributions generated by the Gibbs sampler when $y = 1$ (Player 1 wins). In this example, the initial condition for t was set to 1. The burn-in time in this case is around 100 samples, as the distributions adopt the stationary state after this number of samples. See Appendix A for a visual representation of the burn-in.

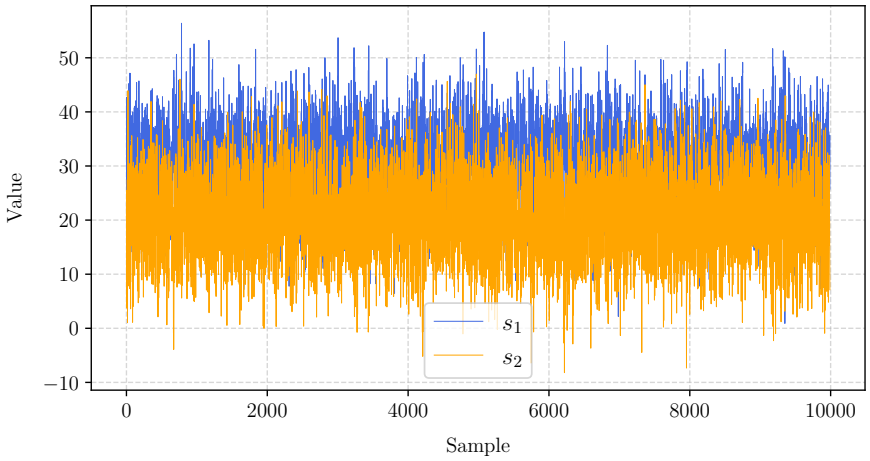

Figure 2: Gibbs sampler ($n = 10\,000$ samples) of the posterior distribution when $y = 1$.

In Figure 3 we recover the representation of s_1 and s_2 skills as a Gaussian distribution. This has been done by inferring the mean and the variance from the samples drawn by the Gibbs sampler and discarding the 100 first samples due to the burn-in. We can see that the distribution of s_1 has a higher mean which is reasonable since $y = 1$. Note that the density functions are only plotted 3σ away from the mean on both sides. As can be seen, most of the samples fall below this curve. This is mentioned because the *TrueSkill* algorithm uses the 3σ mark as a (very) conservative estimate for the skill.

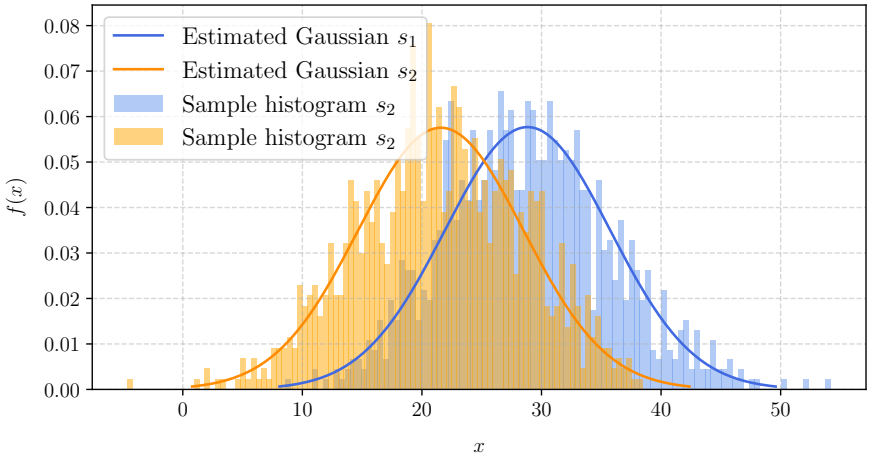

Figure 3: Recovered representation of s_1 and s_2 skills as Gaussian random variables.

In Figure 4 we present the histogram of the generated samples (post burn-in) alongside the fitted Gaussian posterior for six different sample sizes. Additionally, we report the time taken to generate these samples. This analysis highlights the trade-off between the accuracy of the estimates and the computational time when determining the optimal number of samples to use after burn-in. We can observe that 10 000 is a reasonable number of samples since the computational time is short (just around 1 second) and with a high accuracy of the estimate.

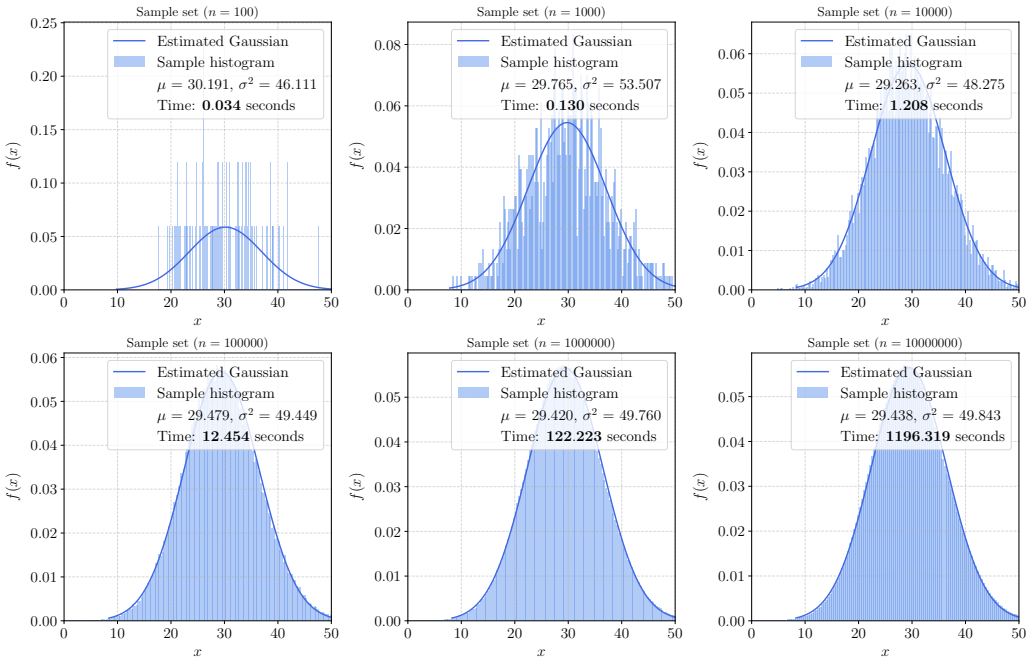

Figure 4: Estimated Gaussian distributions for s_1 for different amounts of samples.

Figure 5 shows a Gaussian approximation of the posterior distribution of the skills, together with the prior for s_1 respective s_2. The plot shows that the variance has decreased for the posterior distribution, and the distribution of s_1 has a higher mean than the prior while the distribution of s_2 has a lower mean than the prior. This is reasonable since $y = 1$t (Player 1 wins the game) was given.

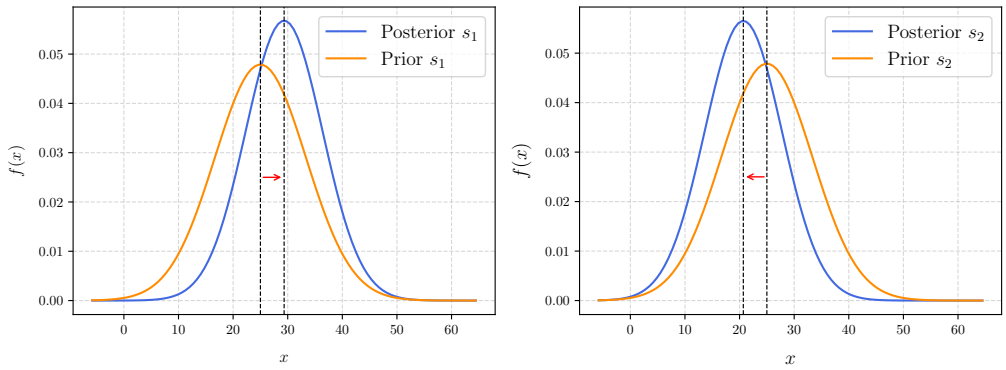

Figure 5: Prior of s_1/s_2 vs posterior of s_1/s_2.

Q5 - Assumed density filtering

In Table 1 (left) we display the final ranking obtained using *assumed density filtering* (ADF) with Gibbs sampling (10000 samples, choice motivated in Figure 4). The result will vary slightly in each execution round because the final ranking is not deterministic. The variance describes how spread the values are from the mean, i.e., the uncertainty of a team's skill. The bigger the variance of the two teams playing, the more likely it is that the less skilled team wins.

Instead of iteratively updating the skills across the matches, we can change the order of the matches in the dataset at random and see if the result will change. This can be seen by looking at the ranking in Table 1 (right). Along with the built-in stochastic of the sample, skills also change significantly. For instance, winning the first game offers a better indication of a team's skill than winning the last game.

Rank	Team	μ	σ
1	Juventus	29.480	1.092
2	Napoli	28.837	0.892
3	Milan	27.940	1.053
4	Torino	27.643	0.976
5	Inter	27.439	0.874
6	Atalanta	27.409	0.788
7	Roma	27.293	0.860
8	Lazio	25.842	0.823
9	Sampdoria	25.287	0.801
10	Bologna	24.450	0.860
11	Spal	24.184	0.832
12	Udinese	24.104	0.997
13	Empoli	23.847	0.844
14	Genoa	23.738	0.956
15	Parma	23.584	0.892
16	Sassuolo	23.524	1.023
17	Cagliari	23.380	0.949
18	Fiorentina	23.134	1.009
19	Frosinone	20.937	1.016
20	Chievo	18.457	1.484

Rank	Team	μ	σ
1	Napoli	31.153	1.430
2	Juventus	30.431	1.644
3	Milan	28.838	1.405
4	Torino	28.669	1.450
5	Roma	28.061	1.377
6	Atalanta	27.946	1.223
7	Inter	27.926	1.388
8	Lazio	26.187	1.327
9	Sampdoria	25.639	1.255
10	Bologna	25.229	1.308
11	Sassuolo	24.178	1.493
12	Spal	24.057	1.267
13	Udinese	24.029	1.334
14	Cagliari	23.958	1.426
15	Genoa	23.901	1.431
16	Empoli	23.814	1.158
17	Parma	23.423	1.298
18	Fiorentina	23.295	1.562
19	Frosinone	21.223	1.582
20	Chievo	17.750	2.250

Table 1: Final ranking processing the matches sequentially (left) and final ranking processing the matches at random (right).

Q6 - Using the model for predictions

In this section, we use the ADF-model from Question 5. Before updating the skills parameters, we predict the team with the highest skill to win the next match based on the estimates from the last iteration. Then, the prediction is compared with the actual result of the match. This way, we can calculate our prediction rate averaging the predictions across all matches. In Table 2 we can observe the prediction rates taking into account all the games and all the games without the draws. Note that this method obtains 177 correct predictions (out of 380), which is 0.465 of the games being correctly predicted. When not taking the draws into account the prediction rate is 0.65, clearly outperforming random guessing, which would have a probability of 1/2, given two possible outcomes of the game.

Total number of games	Number of games without draws	Correct predictions	Prediction rate (all matches)	Prediction rate (without draws)
380	272	174	0.465	0.650

Table 2: Number of matches, correct predictions, and prediction rates.

Q7 - Factor graph

The factor graph of our distribution looks as follows:

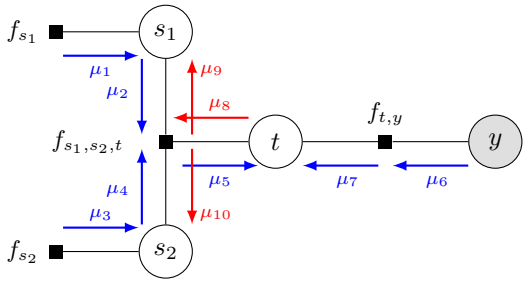

Figure 6: Factor graph of the model.

The joint distribution is described as $p(s_1, s_2, t, y) = f_{s_1}(s_1) f_{s_2}(s_2) f_{s_1,s_2,t}(s_1, s_2, t) f_{t,y}(t, y)$, where

$$\begin{aligned} f_{s_1}(s_1) &= p(s_1) = \mathcal{N}(s_1; \mu_1, \sigma_1^2), \\ f_{s_2}(s_2) &= p(s_2) = \mathcal{N}(s_2; \mu_2, \sigma_2^2), \\ f_{s_1,s_2,t}(s_1, s_2, t) &= p(t|s_1, s_2) = \mathcal{N}(t; s_1 - s_2, \beta^2), \\ f_{t,y}(t, y) &= p(y|t) = \delta(y = \text{sign}(t)). \end{aligned}$$

Q8 - A message-passing algorithm

Explicit forms for the messages needed to compute $p(t|y)$ are:

$$\begin{aligned} \mu_6(y) &= \delta(y = 1), \\ \mu_7(t) &= \sum_y f_{t,y} \cdot \mu_6(y) = \sum_y f_{t,y} \cdot \delta(y = 1) \\ &= \sum_y \delta(y = \text{sign}(t)) \delta(y = 1) = \sum_y \delta(\text{sign}(t) = 1) = \delta(t > 0). \end{aligned}$$

The message $\mu_7(t)$ will result in a non-Gaussian (truncated Gaussian) $p(t) = \mu_7(t)\mu_5(t)$ and we need to do moment-matching at this node. For this, we need all incoming messages. To get $\mu_5(t)$ we start from the left in the graph:

$$\mu_1(s_1) = f_{s_1}(s_1) = \mu_2(s_1), \qquad \mu_3(s_2) = f_{s_2}(s_2) = \mu_4(s_2),$$

$$\mu_5(t) = \iint f_{s_1,s_2,t}(s_1, s_2, t) \mu_2(s_1) \mu_4(s_2) ds_1 ds_2 = \int p(t|s) p(s) ds = \mathcal{N}(t; \mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2 + \beta^2),$$

where to calculate $\mu_5(t)$ we used *Corollary 2 (Affine transformation – Marginalization)* in Wahlström [2022] together with the independence of s_1 and s_2. The remaining messages are the ones used to backpropagate the messages (marked in red in Figure 6) back to s_1 and s_2. The message $\mu_8(t)$ is the result of dividing the outgoing message $q(t) = \mathcal{N}(t; \tilde{\mu}_t, \tilde{\sigma}_t^2)$ (approximated using moment matching) by the incoming message $\mu_5(t)$:

$$\mu_8(t) = \frac{q(t)}{\mu_5(t)} \propto \mathcal{N}(t; \hat{\mu}_t, \hat{\sigma}_t^2).$$

The remaining two messages are $\mu_9(s_1)$ and $\mu_{10}(s_2)$:

$$\begin{aligned} \mu_9(s_1) &= \iint \mathcal{N}(t; s_1 - s_2, \beta^2) \mu_8(t) \mu_4(s_2) dt ds_2 \qquad \text{(Gaussian shifting)} \\ &\propto \iint \mathcal{N}(t + s_2; s_1, \beta^2) \mu_8(t) \mu_4(s_2) dt ds_2 \qquad \text{(Gaussian scaling)} \\ &\propto \iint \mathcal{N}(s_1; t + s_2, \beta^2) \mu_8(t) \mu_4(s_2) dt ds_2 \qquad \text{(Gaussian marginalization)} \\ &= \mathcal{N}(s_1; \hat{\mu}_t + \mu_2, \sigma_2^2 + \beta^2 + \hat{\sigma}_t^2). \end{aligned}$$

In the last step we used the fact that since $t \sim \mathcal{N}(t; \hat{\mu}_t, \hat{\sigma}_t^2)$ and $s_2 \sim \mathcal{N}(s_2, \mu_2, \sigma_2^2)$, we can deduce that the sum $t + s_2$ follows a normal distribution given by $t + s_2 \sim \mathcal{N}(t + s_2, \hat{\mu}_t + \mu_2, \hat{\sigma}_t^2 + \sigma_2^2)$.

Message $\mu_{10}(s_2) = \mathcal{N}(s_2; \mu_1 - \hat{\mu}_t, \sigma_1^2 + \beta^2 + \hat{\sigma}_t^2)$ has been determined following the same procedure.

Now, we have all the ingredients to compute the posterior distribution for each of the two skills given the result of one game between two players:

$$p(s_1|y) \approx q(s_1) \propto \mu_1(s_1)\mu_9(s_1) \propto \mathcal{N}\left(s_1; \frac{(\hat{\mu}_t + \mu_2)\sigma_1^2 + \mu_1(\beta^2 + \sigma_2^2 + \hat{\sigma}_t^2)}{\sigma_1^2 + \sigma_2^2 + \beta^2 + \hat{\sigma}_t^2}, \frac{\sigma_1^2(\beta^2 + \sigma_2^2 + \hat{\sigma}_t^2)}{\sigma_1^2 + \sigma_2^2 + \beta^2 + \hat{\sigma}_t^2}\right),$$

$$p(s_2|y) \approx q(s_2) \propto \mu_3(s_2)\mu_{10}(s_2) \propto \mathcal{N}\left(s_2; \frac{(\mu_1 - \hat{\mu}_t)\sigma_2^2 + \mu_2(\beta^2 + \sigma_1^2 + \hat{\sigma}_t^2)}{\sigma_1^2 + \sigma_2^2 + \beta^2 + \hat{\sigma}_t^2}, \frac{\sigma_2^2(\beta^2 + \sigma_1^2 + \hat{\sigma}_t^2)}{\sigma_1^2 + \sigma_2^2 + \beta^2 + \hat{\sigma}_t^2}\right).$$

In Figure 7 it is shown both the posterior distribution of the skills after one match given the result that Player 1 one the game, computed with a message passing algorithm and the Gaussian approximation from Gibbs sampling.

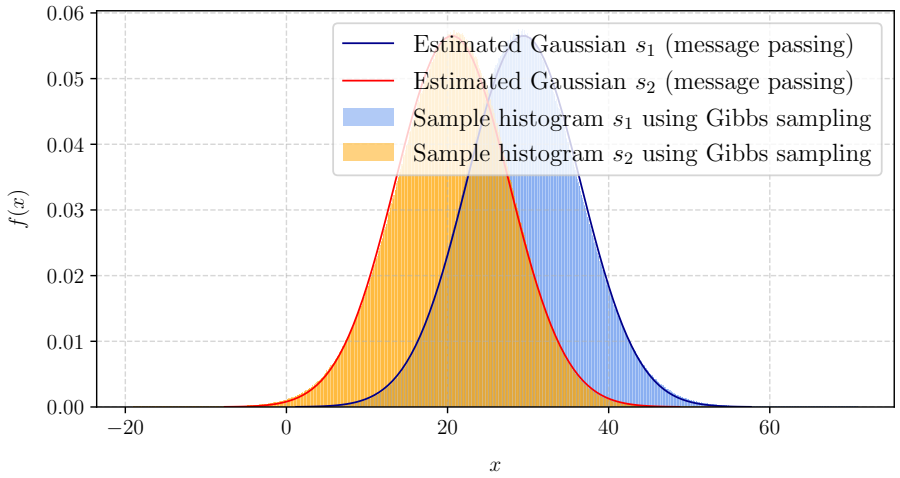

Figure 7: The posterior computed with message passing plotted together with the Gaussian approximation of the posterior, for s_1 respective s_2, using Gibbs sampling.

The posteriors are practically the same as seen in Table 3, which means that both these methods come to approximately the same posterior distributions. To achieve maximum accuracy, Gibbs sampling was conducted over a total of 10 000 000 million samples.

	Player	Mean μ	Sigma σ^2
Gibbs sampling	1	29.4258	49.7265
	2	20.5730	49.7806
Message passing	1	29.4327	49.7957
	2	20.5673	49.7957

Table 3: Results using Gibbs sampling with 10 000 000 samples and message passing.

Q9 - Your own data

League of Legends Champions Korea (LCK) is the primary competition for *League of Legends* electronic sports in South Korea. Contested by ten teams, the league runs two seasons per year and serves as a direct route to qualification for the annual League of Legends World Championship. A dataset consisting of all matches played in the LCK over the spring and summer splits has been employed for testing the developed Trueskill methods. The idea is to assess the ability of the teams in the face of the World Cup that will be held this October in Seoul where the LCK will have 4 representatives.

The data has been extracted from the official `Riot Games webpage`, the company that published League of Legends back in 2009. The dataset that has been created follows exactly the same structure as the Serie A dataset, in this way, all the code previously used has been reused without modification. The dataset has been evaluated with the methods from Question 5 and Question 6 and the results have been displayed in Appendix B.

We note that the team with the most skill in both methods is Gen. G, who is precisely the winner of the two splits that have been contested this year. However, if we compare the two tables the skills do not perfectly correspond to official rankings but show a lot of similarity. It should also be noted that the 4 teams with the most skill have been the 4 selected to represent Korea in the World Cup.

Finally, we note that we obtained a fairly high prediction rate of 0.770 over the total number of games. This is because, in a League of Legends game, there are no draws as the winner of a match is the best of 3 games (except in the split playoffs where it is the best of 5 games).

Q10 - Open-ended project extension

To avoid players getting stuck at a rank due to the TrueSkill algorithm constantly reducing the standard deviation of each player, we introduced an additive dynamics factor. It was added after Gibbs sampling through a grid search to find the optimal value, as depicted in Table 5(a).

Another tested addition was the introduction of draws to the algorithm. This was achieved by evaluating the value of t. If the value was smaller than a predefined draw margin, the prediction would result in a draw ($y = 0$). If the value was equal to or larger than the draw margin, y was determined in the same manner as before. This aspect was also examined through a grid search involving various values for the draw margin, as illustrated in Table 5(b).

Using an additive factor worsened the prediction rate and was therefore discarded when implementing the second extension. However, introducing a draw margin leads to a better result. The optimal prediction rate was achieved with a draw margin of 0.8, giving a prediction rate of 0.503, compared to not having a draw margin, which resulted in a prediction rate of 0.474.

Discussion and conclusions

Throughout the project, we defined a probabilistic model to estimate player skills in competition, computed the model from different viewpoints, and determined the posterior distribution of the skills of players given a match result using different approaches such as Gibbs sampling and message passing. Our assumed density filtering using Gibbs sampling model achieved a 65% prediction accuracy (no draws) and 46.5% (with draws) with consistent posterior skill distributions.

When applying the TrueSkill algorithm on the LCK dataset it produced a considerably high prediction rate of 77%. It should however be noted that there were no draws in this dataset which may skew the result to appear more favorable.

We also tried to improve our implementation by adding two new features: an additive dynamics factor to increase player variance and introducing draws with a specific margin. While the additive factor had a negative impact on predictions, optimizing the draw margin showed a slight improvement in the prediction rate.

Comparing the prediction rates when considering draws and no draws makes us believe that our implementation may not handle draws that well. This is something worth considering for a potential future improvement. Nevertheless, our implementation is still significantly superior to predicting at random.

References

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: A bayesian skill rating system. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/f44ee263952e65b3610b8ba51229d1f9-Paper.pdf.

Niklas Wahlström. Lecture 2 notes in Advanced Probabilistic Machine Learning. Uppsala University, 2022.

Appendix

Appendix A. Question 4 - Burn-in

In this section, we show a smaller number of samples to be able to see the burn-in more clearly. Figure 8 shows that a reasonable choice is around 100 samples because it is the time it takes for the cumulative means of s_1 and s_2 to stabilize.

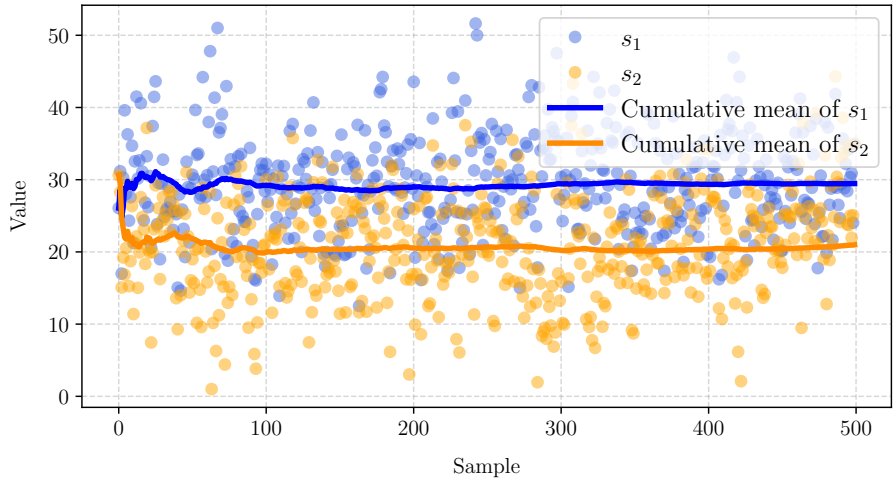

Figure 8: Gibbs sampler ($n = 500$ samples) of the posterior distribution when $y = 1$.

Appendix B. Question 9 - Results

All the code previously used in Question 5 and Question 6 has been reused without modification.

Rank	Team	μ	σ
1	Gen. G	26.80	1.45
2	KT Rolster	25.74	1.32
3	T1	23.79	1.26
4	Dplus Kia	22.72	1.22
5	Hanwha Life Esports	22.41	1.21
6	Liiv SANDBOX	19.55	1.22
7	DRX	19.22	1.11
8	BRION	18.49	1.12
9	Kwangdong Freecs	17.90	1.17
10	Nongshim RedForce	17.60	1.22

Rank	Team	μ	σ
1	Gen. G	30.12	1.84
2	T1	28.13	1.71
3	KT Rolster	28.06	1.71
4	Dplus Kia	26.14	1.50
5	Hanwha Life Esports	24.34	1.44
6	Liiv SANDBOX	23.56	1.64
7	DRX	21.57	1.47
8	Kwangdong Freecs	20.81	1.52
9	BRION	20.80	1.55
10	Nongshim RedForce	18.00	1.67

Table 4: Final ranking processing the matches sequentially (left table) and final ranking processing the matches at random (right table).

Total number of games	Correct predictions	Prediction rate
196	151	0.770

Table 5: Number of matches, correct predictions, and prediction rate.

Appendix C. Question 10 - Results

Additive Factor	Prediction Rate
0.1	0.461
0.2	0.455
0.3	0.447
0.4	0.439
0.5	0.447
0.6	0.447
0.7	0.447
0.8	0.447
0.9	0.450
1.0	0.450

(a)

Draw Margin	Prediction Rate
0.2	0.487
0.4	0.487
0.6	0.492
0.8	0.503
1	0.497
1.5	0.495
2	0.484
3	0.442
4	0.437
5	0.392

(b)

Table 6: Prediction rate for each value of additive dynamics factor τ (left). Prediction rate for each draw margin (right).