Razvan Kusztos
Girton College
rek43@cam.ac.uk

Computer Science Tripos, Part III

LE49: Probabilistic Machine Learning

Dota2 Player Ranking Using Trueskill$^{\text{TM}}$

Report

27 February 2018

Word count: 2630

1 Introduction

Player ranking has many important uses in competitive sports or games. Firstly, they provide a basis to matchmaking systems, improving the experiences of both players and watchers. They create balanced games for the participants, while keeping them dynamic for audiences. Secondly, ranking players can be used to determine tournament qualification (such as the access to ranked matches in Dota2). Lastly, ranking triggers the human ingrained response to competition, attracting players to stay for longer, thus benefiting the industry.

The essence of ranking algorithms is determining a real skill score for each player, such that its magnitude is representative of the player's future performances. In a probabilistic setting, these skills must also be depictive of the probability of future games' outcome.

One of the most influential statistical ranking systems is due to Arpad Elo (1959) [1], and has been used extensively in chess ranking [2]. In the online game scene, it has been used in World of Warcraft's PvP[1] matchmaking [2].

The core of the *Elo* algorithm is that each player's game performance is normally distributed around their skill, $p \sim \mathcal{N}(skill, \beta^2)$. The probability that one player wins is then proportional to the probability that its performance is higher than its adversary's.

$$P(p_1 > p_2 | skill_1, skill_2) = \Phi\left(\frac{s_1 - s_2}{\sqrt{2}\beta}\right),$$

where Φ is the cumulative density of a normalised Gaussian distribution.

The *Glicko* rating systems [3], [4]. were introduced as improvements to *Elo*. They replace the constant variance parameter used by the *Elo* system by shifting to a Bayesian framework. In this setting, the player skill itself is modelled as a sample from a prior distribution and, with each game, the uncertainty (variance) we hold about their skill decreases.

Unfortunately *Glicko* is limited to single player games. Microsoft TrueSkill$^{\text{TM}}$was developed to address this limitation, by allowing individuals' skills to be updated given the team's performance, as well as considering games with more than 2 competing entities.

An important question that needs yet to be addressed is how to enhance these existing systems by incorporating other information about the players, rather than just win vs. loss information. For instance, in chess, one might incorporate knowledge of whether the player controls the black or the white. A way in which this problem has been addressed is due to *Zhang et al.* [11]. They suggested that a skill, rather than being a single real value, could be modelled as a vector of skill components: *a virtual team of [the player's] various skills.* Of particular interest to the current work is that they validate their algorithm on *DotA*, a precursor of the game I will discuss in this work.

In the rest of this report, I will introduce factor graphs and the message passing algorithms, necessary for understanding the *TrueSkill*^{TM} original implementation. I will then describe the assumptions made in the *TrueSkill*^{TM} model. Next, I will introduce the particularities of the *Dota2* game, a multiplayer online battle arena (MOBA) that has been part of the recent advent of e-sports. Lastly, I incorporate these considerations into an implementation that I evaluate on both real and synthesized datasets.

[1]player vs. player

[2]http://wowwiki.wikia.com/wiki/Arena_personal_rating

2 Implementation

2.1 Factor Graphs

The *Factor Graph* is a graph-based graphical model. Like *Markov Random Fields* or *Bayesian Networks*, it is used to model multivariate probability distribution while making explicit the conditional independence assumptions of the random variables. The *Factor Graph* is based on a bipartite undirected graph with two types of nodes: **variables** (depicted as circles), and **factors** (depicted as squares). The factors represent functorial combinations of the neighbouring variables. The bipartite structure is achieved by enforcing the constraint that edges can only connect nodes of different types.

Another way to see it is that the factor graph expresses a factorisation of a joint distribution:

$$g(x_1, x_2, x_3, ..., x_n) = \sum_{j \in J} f_j(\mathbf{x}_j),$$

where J is the set of factors and $\mathbf{x}_j$ is the list of variables that are adjoint to the node representing the factor f_j.

We are generally interested in performing marginalisation. This is usually required in practice, for example when computing likelyhood functions.

$$g_i(x_i) = \sum_{x_j, j \neq i} g(x_1, x_2, ..., x_n)$$

2.2 Message Passing

Marginalisation approximations are performed using the so called *sum-product algorithm*. Also known as *Belief propagation* [9], this algorithm was designed to be a general, easily parallelisable, exact inference of graphical model based on trees. However, it is known to provide approximations on general graphs [10], including cyclic graphs [8].

It has been shown that this algorithm is an instance of the *Expectation propagation* [7] on factor graphs. The algorithm approximates marginals by propagating *messages* along the edges of the factor graphs. In the context of *Expectation propagation*, messages are nothing but factor approximations obtained in the main loop of the algorithm [7].

Concretely, message passing can be defined through the following equations:

$$\begin{aligned}
p(v_i) &= \prod_{f \in F_{v_i}} m_{f \to v_i}(v_i) \\
m_{f \to v_i}(v_i) &= \int ... \int f(\mathbf{v}) \prod_{i \neq j} m_{v_j \to f}(v_i) dv_j \\
m_{v_i \to f}(v_i) &= \frac{p(v_i)}{m_{f \to v_i}}
\end{aligned}$$

In the previous, f refers to a factor and v to a variable node. I use the standard notation $m_{f \to v}$ to represent the message from node f to node v.

2.3 TrueSkill Implementation

For simplification, I will introduce this problem with no-draw games consisting of two teams, each with two players.

We model all players' skills as being drawn from a prior normal distribution. This encodes the knowledge we have about the players before beginning to analyse their game outcomes. Then, we calculate the team performance as being the sum of the members' skills with an added *Gaussian noise*, representing a *luck* element. Lastly, the probability of the team winning is given by the probability the first team's performance is higher than the second team's performance

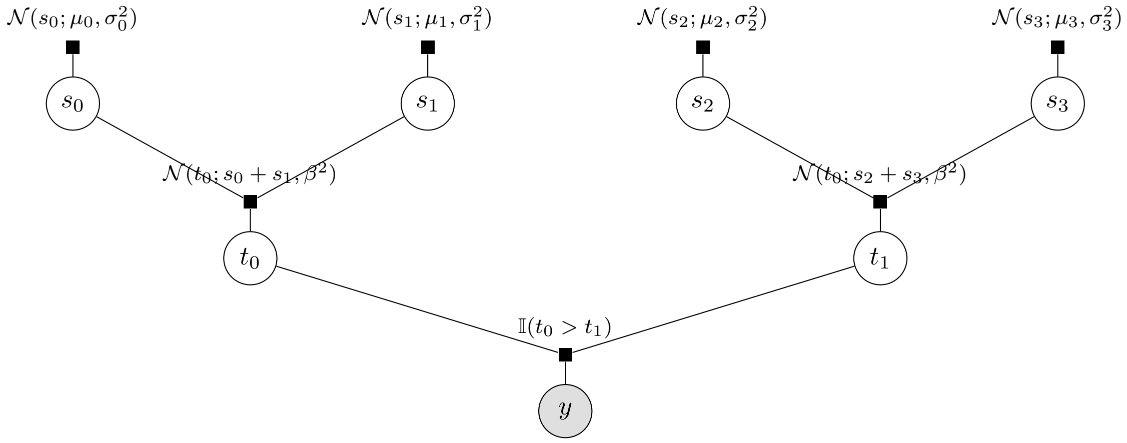
$\mathcal{N}(s_0; \mu_0, \sigma_0^2)$
$\mathcal{N}(s_1; \mu_1, \sigma_1^2)$
$\mathcal{N}(s_2; \mu_2, \sigma_2^2)$
$\mathcal{N}(s_3; \mu_3, \sigma_3^2)$
s_0
s_1
s_2
s_3
$\mathcal{N}(t_0; s_0 + s_1, \beta^2)$
$\mathcal{N}(t_0; s_2 + s_3, \beta^2)$
t_0
t_1
$\mathbb{I}(t_0 > t_1)$
y

Figure 1: Original TrueSkill$^{\text{TM}}$model for a game with two teams of two members each. Note that, in the above, I only show a small part of the full graph, related to a single game. In reality, we have N players and G games.

We are interested in computing the marginal distribution, $\mathbb{P}(s_i|\mathbf{s}, \mathbf{t}, \mathbf{y})$ for each player i. We can do these by applying the sum product algorithm. The derivation of the message passing equation is an exercise in *Gaussian* distribution manipulations [6]. Notice however that we need to make two further approximations.

Firstly, because of the presence of cyclic structures (since the same player can take part in multiple games), we will have to make use of the *loopy belief propagation* ideas and iterate a few times through the steps of the algorithm.

Secondly, the efficiency of *expectation propagation* is due to the small amount of moments that we need to to propagate through the network. In the case of the *normal distribution*, two moments suffice (μ and σ). However, for the factor computing $\mathbb{I}(t_0 > t_1)$, we can apply *moment matching* to approximate its result to a *Gaussian*. This reduces to applying *moment matching* on a truncated *Gaussian*, say: $t \sim \frac{1}{Z_t}\delta(t > 0)\mathcal{N}(t; \mu, \sigma^2)$. By using the notation:

$$\Psi(z) = \frac{\mathcal{N}(z; 0, 1)}{\Phi(z)}$$
$$\Lambda(z) = \Psi(z)(\Psi(z) + z),$$

We can approximate the distribution of t by a Gaussian whose moments are [3]:

$$\mathbb{E}(t) = \mu + \sigma\Psi(\frac{\mu}{\sigma})$$
$$\text{Var}(t) = \sigma^2(1 - \Lambda(\frac{\mu}{\sigma}))$$

[3] http://mlg.eng.cam.ac.uk/teaching/4f13/1718

Notice that under this model, the only factors that are open to modification are the prior distribution of the skills and the outcome of the game. There is no other way to incorporate new assumptions that are valid in particular games we apply this model to. In the following, I will present a few simple modification that could better fit the game of DotA2, as well as validate them on synthesized and real datasets.

2.4 DotA2

Dota2 (Defense of the Ancients 2) is a Multiplayer Online Battle Arena(MOBA). The *Arena* is a square map containing two adversary bases on opposite corners, united by three lanes heavily guarded by turrets. There are two teams, referred to as the *Radiance* and *Dire*, each of which is composed of five human players. The goal of a team is to infiltrate the enemy base and destroy their *Ancient*, a static structure. Upon achieving this feat, the game terminates.

Each human player controls a *hero* which they can chose before the game commences. The *heroes* are all different, requiring particular strategies. During clashes between teams, *heroes* might get killed, which confers their slayers *gold* and *experience*. Upon death, *heroes respawn* after a penalty time during which the player can only observe the game.

Although this is an incomplete description of the game[4] , it suffices for understanding the metrics provided by specialised datasets, such as: duration of the game, hero name, kills, deaths, gold per minute, xp per minute etc.

2.4.1 Dota2 Ranking Dataset.

For running experiment in following sections, I will be using a *Kaggle*-hosted dataset [5], created by querying the OpenDota service [6]. This dataset contains metadata about matches (e.g. duration, geographic information), as well as some of the players that took part in it (e.g. level, experience, gold, damage dealt, the hero chosen, deaths, kills etc.). Furthermore, it contains timestamps of all the events that took part in the game, such as items bought, and even the text log. As the data is very rich, constructing a probabilistic model that integrates all these variables would require some amount of domain knowledge. Nonetheless, in the next sections, I will outline a few assumptions that we can check about the data.

2.5 Assumptions about the data

2.5.1 Matchmaking bias

The platform implements a matchmaking routine [7]. Also based on a probabilistic assumption, their algorithm assigns skill levels, called MMR - Match Making Rating. This number, originally starting at the value of 3000 is increased with every win and decreased with every loss. The difference depends on the MMR difference between the two teams.

Players will only play games with peers of a similar skill. This contradicts the independence assumption between the players.

It is worth noting that the data does not account for the entire history of all the players that occur in the present matches. This makes simple heuristics for approximating skills uninteresting, such as the simple win/loss ratio. However, we can still observe the matchmaking bias by analysing some other features of the players, such as their gold per minute or xp per minute gains. These numbers are more likely to be representative of the player's skill. A visualisation of the the bias

[4] https://dota2.gamepedia.com/Dota_2_Wiki

[5] https://www.kaggle.com/owajawa/dota-2-skill-rating-with-trueskill/data

[6] https://www.opendota.com/

[7] https://dev.dota2.com/showthread.php?t=129665

is shown in Figure 2. The relatively low shift is possibly due to a selection bias in the data, by picking all matches to be in the same league.

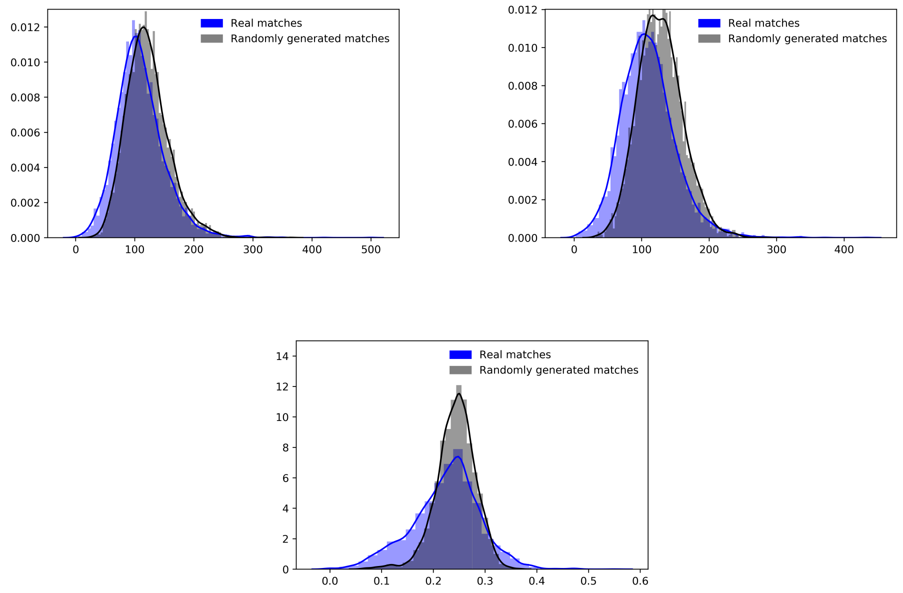

Figure 2: Graphs showing the range of values in standard deviations of player metrics, computed in the context of individual matches. The blue line corresponds to the actual data, whereas the gray line is obtained assigning the players randomly to matches, mimicking a random dataset. The first two plots show *gold per minute* and *xp per minute*, illustrating the bias in the real data. The last plot shows *win ratio*.

2.5.2 Anonymous players

A substantial amount of players choose to play anonymously, accounting to more than 99% of matches. In this case, the system does not preserve any identification that could trace back the individual from the gameplay information. This turns out to be a critical problem in modeling the calculation of *team performances*.

As a preprocessing step, I decide to ignore the matches containing teams with less than two non-anonymous players. Then, I modify the calculation of the team performances to be the mean of the skills of non-anonymous players. This is consistent with the assumption that peers are close in skills.

2.5.3 Analysis of feature importance

The feature space of the players is relatively big, given the sparsity of the data. We can therefore employ supervised techniques for picking only the most important ones. Inspired from the work of *I. Guyon et al.* [5], we can pose the problem as an instance of a supervised machine learning task and solve it using linear SVM. The relative coefficients of the features could be representative of feature importance (the original paper uses the square of these values).

The problem could be posed as a supervised learning problem, we can split the match history into *past* and *future*. Taking the features of players analysed in the *past* can be used to infer the match results in the *future*.

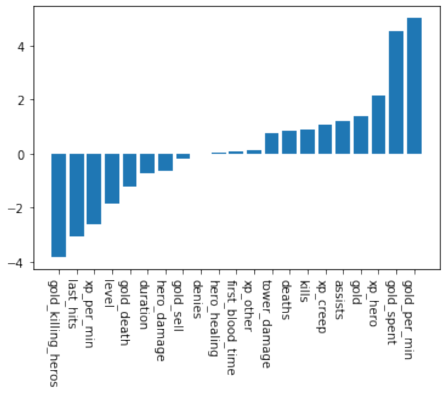

Figure 3: This plot presents the relative importance of the features, as explained in section 2.5.3. This clasifier obtained a low score, but nonetheless better than random. I tried linear SVM (51.8%) and RBF SVM (54.0%)

3 Results

3.1 Synthesized dataset

In order to validate the implementation of the algorithm, I have used a simple, synthesized dataset. This is generated by first picking the *team performance* and then forming teams of players whose skills are close in magnitude to the team performance.

Making a synthesized dataset to capture the richness of the players' features would require specialised knowledge, so this dataset is being used solely to validate implementations of the baseline algorithms.

3.2 Baseline results

An important step toward integrating the features into calculation is determining the predictive power of the wins and loss history. From the results, I deduce that the statement of the problem of ranking as a supervised machine learning task requires extra thought. During these experiments, I have split the match history into *training* (80%) and *testing* (20%). *Elo* and *Trueskill* scores were inferred for players that took part in the training matches. The reported results are the percentage of games in the testing matches whose outcome was correctly predicted, using the players' scores from the previous steps. Players whose scores were not computed in the training phase were replaced with anonymous players. The results are shown in Table 1.

In all the future results, the accuracies given are the result of averaging a set of N=5 results, to account for the random components of the algorithm.

3.3 Integrating numeric features

Although the SVM was not capable of extracting more information from the features, it was worth trying integrating this information in the probabilistic model. To this end, I suggest two possibilities:

Algorithm	**Dataset**	**Accuracy**
Linear SVM	Real	51.8%
Elo	Real	52.4%
Elo	Synth	55.4%
Trueskill (lib)	Real	51.0%
Trueskill (lib)	Synth	54.3%
Trueskill (imp)	Real	51.2%
Trueskill (imp)	Synth	55.4%

Table 1: Summary of *baseline* results. Trueskill (lib) is using the a python library(http://trueskill.org/), based on an online learning scheme via *Gaussian density filtering* [7]. My implementation is inspired from http://mlg.eng.cam.ac.uk/teaching/4f13/1718/, only made more efficient by using sparse matrices. The Elo implementation is inspired from https://www.kaggle.com/kplauritzen/elo-ratings-in-python

.

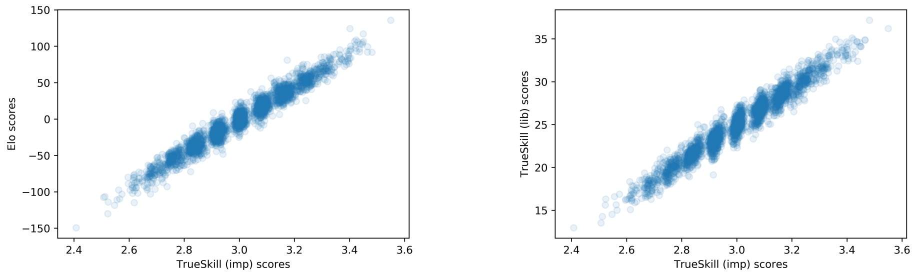

Figure 4: Figures showing the correlation between skills computed by the three presented algorithms, indicating a degree of equivalence.

- Using the relative magnitude of each players' features as part of the prior
- Training an alternative model that considers as winners the team with the bigger mean feature.

The results are presented in Table 2.

Feature	**Accuracy**
gold per minute	51.7%
kills / deaths	51.1%
(kills + assists)/deaths	51.5%
gold spent	51.9%

Feature	**Accuracy**
gold spent	49.9%
gold per min	50.0%
kills	49.9%
assists	50.9%

Table 2: Summary of the results by integrating features. On the left, I present the results when feature information is added to the prior of the skills. On the right, I show the variant of pretraining the model based on feature information, and then using the results skills as priors.

3.3.1 Integrating Hero information

In order to integrate hero information, I suggest considering players as virtual teams containing the human component and the hero. In order to simplify the model, I first calculate the *skills* of heroes, simply by replacing the player ids with the hero ids in the TrueSkill algorithm. Then, I average this value with the value of skills computed as in the baseline in Section 3.2. This final value is used for predicting future games. The results could be seen in Table 3.

Model	**Accuracy**
Baseline Trueskill	51.2%
Hero Trueskill	54.2%(±0.8%)

Table 3: Comparing the results of the baseline and those of the algorithm that integrates hero information

4 Conclusions

In this project, I attempted to illustrate benefits of integrating the richness of of player's previous gameplay information in a standard, probabilistic machine learning algorithm. In particular, the hero information seemed to be the most fruitful (Section 3.3.1).

On main point that came across during the experiments was the difficulty of posing this problem as a supervised machine learning problem. The TrueSkill algorithm under-performed in all models, including when run on synthesized datasets. There seems to be a fundamental problem in extrapolating training data to testing data, obtaining scores only marginally higher than random. However, in the case of synthesized datasets, the algorithms' scores were capable of obtaining training accuracies one standard deviation away from the scores obtained with the truth values. This could possibly hint at the need of introducing more regularisation techniques, perhaps bespoke variance terms when computing team performance for each player. More generally, exploring how to better pose this problem in a supervised setting is potentially a good start for future work.

A second problem was that due to the little amount of data and high variability of the players, the true advantage of using a probabilistic framework could not be explored. The variances were often trumping the means in absolute magnitude, making it hard to view the results in a probabilistic mindset. By exploring correlations between the players rating and their features, assuming access to more data, we could potentially in future work use variance reduction techniques, such as using control variates in a MCMC formulation of TrueSkill.

References

[1] Arpad E Elo. *The rating of chessplayers, past and present.* Arco Pub., 1978.

[2] Mark E Glickman. A comprehensive guide to chess ratings. *American Chess Journal*, 3:59–102, 1995.

[3] Mark E Glickman. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, 1999.

[4] Mark E Glickman. Example of the glicko-2 system. *Boston University*, 2012.

[5] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[6] Ralf Herbrich, Tom Minka, and Thore Graepel. TrueskillâĎć: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576, 2007.

[7] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[8] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[9] Judea Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach.* Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[10] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier, 2014.

[11] Lei Zhang, Jun Wu, Zhong-Cun Wang, and Chong-Jun Wang. A factor-based model for context-sensitive skill rating systems. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 2, pages 249–255. IEEE, 2010.