CSE702 Week 4B: Main Analyzer Workflow [Tuesday evening was a full demo.]

The analyzer has been trained with data from in-person chess in 2010--2019. Updating the training---before or after applying the Sonas correction of ratings---is a separate matter. We will be mostly concern just with operating the program with the calibrations given.

For lamentable technical reasons, the calibration is done completely separately for each chess engine---and engine version---used for testing. Thus you have to give the code for the engine version at startup. If the executable is called `ir` (yours may be `irw` or `IRW` on Windows), do

`ir` <engine> [mode]

engine = `SF11` or `Kom13` or `SF7` or `Kom10`. (Stockfish 16 still not up.)
mode = `UW` for "unit weights", or `EWN` for "expectation weights, normalized."

Once it loads and builds stuff, and you load a set of games, the main workflow items are:

1. Test a player or players at a prescribed competence level. This `perfTest` does not use regression over the loaded data. The main outputs are ***z-scores*** of the abstract form
$$z \;=\; \frac{(actual) \;-\; (projected)}{(std.\,dev.\;\sigma)}$$
2. Use regression over loaded data for player(s) P to determine the corresponding rating level IR_P. The latter is called the **Intrinsic Performance Rating** (**IPR**).

The main distinction between these workflows bears repeating:

1. Performance tests are calibrated **using large data only**. The small data of a player's games in one tournament are used only for tallying the $(actual)$ figures. The $(projected)$ and $(std.\,dev.)$ components are based on regressions over datasets of 100,000s of positions from 1,000s of games for each rating level 1025,1050,...,2775,2800,2825+.
2. The IPR is obtained by regression over t**he player's own small data**. For this reason, the two-sigma error bars in an IPR output $IR_p \;\pm\; e_p$ are **error bars of measurement only**.

It is tempting to "Studentize" an IPR measurement against the player's rating R_P to create something that looks like a z-score:
$$\frac{IR_p \;-\; R_P}{(e_p\,/\,2)}.$$

The two sins here are that e_p is not a population standard deviation as William Sealy Gosset required and that e_P itself is based on small data with wide potential variation. Whereas, σ in the z-score formula comes from large populations.

Model Parameters and Virtual Players

The only inputs to the model are:

(a) the numerical values of possible chess moves in a given position, as given by the strong chess program used to build (an instance of) the model, and
(b) the model parameters, which fall into multiple ranks from completely fixed to completely free.

The numerical values in (a) are *the only elements specific to chess in the entire model*. If the numbers came from programs for Shogi or Go or Othello or any other pure game of strategy, the operations of the model would be entirely unchanged. (The training over large sets of recorded games by players of various skill levels would be particular to the game, of course.) Whether the model can be made sharper by incorporating more specific chess knowledge---such as labeling which moves are advancing, capturing, with the Knights, etc.---is a possible open qustion to pursue.

The two main model parameters---both treated as completely free---are:

- s for *sensitivity*: how finely the player can react to small differences in the values of moves.
- c for *consistency*: how well the player can avoid large mistakes.

It has been a longstanding desire to have a third main and freely regressable parameter:

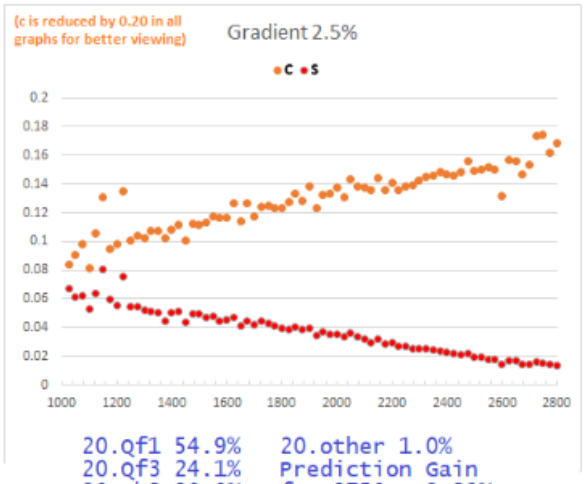
- d for habitual *depth of thinking*.

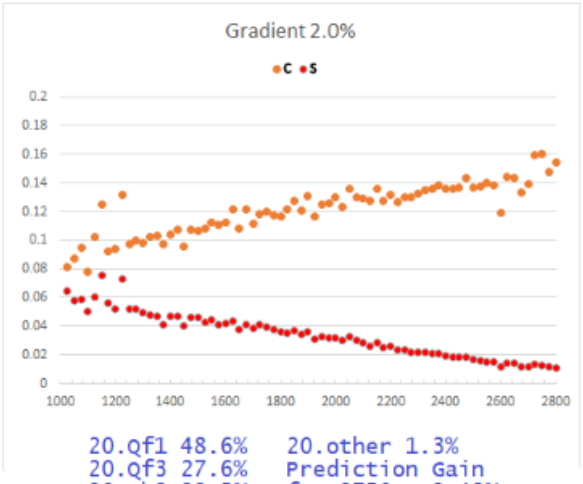
However, making d a free parameter causes numerically unstable results. The same held true for a paremeter h for "heave"---meaning the tendency to be eager in the sense of a ship riding over the water. The idea is currently applied only in the form of a parameter e_v that governs the relative probability of moves whose final values are equal, but whose values differed at lower depths of search. The program code allows treating e_v as a completely free parameter, but then the fitting method of making it an unbiased estimator fails to "close" for individual players the way it does over large sets of training data. So what I currently do instead is use the mapping $R \mapsto e_v$ given by the training fit to set e_v as a function of IR_p in the IPR workflow. Then only s and c are freely fitted to any player's games.

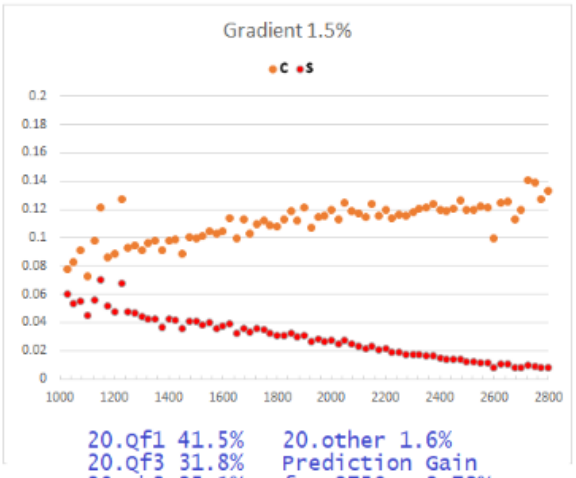
I call a parameter setting $Y = (s, c, \dots)$ a *virtual player.* The IPR regression finds the closest virtual player to the given set of games. (My 2013 paper with Tamal Biswas used Z for virtual player or "agent", but that letter might confuse with z-score.)

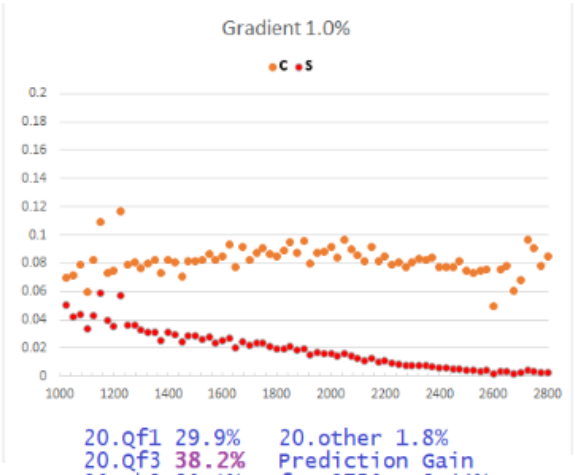
> **Important fact**: The virtual players obtained by regression for the seventy-three individual training sets of games by Elo 1025, 1050, ..., through Elo 2775, 2800, and 2825+, when plotted in the 2-dimensional (s, c) plane, fall closely into a 1-dimensional curve (which should in turn be rectifiable as a straight line after the Sonas correction). This not only smooths out into a continuous set of values (s_R, c_R) which I call the **central fit**, it gives individual (close-to-)linear regressions for the s_R and c_R values individually.

The diagrams in my graphic about a hyper-parameter discussed in the next section not only show the individual lines of central fit values, but also how they "collapse" when that hyperparameter is set too aggressively. These are the actual s_R and c_R values, not the ones smoothed by regression:









The upshot is that other aspects of the model are tweaked so that the "Important Fact" holds true to best advantage.

So the `perfTest` workflow does this:

1. Input: a rating R and the set of games to test.
2. Compute $\hat{s}_R$ and $\hat{c}_R$ from the regressions for the central fit.
3. Do projections for the virtual player $Y(\hat{s}_R,\hat{c}_R,...)$, where the "..." includes e_v and other player-characteristic parameters that are expressly tied to s and c.
4. Compare the player's actual performance against those projections.

Note that only the player's rating R is used for the projections. It is possible that the actual human player P involved has a different tradeoff (s_P, c_P) that produces the same rating R---but gives results meaningfully different from those for $Y(\hat{s}_R, \hat{c}_R, ...)$. If the player is more positional, then s_P may be lower (which is better) and c_P may be lower too (which is worse). This will generally reduce the z-scores of the T1-match and EV-match tests while raising that of the ASD test. If the player is more adept at navigating tactical positions without large mistakes, then c_P will be higher and s_P lower to compensate (since we are saying the rating R is the same). This is a potential defense in a cheating test. However:

- The difference in the overall results is generally small enough to be covered by the policy of giving 25 Elo slack in the tested rating R.
- The z-scores in large tournaments obtained via the central fit conform well to the first and second moments of the standard bell curve.
- The IPR figures are unaffected by the starting values $(s, c, \ldots)$ given.

This is despite the following

Weird Observation: The actual (s_P, c_P) values obtained for players P in the IPR regression producing the IPR values $I = IR_P$ are fairly often skewed away from the central fit values $(\hat{s}_I, \hat{c}_I)$ like so:

- For Standard chess, s_P and c_P are higher than $\hat{s}_I$ and $\hat{c}_I$, s_P markedly so.
- For Rapid chess, there is very little skew.
- For Blitz chess, the skew goes the other way: s_P is lower than projected.

There are issues related to Simpson's paradox going on here, as Lipton and I covered in a notorious GLL article. I have not gotten to the bottom of this---resolving this is a possible seminar project. However, because most online chess during the pandemic was played at Rapid pace, I used these observations to justify **not** building the model separately for Rapid and Blitz chess---nor running through the full millions-of-trials validation process for them.

To summarize, the mapping from Y to the Elo rating denoting the skill of the (virtual!) player is many-to-one. In particular, multiple combinations of s and c correspond to the same Elo rating R. It would be interesting to plot "Elo isobars" of the s and c landscape, representing different tradeoffs between strategic and tactical ability that yield the same rated skill.

IPRs and the Reference File

The IPR is intended to be "of" a player P "on" a set of games G. But neither is true. Instead, the IPR is well-defined as a function of a virtual player $Y(s, c, \ldots)$. It does not put Y on the games G but rather on a fixed set of games called the **reference set**. In the current program release, the reference set has 150 games at all rating levels 1000s thru 2700s. Each rating level features one main player against

6--9 opponents of basically the same rating. The reference set is hand-picked and was originally intended to have the same effect as using the entirety of the training sets as the reference (which would take too long for the code to execute). Here is the workflow:

1. Do regression on the set of games G (taking the side of each game played by P) to fit the closest virtual player Y. This operates **`runFit`** and outputs Y as a **`TrialSpec`**.
2. Do a **`perfTest`** *on the reference set*. Get the projected ASD figure a_Y.
3. The IPR is a function $r(a_Y)$, which is pre-determined by regression over the training sets.

That is to say, we do not actually use the actual ASD a_P by the player P on his/her own games G. That number would not be robust because the games G might have been unusually easy to play---or unusually hard. Instead, the regression computes Y in a way that best reflects how P played in those games, regardless of the total difficulty. Then we give Y the reference set as a *standardized test*.

Note that the actual ASD on the reference set is not relevant either, because the games were played by different players. The final important wrinkle is that the projected ASD comes with its own standard deviation σ_Y. Thus

$$[r(a_Y - 2\sigma_Y),\ r(a_Y + 2\sigma_Y)]$$

becomes the two-sigma confidence interval for the IPR measurement $r(a_Y)$. At the moment, the function r is nonlinear, so the two arms of the interval are not equidistant from $r(a_Y)$. But because I round IPR figures to the nearest **05**, reflecting the fact that the whole code has only 3--4 digit precision, the arms are generally equal within this tolerance, so the code outputs a single $\pm$ error bar.

The reference file is loaded automatically. For Stockfish 11 it has the required name **`RefSF11.aif`** and similarly **`RefSF7.aif`**, **`RefKom13.aif`**, and **`RefKom10.aif`** for the other engines. It is initially loaded as both the "reference trial" and the "focus trial", the latter as main data. You can play around with it. Menu option `[6] clearTurns` zaps only the focus trial data.