CSE702 Week 6: Experiments With the Analyzer

There is a whole host of possible experiments that are easy to conduct in the program. I finished the previous major update just in time for the pandemic, which put the cheating application into overdrive. Then came the Carlsen-Niemann case, and on its heels, the doubt about ratings explored all last year. So a lot of basic work remains unprobed. Well, we can poke around to gain a feel for what might be interesting to pursue further---and just get experience with the C++ program.

Most of the examples I thought of had the form "test quality metric Y on sample X", so I've made a table and given enough scripting examples to illustrate every row and column individually, so you can put together your own combos. They apply equally well to the three chess engines used for Multi-PV analysis: Stockfish 11 (`SF11`), Komodo 13.3 (`Kom13`), and Stockfish 7 (`SF7`). The program invocation is simply

`ir SF11 UW` or `ir Kom13 UW` or `ir SF7 UW`

(If you don't copy the executable to your own folder and are working in a subfolder of `/.../CSE702/` then you can do `../ir SF11 UW` etc., else use the full path `/projects/regan/Chess/CSE702/ir`. You will also need to copy lines posted on Piazza into your `.cshrc` and do `source ~/.cshrc` to activate a recent enough Gnu C++ system to run the code.)

The program then prompts to accept the path `/projects/regan/Chess/CSE712/AIF/`; any reply other than `n` or `no` (followed by an alternate path) is treated as saying yes---so you can accept the path and give a useful session name at the same time. After a half-minute of chugging, the program displays the main menu. The above must be on a line by themselves because they are an OS-level invocation of the `ir` program and the next two tokens are its command-line arguments. (`UW` means "unit weights" mode; the alternative is `EWN` for "expectation weights, normalized-by-rating" which for various reasons already discussed is a mess right now.) The session ID on the next line cannot have internal spaces---thereafter, space and carriage return are treated the same. All of the following script examples begin from there.

A Grid of Tests

There is actually a whole third dimension of particular ratings to use as the anchor for the tests. For instance, you can limit row A to players rated under 1700, then repeat it for above 2500, then 2100±50, say. Likewise rows B, C, and everything else. This could test whether the rating system keeps its theoretical linear invariance. However, we'll either ignore particular ratings and presume the invariance, or we'll fix the rating level to be 2100±50 (or whatever). The issue of limiting the sample to a rating range for fairer comparison actually comes up right away.

Here is the first grid, followed by the script for box A1 (and the rest of row A):

Setting\Metric	1. IPR	2. T1 (vs. proj)	3. ASD	4. Error 050+	5. Error 200+
A. Playing nearly equal-rated oppt.	2205 ± 30 overall among 2100 to 2250, it is right at 2250 (!)	46.96% (fitted so proj is the same)	0.1340 (fitted so proj is the same)	4938.00 (8.26%) versus proj. 4776.87 (7.99%)	773 vs. 765.43 projected.
B. Oppt. 200+ Elo higher	Played a little worse (not significant?)	46.30% (proj. 47.35) with the 2250 fit	0.1465 (vs. 0.1391: proj.)	1102 vs 1019.74 proj	178 vs. 187.7 projected.
C. Oppt. 200 or more lower	Played better				
D. Playing ahead 1.30+	About the same				
E. Playing 1.30 or more behind.	About the same--- except ...		...overall projected and actual error went up.		

```
/projects/regan/Chess/CSE702/ir SF11 UW
A1script
addOutputFile A1results.txt A1results
attach tleq60 n
clearTurns addTurns /projects/regan/Chess/CSE702/ClockedAIF/FIDE45*SF11*aif
newFilters
EloDiffWithin elodiffwithin25 leq 25
done
showTrial n
perfTest goTest
perfTest useRating 2187 ratingSlack 0 goTest
```

To do row D.

```
newFilters
EvalPlayer ptmevalleqm1.30 leq -1.30
```

```
done

newFilters
EvalPlayer ptmevalgeq1.30 geq 1.30
done
```

Although all your results will be saved in the file "`IRsessionData.txt`", the third line saves just the results for this run in the separate file "`A1results.txt`". That file gets a separate session ID too. The next line loads a predefined filter that cuts off games after turn 60---IPR and the cheating-test z-scores have been calibrated this way. Loading it first saves some time of re-filtering, though loading the Olympiad files still takes several minutes---it's several GB of text data. Before doing so, we cleared out the 150 games of the reference set RefSF11.txt, though those games are still loaded in the "reference trial"---which is used only for IPR calculations.

The fifth line loads all the files from last September's Budapest Olympiad (for the engine Stockfish 11). We saw that the Elo ratings from the Olympiad games were reasonably well adjusted from the Sonas correction and pandemic lag, not as well as in January 2025 but usable. It's about 5GB of text data per chess program, though the resources used can go over 16GB, 13% of memory.

The next lines do the selection of Row A. Because this predicate applies to a whole game, rather than moves within a game, it is marked "GF" for "game filter" (not gluten-free). If you define or otherwise attach a GF before (re-)loading data, then only those games matching the filter will load. This is recommended for memory savings and speed if you know you won't need the other games for the whole run. The **`showTrial`** line shows that 21,999 positions meet the five attached filters, out of 565,566 positions total. (The code boots the other positions out of memory, so the usage goes down.) The simple **`perfTest goTest`** line gives a way to peek that the average rating of the players---weighted on a per-position basis---is 2186.63. The final line tests how well the actual games match the model's projections according to the "central fit" $s = 0.05408$ and $c = 0.33135$ (and $ev = 2.20845$) of the main parameters for 2187 rating. (If you enter just `perfTest useRating 2162 goTest` the system gives the same result, since default is adding 25 for slack.) Answer: the model does fairly well.

Now doing **`runIPR 2187 FIDE45OlyAllDiffLeq25SF11IPR`** fits the data exactly. The 2187 argument is an initial guess of the answer and usually does not matter. The third argument becomes the name of the **`TrialSpec`** object that conveys the fit and should be given at the same time (it is prompted for later). This gives the Intrinsic Performance Rating (IPR) for column 1. The output 2204.71 ± 28.48 rounds to 2205 ± 30. The 2187 average rating is within the two-sigma error bars, so at least that is good. The "actual" figures for T1-match, errors of at least 0.50, and errors of at least 2.00 should not change (the latter two are raw evaluation differences, not scaled), but ASD does change because the rating basis has changed. Do **`perfTest goTest`** once more to see: the T1-match stayed 47.21% but the reported ASD slipped from 0.1410 to 0.1406 PEPs per move" (PEP = "Pawn in Equal Position"). Actual errors of size 0.50 or more number 1756, of which 290 were errors of 2.00 or

more (raw centipawn figures, not PEPs). Those are versus projections of 1690.68 and 277.53, respectively.

Now to do Row B, we need to define a different filter. The script is the same as above until the filter definition, which is now

```
newFilters
EloDiff opptgeq200higher leq -200
done
```

OK, why "**leq**" and minus? As the menu helpfully says, the target value is **Player Elo - opponent Elo**. So you want that difference to be "more minus" than -200, which means ≤ -200. Admittedly a little mind-bending, but it makes the interface consistent.

You don't have to restart from the beginning---you can tack this on to the previous run. If you paste in those lines, you get the message "`All tuples filtered out---may see NaNs or worse!`" That's because we left the previous filter attached and it's logically contradictory. We can do

```
detach elodiffwithin25 n
```

to deactivate it. (The '**n**' responds to a yes/no prompt about detaching more than one filter.) It was actually good to work in this order, because if we'd detached it first, we'd have yo-yoed up to over 500,000 active data points, then have to destruct most of them again.) Now do

```
perfTest goTest
```

since we still have the previous IPR fit loaded. Immediately we see that the players underperform, but wait---there's a selection change here. We see the new selection of 75834 positions has avg. Elo 1936.62 versus 2253.03. If we then do

```
perfTest useRating 1937 ratingSlack 0 goTest
```

then the result is incredibly close in T1-match, less so in ASD but still reasonable. For a better comparison to the previous sample, maybe we want to look at players themselves near 2187---?:

```
newFilters
PlayerEloWithin ptmelo2187pm12 2187 25
done
```

Now this gives 3,994 turns with avg. Elo 2186.79 versus 2476.02. The performance is once again underwater---not by as much but the comparison is more meaningful. We get IPR 2155 +- 65. Is this a significant difference? Maybe not... OK, let's try versus 200 or more lower. Now do:

```
newFilters
EloDiff opptgeq200lower geq 200
done
```

Well, we need to detach "`opptgeq200higher`" and then reset the IPR fit by

```
loadTrialSpec FIDE45OlyAllDiffLeq25SF11IPR
perfTest goTest
```

[Monday's session got almost this far. The idea for Tuesday is to explore rows D and E, then do some more freewheeling choices out of the following, each taking an option or combo of options.]

Tuesday: Let's switch things up so that we load only games with at least one player rated between 2100 and 2250.

```
/projects/regan/Chess/CSE702/ir SF11 UW
A1script
addOutputFile A1results.txt A1results
attach tleq60 n
newFilters
PlayerElo elogeq2100 geq 2100
PlayerElo eloleq2250 leq 2250
done
clearTurns addTurns /projects/regan/Chess/CSE702/ClockedAIF/FIDE45*SF11*aif
newFilters
EloDiffWithin elodiffwithin25 leq 25
done
showTrial n
perfTest goTest
perfTest useRating 2187 ratingSlack 0 goTest
```

It still loads almost 200,000 positions. But this focuses the sample on players near the 2187 average we found.

Here are some other selections to test, with itemized directions:

1. Load a tournament file or files (at least 28 players, so loading WijkTataA and WijkTataB together is fine).
2. Attach the **`tleq60`** filter.
3. Then do **`runIPR 2500`** [name] to fit the file overall. (The 2500 is just an initial guess; you can substitute the average Elo in your data if you wish but it doesn't matter much.)

4. Do **`perfTest goTest`** and inspect the T1%, ASD, and *rates* of small and large errors.
5. Then define the filter(s) to make row selection.
6. Do **`perfTest goTest`** again to judge performance on the selection. This will again pick up the "Spec" (≡ model instance) from step 3. Fill in the T1, ASD, and error boxes of the row contrasted against the values in step 4.
7. Finally do **`runIPR`** ... again to fit that selection and fill the IPR column value. (We're really interested in the IPR change from step 3, since you can choose any tournament from amateur to elite level. Note that if we get similar results from all levels of chess skill, there's more reason to regard the results as general.)

Setting\Metric	1. IPR	2. ASD (vs. proj)	3. T1%	4. Error 050+	5. Error 200+
F. Turns 17-24 Zaid					
G. Turns 25-32					
H. Turns 33-40 Radhika					
I. Turns 41-48 Gishnu					
J. Used $\leqslant$ 8sec. Yash					
K. Used $\geqslant$ 5min. Ahmad	Pre: 2100± Post: 1600±	.1492 .20 actual rb2100 Post: both .226	Pre: 39.45 Post: 31.95%	Pre: 8.7 proj 12.11 ac Post: 12.4 proj, 12.11 ac	
L. UEE $\leqslant$ 0.5 Dylan					
M. UEE $\geqslant$ 3.0 Yujie					
N. $\delta_{10} \leq 0.25$ Shuwei					
O. $\delta_2 \geq 0.70$ Jiaheng					