CSE702 Week 5A: Model Particulars and Principles

In which we describe how and why the model has evolved to its present complicated state, and what simpler principles all this is supposed to represent. A meta-question is whether the model can be simplified by more-standard machine-learning means while preserving those principles.

The first three sections are actually necessary "window-dressing" before we even get to the predictive heart of the model.

Hyperparameters and Tiers

The term hyperparameter IMHO does not capture all the following tiers of nuance. They reflect also the idea that after large-scale training to build a *model*, an application to a particular class of inputs may have a shorter phase of customization to create a *model instance*. The latter phase is supposed to set the *main parameters*, but may involve other parameters in a dependent manner. In our case the latter phase is the IPR regression on the games of a particular player or tournament, and the main dependency is along the backbone of Elo ratings.

1. A switch or numerical setting that affects training but is not part of the final model instance. Canonical example: the *learning rate* in neural nets.
2. A switch or setting fixed prior to training that is part of the finished model. Example: neuron firing and refraction rates.
3. A setting determined while training the model but not changed in any model instance. Possible example: interconnection coupling between neural layers.
4. A setting determined during training (or not) that is also changed *dependently* in customization.
5. Parameters that define a customized model instance and have no dependency. Those are the main parameters, and I also call them the free parameters.

The dependency in the chess model can be directly on the value of another parameter or indirectly via an implied Elo rating R. In the latter case, the "parameter" is really a *mapping* from R into a parameter space. The following examples hopefully clarify all this generality.

- Sensitivity s and consistency c are (still) the only free parameters---tier 5.
- The eagerness parameter ev is currently treated as tier 4. The C++ code allows treating it as free, but regressing it reliably finds a global minimum only over the training sets of thousands of games, not for individual players.
- The *gradient* parameter v belongs to tier 2. It is currently set to the conservative value 0.035. This makes the model even less predictive than the 0.025 value led with here, but shores up the model's numerical stability as discussed toward the end of this recent GLL post.
- The *flattening parameters* am, bm (on the minus side) and ap, bp (on the plus side) are rating-dependent. They are best regarded as in tier 3 because they are not intended to be characteristics of a (virtual) player---although in current operation they go thru the same process

as ev. The *expectation parameters* la, lb, lk, lq give the rating-dependent correspondence between the advantage or disadvantage e in a position and the points expectation (loosely, the "chance of winning").

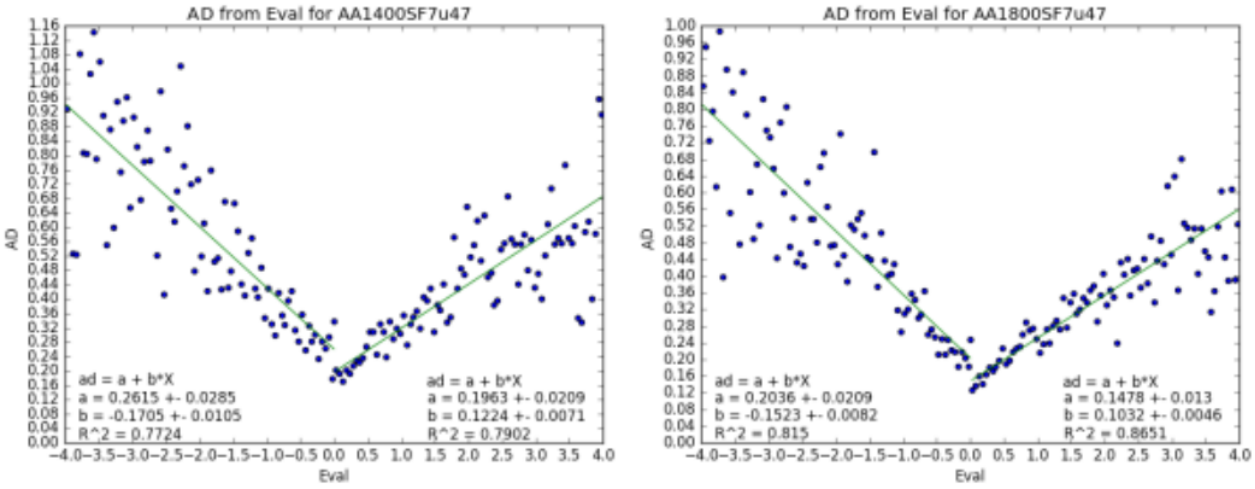
- The program allows freely configuring weights (in menu option [17]) for various criteria in the **loss function** whose minimization governs the regressions used in both the large-scale training and customization. The resulting model (instance) is supposed to be interpreted independent of the way it was obtained, so these un-named weights are in tier 1.

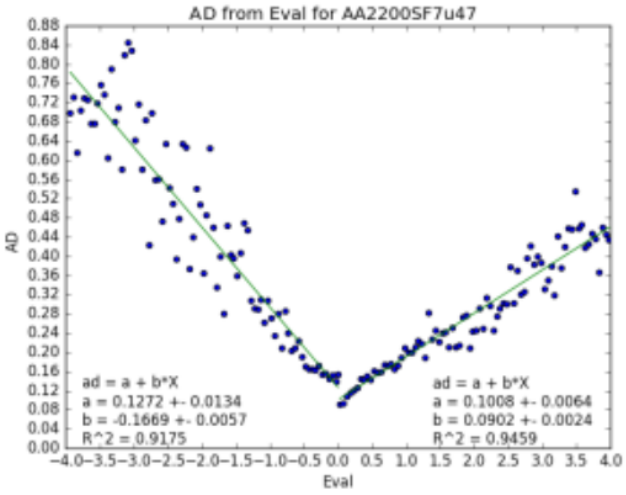
The flattening parameters are used to scale down the raw centipawn loss figures given by the engine for inferior moves, according to the overall position evaluation e. When e is negative, they fit the average centipawn loss (ACPL) δ as $a_m + b_m|e|$, and when e is positive (i.e., the player to move has an advantage), as $a_p + b_p e$. Then the scaled value δ' of an error of raw magnitude δ in a position of value e is given by

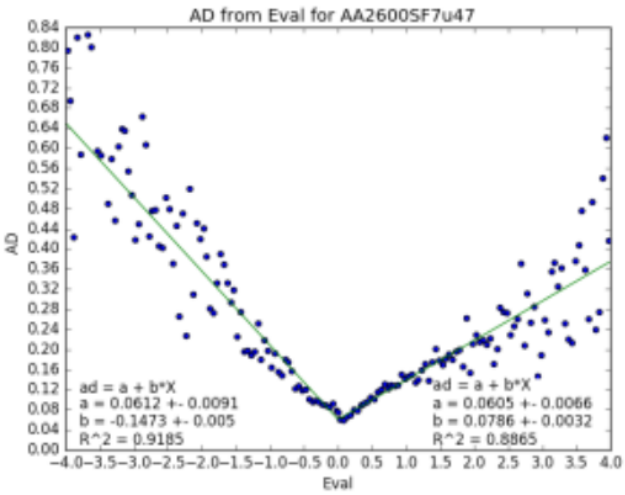
$$\delta' = \frac{a_m \delta}{a_m + b_m|e|} \quad \text{or} \quad \delta' = \frac{a_p \delta}{a_p + b_p e},$$

according as e is negative or positive. Note that if $e = 0$, the raw value δ is left unchanged---which is why I call the scaled units "peps" for "pawns in equal positions." This allows a more-precise flattening than that described in the GLL blog article "When Data Serves Turkey."

Well, the a_m etc. are not single values. They are rating-dependent, as one can tell by looking at the y-axes (which are called AD rather than ACPL) in the following quartet of diagrams from the article:







The laborious model-building process first trains a mapping $f: R \mapsto \left(a_m^R, b_m^R, a_p^R, b_p^R\right)$ by regression over the training sets for all rating levels. Then when R is prescribed in the performance-test workflow, or updated during iterations in the IPR workflow, the corresponding $\left(a_m^R, b_m^R, a_p^R, b_p^R\right)$ are used as above to create the δ' values of each move in each position. Those in turn feed the central utility function in the main model equation that is used to evaluate or regress at that value of R. Thus the IPR workflow is a two-tier regression process. The outer tier sets up the relevant hyperparameters for the current **reference rating** R, while the inner tier regresses the main model parameters for that reference. The result of the inner tier updates R to a new rating R', and the outer tier goes again unless $|R - R'|$ is within a prescribed tolerance (1 Elo point in my settings). The notion of "reference rating" needs a separate section.

Reference Ratings, Expectation, and Difficulty

A motivating question is: **What makes a chess position difficult?** That is to say, difficult relative to other positions. What characterizes especially difficult positions?

At the other end, what about the spectrum of ways a position can be easy to play?

- The position is easy because the best move is so obvious that almost any player would find it.
- The position is easy because it has many equally-good moves---it is virtually impossible to play a bad move.

The latter is less clearly a case of "easy" because the crisis in the position might arrive at a later move. There is already one key aspect of both these cases: they are both worded in terms of what human players *would* do. That is, they are **subjective**. They are not phrased in terms of raw data associated to the positions, which I have termed **objective**. This is actually maybe the clearest instance thus far of why I have been using the terms this way and why I regard my predictive analytic model as *subjective*.

> **Open Question**: Can you devise a good measure of difficulty that is objective in this sense, without using projections?

The answer I've implemented is:

> The difficulty is measured by the projected loss in evaluation---or points expectation---by a human player having to play the position.

This however runs right into an issue:

> "...by a human player" **of what skill level?**

I tried to implement the idea of a "Reference 2000 Player"---i.e., fix the use of the projections for $\Upsilon(s_{2000}, c_{2000}, \dots)$. But the digital dynamics of my code in operation do not (did not?) work well with this. What worked instead is to "piggyback" the reference level:

- In the `perfTest` workflow, the reference is the prescribed rating R. Thus if Carlsen is being tested, the difficulty is for someone of world champion rating class. If a 1600-player is being tested, the difficulty is measured with characteristics centered on 1600.
- In the **IPR** workflow, the reference R is updated along with more being learned about the rating quality of the games played.

Measuring as ASD in centipawns units, the difficulty ≡ the projected loss by the central-fit virtual player $\Upsilon_R = \Upsilon(s_R, c_R, \dots)$ corresponding to the current reference rating R. for a given game turn t, this is:

$$\mathcal{D}_R(t) = \sum_{\text{legal moves } m} \Pr_{\Upsilon_R}(m)\delta'(m) ,$$

where $\delta'(m)$ is the ASD for the move m. A property that may have a little hidden subtlety is:

- The lower the rating R, the higher the probability of inferior moves (i.e., those m with $\delta'(m) > 0$), hence the higher $\mathcal{D}_R(t)$.

We want a measure that does not depend on the rating R---at least not so obviously. Here are three ways to do this:

1. Use a fixed value for R, such as $R = 2000$ as mentioned above. This leaves $\mathcal{D}(t) = \mathcal{D}_{2000}(t)$ dimensioned in units of centipawns.
2. Divide $\mathcal{D}_R(t)$ by $\widehat{\mu}_{ASD}(R)$ to make a dimensionless value. Is it close to the same over all R?
3. Instead of the ASD $\delta'(m)$ for the move m, use the corresponding points expectation change, which could be denoted $\epsilon_R(m)$. The conversion from centipawn evaluations to points expectation also depends on R, in a way that (hopefully!) fully offsets the dependence of the

probabilities on R. (This conversion is governed by the parameters la, lb, lk, lq mentioned above, which define a general logistic curve.) The units are dimensionless ones of expectation.

I have implemented strategy 3. Thus the difficulty of the position at game turn t is

$$\mathfrak{e}_R(t) \;=\; \sum_{\textit{legal moves } m} \Pr_{Y_R}(m)\epsilon_R(m)\;,$$

which is the projected loss of points expectation. Note that $\mathfrak{e}_R(t)$ is non-negative: by definition, your points expectation stays the same only if you make an optimal move, since the value of an optimal move is taken as the value of the position on the whole. [There is a "Murphy's Law" type situation here: If the value of the position is 0.00 according to the engine, would you rather it be your turn to move or your opponent's turn? If it is your turn, you have the first opportunity to make a game-losing blunder. The main contribution by Jason Zhou to this 2014 paper was to show with large data that for human players---and not computer players---the onus of having to make the next move depressed the points share (wins + 0.5*draws) by 0.02--0.03. This is the main reason for having the parameters lk, lq which allow the logistic curve to be off-center and slightly warped.]

Ideally, R should not be present as a subscript in $\mathfrak{e}_R(t)$. The fact that the probabilities come from the central fit is already a hint that an exact offset of the two uses of R is not foreordained. In fact, there is some drift with R, but it is regressed as b_R and divided out to make a measure independent of R.

There is one further normalization that does not depend on R. As defined, since $\epsilon_R(m)$ is always a number between 0 and 1 and the probabilities sum to 1, the value $\mathfrak{e}_R(t)$ is always between 0 and 1. It is generally quite a bit less than 1. We can compute the average value w_0 over all positions in the training sets across the rating spectrum. Then the value

$$\mathfrak{e}(t) \;=\; \frac{\mathfrak{e}_R(t)}{b_R\; w_0}$$

has average value 1.0 and is reasonably independent of the reference value R. This is the **normalized expectation weight** of the position, and is what you get when you use the **EWN** option at startup (rather than **UW**, "unit weights", for all positions having weight 1).

Thus the bprinciple of EWN weights is that **positions are weighted according to their difficulty**, which in turn is measured as **projected** expectation loss. From the internal use of the model's own projections, this is not only *subjective* but subject to a potential vicious cycle. It also does not really break free of the dependence on the reference rating R, because you need to use R to compute it to begin with. (And the normalization is far from perfect...)

One motive for EWN is to try to catch "smart cheating"---the idea that players will cheat only in positions where a lot is at stake, which get up-weighted in EWN mode. A downside of this is that "dumb

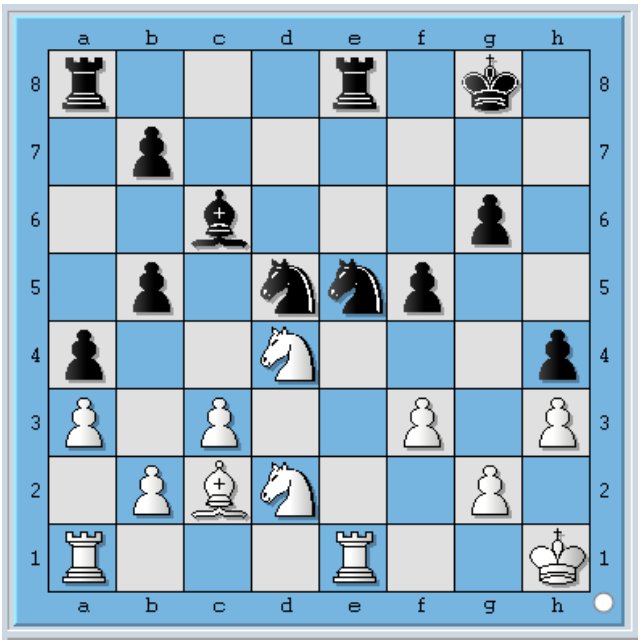
cheating"---cheating on every move---becomes harder to catch, because making the weights "choppy" increases the standard deviation in the denominator of the z-score. The latter was so legion for tournaments moved online during the pandemic that I left EWN unused. There are, however, two non-cheating rationales:

- It promotes a measure of "Challenge Created": Which players are able to make the games more difficult for their opponents---and maybe also for themselves?
- There is a material analogy to the probability-of-selection principle used in Google's "quantum supremacy" experiment---which I wrote about here. Thus the EWN measure is in some way "organic". (Its results have alas so far been *wimpy*---and Google's experiment has also gone under a cloud.)

Doing research to see if the difficulty measure really affects human players is a wide-open idea for a seminar project that could lead to a publishable paper. The pandemic put this on my own back-burner. Well, that and some caveats and qualms...

An Example and Some Qualms

Here is the analysis by Komodo 13.3 of a position in a game that Hans Niemann, playing White, lost against Sam Shankland 15 moves later in the 2023 US Championship, which was held in St. Louis last October:



This is a fairly complicated position in real chess-playing terms. Black has a slight upper hand, but no crisis is looming yet.

```
[GID "U.S. Championship;chess24.com;2023.10.07;3;Niemann, Hans Moke;Shankland, Sam;0-1"]
[EID "Komodo 13.3 64-bit "]
[Turn "31-w"]
```

```
[MovePlayed "Re2"]
[EngineMove "Rad1"]
[Eval "-071"]
[PrevEval "-073"]
[NextEval "-073"]
[Depth "23"]
[NodeCount "201004027"]
[FEN "r3r1k1/1p6/2b3p1/1p1nnp2/p2N3p/P1P2P1P/1PBN2P1/R3R2K w - - 0 31"]
[RepCount "0"]
[NumLegalMoves "39"]
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Rad1	+041	+035	+029	+000	-005	-025	-023	-051	-021	-020	-033	-036	-029	-042	-043	-053	-061	-067	-068	-070	-070	-071	-071
Rab1	+016	+009	+021	-036	-018	-032	-026	-039	-042	-030	-045	-055	-033	-054	-052	-052	-061	-061	-067	-070	-070	-071	-071
Ne2	-048	-091	-040	-040	-044	-084	-065	-061	-045	-045	-062	-050	-058	-062	-060	-061	-067	-070	-070	-070	-071	-071	-071
Reb1	-030	-052	-010	-059	-042	-067	-058	-079	-107	-080	-083	-066	-074	-073	-061	-061	-066	-068	-070	-070	-071	-071	-071
Ra2	-003	-029	-010	-073	-079	-057	-036	-053	-051	-052	-038	-051	-051	-055	-061	-061	-070	-068	-070	-070	-071	-071	-071
Rf1	-029	-080	-010	-032	-031	-031	-034	-107	-125	-080	-085	-070	-070	-078	-061	-061	-064	-067	-070	-070	-071	-071	-071
Red1	-006	-057	-010	-036	-032	-042	-049	-056	-125	-088	-094	-070	-074	-066	-082	-071	-065	-067	-069	-070	-071	-071	-071
Nf1	+017	-029	-062	-058	-058	-058	-058	-101	-097	-101	-094	-082	-071	-070	-069	-070	-069	-080	-069	-070	-071	-071	-071
Rac1	+018	+012	+021	-048	-044	-032	-017	-045	-052	-044	-033	-038	-036	-054	-057	-061	-067	-067	-070	-070	-071	-071	-071
Rec1	-029	-052	-010	-059	-058	-051	-051	-051	-052	-052	-062	-060	-058	-057	-061	-061	-061	-067	-070	-071	-071	-071	-071
Rg1	-030	-044	-008	-052	-040	-051	-051	-047	-058	-052	-062	-051	-055	-055	-059	-061	-064	-067	-070	-071	-071	-071	-071
Re2	+008	+022	+035	-039	-023	-040	-010	-051	-042	-023	-033	-029	-029	-032	-044	-061	-064	-067	-069	-071	-071	-071	-071
Kg1	+021	+022	+028	-018	-018	-040	-040	-039	-031	-030	-048	-032	-031	-054	-046	-061	-061	-067	-069	-071	-071	-071	-071
Kh2	+022	+022	+013	-008	-008	-042	-034	-034	-050	-044	-035	-038	-051	-051	-052	-061	-061	-066	-069	-071	-071	-071	-071
Nxc6	-044	-044	-030	-040	-044	-085	-090	-110	-103	-119	-121	-135	-119	-123	-117	-149	-086	-088	-094	-086	-095	-089	-097
b3	-076	-076	-062	-088	-077	-120	-117	-112	-117	-132	-080	-109	-100	-113	-086	-089	-106	-101	-132	-120	-104	-118	-113

...

The numbers say that the choice among *fourteen* reasonable legal moves *makes zero difference*. (Well, that is provided White will play optimal moves later on when crunch-time comes.) The result is that my program gives this position a weight under 0.02. But it is not a trivial position.

That is to say, the definition of difficulty does not allow for a "slippery slope effect" that is felt very real in human play. This is palpable when one is defending the classic endgame of King+Rook versus King+Rook+Bishop. The engine may say that all your non-suicidal moves keep the evaluation at 0.00, meaning a drawn game. But some of those moves may put you into the "zone of one mistake"---where anything but one particular move may put things into a lost position.

One consequence is that my model overestimates the quality of play in numerous late-stage endgames. This happens in the screening test as well and is one of a couple reasons my standard setting cuts off the sample after turn 60 in any game. (Another is that players may be too short of time to cheat by then.) My full model *should* be able to adapt to such situations, but (a) this is where its severely underfitted nature is not flexible enough, and (b) I am not the only one with large-scale models who has observed this situation and uses a similar cutoff.)