Trueskill has five parameters that can be adjusted to suit the game: mu, sigma, beta, tau and draw_probability.

We'd like to find the best parameters for our game, the Forged Alliance Forever 1v1 ladder.

To that end, I got my hands on some data, wrote some c/c++ functions to calculate marginal likelihood of outcomes and put python bindings on them. Then I searched the trueskill parameters for the maximum marginal likelihood of the outcomes observed in the data - ie in machine learning parlance, I "learned" the model parameters.

The optimum result was:

- mu = 1500
- sigma = 500
- beta ~ 240
- tau ~ 18
- draw_probability ~ 0.045

Note that I didn't search mu and sigma. I set them to 1500 and 500 at the outset as my preliminary searches indicated that the system is otherwise under constrained.

The Dataset

Ideally I would get a database dump from one of the hard working FAF administrators or developers. However being the overworked, underpaid beavers that they are, getting one's hands on such a database dump proved more difficult than anticipated.

So instead I exhaustively searched the replay vault for games and results, a slow process that took a few days to finally run out of new players to search for. See "replaydumper.py", which I hooked into the FAF client's replay.py so that it was activated when I hit the replay search button.

The data exactly as obtained from the replay vault is saved in "replaydumper.json". It has three top level keys:

- "replays": data about the replay including map, start time, finish time
- "replayPlayers": more detail about each replay, including player names, faction selections and the all important final score
- "playerAliases": a dictionary keyed by player names as the appear in "replayPlayers", valued by what those player names resolve to when queried from the FAF username web page. In the end I decided not to use this information because it appears that the "replayPlayers" names were pretty consistently reflecting the user's current name anyway. At least all of TA4Life's games appear as his current (as of time of query) name, and not as any of his many previous names (eg DEUS_EX, Putin_Trump_2016, etc).

The “replaydump2csv.py” script converts the json into a flat csv with unique integer ids for maps and players. See “replays.csv”. uniquePlayerIds.csv gives the integer player ids and uniqueMapIds.csv gives the integer map ids.

There are about ~190,000 1v1 ladder games covering circa 2014 to Jan 2016, played on ~100 different maps by ~13,000 dedicated gamers.

Maximum Marginal Likelihood

There are already a number of trueskill implementations around. Theres one built into the FAF client, theres the python library http://trueskill.org and then there is the C# https://github.com/moserware/Skills.

The python implementations are too slow for machine learning purposes and, while I’m sure C# is up to the task, I’m not really interested in C#. So instead I ported the key 1v1 functions from Moser to my own C implementation and bunged python/numpy bindings on top of them.

The key function is rate_1vs1. This does what one would expect: it updates players’ ratings based on observed outcomes. But it also calculates a-priori likelihood of the outcomes, which is the basis for our optimisation objective during the “learning” phase.

The likelihood of a draw is:

L[draw] = quality_1vs1 * draw_probability

where quality_1vs1 is the canonical game quality provided by trueskill and draw_probability is the fifth trueskill parameter (which we will need to “learn”).

The likelihood of a win for player 1 is discussed at https://github.com/sublee/trueskill/issues/1. I went with suggestion of **jsnell** on 21 Oct 2015. However one needs to realise that jsnell’s formula seems to give probability of win given that no draw occurred. Therefore it’s necessary to apply bayes law to recover the plain old probability of win:

L[win1] = (1-L[draw]) * Jsnell[win1]

and similarly for player 2:

L[win2] = (1-L[draw]) * Jsnell[win2]

Finally, the “learning” phase is executed by the python script learntrueskill.py. It simply loads the data, generates a random set of trueskill parameters (mu, sigma, beta, tau and draw_probability) and calls rate_1vs1 to retrieve the likelihoods for each game. The log likelihood is summed over all games where both players have played at least 10 games and the result is taken as the optimisation objective. Repeat 100,000 times with different random

parameters. Results get logged to learntrueskill_solution_space.txt so you can examine how the marginal likelihood changes as a function of the parameters.

As it turns out, with mu=1500 and sigma=500, the optimal beta, tau and draw_probability are:

- beta ~ 240
- tau ~ 18
- draw_probability ~ 0.045

The output of learningtrueskill.py is saved in data/learning_results