A Modelling Approach to Rating Players

David Barber
University College London

November 9, 2017

1 Elo System

We want to associate with a player a number s that can represent the strength (or skill) of the player. In the Elo system, the strengths s_A and s_B of two players A and B are related to how likely it is that player A will beat player B (we are considering here games with only win/lose outcomes):

$$p(A \triangleright B|s_A, s_B) = \phi(s_A - s_B)$$

where $\phi(x)$ is a monotonically increasing function that maps the strength difference to a real value between 0 and 1. In his original work, Elo used the cumulative Gaussian probability density function (also called the 'error function')

$$\text{erf}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt$$

although it is also common to use the logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Using such 'sigmoidal' (s-shaped) functions means that, as the relative strength $s_A - s_B$ increases, so does the probability that A will beat B.

1.1 A Tournament version

Let's imagine that A and B play three games. In game 1, player A wins, in game 2 player B wins and in game 3 player A wins. If we knew the strengths s_A and s_B, what is the probability that these results would arise? Assuming that the results are independent (given the strengths) this is

$$p(\mathcal{D}|s_A, s_B) = p(A \triangleright B|s_A, s_B)p(B \triangleright A|s_A, s_B)p(A \triangleright B|s_A, s_B) = p(A \triangleright B|s_A, s_B)^2(1 - p(A \triangleright B|s_A, s_B))$$

where $\mathcal{D}$ represents the games outcome data.

More generally, if A beats B $N_{A \triangleright B}$ times and B beats A $N_{B \triangleright A}$ times we have

$$p(\mathcal{D}|s_A, s_B) = (\phi(s_A - s_B))^{N_{A \triangleright B}} (1 - \phi(s_A - s_B))^{N_{B \triangleright A}}$$

In a maximum likelihood approach we would set s_A and s_B to those values that maximise $p(\mathcal{D}|s_A, s_B)$. This is the same as

$$\underset{s_A, s_B}{\text{argmax}} \ \log p(\mathcal{D}|s_A, s_B)$$

where

$$\log p(\mathcal{D}|s_A, s_B) = N_{A \triangleright B} \log \phi(s_A - s_B) + N_{B \triangleright A} \log (1 - \phi(s_A - s_B))$$

In general, there is no closed form solution to this and we need to use iterative optimisation methods to find the maximum likelihood strengths.

1.1.1 Gradient optimisation

The gradient of the log likelihood wrt s_A is given

$$\frac{\partial}{\partial s_A}\log p(\mathcal{D}|s_A,s_B) = N_{A\triangleright B}\frac{\phi'(s_A-s_B)}{\phi(s_A-s_B)} - N_{B\triangleright A}\frac{\phi'(s_A-s_B)}{1-\phi(s_A-s_B)}$$

where $\phi'(x)$ is the derivative of $\phi(x)$ and

$$\frac{\partial}{\partial s_B}\log p(\mathcal{D}|s_A,s_B) = -N_{A\triangleright B}\frac{\phi'(s_A-s_B)}{\phi(s_A-s_B)} + N_{B\triangleright A}\frac{\phi'(s_A-s_B)}{1-\phi(s_A-s_B)} = -\frac{\partial}{\partial s_A}\log p(\mathcal{D}|s_A,s_B)$$

A simple gradient ascent scheme would then update

$$s_A^{new} = s_A + \epsilon\frac{\partial}{\partial s_A}\log p(\mathcal{D}|s_A,s_B), s_B^{new} = s_B + \epsilon\frac{\partial}{\partial s_B}\log p(\mathcal{D}|s_A,s_B)$$

where ϵ is the learning rate. These equations would be iterated to convergence.

If there were three players, A, B, C we would have

$$\begin{aligned}\log p(\mathcal{D}|s_A,s_B,s_C) =& N_{A\triangleright B}\log\phi(s_A-s_B) + N_{B\triangleright A}\log\left(1-\phi(s_A-s_B)\right)\\ &+ N_{A\triangleright C}\log\phi(s_A-s_C) + N_{C\triangleright A}\log\left(1-\phi(s_A-s_C)\right)\\ &+ N_{B\triangleright C}\log\phi(s_B-s_C) + N_{C\triangleright B}\log\left(1-\phi(s_B-s_C)\right)\end{aligned}$$

Gradient based maximum likelihood training for s_A, s_B and s_C would follow then similarly to the above.

1.2 An online version

In the standard Elo case, the assumption is that game results arrive in an online fashion and that we wish to update the strengths of the two players as soon as a result is announced. In this case we want to make an adjustment to our current strengths. One way to think about this is to imagine that there is a dataset of results and that we simply sample a result from this set (this is an extreme form of the minibatch setting in which the minibatch contains a single game result). Let's assume that we sample a game result that A beats B. Then we would update the strengths to increase $p(A\triangleright B)$:

$$s_A^{new} = s_A + \epsilon\frac{\phi'(s_A-s_B)}{\phi(s_A-s_B)}, \qquad s_B^{new} = s_B - \epsilon\frac{\phi'(s_A-s_B)}{\phi(s_A-s_B)}$$

For a specific choice of ϵ this is the Elo system. After each game outcome, the two players skills are updated according to the above formula.

Note that for the logistic sigmoid function $\sigma'(x) = \sigma(x)(1-\sigma(x))$ so that the above becomes simply

$$s_A^{new} = s_A + \epsilon(1-\sigma(s_A-s_B)), \qquad s_B^{new} = s_B - \epsilon(1-\sigma(s_A-s_B))$$

2 A Bayesian Approach

One criticism of the Elo system is that it doesn't account for uncertainty in estimating the strengths of the players[1]. If, in a tournament between A and B, only a few games have been played we would expect to have little certainty about the strengths of the players. On the other hand, if there are a great number of games between A and B in the tournament, then we would expect to be more confident about the skill levels of the players.

[1] Glicko, however, does handle uncertainty in the skill rating `https://en.wikipedia.org/wiki/Glicko_rating_system`

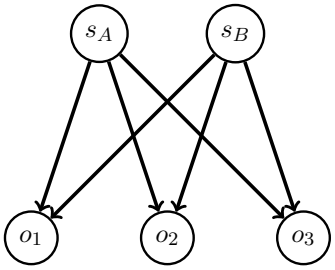
More generally, we are interested in the posterior

$$p(s_A, s_B|\mathcal{D}) = \frac{p(\mathcal{D}|s_A, s_B)p(s_A, s_B)}{p(\mathcal{D})}$$

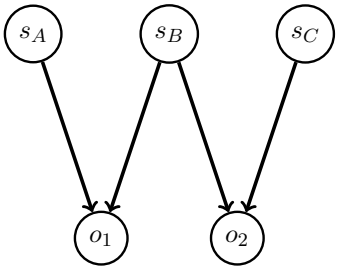
It seems reasonable to assume that, a priori, the skill levels are independent (and from the same distribution)

$$p(s_A, s_B) = p(s_A)p(s_B)$$

As a Belief Network, for game outcomes $o_1 = A \triangleright B$, $o_2 = B \triangleright A$, $o_3 = A \triangleright B$ we have:



Similarly, if we have a tournament between A, B, C and the results are $o_1 = A \triangleright B$, $o_2 = B \triangleright C$ we would have



For a tournament between P players we would have a bipartite graph with the nodes in the upper part being the P strengths of the players and the nodes in the bottom part would represent the results of all game outcomes. This would then give a posterior distribution

$$p(s_1, \ldots, s_P|\mathcal{D})$$

We could then use this distribution to make inferences. For example, what is the probability that player i will beat player j? From the posterior we first find the marginal $p(s_i, s_j|\mathcal{D})$ (note that whilst a priori the strengths are independent, a posteriori they will generally not be) and

$$p(i \triangleright j|\mathcal{D}) = \int \phi(s_i - s_j)p(s_i, s_j|\mathcal{D})ds_i ds_j$$

In general, computing these integrals will be difficult and we will need to make approximations.

2.1 Summarising the posterior

Whilst not strictly necessary, it is useful to summarise some of the information in the posterior. In doing this we will lose some information, but this will hopefully not be too severe. A very natural way to summarise the information is to calculate the marginal skill of each player

$$p(s_i|\mathcal{D}) = \int p(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_P|\mathcal{D})ds_1 \ldots ds_{i-1}ds_{i+1} \ldots ds_P$$

In this case, rather than having a single skill level for a player, we can summarise the tournament information as a skill distribution for each player. This skill distribution then can capture the uncertainty in our understanding of the player's ability.

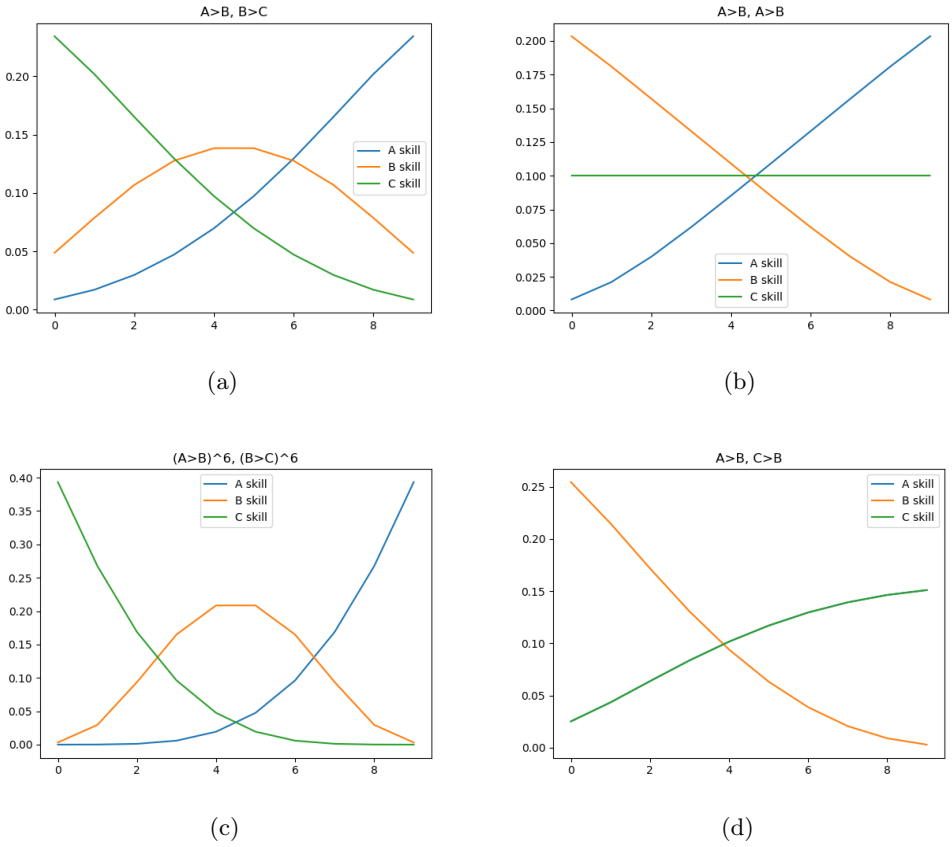


Figure 1: The marginal posterior probability distribution of skill levels. Comparing (a) with (c) we note how the posterior skill levels sharpen and we get more data. (a) $p(A \triangleright C|\mathcal{D}) = 0.897$. (b) $p(A \triangleright C|\mathcal{D}) = 0.672$. (c) $p(A \triangleright C|\mathcal{D}) = 0.994$. (d) $p(A \triangleright C|\mathcal{D}) = 0.5$. See `demoSkill.jl`.

Using this we could then make an approximation

$$p(i \triangleright j|\mathcal{D}) \approx \int \phi(s_i - s_j)p(s_i|\mathcal{D})p(s_j|\mathcal{D})ds_i ds_j$$

Note that, in general, this is not the same as

$$p(i \triangleright j|\mathcal{D}) \approx \phi(\langle s_i \rangle - \langle s_j \rangle)$$

where $\langle s_i \rangle$ is the average skill of player i.

2.2 Demonstration

In figure(1) we show some example tournaments between A, B and C and the resulting marginal skills for each player. Here we assume a discrete set of skill values from 1 to 10 for each player with a uniform prior on the skill value for each player. For this small example we can exactly calculate all quantities by summation, see `demoSkill.jl`. This Bayesian approach shows how to capture uncertainty in the skill estimate and how this will typically decrease as more games are played.

2.3 Approximating the Marginal

If the factor graph corresponding to $p(s_1, \ldots, s_P, \mathcal{D})$ is singly-connected, it is then straightforward to find the marginal $p(s_i|\mathcal{D})$ by simple message passing (the sum-product algorithm). More generally, however, the graph will not be singly-connected. Whilst we could attempt to use the Junction Tree algorithm, this will generally be impracticable since the clique sizes will be large.

Whilst the sum-product algorithm isn't guaranteed to work, we can nevertheless run the algorithm and see what happens. Provided the loop lengths in the multiply connected graph are not too short there is a reasonable chance that the sum-product algorithm will converge to a good approximation of the marginals.

This is essentially the approach that Microsoft take in their TrueSkill algorithm that estimates player ability in the online game Halo. Similar mechanisms are used by online companies to estimate your media preferences (*e.g.* movie A beats movie B) and thereby make appropriate media recommendations.

3 Software and Further Reading

https://github.com/davidbarber/Julia0p6ProbabilisticInferenceEngine
https://papers.nips.cc/paper/3079-trueskilltm-a-bayesian-skill-rating-system
http://www.mbmlbook.com

3.1 Julia Code

```
using PyPlot
A,B,C=1,2,3 # give variables a number
Nlevels=10

function demoSkill()
A,B,C=1,2,3 # give variables a number
Nlevels=10

sigmoid(x)=1./(1+exp(-x))

pLeftPlayerWins=zeros(Nlevels,Nlevels)
for levelleft=1:10
for levelright=1:10
pLeftPlayerWins[levelleft,levelright]=sigmoid(levelleft-levelright)
end
end

#prior_pot=exp.(-0.05*(Nlevels/2-linspace(0,Nlevels,Nlevels)).^2) # peaked around skill 5
prior_pot=ones(Nlevels,1) # uniform prior on skill
prior_pot=prior_pot./sum(prior_pot)
priorA=PotArray(A,prior_pot)
priorB=PotArray(B,prior_pot)
priorC=PotArray(C,prior_pot)

fAbeatsB=PotArray([A B],pLeftPlayerWins)
fBbeatsC=PotArray([B C],pLeftPlayerWins)
fAbeatsC=PotArray([A C],pLeftPlayerWins)

Data = "A>B, B>C"
pABC=fAbeatsB*fBbeatsC*priorA*priorB*priorC # p(A,B,C,Data)
plotstuff(Data,pABC,fAbeatsC)

Data = "A>B, A>B"
pABC=fAbeatsB*fAbeatsB*priorA*priorB*priorC # p(A,B,C,Data)
plotstuff(Data,pABC,fAbeatsC)

Data = "(A>B)^6, (B>C)^6"
pABC=fAbeatsB^6*fBbeatsC^6*priorA*priorB*priorC # p(A,B,C,Data)
plotstuff(Data,pABC,fAbeatsC)

Data = "A>B, C>B"
pABC=fAbeatsB*(1-fBbeatsC)*priorA*priorB*priorC # p(A,B,C,Data)
plotstuff(Data,pABC,fAbeatsC)

end
```

```
function plotstuff(Data,pABC,fAbeatsC)
pABC=pABC./sum(pABC) # normalise to get p(A,B,C|Data)
pAC=sum(pABC,B)
pAbeatsC=sum(fAbeatsC*pAC)
# What is the probability that A will beat C?
println(Data*" : probability that A beats C = $(pAbeatsC.content)")
pA=sum(pABC,[B C]); pB=sum(pABC,[A C]); pC=sum(pABC,[B A])
figure(); title(Data)
plot(pA.content,label="A skill")
plot(pB.content,label="B skill")
plot(pC.content,label="C skill")
legend()
# Can calculate mean skill levels using:
println("mean skill for A=",sum(pA.content.*(1:Nlevels))) # mean skill level for A
println("mean skill for B=",sum(pB.content.*(1:Nlevels))) # mean skill level for B
println("mean skill for C=",sum(pC.content.*(1:Nlevels))) # mean skill level for C
end
```