Take-home Exam

Deadline: January 1, 2023, 23:59

This document contains the instructions for the take-home exam for the course Advanced Probabilistic Machine Learning, SSY316. You will implement a Bayesian method, based on the TrueSkill [1] ranking system, to estimate the skill of players involved in a competition. The exam will be also done in the group of two people that you have already registered in the Canvas. The whole points for the exam is 40.

Introduction

You will work through the whole process of solving a real-world problem using probabilistic machine learning. You are asked to estimate the skill of players involved in a competition based on the results of matches between pairs of players . You will first define a probabilistic model, where all the quantities are represented as random variables, based on the Trueskill Bayesian ranking system developed by Microsoft research for raking on-line matches. The model assigns a skill to each player, in the form of a Gaussian random variable. When two players meet in a match, the outcome of that match is a Gaussian random variable with mean equal to the difference between the two players' skills. The result of the match is 1 (indicating the victory of Player 1) if the outcome is greater than zero, -1 if it is less than zero (indicating the victory of Player 2).
You will use Bayesian inference to find the posterior distribution of the players' skills given observations of the results of matches. Because the posterior distribution is intractable, you will use two different approximation methods based on graphical models.
In the Canvas, you will find the SerieA.csv dataset containing the results of the Italian 2018/2019 *Serie A* elite football (soccer) division; however, the Trueskill model can be used to solve a variety of skill and ranking problems. Later in the exam you will be asked to use a dataset of games/competitions of your choosing.

Grading

A successful submission has detailed and brief answers to all questions. The answer to Q.10 is open-ended and only one, well motivated, extension is strictly required; you are free to implement and extend as much as you please! Use plots only when necessary to convey a point across: do not plot something just because you can.

[1] https://www.microsoft.com/en-us/research/project/trueskill-ranking-system/

Assignments

The following questions need to be addressed in the report. We suggest that you solve the tasks sequentially, as later questions build upon results of previous questions.

Q.1 Modeling (4 points)

Formulate the Trueskill Bayesian model for one match between two players. The model consists of four random variables, two Gaussian random variables s_1 and s_2 for the skills of the players, one Gaussian random variable t, with mean $s_1 - s_2$ for the outcome of the game, and one discrete random variable $y = sign(t)$ for the result of the game (there is no possibility of a draw between the players). Note that there are 5 hyperparameters in the model whose values you have to set. You can read about TrueSkill on the Microsoft research website [2], on the original publication [3], or on Jeff Moser's website [4]. Hint: remember that a "model" in Bayesian parlance is a joint distribution of all the random variables.

Q.2 Bayesian Network (4 points)

Draw the Bayesian network of the model from Q.1 and identify two conditionally independent sets of variables.

Q.3 Computing with the model (4 points)

Compute

- $p(s_1, s_2 \mid t, y)$: the full conditional distribution of the skills. *Hint*: use conditional independence to eliminate y, and derive a closed-form solution.
- $p(t \mid s_1, s_2, y)$: the full conditional distribution of the outcome. *Hint*: using proportionality, $p(t \mid s_1, s_2, y) \propto p(y \mid t)p(t \mid s_1, s_2)$; the second factor is Gaussian while the first one is nonzero only when y and t have the same sign; the result should be a *Truncated Gaussian*.
- $p(y = 1)$ the marginal probability that Player 1 wins the game. textitHint: $p(y = 1) = p(t > 0)$ where t is the Gaussian random variable obtained by marginalizing out s_1 and s_2 from $p(t, s_1, s_2)$.

[2] https://www.microsoft.com/en-us/research/project/trueskill-ranking-system/

[3] Herbrich, Minka, Graepel, "TrueSkill(TM): A Bayesian Skill Rating System," Advances in Neural Information Processing Systems, January 2007, MIT Press

[4] http://www.moserware.com/2010/03/computing-your-skill.html

Q.4 Gibbs Sampler (4 points)

In this question, you 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.
Using the results of Q.3, implement a Gibbs sampler that targets the posterior distribution $p(s_1, s_2 \mid y)$ of the skills given the result of one game between two players. Consider the same prior distributions for the two players ($p(s_1)$ and $p(s_2)$ have the same hyperparameters). *Hint*: the function *scipy.stats.truncnorm* may be useful.

- Only the stationary distribution of the Gibbs sampler represents the posterior distribution of the skills. Initial samples will depend on the initial condition of the chain and may be far away from the stationary distribution and should be discarded (the so called burn-in).

  Plot the samples of the posterior distributions generated by the Gibbs sampler when $y = 1$ (Player 1 wins). What seems to be a reasonable value for the burn-in? Comment on your choice of burn-in. Then, re-run the experiment. Was your burn-in a good choice also for the second run? Comment.

- To recover the Trueskill representation of skills as Gaussian random variables, we need to transform the samples drawn from the Gibbs sampler into Gaussian distributions. Implement a function that uses the mean and covariance of the samples drawn by the Gibbs sampler to find a Gaussian approximation of the posterior distribution of the skills.

- When deciding how many samples to use (after burn-in), there is a tradeoff between accuracy of the estimate and computational time. Plot the histogram of the samples generated (after burn-in) together with the fitted Gaussian posterior for at least four (4) different numbers of samples and report the time required to draw the samples. What is a reasonable number of samples? Comment on your choice.

- Compare the prior $p(s_1)$ with the Gaussian approximation of the posterior $p(s_1 \mid y = 1)$; similarly compare $p(s_2)$ with $p(s_2 \mid y = 1)$. What has happened? Comment.

Q.5 Assumed Density Filtering (4 points)

The Gibbs sampler from Q.4 processes the result of one match to give a posterior distribution of the skills given the match. We can use this posterior distribution as a prior for the next match in what is commonly known as *assumed density filtering (ADF)*. In this way, we can process a stream of different matches between the players, each time using the posterior distribution of the previous match as the prior for the current match.

- Use ADF with Gibbs sampling to process the matches in the *SerieA* dataset and estimate the skill of all the teams in the dataset (each team is one Player with an associated skill

si). Note that there are draws in the dataset! For now, skip these matches and suppose that they leave the skill unchanged for both players. What is the final ranking? Present the results in a suitable way. How can you interpret the variance of the final skills?

- Change the order of the matches in the *SerieA* dataset at random and re-run ADF. Does the result change? Why?

Q.6 Using the model for predictions (4 points)

Being a Bayesian model, Trueskill can be used to answer probabilistic questions such as "*what is the probability that Player 1 wins against Player 2?*". In particular, we can make predictions based on these probabilities. Invent a prediction function that returns $+1$ if the Player 1 will win and -1 if the Player 2 will win (note that the result should be deterministic and that you are only allowed to use the information of previous matches in this prediction!). Using the *SerieA* dataset, compute the *one-step-ahead* predictions of the results based on the model and compare with the actual results for all matches; for each match

1. compute the prediction and compare to the result in the dataset,
2. update the model using the new match,
3. iterate over the whole dataset.

Report the prediction rate

$$r = \frac{\text{number of correct guesses}}{\text{number of total guesses}}$$

of your method over the whole dataset. Is it better than random guessing?

Q.7 Factor graph (4 points)

Draw the Factor graph of the model and identify the messages. Write explicit forms for the messages in the model and define a protocol for message passing. *Hints*: The final graph should have 4 variable nodes and 4 factor nodes.

Q.8 A message-passing algorithm (4 points)

Use moment-matching to approximate the truncated Gaussian marginal with a Gaussian marginal. Implement the message-passing algorithm to compute the posterior distribution of the skills given the result of one game between two players. Plot the posteriors computed with message passing. In the same plot, show also the histogram and the Gaussian approximation from Gibbs sampling. *Hint*: The posteriors using moment-matching and Gibbs sampling should look very similar.

Q.9 Your own data (4 points)

Test the Trueskill methods you have developed on a dataset of your own choosing. Some suggestions are other team sports (e.g., hockey, basketball, rugby), two-player sports (e.g. tennis), tabletop games, or computer games. Include details about the source of the data and any pre-processing you have done. Be creative!

Q.10 Open-ended project extension (4 points)

In implementing Trueskill, you may have noticed that both the model and the inference algorithms have limitations. In this part of the project you are free to define your own extensions of the model and of the method in some way you find interesting. For example, the model may be tuned more to the specific competition considered:

- Model draws?
- Does one of the players have an advantage *a-priori* (e.g., white in chess)?
- Should skill change over time?

Alternatively, the algorithm may be improved upon in various ways:

- Process more matches together to improve performance?
- Tune the hyperparameters differently/better?
- Use the scores of the matches to improve the estimates?
- Use data from other datasets to improve the predictions?
- Implement a "smarter" prediction function?

Implement at least one extension on the model and test it on the datasets. Do the results change? Can you get better prediction rates?

Instructions for how to write your report and submission:

Hand in your solutions (including figures/plots) to all the questions in this document as one PDF file of a maximum of six pages. Bibliography, simulation code, and additional material can be added as an appendix and the first six pages should contain all important derivations and results, as well as design choices made in the implementation. Your Python 3 Jupyter notebook code should be submitted in a separate ZIP file and all datasets required should be submitted together with the code.