A short introduction to the Kamlbot ranking algorithm

Kolaru

July 21, 2019

1 Fundamental requirements

Let say we want to build a ranking system for 1v1 games in Tooth and Tail (legends say some people do that during their free time... madness). There are several requirements we impose on this ranking:

1. Associate a score to each player that primarily reflects their skill level.
2. Have the scores to stop changing after a while, if the skill level of each player stays the same.
3. Do not change the score of a player if they do not play.

Each of these requirements has consequences for the ranking system. The last one may seem useless, but it excludes all systems that need to update the score of everyone after each game, as for example Google PageRank algorithm (a basic algorithm to rank web pages by importance).

The requirement 2 is more (mathematically) interesting. Assume we have two players, Hopper and Archimedes. If Hopper wins a game against Archimedes, she will gain W points and if she is defeated, she will lose L points. Due to the relative player skill (mixed with random factors inherent to the game, like map generation), Hopper has a probability p to win (and thus probability $(1-p$ to lose). Therefore assuming the skill levels of both Hopper and Archimedes do not change, in average and after many games, the score difference ΔS for Hopper will be

$$\Delta S = pW - (1-p)L. \tag{1}$$

If we want the score to remain constant (as per requirement 2), we want $\Delta S = 0$. On way to achieve that is to choose

$$W = (1-p)\delta, \tag{2}$$
$$L = p\delta, \tag{3}$$

with δ being a number we can choose to be whatever we want. Using this, we get, as requested,

$$\Delta S = pW - (1-p)L \tag{4}$$
$$= p(1-p)\delta - (1-p)p\delta \tag{5}$$
$$= 0. \tag{6}$$

This is fine, but it means that to compute by how much we update Hopper score, we need to estimate Hopper's probability to beat Archimedes. That's where more advanced probability are needed, and where the TrueSkill algorithm come into play.

2 Some probability concepts

To be able to use some more advance mathematics, we first need to model what it is for Hopper to play a game against Archimedes. A very simple way of doing this is to assume each player draw a random number based on their skill level, and the one with the higher number win. The number need to be random for two reasons. First, high skill level means that a player makes a lot of good play and few errors. How much of either exactly and in which order however is impossible to predict, but can still have a decisive impact on the game outcome. That's why even in pure skill based game like chess, the most skilled player do not always win. Second, some parameters simply

does not depend on the players, like map generation or relative deck strength.

We we call the two random numbers that determine a game's outcome *quality of play* and denote it with the symbol X. For example if for a given game we draw for Hopper $X_{\text{Hopper}} = 14.6$ and for Archimedes $X_{\text{Archi}} = 17.3$, then this models a win for Archimedes.

However this does not solve everything, because there are many different ways to draw random number. The most well known probably being the Dungeon and Dragon aptitude check: the player rolls a 20 faced dice and add their character's proficiency to the result to generate the number that will indicate whether the action is a success.

In probability a common way to draw random number is the *Gaussian distribution*, which depends on two parameters, μ determining the average value of a random number drawn following that distribution and σ determining how concentrated around the average these random numbers will be[1]. To get a feeling, there is 67% chance that a number drawn following a Gaussian distribution will falls between $\mu - \sigma$ and $\mu + \sigma$. Similarly there is 95% chance that this number fall between $\mu - 2\sigma$ and $\mu + 2\sigma$.

In other words, the closer to the average μ, the more likely a number is to being drawn, and how much more likely is determined by σ.

The reason TrueSkill uses Gaussian distribution is because they are very common and have a lot of nice properties that makes the mathematics easier (it makes them easier, but they stay complex though). There is really no overreaching reason for that choice except convenience[2] (ELO rating uses a different one fore example).

This relative mathematical simplicity, among other, let us compute the probability p that Hopper beats Archimedes, provided that we know μ and σ for both of them. The problem however is that we need to estimate two parameters (μ and σ) rather than just a score, and doing so in a smart way is far from obvious.

3 TrueSkill algorithm

The way to update μ and σ for a player uses so called *Bayesian inference*. However, this subject is beyond my actual mathematical skills. So for now let simply accept the idea it is done in a good way by the very smart people that wrote the `trueskill.py` library that the Kamlbot is using.

There is a bit more to say about it though. First of all the reason we need such advance mathematics is requirement 3. If we were able to update the level of everyone after each game, we could do something simpler. With this restriction however, it is rather tricky to update the players μ and (especially) σ in a way that reach a stable state and does not wildly oscillate.

Also, the score displayed by the bot is not μ (which is the average level of skill of the player), but rather $\mu - 3\sigma$. The reason for that is to reward consistent play (that should result in lower σ), but most importantly to have new player start with a low score that will most likely increase as they play more game.

Indeed, while new players are initially given $\mu = 25$ and $\sigma = 8.33$ (corresponding to a score of roughly 0), the σ for player having played enough games generally range from 0.8 to 1.0 which is much smaller.

With all that, the 3 requirements we laid out at the start are fulfilled: the score being $\mu - 3\sigma$ it is mainly based on a player's skill level; smart mathematics allow to have a stable ranking; and even smarter mathematics allow to make it efficient by only updating the playing players.

[1] μ and σ are generally referred to has the mean and standard deviation respectively of the distribution, while σ^2 is its variance.

[2] In fact giving the nature of the data (only results of games and no direct measurement of skill level) it is probably impossible to determine if a distribution is better than another, provided it is possible to tune the mean and variance of both.