CSE702 Spring 2025 Week 4: Predictive Analytics

A *predictive analytic model* :

1. addresses a series of *situations*, each of which involves a set of outcomes $m_1, m_2, \ldots, m_\ell$.
2. generates projected probabilities $p_1, p_2, \ldots, p_\ell$ for the respective outcomes. And:
3. also generates confidence intervals $[a_1 < p_1 < b_1]$, $[a_2 < p_2 < b_2]$,..., $[a_\ell < p_\ell < b_\ell]$ for these probabilities.

In my usage, point 3 distinguishes "predictive analytics" from mere "analytics." But what on earth does it mean to speak of a "95% probability interval" for one of your own projected probabilities? An outcome m_i either happens or it doesn't.

The point comes more into focus if we imagine analyzing a physical coin. Suppose the coin has a raised head and a slightly concave tail relative to its rim. Then we may estimate the probability p of tails at 0.51. Moreover, we want to be able to assert 95% confidence that the true physical probability $\check{p}$ is between 0.505 and 0.515. What does this *mean*? Basically this:

- Say that a "trial run" r flips the coin 1,000 times and records the proportion t_r of tails.
- The assertion says that if we do 1,000 trial runs, then at least **950** of them will have $0.505 \leq t_r \leq 0.515$.

We've had to flip the coin a million times total to explain the concept. But what we did was not just estimate the coin, we tested and verified the *claimed precision* as well as accuracy of our estimate. That is to say, we did analytics of the prediction itself. Thus: **predictive analytics**.

(By the way, note this article about dependence on how the coin faces initially.)

The point about confidence intervals becomes more concrete if we add a fourth point to the definition: *A predictive analytic model*

4. projects risk/reward quantities v_i associated to the outcomes m_i.

Understood simply, the projected loss/value in the single situation is $E[v] = \sum_{i=1}^{\ell} p_i v_i$. But with repeated situations $t = 1, \ldots, T$ we can project in that dimension too. If outcome m_1 at each time t, which we can label $m_{1,t}$, is the costly one, then the projected total loss is $\sum_{t=1}^{T} p_{1,t} v_{1,t}$. Or for total value/loss, we sum over both dimensions to get the expectation: $\sum_{t=1}^{T} \sum_{i=1}^{\ell_t} p_{i,t} v_{i,t}$.

Then the confidence intervals around $p_{1,t}$, or around all the projections $p_{i,t}$, translate into confidence intervals for these **aggregate statistics**. It is **not** as simple as saying that they are weighted sums of

$a_{i,t}v_{i,t}$ and $b_{i,t}v_{i,t}$. Consider $E[v]$ in the case $\ell = 2$ of just two outcomes m_1 and m_2, which are exhaustive and mutually exclusive. Further suppose $v_1 = -v_2$, $p_1 = p_2 = 0.5$, and $a_1 = a_2$, $b_1 = b_2$. If you expected the confidence interval to be $[a_1v_1 + a_2v_2,\ b_1v_1 + b_2v_2]$, then surprise! that's $[0, 0]$. If the true $\check{p}_1$ is at the bottom a_1 of its envelope, then we must have $\check{p}_2 = (1 - \check{p}_1) = (1 - a_1) = b_2$. Note that $b_2 = 1 - a_1$ and $a_2 = 1 - b_1$ must be true in general. Then $[a_1v_1 + (1 - a_1)v_2,\ b_1v_1 + (1 - b_1)v_2]$ is true in general, and when $v_2 = -v_1$ it becomes $[(2a_1 - 1)v_1, (2b_1 - 1)v_1]$. But now try the case $\ell = 3$...

For the aggregates over t, the simple sums $\sum_{t=1}^{T} a_{1,t}v_{1,t}$ and $\sum_{t=1}^{T}\sum_{i=1}^{\ell_t} a_{i,t}v_{i,t}$ put the bottom of the envelope far too low when the events for different t are **independent**. What's needed instead is to compute the **variance** var_t for each t. **If** the situations for different t are independent, then we get the overall variance as $\sum_t var_t$ by the rule that variances of independent events add. Taking the square root then gives an overall **standard deviation** σ around the estimate E for the expected value. Then the "two-sigma error bars" $[E - 2\sigma, E + 2\sigma]$ give (slightly more than) 95% confidence of bounding the true expected value.

[Some footnotes: Again it may seem weird to distinguish an "expected expectation" from a "true expectation." But when you are deciding whether to buy any financial instrument with risk, that's what you are hoping to equate---or put in a confidence range. The usual convention in statistics is to use a hat $\hat{}$ for a projected quantity, so we should start by saying that a predicive analytic model generates probability estimates $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_\ell$ and so on. This could get cumbersome, so instead I'm using an inverted hat $\check{}$ for a ground-truth quantity. We will have to be careful not to get overconfident by confusing model projections with true values.

We haven't stated that the probabilities make $c_t = p_{1,t} + \cdots + p_{\ell_t,t}$ equal to 1 for each t. But if they don't, we can postulate a "null event" $p_{0,t}$ of value $v_{0,t} = 0$ and probability $1 - c_t$. Furthermore, if we define ℓ to be the maximum of ℓ_t over t, then we can pad every situation t with $\ell_t < \ell$ to have "dummy outcomes" $m_{\ell+1}, \ldots, m_{\ell_t}$, each of probability 0. Thus we can pretend that ℓ is always the same for any situation t. Neither of these changes should affect either the projected variance var_t or its true counterpart $\check{var}_t$. Dividing by "ℓ" may not be meaningful even apart from the fact that the situations t need not have the same number ℓ_t of possible outcomes, but dividing by T to make averages out of the aggregates is always fine.]

In Chess

The situations t are chess positions with a given player to move. We can think of t as meaning "game turn." Then:

- The $m_1, \ldots, m_\ell$ are the legal moves in the position.
- Each m_i has a value v_i given by one or more strong chess programs (called **engines**).
- Traditionally v_i is in **centipawn units**: a possibly-negative integer of $1/100s$ of a pawn. When written to two decimal places we speak of "pawn units." Thus -150cp and -1.50 pawns are the same, meaning that the value is figuratively a pawn-and-a-half disadvantage.
- The engine itself orders the moves $m_1, \ldots, m_\ell$ in nonincreasing order of value: $v_1 \geq v_2 \geq \cdots \geq v_\ell$. Even though v_2 and further values may equal v_1, move m_1 is called the `bestmove` and is the one the engine will play in a game.

Now suppose we have generated projected probabilities $p_1, \ldots p_\ell$ for the choice of moves. Then:

- p_1 is the projected chance of making the computer's first move. Its variance is $p_1(1-p_1)$.
- $p = p_1 + p_2 + p_3$ is the projection for making one of the top three moves. Its variance is $p(1-p)$. If $\ell \leq 3$, so that this adds to 100%, then the variance is zero.
- $p_{EV} = \sum_{i:v_i=v_1} p_i$ is the projected probability of making a move that is either the first move or has equal-optimal value. Again the variance is $p_{EV}(1-p_{EV})$.

We may exclude positions with only one legal move as trivial. About 8--10% of posiitons have tied-optimal moves. Just over half of those are with $v_1 = v_2 = \ldots = 0.00$. This can depend on how long the engine is run in its Multi-PV mode and whether there is a turn-number cutoff to discard dead-drawn endgame positions.

- $E[v] = \sum_{i=1}^{\ell} p_i v_i$ is the projected position value after making the move. The projected centipawn loss is $E[\delta] = v_1 - E[v] = E[v_1 - v] = \sum_{i=1}^{\ell} p_i \delta_i$ where $\delta_i = v_1 - v_i$. Note that we could sum the latter from $i = 2$.

To compute the associated projected variance, we need to invoke the formula

$$Var(X) = E\left[(X - E[X])^2\right] = E\left[X^2\right] - E[X]^2.$$

Incidentally, in the case of first-line match, X is the indicator function: $X = 1$ if m_1 is played, else $X = 0$. Then $E[X] = E\left[X^2\right] = p_1$. So the variance is $p_1 - p_1^2 = p_1(1-p_1)$. Now in the case of the average centipawn loss, $E\left[\delta^2\right]$ is just $\sum_{i=1}^{\ell} p_i(v_1 - v_i)^2$. There isn't any better way to compute it than $Var(\delta) = E\left[\delta^2\right] - E[\delta]^2$. (I'm not sure exactly what the convention on square brackets versus parens is supposed to be, but I use parens to mean "variance of" as a standalone quantity, whereas brackets mean the item inside gets expanded.)

Is the variance of the value itself the same? Yes: The expectation of the value is $E[v_1 - \delta]$, which equals $v_1 - E[\delta]$. Now using the first definition of variance,

$$E\Big[(v_i - (v_1 - E[\delta]))^2\Big] \;=\; E\Big[(v_i - v_1 + E[\delta])^2\Big] \;=\; E\Big[(E[\delta] - \delta)^2\Big] \;=\; E\Big[(\delta - E[\delta])^2\Big] \;=\; Var(\delta).$$

One good conceptual point of using "deltas" comes from the scaling. This uses the generalization that taking a difference $v_1 - v_i$ is the same as integrating the "unit metric" $d\mu(x) = 1$ from $x = v_i$ to $x = v_1$. Now suppose we want to consider other metrics. Suppose we say the incremental value of an extra centipawn ($= 0.01$) value is at face value when the game is dead-even but tapers off the more one side has an advantage. Then we want $d\mu(0) = 1$ and $d\mu(x) < 1$ for $x \neq 0$. If it tapers off in "affine linear proportion" to the absolute value of x, then the metric we want is

$$d\mu(x) \;=\; \frac{1}{1 + C|x|}dx$$

for any fixed constant C. Now suppose for sake of convenience that v_i as well as v_1 is positive (that is, the move m_i is a mistake but it doesn't cost all the advantage). Then the **scaled difference** is

$$\int_{x=v_i}^{x=v_1} d\mu(x) \;=\; \int_{x=v_i}^{x=v_1} \frac{1}{1+Cx}dx \;=\; \frac{1}{C}\ln(1+Cx)\Big|_{x=v_i}^{x=v_1} \;=\; g(v_1) - g(v_i),$$

where g is the function $g(x) = \frac{1}{C}\ln(1 + Cx)$. The essence is that we can precompute all the "scaled values" $v' = g(v)$ ahead of time, and then $\delta' = v_1' - v_i'$ becomes the simple "**scaled delta**." The definition of variance for scaled difference is then much the same as for unscaled difference, just with δ' in place of δ. Note also that $g(0) = 0$, so the "constant of integration" can be taken as zero.

If $v_1 \leq 0$ then $v_i \leq 0$ too since $v_i \leq v_1$ and the calculation is much the same. If $v_1 > 0$ but $v_i < 0$ (a mistake that puts you suddenly at a disadvantage), then you have to break up the computation into two pieces, one from v_1 down to 0 and then from 0 down to v_i. But actually, simply making $g(v_i)$ negative when v_i is negative handles this case gracefully too.

A final point is that this metric formulation immediately explains why the slope of average error is steeper on the negative side in the diagrams linked here. The scaling represents the psychologically perceived magnitude of the error. An average error e made when you are a pawn behind is greater in the diagram than an average error d when you are a pawn ahead. But e goes through a thinner part of the metric from -1.00 to $-1.00 - e$, whereas d goes from $+1.00$ to $1 - d$ through the fattest part of the metric. The thin/fat difference makes the scalings e' and d' come out pretty much equal. [In fact, my code makes $g(x)$ follow the slopes of those lines directly, as a function of rating, rather than use the particular logarithmic metric.]

Using **expectation loss** instead of (scaled) centipawn loss is a headache because the expectation values depend on rating all the time, but the mathematical procedure is similar.

Aggregate Stats

Now we add these up and take averages over sets of T-many positions. We immediately get projections for our major raw metrics:

- Projected **T1**-Matches: $\sum_{t=1}^{T} p_{1,t}$. As a percentage ("MMP" in my files): $\frac{1}{T}\sum_{t=1}^{T} p_{1,t}$.
- Projected **EV**-Matches: $\sum_{t=1}^{T} p_{EV,t}$, average version $\frac{1}{T}\sum_{t=1}^{T} p_{EV,t}$
- Average Centipawn Loss (**ACPL**, unscaled): $\frac{1}{T}\sum_{t=1}^{T} E[\delta_t]$.
- Average Scaled Difference (**ASD**): $\frac{1}{T}\sum_{t=1}^{T} E[\delta']$.

If game turns were independent, the variances of the summed quantities would simply add over t. The variances of the averages wound then divide by T^2. But...but... The resulting *adjusted variances* give rise to *adjusted sigmas* σ' and ***adjusted z-scores*** via the recipe:

$$z' = \frac{actual - projected}{\sigma'}$$

[Show example from the model's code on the CSE machine **`metallica`**.]

What justifies calling this a "z-score"? A fraction of the form $\frac{sample\ mean}{population\ \sigma}$ or $\frac{obs - sample\ mean}{population\ \sigma}$ is called "Studentized" (after William Sealy Gosset, who used the pseudonym "Student" so as not to disturb sales of Guinness ale). Note that in our case, σ' is a projected sigma---though the fact of its being adjusted actually means that it has been figured over the large population of (presumably) non-cheating players in the training sets of a lakh-plus games. Likewise the projected values in the model are trained over that population. So the fraction is legit. But what makes it a z-score, also called standard score? The answer is: *proximity to a normal model using large enough data*.

Central Limit Theorem and Its Assumptions

Any two normal distributions $N(\mu_1, \sigma_1)$ and $N(\mu_2, \sigma_2)$ can be mapped onto each other.

- $N(0, 1)$ is **standard**.
- $N(50, 5)$ approximates the distribution of flipping a fair coin 100 times.

Let's consider just the T_1-match for the time being. Binomial approximation. Main thing to notice is

that even when p is skewed far away from 0.5, the distribution and the approximating bell curve are still symmetrical around p. The bell curve has the symmetry by definition.

Central Limit Theorem: For **any** distribution D over a numeric domain, the mean d_n of n samples drawn independently from the same D is distributed in a way that converges to $\mathcal{N}(\mu, \sigma)$ for some fixed μ and σ, where $\mu = E_{d \leftarrow D}[d]$.

In the context of chess, CLT seems to say that whatever goes on distributionally inside players' heads, the distribution of the T_1-match %, being a mean of an independent(??) sample, approaches the normal distribution as the number n of samples used in the mean grows. The "Rule of 30" is a convention that $n = 30$ is usually good enough.

CLT presumes

- **independent** samples
- from the **same** (unknown) distribution

Neither assumption holds in chess:

- Consecutive chess moves by a player in a game used to form the sample are not independent. Carlsen-Anand Double Blunder example: Anand missed his opportunity because he was fixated on moving his rightmost pawn down the board.
- The moves in a sample are all from different positions.

Nevertheless, the distributions obtained are fairly close to normal.
[show spreadsheet examples]

Model Parameters and Virtual Players

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 parameter 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

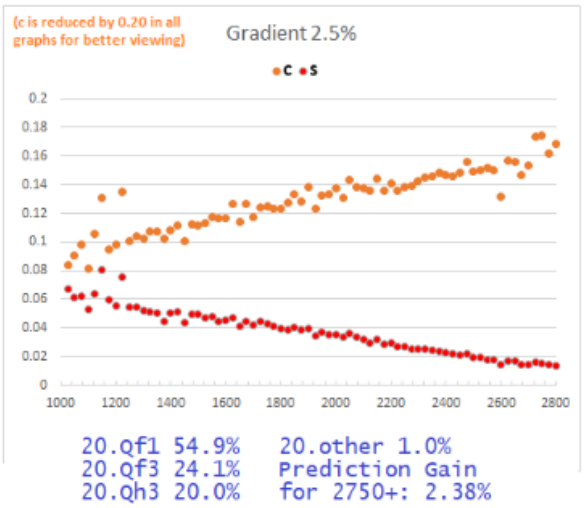
e_v as a function of IR_p in the IPR workflow. Then only s and c are freely fitted to any player's games.

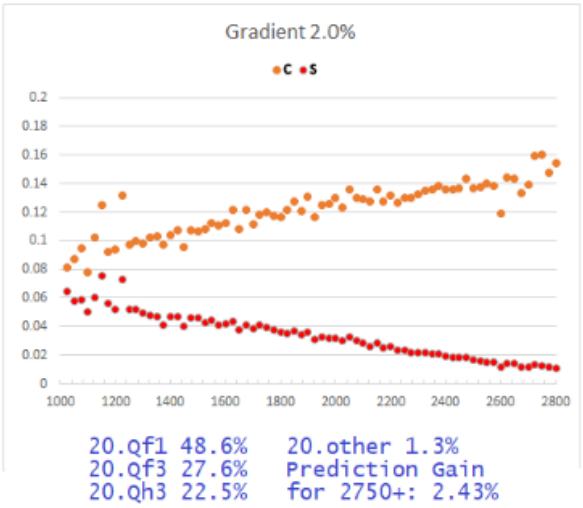
I call a parameter setting $Y = (s, c, \ldots)$ a *virtual player.* The IPR regression finds the closest virtual player to the given set of games. (My 2013 paper with Tamal Biswas used Z for virtual player or "agent", but that letter might confuse with z-score.)

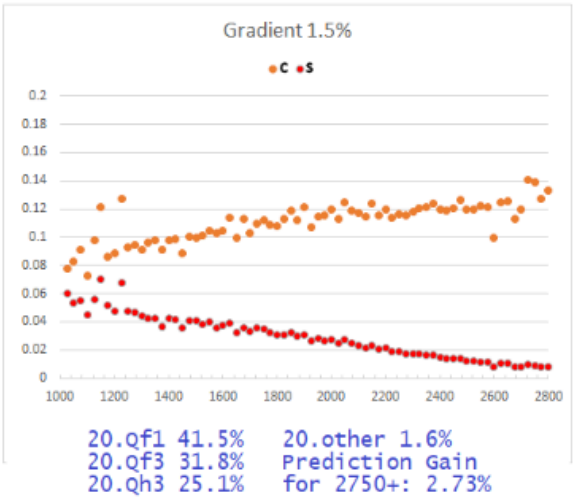
> **Important fact**: The virtual players obtained by regression for the seventy-three individual training sets of games by Elo 1025, 1050, ..., through Elo 2775, 2800, and 2825+, when plotted in the 2-dimensional (s, c) plane, fall closely into a 1-dimensional curve (which should in turn be rectifiable 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.

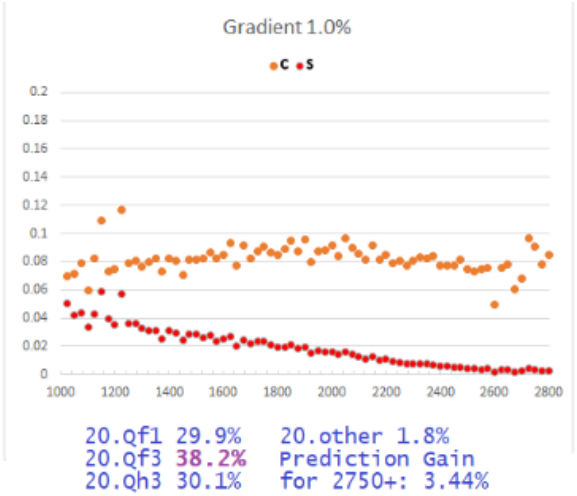
The model also has various hyperparameters which are not (intended to be) player-specific.

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:









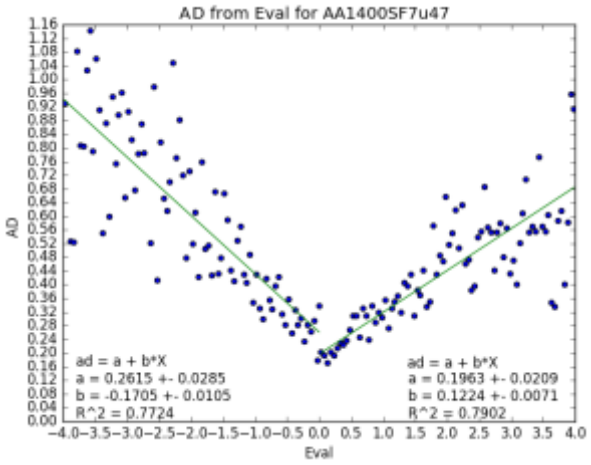
The upshot is that other aspects of the model are tweaked so that the "Important Fact" holds true to best advantage.

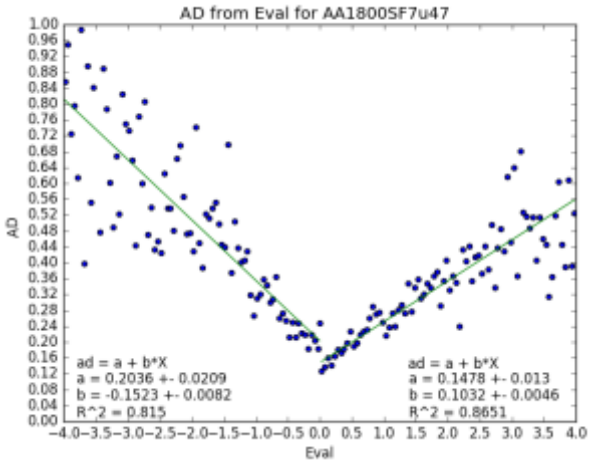
Hyperparameters

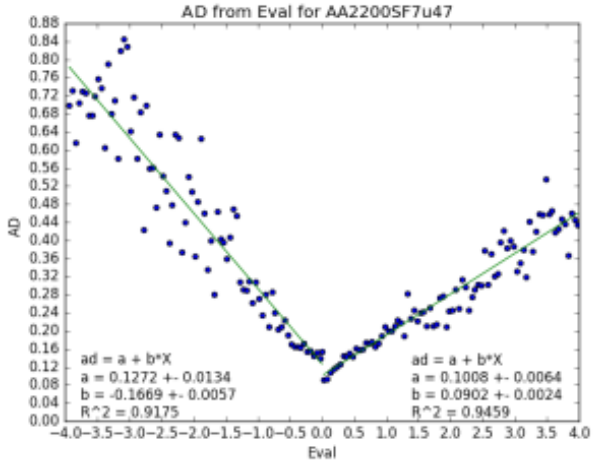
The model's numerous hyperparameters fall into three main groups, of which only the first part of the first group needs to be thought of operationally.

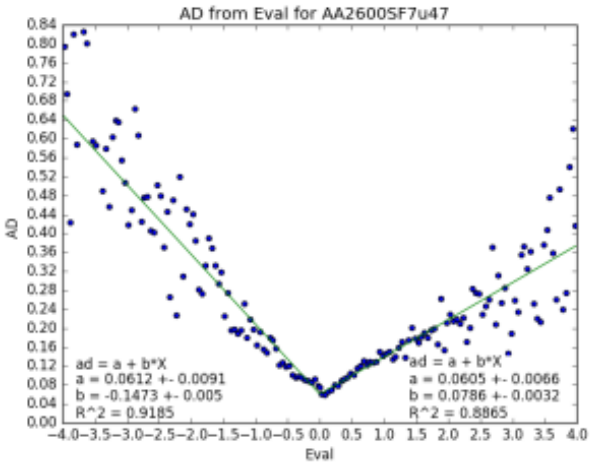
Rating-dependent:

- Scaling parameters a_m, b_m, a_p, b_p. These fit the lines in the diagrams of average error as a function of rating and the overall evaluation x of the position:









Here the line for evaluations $x \leq 0$ on the "m"inus side is $a_m + b_m x$, while that on the "p"lus side for $x > 0$ is $a_p + b_p x$. Notice that the minus-side intercepts a_m are higher than the a_p values, considerably so for lower-rated players. I explored the asymmetry and a possible main reason for it in this article. Basically the point is that when $x > 0$, a large mental error may be mitigated by a fortunate escape to a draw by perpetual check, whose 0.00 value will make the error no larger than the advantage x itself. Whereas, the case $x \leq 0$ has no such safety net and the loss of value can be bottomless. These parameters are used by the model to "flatten" the magnitude of error in positions of any value x to be a_m or a_p on average, independent of x (other than its sign).

- Parameters uz, vz, wz controlling the neighborhood of the value 0.00. They stand for "under 0", "over 0", and "width of 0", respectively. The last is unused---zeroed out. They smooth the difference between a_m and a_p; one can say they effectively make the regression lines in the above diagrams meet up at 0.00.

The code uses a mapping $R \mapsto (a_m, b_m, a_p, b_p, uz, vz)$. The 6-tuple is said to have **rating basis** R. This is fine as long as R is given up-front. When we want to estimate R for a set of games, we start with an initial guess R_0 and its associated 6-tuple. Thereafter, when we update the estimate to R', we then also need to update the 6-tuple. This disturbs the previous fit, so we iterate again to make R'' and update the tuple again, and so on... In almost all cases this converges quickly, but occasionally the process races to negative infinity or yo-yos---the code stops after 20 iterations regardless. Where things get ***even dicier*** is when you apply a data filter F that depends on ASD or some other use of scaled values and ***then*** change the rating basis. The selection of data is left undisturbed unless and until you apply a re-filtering operation. *More about this later---continuing with hyperparameters for now.*

- Scaling parameters c_m and c_p. **Not used**. They fit the "C" in the previously discussed logarithmic scaling formula, for $X \leq 0$ and $X > 0$, respectively.
- Expectation-loss logistic-curve parameters la, lb, lk, lq. **Used only for show.** These fit the generalized logistic curve $A + \dfrac{K - A}{1 + Qe^{-Bx}}$ in an effort to convert from x to the scoring

expectation (for a player of a given rating R *facing an opponent of the same rating*).

- The **prediction-error parameter** ft. It is the same as the epsilon ϵ in "error model 5" in this article and stands for "fallibility threshold" beyond which deviations can be ascribed to the player. It keeps a fairly steady value between 0.04 and 0.05 until the rating goes above 2600, whereupon it rises above 0.07 at 2800. This shows the model's input values losing some of their authority for the world's most elite players. Extrapolating shows the model seriously losing resolution above Elo 3100 or so. The parameter is used to calibrate a second kind of z-score based on "deviance from the range of human predictability." **Operationally we can ignore this too**.

Rating-Independent:

- The main one is the "**gradient**" v. It controls the influence of lower-depth move values. It is currently set to a very conservative value 0.035. Smaller values give more weight to "traps" and lower-depth temptations. Originally v was supposed to be the "variance" of a main depth-of-thinking parameter d. The latter is kept as a relic with the fixed value $d = 20$ but is otherwise unused.
- There is a cap ec on evaluations and dc on differences in value. Both are set to 20.00 in pawn units, i.e., 2,000 centipawns. Most chess programs use 1000.00 as the value of checkmate. So if you can play checkmate but blunder and allow the opponent to give checkmate instead, that would be -2000 as a raw magnitude of error, which would overwhelm any other errors you made. It gets capped instead at -20, as if blundering two queens. The scaling then reduces the magnitude a whole lot more.
- The parameter pp is a "patch power" alternative to e_v which will be briefly discussed later. **Currently not used**.
- Various unnamed parameters in menu option 2 have values that can likewise be customized but are best left fixed. The most important one defines a fixed **virtual depth** scale in which the highest depth actually obtained in analysis is mapped to depth 20. The mapping is independent of the chess program, but its reality---along with the scaling---is why the model has to be built separately for each chess program used.

Tethered Parameters:

- The parameter called "co" is tied to have the same value as the sensitivity parameter s. We can ignore why and just think of s.
- The parameter e_v is trained freely on large data, but treated as tethered to the other main parameters s and c when regressing on small data---else fits do not converge.

The net effect is treating e_v as another rating-dependent hyperparameter---thus we really get a mapping $R \mapsto (a_m, b_m, a_p, b_p, uz, vz, e_v)$. **The fact of this mapping is the only thing that really needs bearing in mind.**

Unused Parameters and the Full General Terms

There are a whole bunch of other parameters that are unused, but whose potential usage may explain some choices that were made. They are covered in a long comment in the `IRall1file.cpp` code that begins, "The $64,000 place where model parameters are reflected, aside from the possible (and deferred) impact of d and v on the weighting of depths." As explained in the next set of notes, Tamal Biswas and I allowed for a fully general linearization of parameters multiplying the values v_1 and v_i, their (rating-dependent!) scalings v'_1 and v'_i, and the corresponding points-expectation values e_1 and e_i (which are also rating-dependent!) by the "objective term"

$$T_1 \;=\; p \cdot v_1 \;+\; q \cdot v_i \;+\; r \cdot v'_1 \;+\; s \cdot v'_i \;+\; t \cdot e_1 \;+\; u \cdot e_i$$

(which is not to be confused with "T1-match"). To be sure, there is redundancy especially with the last two. Results trying for full generality were not promising---too much numerical wonkiness---so the ones other than s are kept zeroed out. Actually, s is tethered to always equal $-r$, so that s multiplies the scaled difference $v'_1 - v'_i$ which is just the scaled delta δ_i of the i-th best move. To reduce clutter, the "eval handler" (main menu option 2) has default-enabled settings `mulDiffs` and `invertParams` that make s do this while keeping $r = 0$ in the display and that make s a divisor rather than multiplier. Likewise, the lower-depth "swing values" are amalgamated into w_1, w_i, and their scaled analogues w'_1, w'_i and linearized with four other parameters to make the "subjective term"

$$T_2 \;=\; e \cdot w_1 \;+\; f \cdot w_i \;+\; g \cdot w'_1 \;+\; h \cdot w'_i\,.$$

Again, h is tethered to $-g$ so that it actually multiplies the "perceived inferiority" $w'_1 - w'_i$, though it stays as a multiplier. Moreover, there is an option to make s a divisor of both T_1 and T_2. But in fact, all four of e, f, g, h are kept zeroed out, so there is effectively no freely-separate swing term (`noSwing` in the code). There is also a third term T_3 with parameters that come into play in the following cases:

- tz ("to zero"): move m_i has value $v_i = 0$ but $v_1 > 0$;
- fz ("from zero"): $v_1 = 0$ and $v_i < 0$;
- bz ("both zero"): $v_1 = v_i = 0$;
- sf ("sign flip"): $v_1 > 0$ but $v_i < 0$, i.e., move m_i converts advantage to disadvantage.
- ne ("negative-eval case"): $v_1 < 0$.

The above-described use of uz and vz seems to do a good-enough job of reflecting these cases, so these last five parameters are also kept zeroed out---and likewise powers T_2^a and T_3^b (to go with T_1^c, except there are also settings that could allow adding these terms before powering) are ignored. The element of "swing" is handled in a tightly-clamped regime that has only the free parameter e_v, multiplying only cases where $v_i = v_1$ (not necessarily both equal to 0.00), under a strict separation of the projection mechanism from measuring actual results.