Elo-R Draft

March 21, 2016

1 Introduction

Competitions, in the form of sports, games and examinations, have been with us since antiquity. Whether the purpose is entertainment, training, or selection for a particular role, contestants and spectators alike are interested in estimating the relative skills of contestants. Various competition formats, such as tournaments, exist to declare a champion. However, we may be interested in comparing players who rarely, if ever, play directly against one another. When there is substantial variance in a player's performance, we may be interested in an aggregate estimate of skill based on the player's entire history of games.

Skills are easiest to compare when there is a standard quantifiable objective, such as a score on a standardized test or a completion time in a race. However, most team sports, as well as games such as chess, have no such measure. Instead, a good player is simply one who can frequently win against other good players. The Elo rating system assigns a quantitative measure of skill in such a scenario. Typically, an average player is rated 1500, while a top player may exceed 2500. The scale is arbitrary, but can be used to rank players relative to one another, as well as to forecast the probable outcome of future matches.

To make things concrete, let's label the players $1, \ldots, N$. The original Elo system deals with 1v1 zero-sum games with binary (win/lose) outcomes. Player i has a rating r_i, a real number that should be higher the more skilled they are. After two players compete, the loser gives some number of points to the winner, so that the sum of ratings is conserved. The number of points transferred depends on how "surprising" the outcome was. To determine the level of surprise, we need some model of the likelihood of match outcomes between players of known ratings. The logistic function is a common and well-studied choice: in this model, the outcome probability decays exponentially with the rating difference between the losing and winning player.

Elo, and variants such as Glicko, are popular choices in the 1v1 setting where each match has a winner and loser. While Elo is not the most accurate predictor of future outcomes, it's accurate enough and has other good properties: the incremental update following each match is simple, fast, and only affects the ratings of participants. Now let's consider a more general setting in which an arbitrary number of players compete simultaneously at a task. The outcome is a ranking, i.e., a total order among all players in the match. That is, we recognize which player came in 1st place, 2nd, 3rd, etc. We ignore the relative margins of victory, as they depend on details of the game that may be difficult to model: if the number of players is sufficiently large, differences in ordinal ranking should encapsulate most of the information about margins of victory.

This model is used in the international programming competition websites Codeforces [1] and TopCoder [2]. Each publishes its own rating system, but without much theoretical justification to

accompany the derivations. By founding Elo-R upon a more rigorous probabilistic model, we achieve better properties in practice as well. Compared with the Codeforces system, Elo-R achieves faster convergence, more robustness against unusual performances, and less inflation. Compared with the TopCoder system, Elo-R is more robust against unusual performances, and monotonic in the sense that performing better in a match will never result in a lower rating. The non-monotonicity of TopCoder has been studied in the analysis of [3]. Furthermore, Elo-R retains simplicity and efficiency roughly on par with the other systems.

In section 2, we describe Elo-R ratings as a weighted average of past performances, with weights depending on the age of the performance and how atypical it is. In section 3, we develop a Bayesian performance model and use it to derive the Elo-R system, which turns out to match the weighted average discussed previously. Then in section 4, we fill in the remaining details and parameters. Finally, in section 5, we conclude with a brief summary of the system's properties.

2 Performance Averaging

Recall that the outcome of a match consists of a total order ranking among all participating players. To model how the outcome came about, suppose player i's true (unknown) skill is s_i. Player i's performance in the match can be viewed as a number p_i, drawn from a logistic distribution centered at s_i. If i beat j in the match rankings, a result we denote as $i \succ j$, then we know that $p_i > p_j$. Note that the event $p_i = p_j$ has probability zero. In practice, some games allow ties. In order to sidestep the issue of modeling ties separately, we will simply treat them as half a win and half a loss; such a division will be very straightforward with the formulas we'll get.

While we can't measure s_i directly, we can estimate p_i by looking at the ratings of other players and at i's position in the ranking. This suggests the following two-phase approach: from the match ranking, estimate p_i for each player. Then, update the rating r_i to a weighted average of past p_is. Note that s_i, p_i, r_i are all measured on a common scale. As we'll soon see, less weight will be put on performances that are either outdated or unusual for the player.

Let's focus on the second phase for now: having computed the latest p_i, we want to combine it with past performances of the same player to get a more precise estimate of their skill. To clarify the notation in this section, we add an extra subscript k, ranging from 1 to the number of matches M in which player i has competed, so that $p_{i,k}$ denotes the performance of player i on their kth match, and $r_{i,k}$ denotes their rating upon completion of the kth match. The sample mean is perhaps the most obvious statistic to take. That is, let

$$r_{i,M} = \frac{1}{M}\sum_k p_{i,k} = \left(1 - \frac{1}{M}\right) r_{i,M-1} + \frac{1}{M} p_{i,M} \tag{1}$$

In addition to being simple, this method has the advantage of being a function of only the number of matches M, the previous rating $r_{i,M-1}$ and the new performance $p_{i,M}$, so there is no need to remember all of the $p_{i,k}$. We say that such a method is *memoryless*.

However, there are two key problems here. The first is that, as a player accumulates more experience, the relative weight of a new performance approaches zero; if an established player improves their skill, it will take a long time for their rating to reflect it. This is because all performances are weighted equally, even those which are too outdated to accurately reflect a player's current skill. The method would be more suitable if a player's true skill were an eternal constant,

never changing, and each $p_{i,k}$ were a measurement of this constant. To properly account for a player's lifelong evolution, we should place less weight on older performances.

The second problem with the sample mean is lack of robustness against outliers. A single extreme performance, perhaps due to illness or a power failure (if the competition takes place over a web server), will drastically affect a player's rating. Thus, we wish to reduce the weight of outliers.

Before we proceed, note that the above equation can be rewritten as

$$\sum_k w_k(p_{i,k} - r_{i,M}) = 0 \tag{2}$$

where $w_k = 1$ for all k. We say that $r_{i,M}$ is a *weighted mean* of the elements $\{p_{i,k}\}_{k=1}^M$ with corresponding weights $\{w_k\}_{k=1}^M$. As a simplifying assumption, let's imagine the $p_{i,k}$ are independent measurements of the present true skill $s_{i,k}$, with Gaussian (normal) noise variances of τ_k^2. If we treat $p_{i,1}$ as the prior, then our posterior belief on the true skill is a normal distribution centered at the weighted mean $r_{i,M}$ given by the following equation:

$$\sum_k \frac{1}{\tau_k^2}(p_{i,k} - r_{i,M}) = 0 \tag{3}$$

All we did here is set $w_k = 1/\tau_k^2$. If the w_ks (or equivalently, the τ_ks) form a geometric sequence, we maintain the memoryless property. We'll derive the weights later; for now, it's simplest to imagine them decreasing by a constant ratio for each match backward in time.

In order to bound the effect of any individual outlier, we also require that $w_k|p_{i,k}|$ approach a positive constant as $p_{i,k}$ approaches $\pm\infty$. We still want to treat non-outliers in essentially the same way as before, so their weight should be close to $1/\tau_k^2$ for performances close to the weighted mean, which is $r_{i,M}$ by definition. To meet these criteria, we opt for

$$w_k = \frac{1}{\tau_k(p_{i,k} - r_{i,M})} \tanh \frac{p_{i,k} - r_{i,M}}{\tau_k} \tag{4}$$

This choice will be justified by a proper derivation later: essentially, it comes from modeling performances according to the heavy-tailed logistic distribution in place of the Gaussian noise discussed previously. Heavier tails are more forgiving of outliers. Substituting (4) into (2), the equation defining $r_{i,M}$ becomes:

$$\sum_k \frac{1}{\tau_k} \tanh \frac{p_{i,k} - r_{i,M}}{\tau_k} = 0 \tag{5}$$

It is monotonic in $r_{i,M}$, so it can be solved using binary search. A straightforward linear approximation shows that in the limit as the performances vary much less than any of the τ_k, the rating behaves like a mean with weights $1/\tau_k^2$. In the limit as the performances vary much more than any of the τ_k, the rating behaves like a median with weights $1/\tau_k$. While this equation is not memoryless, all τ_k corresponding to old performances will be very large, so their terms will behave like their linear approximations. Any sum of linear terms is itself linear, so we can merge the old terms. $p_{i,0}$ here is the weighted sum of the old performances, and the merged weight $1/\tau_0^2$ is the sum of the weights of these old performances. k now ranges only over indices of the new, unmerged, performances:

$$\frac{p_{i,0} - r_{i,M}}{\tau_0^2} + \sum_k \frac{1}{\tau_k} \tanh \frac{p_{i,k} - r_{i,M}}{\tau_k} = 0 \tag{6}$$

As a result, we only need to remember a small number of recent performances. Instead of growing without bound with the size of the history, the number of terms in the sum can be set to a small constant while merging the rest. If we decide to remember zero performances, it reverts to the linear memoryless method. Hence, this general formula encapsulates everything we've covered thus far. For greater accuracy, it's best to remember at least a few performances. We will see that (6) is equivalent to maximizing the product of a collection of normal and/or logistic probability density functions (pdfs). τ_k^2 corresponds roughly to the pdf's variance, although it differs by slightly different constant factors for the normal and the logistic; we choose to stick to this parametrization in order to maintain the equivalence involved in the linear approximation.

3 Approximate Bayesian Derivation

Now we derive our rating algorithm in full. If a player's true skill is s_i, it makes sense to set their rating r_i to our best estimate of s_i, given all the evidence from past matches. In general, our belief on s_i will be proportional to a product of normal and logistic pdfs. We may think of one of these pdfs as a prior belief, and the others as likelihoods of independent measurements of s_i. Given this belief distribution, the rating r_i is defined as the *maximum a posteriori* (MAP, a.k.a. posterior "mode") estimate of s_i; i.e., the point at which the belief density is maximized.

Recall the competition model: each player i performs at level p_i, drawn independently from a logistic distribution centered at s_i. The ranking we observe is a total ordering on the players based on their performances: i outranks j, written $i \succ j$, iff $p_i > p_j$.

Fix i, the player whose rating we presently compute. Let e be the evidence consisting of, for each j, whether $i \succ j$ or $i \prec j$. That is, we ignore relations like $j \succ k$ which affect the relative ordering of player pairs that don't include i. Our goal is to approximate the posterior distribution of s_i given e:

$$f(s_i \mid e) \propto f(s_i) \Pr(e \mid s_i) = f(s_i) \int \Pr(e \mid p_i) f(p_i \mid s_i) dp_i \tag{7}$$

Since the integral is hard to evaluate, we take a leap of faith here and treat $\Pr(e \mid p_i)$ as a (possibly scaled) delta-function that spikes at the MAP estimate of p_i. This is justified as $N \to \infty$, because the evidence e would overwhelmingly concentrate p_i into a narrow range. Having fixed p_i, the previous equation simplifies to

$$f(s_i \mid e) \propto f(s_i) f(p_i \mid s_i) \tag{8}$$

Our update algorithm thus divides into two phases: first, it must determine the player's performance p_i in the match. In the second phase, p_i is used to update the belief distribution on s_i, which is then compactly summarized by finding the point r_i that achieves its maximum.

3.1 Performance Estimation

To compute the MAP of p_i, we must maximize

$$f(p_i \mid e) \propto f(p_i) \Pr(e \mid p_i) \tag{9}$$

Clearly, $p_i = r_i + (s_i - r_i) + (p_i - s_i)$. The first term is a known constant: the player's current rating. We have a prior belief on s_i; we'll eventually see it has a fairly general form but, for the

purposes of this phase, we approximate it by a logistic random variable with mean r_i. We also modeled p_i as a logistic random variable centered at s_i. Hence, the second and third terms have mean zero and known variance.

If we approximate the random terms by normal random variables, then their sum is conveniently also normal with a straightforward mean and variance. However, since the terms were originally logistic, we model their sum as another logistic variable with this mean and variance. To make things concrete, the model I implemented for Codeforces has the parameters $\sigma_i = 100$ (by default, but see discussion of new players), $\delta = 250$, and $\tau_i^2 = \sigma_i^2 + \delta^2$. The corresponding logistic pdfs are:

$$f(s_i) \approx \frac{2e^{2(s_i - r_i)/\sigma_i}}{\sigma_i \left(1 + e^{2(s_i - r_i)/\sigma_i}\right)^2} \tag{10}$$

$$f(p_i \mid s_i) = \frac{2e^{2(p_i - s_i)/\delta}}{\delta \left(1 + e^{2(p_i - s_i)/\delta}\right)^2} \tag{11}$$

$$f(p_i) \approx \frac{2e^{2(p_i - r_i)/\tau_i}}{\tau_i \left(1 + e^{2(p_i - r_i)/\tau_i}\right)^2} \tag{12}$$

Under these modeling assumptions,

$$\Pr(e \mid p_i) = \prod_{j \succ i} \Pr(p_j > p_i) \prod_{j \prec i} \Pr(p_j < p_i) \tag{13}$$

$$\approx \prod_{j \succ i} \frac{1}{1 + e^{2(p_i - r_j)/\tau_j}} \prod_{j \prec i} \frac{e^{2(p_i - r_j)/\tau_j}}{1 + e^{2(p_i - r_j)/\tau_j}} \tag{14}$$

$$\propto \frac{e^{2p_i \sum_{j \prec i} 1/\tau_j}}{\prod_{j \neq i} 1 + e^{2(p_i - r_j)/\tau_j}} \tag{15}$$

Taking logarithms, there exist a constant C such that

$$C + \ln f(p_i \mid e) = \ln f(p_i) + \ln \Pr(e \mid p_i) \tag{16}$$

$$\approx \ln \frac{2}{\tau_i} + (p_i - r_i)\frac{2}{\tau_i} - 2\ln\left(1 + e^{2(p_i - r_i)/\tau_i}\right) + 2p_i \sum_{j \prec i} \frac{1}{\tau_j} - \sum_{j \neq i} \ln\left(1 + e^{2(p_i - r_j)/\tau_j}\right) \tag{17}$$

To maximize this expression, differentiate it w.r.t. p_i, divide by 2 and set to zero:

$$0 = \frac{1}{\tau_i}\left(1 - \frac{2e^{2(p_i - r_i)/\tau_i}}{1 + e^{2(p_i - r_i)/\tau_i}}\right) + \sum_{j \neq i} \frac{1}{\tau_j}\left(\mathbb{1}(j \prec i) - \frac{e^{2(p_i - r_j)/\tau_j}}{1 + e^{2(p_i - r_j)/\tau_j}}\right) \tag{18}$$

$$= \sum_{j \preceq i} \frac{1}{\tau_j}\left(\frac{1}{1 + e^{2(p_i - r_j)/\tau_j}}\right) - \sum_{j \succeq i} \frac{1}{\tau_j}\left(\frac{e^{2(p_i - r_j)/\tau_j}}{1 + e^{2(p_i - r_j)/\tau_j}}\right) \tag{19}$$

$$\propto \sum_{j \preceq i} \frac{1}{\tau_j}\left(\tanh \frac{p_i - r_j}{\tau_j} - 1\right) + \sum_{j \succeq i} \frac{1}{\tau_j}\left(\tanh \frac{p_i - r_j}{\tau_j} + 1\right) \tag{20}$$

Use binary search to solve for p_i. This defines the *performance* of player i in the match.

The terms in parentheses can be thought of as a measure of surprise at the outcomes between i and j: they are the probability of the outcomes opposite to what actually occurred, when the performance of player i is fixed to p_i. In addition to the actual outcomes, the sum also contains two artificial outcomes in which player i wins once and loses once against clones of itself. These regularizing outcomes come from the prior, and prevent the first- and last-place players from achieving $p_i = \pm\infty$. The equation chooses p_i in such a way that the total surprise from wins is equal to the total surprise from losses.

For a more intuitive interpretation, note that if the τ_js were all equal, this amounts to finding the performance level p_i at which one's expected rank would match player i's actual rank, clones included.

3.2 Posterior Maximization

Recall our approximation $f(s_i \mid e) \propto f(s_i)f(p_i \mid s_i)$: the posterior $f(s_i \mid e)$ takes the prior $f(s_i)$ and multiplies it by a new logistic pdf $f(p_i \mid s_i)$. It will inductively be the case that our posterior is proportional to a product of normal and logistic pdfs. Since any product of normal pdfs has a normal kernel, WLOG we can assume there is only one normal in the product:

$$f(s_i \mid e) \propto e^{-(s_i-\mu_0)^2/\tau_0^2} \prod_k \frac{e^{2(s_i-\mu_k)/\tau_k}}{\left(1+e^{2(s_i-\mu_k)/\tau_k}\right)^2} \tag{21}$$

$$C + \ln f(s_i \mid e) = -\frac{(s_i-\mu_0)^2}{\tau_0^2} + 2\sum_k \left(\frac{s_i-\mu_k}{\tau_k} - \ln(1+e^{2(s_i-\mu_k)/\tau_k})\right) \tag{22}$$

We will discuss the parameters μ_k and τ_k^2, which correspond roughly to mean and variance of the corresponding pdfs, in the next section. To maximize $\ln f(s_i \mid e)$ after being given these parameters, differentiate it w.r.t. s_i, divide by 2 and set to zero:

$$0 = \frac{\mu_0 - s_i}{\tau_0^2} + \sum_k \frac{1}{\tau_k}\left(1 - \frac{2e^{2(s_i-\mu_k)/\tau_k}}{1+e^{2(s_i-\mu_k)/\tau_k}}\right) \tag{23}$$

$$0 = \frac{\mu_0 - s_i}{\tau_0^2} + \sum_k \frac{1}{\tau_k}\tanh\frac{\mu_k - s_i}{\tau_k} \tag{24}$$

Solve for s_i using binary search: this is the MAP estimate we will use as the new rating r_i. When the μ_ks are the performances of player i in previous matches, this is identical to the weighted mean from equation (6). If time and memory are a concern, we can apply the same trick to merge old tanh terms (i.e. indices k for which τ_k is large) into the normal term.

If all but the last M matches for each player are merged, and all binary searches are performed to an accuracy of ϵ, then a match with N players is processed in $O((N^2 + NM)\log(1/\epsilon))$ time. The ratings accumulate $O(\epsilon)$ numerical error per match, and likely a lot less in the long run due to statistical averaging. The Elo-R update is summarized in Algorithm 1.

4 Noise, Uncertainty, and Evolution

We now know how to compute p_i from a match's rankings, and how to compute r_i from the general form of the posterior belief over s_i. It remains to link these steps by building the posterior belief corresponding to a history of p_is. For new players, we can consider assigning either a logistic or normal prior. Since we want fast convergence, but also believe that extremely high ratings should take time to earn, we opt for a normal prior with $\mu_0 = 1500$ and $\tau_0 = 350$. We can think of this as the "posterior" belief in any player's skill upon completing zero matches.

In the general case, we wish to go from the posterior distribution after $M-1$ matches, which we'll denote by $f'(s_i)$; to the prior distribution just before the Mth match, denoted by $f(s_i)$; and finally to the posterior distribution after the Mth match, denoted by $f(s_i \mid e)$. Here e denotes the evidence from the Mth match.

As previously discussed, we need to model changes in a player's skill between matches. Although $f'(s_i)$ and $f(s_i)$ are based upon the same body of evidence, they are not identical: $f(s_i)$ has more uncertainty because s_i was changed by a noise term in the interim between the two matches. Once we have $f(s_i)$, estimating $f(s_i \mid e)$ is a simple matter of multiplying $f(s_i)$ by $f(p_i \mid s_i)$. The latter is a logistic term with $\mu_k = p_i,\ \tau_k = \delta$, where p_i is computed as above and δ is a specified constant.

How should we model the increase in uncertainty between $f'(s_i)$ and $f(s_i)$? It can be shown that a product of normal pdfs with variances $\{\sigma_k^2\}$ corresponds to a normal with variance $1/(\sum_k 1/\sigma_k^2)$. While general products of probability distributions don't have this property, let's define by analogy the *uncertainty* of our normal-logistic product beliefs to be $1/(\sum_k 1/\tau_k^2)$. Recall when we introduced the constant σ_i to represent the uncertainty in our belief. With our new definition of uncertainty, σ_i is no longer a constant but is instead given by:

$$\frac{1}{\sigma_i^2} = \sum_k \frac{1}{\tau_k^2} \tag{25}$$

When we multiply $f(s_i)$ by the logistic pdf $f(p_i \mid s_i)$, the latter corresponds to new information about s_i. Here, the term *information* refers to the reciprocal of uncertainty. If we continue to add information, the uncertainty will tend toward zero. As discussed in the first section, we must prevent this in order to respond to a player's evolution over time. To this end, let's say s_i changes between matches by a noise term centered at 0 with uncertainty η^2. If we want the amount of noise to increase with physical time, we could model it as Brownian motion. I chose instead to add a constant lump term right before each match.

Thus, s_i after the Mth match is equal to s_i before the match, plus a noise term. Deriving $f(s_i)$ from $f'(s_i)$ amounts to accounting for this noise. A crude way to approximate its effect would be to increase all of the τ_k. If we want the weight of old matches to decay exponentially in some natural limiting cases, then it makes sense to multiply all of the τ_k by the same constant K. Furthermore, in analogy to additivity of variances in response to a random noise offset, K should be chosen in such a way that the belief uncertainty is increased by η^2. Thus, if $f'(s_i)$ has a factor with uncertainty τ_k, then $f(s_i)$ has the same factor but with its uncertainty changed to $K\tau_k^2$ where:

$$\frac{1}{\sum_k 1/(K\tau_k^2)} = \frac{1}{\sum_k 1/\tau_k^2} + \eta^2 \tag{26}$$

$$\text{Therefore, } K = 1 + \eta^2 \sum_k 1/\tau_k^2 \tag{27}$$

The noise constant η is chosen such that σ_i approaches the limiting constant σ^* as the number of matches goes to infinity. Setting $1/\sigma_i^2$ to $1/\sigma^*$ both before and after the match yields the fixpoint equation:

$$\frac{1}{\sigma^{*2}} = \frac{1}{\sigma^{*2} + \eta^2} + \frac{1}{\delta^2} \tag{28}$$

$$\text{Therefore, } \eta^2 = \frac{1}{1/\sigma^{*2} - 1/\delta^2} - \sigma^{*2} \tag{29}$$

My Codeforces model uses $\delta = 250$ and $\sigma^* = 100$, yielding a noise of $\eta \approx 43.64$. The uncertainty σ_i for a new player is 350, which asymptotically approaches $\sigma^* = 100$ as they gain experience. In order to give low ratings for new players, we can publish $r_i - 2(\sigma_i - \sigma^*)$ for the ratings in place of r_i. Thus, a new player's published rating is $1500 - 2(350 - 100) = 1000$. As the difference term vanishes, an experienced player's published rating would be very close to r_i.

Algorithm 1 *update*(i)

$K := 1 + \eta^2/\sigma_i^2$
for all k **do**
  $\tau_k := \sqrt{K}\tau_k$
end for
$p_i :=$ SOLVE $\sum_{j \preceq i} \frac{1}{\tau_j}\left(\tanh \frac{p_i - r_j}{\tau_j} - 1\right) + \sum_{j \succeq i} \frac{1}{\tau_j}\left(\tanh \frac{p_i - r_j}{\tau_j} + 1\right) = 0$
Append $(\mu = p_i, \tau = \delta)$ to the list of pdfs in the belief product
$r_i :=$ SOLVE $\frac{\mu_0 - r_i}{\tau_0^2} + \sum_k \frac{1}{\tau_k} \tanh \frac{\mu_k - r_i}{\tau_k} = 0$
$\sigma_i := 1/\sqrt{\sum_k 1/\tau_k^2}$

5 Properties, Comparisons and Conclusions

TODO.

The rating is a monotonic function of the list of past performances.

If you swap two performances in the list, the rating goes up if the better performance moves forward in time, and down if the better performance moves backward.

The performance p_i is measured in the same units as rating, and has the property that within a given contest, a higher ranking contestant has higher p_i than a lower ranking one. (in case of ties, a higher-rated contestant has slightly higher p_i)

References

[1] Codeforces. http://codeforces.com/blog/entry/20762.

[2] Topcoder algorithm rating system. http://cshelp.wpengine.com/data-science/srm-and-mm-rating-systems/algorithm-rating-system.

[3] RNDr Michal Forišek. Theoretical and practical aspects of programming contest ratings. 2009.