ELO REGRESSION
EXTENDING THE ELO RATING SYSTEM

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Jonathan Dorsey

May, 2019

ELO REGRESSION

EXTENDING THE ELO RATING SYSTEM

Jonathan Dorsey

Thesis

Approved:	Accepted:
Advisor Dr. Sujay Datta	Dean of the College Dr. Linda Subich
Faculty Reader Dr. Mark Fridline	Dean of the Graduate School Dr. Chand Midha
Department Chair Dr. Timothy O'Neil	Date

ABSTRACT

The Elo rating system has long been a mainstay for comparing the strengths of chess players. The utility of the system lies in its computational simplicity and its prospective nature: the ratings for any given time period are calculated using only information from previous time periods. By calculating ratings for each time period of interest, Elo can be used to model the relative changes in players' skills over time in an ad hoc fashion. The main weakness of this approach is that the prospective nature of Elo means not all information is being used.

The main purpose of this research is to display a modified version of the Elo rating system that better handles retrospective ratings. We achieve this by extending Elo to explicitly incorporate time as a covariate. We call our modified system "Elo regression." We find that Elo regression outperforms Elo, in terms of predicting the outcomes of held out games, on two different data sets, and performs equivalently on a third.

A secondary purpose of this work is to display the benefits of reinterpreting pre-existing algorithms as particular methods for fitting a probabilistic model. Once an algorithm has been understood this way, principled modification of the algorithm is possible via changing the underlying model, the method of parameter estimation, and the method of calculating or approximating those estimates.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The Elo rating system, or just "Elo," is a method of assigning ratings to players based only on the outcomes of games, in order to assess their relative skills. Elo was invented by its namesake, physicist Arpad Elo, in the 1950s [9]. Since its adoption by the World Chess Federation (FIDE) in the 1970s, it has been the primary metric for measuring the strength of chess players. Chess ratings are very important to players, as FIDE uses ratings to determine invitations to tournaments, pairings at tournaments, and titles such as Grandmaster [5]. Elo is also applied to other zero-sum games, such as soccer and basketball [7, 15].

The fundamental idea behind Elo is that ratings can be used to predict the expected outcomes of future games. After each game, the players' ratings can be updated based on the discrepancy between the actual outcome and the expected outcome. This way, the ratings evolve over time and correct themselves based on the observed outcomes.

The main strength of the system is its computational simplicity, which allows chess federations to maintain the ratings of hundreds of thousands of players with modest computational resources [4]. Another benefit of Elo's simplicity is that players can in principle compute their rating updates themselves by consulting tables and using a pocket calculator. This gives the ratings a sense of fairness since the details of the

system are not hidden from the average player behind complicated mathematics and algorithms.

Until now, we have been focusing on *prospective* uses of Elo ratings. That is, looking to the future to decide who should be invited to a tournament, or who will be worthy of the title of Grandmaster. This is what chess federations use Elo ratings for. Elo is also used to retrospectively or historically to analyze players' abilities over time [9, 1, 2]. This exercise is of general historical interest to chess players and fans, since they want to compare chess players across history and find the players with the greatest legacies. This is a very different way to use ratings. As an analogy, consider the differences between singling out the best and brightest current scientists and deciding whether, in hindsight, a scientist's work has been historically important enough to warrant a lifetime achievement award. What seems important in the moment may not pan out in the future. This is similar to the situation in chess ratings, where the players who seem the best right now turn out, with the benefit of hindsight, to be overrated. Thus, a rating system must decide to what extent new information informs only new ratings, or whether to use this new information to try to retroactively correct old ratings. We call any system that uses new information to retroactively correct old ratings a *retrospective* rating system. A system which never changes its past ratings, that is, it only uses information from the past to predict the future, we call a *prospective* rating system. The decision about how much to retroactively adjust ratings is implicitly an assumption about how quickly skill can change over time. If skill can change arbitrarily quickly, then there is no justification for revising old ratings. However, if we believe there are constraints on how quickly skill can change, then when a player's rating quickly drops, we may infer that it must have been too high to begin with.

As we mentioned above, Elo can be used to model skill over time: since a player's rating changes after each game, keeping track of his or her rating over time provides

some measure of how that player's skill is changing. Here and throughout the rest of the paper we use *skill* to refer to the "true" underlying ability of the player, and *rating* to refer to an estimate of skill. We put "true" in quotes because it is doubtful that any single number accurately reflects all aspects of chess ability. The fact that Elo is a prospective rating system, which, to reiterate, means that at each time period it only uses data from the past to predict the future, limits its accuracy for historical analysis, where we are concerned with finding the best ratings for each time period using data from all time periods. Moreover, the Elo system does not explicitly attempt to model the dependence of skill on time. The relationship is the ad hoc result of the Elo updating method. In this paper, we propose a way to combine the core underlying model of Elo with linear regression, so that we may explicitly regress skill on time. We call the resulting model "Elo regression." Using an appropriate basis expansion with linear regression allows for flexible modeling of any relationship between skill and time, not merely straight line relationships.

By making the relationship between skill and time more formal, we can get better predictive performance on held out data, which we use as a proxy for learning the underlying skills of the players. We also gain more control over the assumptions of the model, for example, through the basis expansion we choose, which lets us incorporate domain knowledge and common sense. Another benefit is that the model is conceptually very simple, since it is the combination of the simple Elo system with the well-understood technique of linear regression. The main downside of this approach is the heavy computational burden, which will be discussed below.

Several other chess rating systems have been proposed as improvements over the Elo system. Perhaps the most well-known are the Glicko and Glicko2 rating systems [10, 11]. There is also the TrueSkill system, which was originally designed for rating Xbox players [12]. Both the Glicko rating systems and TrueSkill are based on a Bayesian model that accounts for uncertainty in the player's current rating. Thus,

these systems trade off some of the simplicity of Elo for more accuracy. Regardless, these systems follow the same prospective, sequential-update-based approach as Elo, as none of them were designed for retrospective analysis. Notably, however, a modified version of TrueSkill, called TrueSkill Through Time, has been used to explicitly model chess skill through time [8]. The primary difference between Elo regression and TrueSkill Through Time is that Elo regression allows specifying a parametric form for the relationship between skill and time, while TrueSkill Through Time does not. Moreover, Elo regression allows games to occur on a continuous time scale, while TrueSkill Through Time uses a discrete time scale. TrueSkill through time has the major benefit of using Bayesian methods to automatically estimate an uncertainty associated with each rating. We will not discuss other approaches any further.

CHAPTER II

THE ELO RATING SYSTEM

This chapter presents the technical details of Elo following the classic exposition from [9]. Then we discuss a probabilistic interpretation of the rating system. That leads nicely into a discussion of our modified version of Elo, which we take up in the next chapter.

The Technical Details of Elo

The Elo rating system works as follows. Every player i has a rating R_i. We will discuss where these ratings initially come from later. When two players i and j compete, Elo produces an expected score for each player. The expected score for player i is

$$E_i = \frac{1}{1 + 10^{(R_j - R_i)/400}} \tag{1}$$

and likewise the expected score for player j is

$$E_j = \frac{1}{1 + 10^{(R_i - R_j)/400}}. \tag{2}$$

If we let $Q_i = 10^{R_i/400}$ and $Q_j = 10^{R_j/400}$, then we have

$$E_i = \frac{Q_i}{Q_i + Q_j} \tag{3}$$

$$E_j = \frac{Q_j}{Q_i + Q_j} \tag{4}$$

which shows that $E_i + E_j = 1$.

In chess, a score of 1 represents a win, a score of 0 represents a loss, and a score of 1/2 represents a draw. Thus, expected scores of 0.8 for white and 0.2 for black are consistent with white winning 80% of the games and losing the other 20%, but those expected scores are also consistent with white winning 60% of the games and drawing the other 40%. That is, Elo and chess scoring both treat a draw as exactly equivalent to half of a win and half of a loss. We discuss this point further in the next section and touch on it again when generating synthetic data.

From the game the players actually play, we get the real scores S_i and S_j. We then update the ratings R_i and R_j based on the discrepancy between the expected outcome and the actual outcome. The update equations follow Elo's original suggestion to change each rating by an amount proportional to the difference between the real and expected scores. This way a large discrepancy results in a large correction, and a small discrepancy results in a small correction. Writing R_i' and R_j' for the new ratings, we have

$$R_i' = R_i + K_i(S_i - E_i) \tag{5}$$

$$R_j' = R_j + K_j(S_j - E_j) \tag{6}$$

where K_i is called the K-factor for player i, and similarly, K_j is the K-factor for player j. The K-factor decides how much a player's rating can change per game. Notice

that if there is a common K factor $K = K_i = K_j$, then the updates sum to zero:

$$K(S_i - E_i) + K(S_j - E_j) = K((S_i + S_j) - (E_i + E_j)) \tag{7}$$

$$= K(1 - 1) \tag{8}$$

$$= 0. \tag{9}$$

Thus, rating points are transferred from the loser to the winner, without being created or destroyed.

Commonly, K-factors are set between 10 and 40. Different chess federations have different rules about how K-factors are set for different players. If the K factor for a player is set too small, then the ratings will be slow to converge on the true skill of that player, especially as that skill changes over time. If the K factor for a player is set too high, then his or her rating will only reflect his or her recent performances and thus can fluctuate too wildly. Typically K-factors are set higher for newer players, so that their ratings can adjust more quickly, and lower for older or higher-rated players, so as to keep their ratings from oscillating too wildly [4].

Besides the exact method for choosing the K-factor, the other main choice we have brushed over is how brand-new, unrated players are handled. Typically unrated players are given a provisional rating which is calculated by a method that adapts to their performance more quickly than the traditional methods. Also, games involving a provisionally rated player affect only the rating of that player. For example, players may be given a rating after their first tournament that depends not only on how many wins, losses, and draws they achieve, but also directly on the average rating of their opponents. Since chess tournaments use a Swiss format where players tend to be paired with other players of a comparable skill level, especially in the later rounds, the average rating of a player's opponents in a tournament contains valuable information about that player's skill.

A Probabilistic View of Elo

The Elo rating system can be cast as a very particular way of fitting a probabilistic model. This view is probably not novel, but we could not find a definitive reference for it.

Let each player i have a skill level θ_i, and let $\vec{\theta}$ refer to the vector of all such skills θ_i. When players i and j play, the outcome is a random quantity that depends on their difference in skill. Rather than merely asking if $\theta_i > \theta_j$ to determine the winner, which is the same as asking if $\theta_i - \theta_j > 0$, we can ask if a logistic random variable with mean $\theta_i - \theta_j$ and scale parameter $s = 1/c$ is greater than zero. The probability of this happening is $\sigma(c(\theta_i - \theta_j))$ where

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

is the unit logistic function, which is the cumulative distribution function of the standard logistic distribution. Notice that if we set $c = \ln(10)/400$, then the expression for the probability of player i winning is equal to the expected scoring function from the Elo equations above:

$$\sigma\left(\frac{\ln(10)}{400}(\theta_i - \theta_j)\right) = \frac{1}{1 + \exp\left(-\frac{\ln(10)}{400}(\theta_i - \theta_j)\right)} \tag{11}$$

$$= \frac{1}{1 + 10^{(\theta_j - \theta_i)/400}}. \tag{12}$$

These two interpretations agree if there are no draws, since then the expected score is just the probability of a win times 1, plus the probability of a loss times 0. The choice of c does not lead to different inferences, but merely decides the scale of the ratings.

So, assuming that games cannot end in draws, we have a parametric model of how the outcome of a game is decided. We can extend it to multiple games by assuming

that each game is independent of every other game. Thus, the total likelihood is a product of factors of the form $\sigma(c(\theta_i - \theta_j))$, one for each game, where i and j are the two players who participated in the game, and i is the winner of the game. Given a set of games, we can estimate the underlying skills, $\vec{\theta}$, by maximum likelihood. That is, choosing as our estimate of $\vec{\theta}$ the values that maximize the probability of the games we observed. As above, we will call the estimated skills, "ratings."

Before discussing the estimation procedure further, there are some issues to address. Strictly speaking, there is no one maximum likelihood estimator, since the model is not identifiable. This means that different configurations of the parameters lead to exactly the same distribution of game outcomes. This happens because the outcomes depend only on the differences of the skills, not on the skills themselves. So translating all the skills by the same amount does not change anything. This issue is handled by the gradient ascent approach we use, which will be described below. The other issue is that the likelihood is unbounded when a player has all wins (or all losses), since we can always increase the likelihood by driving his or her rating up (or down in the case of losses). This problem rarely occurs in real data, and can be accounted for by using a penalized maximum likelihood approach. This is akin to the issue of complete separation in logistic regression.

As is typical with maximum likelihood estimation, we will actually maximize the log likelihood since its derivatives are easier to calculate. The log likelihood is a sum of terms of the form $\ln \sigma(c(\theta_i - \theta_j))$. So the partial derivatives are the sums of the derivatives of each of these terms:

$$\frac{\partial}{\partial \theta_i} \ln \sigma(c(\theta_i - \theta_j)) = \frac{\frac{\partial}{\partial \theta_i}\sigma(c(\theta_i - \theta_j))}{\sigma(c(\theta_i - \theta_j))} \tag{13}$$

$$= \frac{c\sigma(c(\theta_i - \theta_j))(1 - \sigma(c(\theta_i - \theta_j)))}{\sigma(c(\theta_i - \theta_j))} \tag{14}$$

$$= c(1 - \sigma(c(\theta_i - \theta_j))). \tag{15}$$

Similarly, we find

$$\frac{\partial}{\partial \theta_j} \ln \sigma(c(\theta_i - \theta_j)) = -c(1 - \sigma(c(\theta_i - \theta_j))). \tag{16}$$

And of course, the other partial derivatives with respect to θ_k for $k \neq i$ and $k \neq j$ are zero. Thus, we have all the information needed to calculate the gradient of the log likelihood.

We can use the gradient to optimize the log likelihood through gradient ascent, which works as follows:

1. Choose some initial ratings: $\vec{R}_0$.
2. Choose a small step size $\varepsilon > 0$.
3. Repeat the following two steps until the log likelihood stops increasing:
   (a) Calculate the gradient of the log likelihood at the current ratings: $\nabla l(\vec{R}_i)$.
   (b) Update the ratings by taking a small step in the direction of the gradient: $\vec{R}_{i+1} = \vec{R}_i + \varepsilon \nabla l(\vec{R}_i)$.

Since the gradient points in the direction of steepest increase, this procedure will gradually move the ratings closer to a local maximum of the log likelihood, provided we take small enough "steps."

A crude, but sometimes useful, approximation of gradient ascent is when we estimate the gradient of the log likelihood at each step by only using one data point. Recall that the gradient of the log likelihood is a sum of terms, one for each piece of data we have. If we use each game exactly once, in the order the games are played, then the resulting "online" learning algorithm is exactly the original Elo rating system with a global fixed K-factor.

To see this, recall the partial derivative of the log likelihood with respect to the winner of the game and note that

$$\frac{\partial}{\partial \theta_i} \ln \sigma(c(\theta_i - \theta_j)) = c(1 - \sigma(c(\theta_i - \theta_j))) = c(S_i - E_i) \tag{17}$$

because $S_i = 1$ on the assumption that player i wins and $E_i = \sigma(c(\theta_i - \theta_j))$ as we noted earlier. Likewise,

$$\frac{\partial}{\partial \theta_j} \ln \sigma(c(\theta_i - \theta_j)) = -c(1 - \sigma(c(\theta_i - \theta_j))) = c(S_j - E_j) \tag{18}$$

because $-(S_i - E_i) = S_j - E_j$ as we already noted. Thus, if we apply the online gradient ascent updates we see that the new ratings are

$$R_i' = R_i + \varepsilon c(S_i - E_i) \tag{19}$$

$$R_j' = R_j + \varepsilon c(S_j - E_j) \tag{20}$$

which are exactly the same as the Elo updates when the K-factor is set to εc, or equivalently, when we set the step size ε to K/c.

Thus, if we perform an online form of gradient ascent, where the data is streamed to the model in chronological order and only used once before being discarded, we have the exact Elo rating system, with fixed K-factors.

Now we can see how gradient ascent handles the identifiability issue. In general, a gradient ascent update looks like a sum of many Elo updates as described above. As we noted in the earlier Elo section, these updates neither create nor destroy rating points; they only move them around. So if we use gradient ascent, then we are constrained to a subspace of the parameter space on which $\sum \vec{\theta}$ is constant. When the parameter space is restricted in this way, the model is identifiable.

The other problem we have brushed aside until now is how to handle the likelihood of a draw, because we felt it distracts from the main thread of the exposition. The model as we have described it thus far can only handle wins and losses: we use the logistic function to convert a difference of skills directly into a probability of winning. At first there is no clear and natural way to convert a difference of skills into a probability of winning and a probability of drawing (the probability of losing then being one minus their sum). Elo skirts around the issue by talking in terms of expectations rather than probabilities, effectively treating draws as 1/2 a win and 1/2 a loss. In fact, there is a natural way to rephrase this approach in terms of probabilities. We model the outcome of each real game as depending on the outcomes of two independent "pseudo-games." If one player wins both pseudo-games, then he or she wins the game, and his or her opponent loses the game. If each player wins one pseudo-game, then the game is a draw. Thus, we make it explicit that a draw is 1/2 a win and 1/2 a loss.

Letting $p = \sigma(c(\theta_i - \theta_j))$ be the probability of player i winning a pseudo-game against player j, we have that the probabilities of player i winning, losing, and drawing against player j are p^2, $(1-p)^2$, and $2p(1-p)$ respectively. The expected score of a game generated in this fashion is still $p = \sigma(c(\theta_i - \theta_j))$, since

$$1 \times p^2 + \frac{1}{2} \times 2p(1-p) + 0 \times (1-p)^2 = p. \tag{21}$$

We can implement this procedure for handling draws by halving the gradient ascent step size and recoding the data so that each game becomes two games, according to the pseudo-game idea above.

CHAPTER III

ELO REGRESSION

Now that we have discussed the probabilistic view of Elo, we are ready to modify it in a principled way. We noted that Elo models the outcome of a game as depending on the logistic function applied to a difference in skills. The maximum likelihood estimates for the parameters of this model are then approximated by online gradient ascent. Thus, we have three avenues for experimentation: changing the underlying model, changing the method of estimation, and changing the method of calculating or approximating those estimates. We make changes in all three areas. The resulting model we call "Elo regression," since it explicitly regresses skill onto time. Below we discuss the mathematical details and the strengths and weaknesses of Elo regression as compared to Elo.

Basis Expansion

To change the underlying model, we will leave the logistic function aspect alone and simply replace the fixed skill θ_i of player i with a function $\theta_i(\vec{x})$ that depends on a vector of covariates $\vec{x}$. We use linear regression together with a basis expansion to model non-linear relationships. We choose this approach because it is probably the simplest way, mathematically, to model θ_i as a function of covariates.

For our purposes throughout the rest of this work, the only covariate we consider is time, which we denote by t. Thus, we model every player's skill as a function of

time like so:

$$\theta_i(t) = \sum_k \beta_{ik} f_k(t). \tag{22}$$

It bears repeating that using linear regression does not constrain the skill level θ_i to be a linear function of the covariates, since we can choose non-linear basis functions f_k. Indeed, by choosing different basis functions, this parametric form can model polynomials, piecewise linear functions, natural cubic splines, and more.

The basis functions we choose to use for the remainder of this work are the so-called "Gaussian radial basis functions,"

$$f_k(t) = \exp\left(-\frac{(t - C_k)^2}{L^2}\right) \tag{23}$$

where C_k and L are the center point and length scale of the basis function, respectively. The length scale acts as a regularization term: larger length scales lead to smoother functions. The centers must be chosen so that there are enough of them in the right places to capture the variability of the curve being modeled. In practice we choose the centers and length scale using some form of cross-validation. See the figures below for more information about how these basis functions model smooth curves.

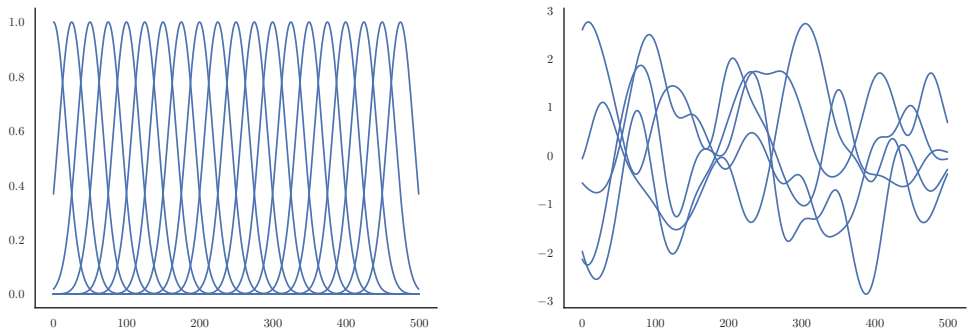

Figure 1: Left: Gaussian radial basis functions with uniformly spaced centers plotted individually. Right: Random linear combinations of the basis functions on the left.

Penalization

We change the method of estimation only slightly, by switching from maximum likelihood estimation to penalized maximum likelihood estimation. We use an L_2 penalty, also known as Tikhonov regularization, ridge regression, or weight decay. That is, instead of maximizing the log likelihood, we maximize the log likelihood minus a multiple, λ, of the L_2 norm of the coefficients. Using an L_2 penalty shrinks the parameter estimates toward zero [14]. In combination with the way radial basis functions put each coefficient on equal footing, the L_2 penalty shrinks the overall ratings toward zero in a nice uniform way. By arranging to make zero the average skill level, the L_2 penalty shrinks our skill estimates toward the mean. We used only a touch of L_2 penalty, to make sure the estimates always exist and to provide a small amount of regularization.

Minibatch Gradient Ascent

We change the method of calculating the estimates from online gradient ascent to minibatch gradient ascent. Minibatch gradient ascent works by using a small random sample of the data (called a batch or minibatch) to calculate the gradient of the log likelihood. For example, we can use 100 games instead of a full data set of millions of games. This obviously saves computational effort. The idea is that, after some rescaling, the gradient of the log likelihood is a mean of terms, one for each game. This is to say that the gradient is the expectation of a certain function taken over a population defined by the full data set of games. If we use only a small random sample of the games to compute the gradient, then we are approximating a mean by a sample mean. By the law of large numbers, this approximation has a relatively high quality for relatively small batch sizes. Subject to some technical conditions on step size, minibatch gradient ascent converges almost surely to a local maximum [13].

Thus, minibatch gradient ascent allows us to calculate the gradient with an amount of computational effort essentially independent of the amount of data at hand. This together with the low memory overhead of first-order optimization methods allows gradient ascent to scale to massive data sets and models with many parameters.

Now we show the derivatives needed to implement gradient methods. Once again we will show the simpler case, assuming no draws. As before, the fully correct results can easily be recovered by treating a draw as 1/2 a win and 1/2 a loss.

The parameters of the model are no longer the θ_i but rather the β_{ik} on which they depend. So we need to calculate the partial derivatives with respect to these β_{ik}. As we noted earlier, the choice of c is arbitrary, so we will let $c = 1$. Then the derivative is

$$\frac{\partial}{\partial \beta_{ik}} \ln \sigma(\theta_i(t) - \theta_j(t)) = \frac{\frac{\partial}{\partial \beta_{ik}} \sigma(\theta_i(t) - \theta_j(t))}{\sigma(\theta_i(t) - \theta_j(t))} \tag{24}$$

$$= \frac{\frac{\partial}{\partial \beta_{ik}} \left(\sum_l \beta_{il} f_l(t) - \beta_{jl} f_l(t)\right)}{\sigma(\theta_i(t) - \theta_j(t))} \tag{25}$$

$$= \frac{f_k(t)}{\sigma(\theta_i(t) - \theta_j(t))}. \tag{26}$$

Similarly,

$$\frac{\partial}{\partial \beta_{jk}} \ln \sigma(\theta_i(t) - \theta_j(t)) = \frac{-f_k(t)}{\sigma(\theta_i(t) - \theta_j(t))}, \tag{27}$$

and all other partial derivatives are zero.

We see once again that the derivatives of interest are opposites of each other. This implies that the sum, across all players, of the parameters corresponding to a particular basis function is constant, which once again is enough to ensure identifiability.

Overview of Modifications

To review, Elo regression models player skill as a function of time using linear regression with Gaussian radial basis functions. The exact basis can be chosen by

performance on a validation set. The maximum likelihood estimates are penalized slightly with L_2 regularization to ensure the estimates exist, and then approximated using minibatch gradient ascent.

Comparison to Elo

Elo regression is not designed to be a general purpose rating system and thus has severe limitations in that regard. First, it is much more computationally intensive than Elo, so maintaining ratings for many players across many games would be more costly. Though, it is worth noting that some of the cost would be offset by using the old parameters as the initialization for new estimates instead of training from scratch. Second, the hyperparameter optimization involved in Elo regression and the more complicated math, as compared to Elo, could make players feel that the ratings are arbitrary and esoteric. Finally, non-prospective rating systems would probably annoy players by removing their feeling of control over their ratings. With a non-prospective rating system, a player's rating can change based on other players' performances. Players generally like to feel that ratings are simple: when they win their rating goes up and when they lose it goes down.

However, Elo regression is designed to use all possible information from a data set for retrospective or historical analysis of player skill over time. As we will soon see, it improves upon Elo in this regard.

CHAPTER IV

DATA

Throughout the rest of this thesis we directly compare the performance of Elo and Elo regression on three data sets. This chapter describes and explores the data we use to do so.

The first is a set of real chess game data that was used for a Kaggle competition [6]. This data set is of interest not only because it is real data, but also because of its large scale in terms of number of games and number of players. The second is a synthetic data set designed to show how the models compare when skills are changing relatively quickly, but there are many games per player per time period to learn from. The final data set is again synthetic and designed to showcase the weakness of using prospective ratings for retrospective analysis.

The Kaggle Chess Data

As we mentioned earlier, this data comes from a Kaggle competition, the Deloitte/FIDE Chess Rating Challenge. We selected this data primarily because it was the largest available data set of real chess game outcomes we could find. The competition featured anonymized data from the World Chess Federation's (FIDE) database. Following the end of the original competition, a follow-up data set was released with more players and more games. We used this follow up data.

The follow-up data comes from a recent 135-month period of professional chess games, again, extracted from the database of the World Chess Federation (FIDE). The specific dates of the games are not reported, only the month. All together, the data consists of 3,140,354 games between 87,987 players. The data was constructed to be well-connected. The initial player pool was four prominent world champion players. Then players were iteratively added to the player pool if they had played a certain number of games against other players in the pool. Thus, the resulting dataset consists of those players who are in some sense only a few degrees of separation from a world champion. This excludes many players from isolated regions, as well as players who generally only play within their own age group. Since this data set sidesteps this very real issue, the second synthetic data set is designed to simulate a situation where players are not well-connected.

We split this data into three sets: 70% of it (2,198,247 games) was used as a training set, 15% of it (471,053 games) was used as a validation set, and 15% of it (471,054 games) was used as a test set. The training set was used to fit all the models. The validation set was used for tuning hyperparameters. The test set was used only once at the end for assessing the final accuracy of the models.

With 2.2 million games, 88,000 players, and 135 time periods, we only have around 0.2 games per player per time period. So, while the data set is quite large computationally speaking, it is relatively sparse in terms of information about how each player's skill is changing over time.

Below we include some exploratory graphs to give the reader some sense of the data. First, we can see that most of the games happen in the later half of the data. Looking at the histogram, and noting the logarithmic y-axis, we can see that the distribution of the number of games played per player is very skewed. This means that most players have relatively few games, but there are a few players with many games. Finally, the last graph shows that in each month many new players join the

rating pool. All of these factors contribute to making this a tricky data set to fit a model to. The last point in particular means that there are always many players in each month for which we have no previous data. For a prospective system, any match involving these players is a complete unknown. As a last note, approximately 30% of the games in this data set were draws. Thus, handling draws is a serious concern for chess data.

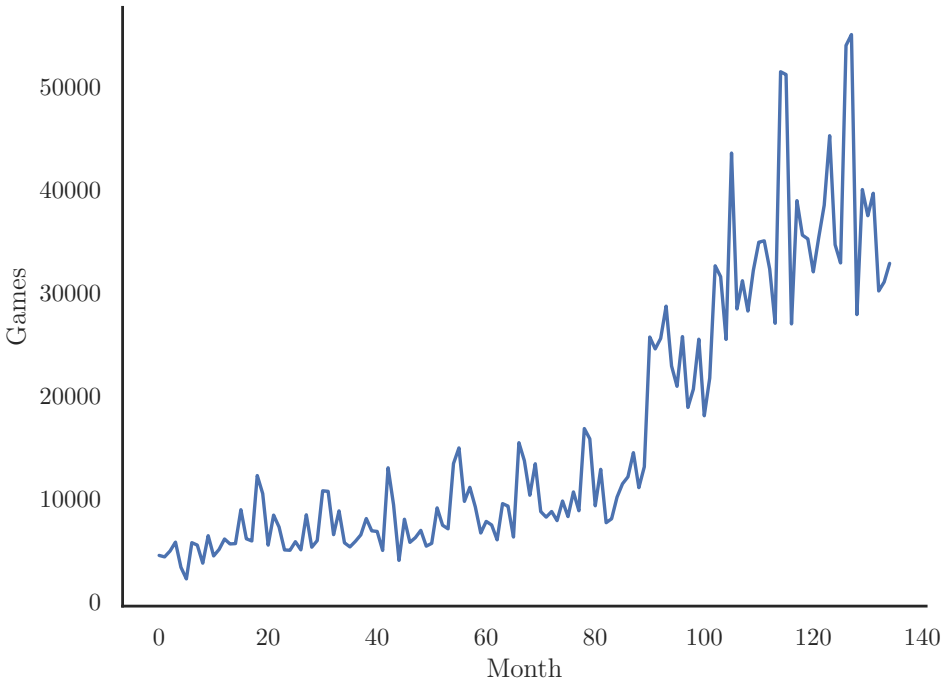

Figure 2: The number of games per month for the Kaggle data.

Synthetic Data Set 1

This data set was designed to be extremely rich with information. The data consisted of games between 100 players across 100 time periods. For each player, we sampled from a 100-dimensional Gaussian distribution with mean zero and covariance matrix K given by

$$K_{ij} = \tau^2 \exp\left(-\frac{(i-j)^2}{2\lambda^2}\right) \tag{28}$$

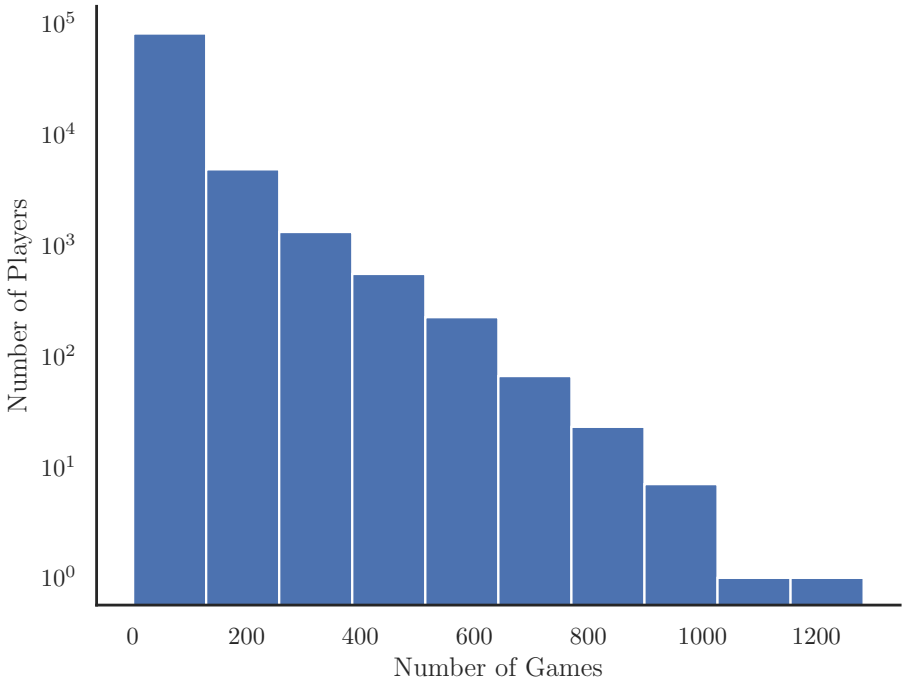

Figure 3: A histogram of the distribution of games per player.

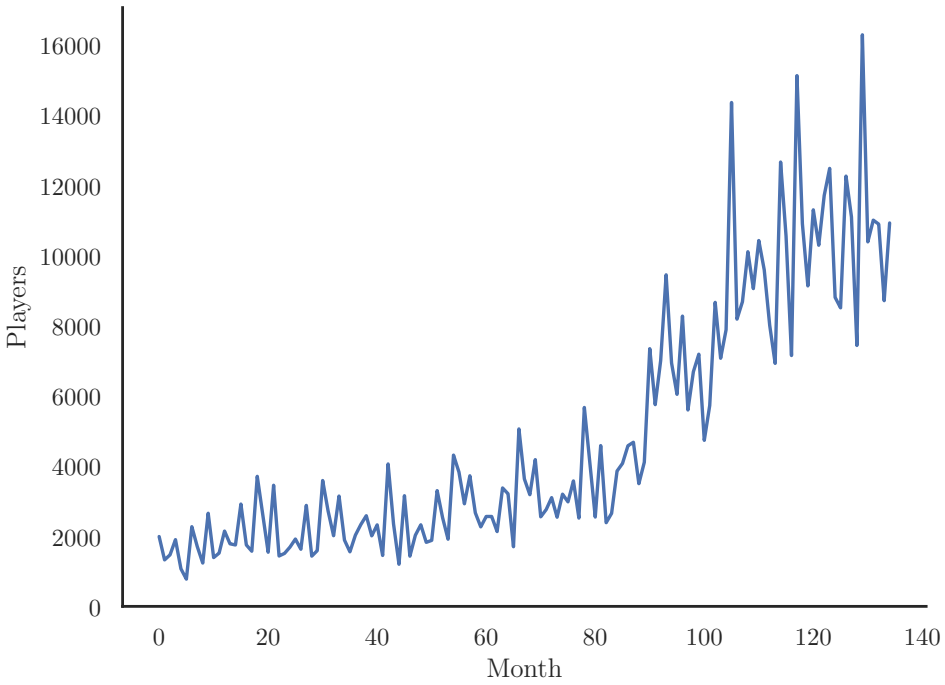

Figure 4: Number of players entering the player pool each month.

with $\tau = 1.5$ and $\lambda = 20$. This produced a discrete curve which gives the impression that skill is varying smoothly with respect to time. See the figure below. The parameter τ controls the amplitude of the resulting curves. The parameter λ controls how quickly the curves tend to vary over time, with larger values of λ corresponding to smoother curves.

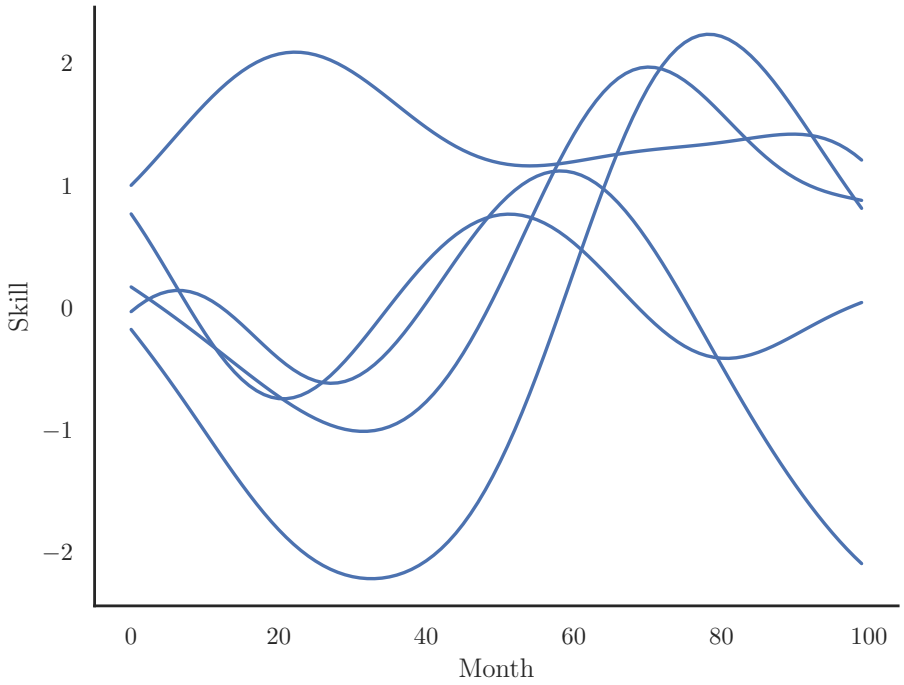

Figure 5: Example skill curves from Synthetic Data Set 1.

Given these skill curves, we can simulate the outcomes of games between pairs of players according to the probabilistic model that underlies both Elo and Elo regression. A white player i is chosen uniformly at random, a black player $j \neq i$ is chosen uniformly at random, and a time period t is chosen uniformly at random. Then we calculate a threshold

$$h = \sigma(\theta_i(t) - \theta_j(t)). \tag{29}$$

where $\theta_i(t)$ is the skill of player i at time t. At this point we can draw a uniform random variable $X \sim U(0, 1)$ and say that i wins if $X < h$, otherwise j wins. But as

we discussed earlier, this procedure will never lead to draws, and thus will produce data uncharacteristic of chess. The appropriate way to simulate data from the model that underlies both Elo and Elo regression is to simulate two pseudo-games: draw two independent variables $X_1, X_2 \sim U(0, 1)$, and if both are less than h then we say white wins. If both are greater than h, then black wins. Otherwise it is a draw. By varying τ, the overall amplitude of the skill curves, we produced data with approximately a 30% draw rate, just like the real data.

We produced one million games in the above fashion, and used 800,000 of them as a training set, and the remaining 200,000 as a validation set. There was no need for a separate test set, since we have access to the complete data generation mechanism.

With 800,000 games across 100 times periods among 100 players, we have 80 games per player per time period for the models to learn from. This is a huge amount of temporal information compared to the Kaggle data set. We do not bother to produce any exploratory graphs like we did for the Kaggle data since we already know exactly how this data was generated.

Synthetic Data Set 2

This data set was designed specifically to display the weakness of a purely prospective rating system. This data was simulated in much the same way as the first synthetic data set, but with a few important differences. First, there were 200 time periods instead of just 100. Second, the 100 players were split into two groups. The second group was made stronger overall than the first group by adjusting the means of the Gaussian distributions from which the skill curves were sampled. The second group had a constant mean of 1.5, while the first group had a constant mean of -1.5. Then, for the first 100 time periods, players only played within their own group. For the second 100 time periods, any two players could play each other. We set $\tau = 1.25$, to make the draw rate around 30% again. Finally, we set $\lambda = 70$, so that

the skills were varying slowly enough that it would make sense to infer that group 2 was stronger than group 1 in the first 100 time periods based on the results of the last 100 time periods. Below we present a figure showing some example skill curves and displaying the separation between groups.

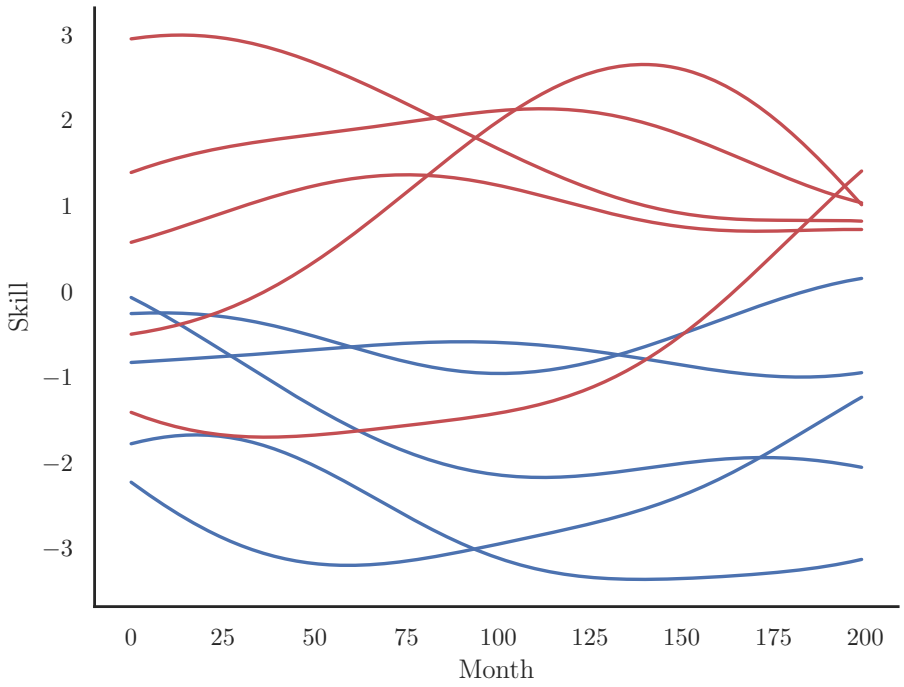

Figure 6: Example skill curves from Synthetic Data Set 2. Blue curves are from group 1, while red curves are from group 2.

Thus, for the first 100 time periods, it is impossible to compare the two pools of players, since they have never interacted. A rating system must assign ratings to all players, however, and therefore any rating system is forced to compare the two groups. For the second 100 time periods, comparison becomes possible. If a rating system like Elo only uses past information at each time period, then for the first 100 time periods its ratings cannot reflect that one group is likely better than the other. Meanwhile, a rating system that is allowed to use information from the future can deduce meaningful ratings even for the first 100 time periods. Though this situation

may seem contrived, less exaggerated versions of this situation do happen in real life. For example, if players from two different countries rarely meet in tournament then the ratings for players from the different countries can gradually become "out of sync" without enough contact. A similar form of isolation can happen when players play within strict age divisions.

Once again, we do not bother to produce any exploratory graphs like we did for the Kaggle data since we already know exactly how this data was generated.

CHAPTER V

METHODS

Models Tested

We compared standard Elo, Elo regression, and a constant model. Elo and Elo regression have already been described in detail above. The constant model assumes that skill is constant over time. It is equivalent to performing a modified Elo regression where the basis consists solely of an intercept. It is also equivalent to Elo regression as $L \to \infty$. Finally, the constant model is also approximately equivalent to a modified version of Elo where each game is used to update the ratings repeatedly until the ratings stop changing. We included this model because it acts as an interesting control to test whether there is anything to be gained by trying to model the change over time at all.

Measuring Performance

Our ultimate goal is to perform accurate inference of the player skills over the time period during which we have data. The problem is that for real data we can never directly observe the player skills, only the outcomes of games. Therefore, we will mainly evaluate the models based on their prediction accuracy on held-out data, as a proxy for accurately inferring the skills.

We would like to know how different players would compare to each other in the past, even if they never played each other in our data. In the case of real data, using

prediction on held out games cannot really estimate this, since the held out data only includes games that really happened. In the case of simulated data, however, we can investigate this. This is what the second synthetic data set exists to test.

The main metric we use for prediction accuracy is the binomial deviance, which is a sum of terms of the form

$$-\left(S \ln E + (1 - S) \ln(1 - E)\right) \tag{30}$$

one for each game, where S is the actual score of the white player and E is the predicted score of the white player. This was the metric used in the Kaggle competition from which the real data set came, and it was used based on the advice of Mark Glickman, noted chess statistics expert [3]. With a little work, we can see that the binomial deviance is proportional to the negative log likelihood of all the logistic-based models we use in this paper. Assuming the white player is player i and he or she wins a game over player j, the log likelihood gains a term of the form

$$\ln \sigma(\theta_i - \theta_j) \tag{31}$$

where $\sigma(\theta_i - \theta_j)$ is E, the expected score of the white player. The actual score $S = 1$ by assumption, so the second term of the deviance vanishes, and we can see that the term from the log likelihood is the opposite of the term from the deviance. The case of the white player losing follows similarly.

When player i draws with player j, we average a win and a loss to get a term of the form

$$\frac{1}{2} \ln \sigma(\theta_i - \theta_j) + \frac{1}{2} \ln \sigma(\theta_j - \theta_i). \tag{32}$$

Again, $\sigma(\theta_i - \theta_j) = E$, and $\sigma(\theta_j - \theta_i) = 1 - E$, and $S = 1 - S = 1/2$, so again we have the negative binomial deviance. Thus, minimizing the binomial deviance is equivalent to maximizing the log likelihood.

The defining feature of the binomial deviance is that it harshly penalizes a confident, incorrect prediction. The penalty for a perfectly sure, but incorrect, prediction (predicting a score of 0 or 1 while the true value is the opposite) is infinite. On the other hand, perfectly predicting a win or loss contributes zero to the binomial deviance, since $\ln 1 = 0$. Predicting all ties always results in a binomial deviance of $\ln 2 \approx 0.693$ because

$$-\left(S \ln \frac{1}{2} + (1 - S) \ln \left(1 - \frac{1}{2}\right)\right) = -(S + (1 - S)) \ln \frac{1}{2} = \ln 2. \tag{33}$$

Unlike a win or loss, a perfectly predicted tie does contribute to the binomial deviance, so the lowest possible deviance depends on the number of ties in the test data. Top-level chess games end in a draw approximately 30% of the time, so the minimum possible binomial deviance is around $0.3 \ln 2 \approx 0.208$. This is of course practically impossible, since it requires perfectly predicting every single game.

We also provide a somewhat cruder, but more easily interpreted, measure of predictive accuracy in the form of correctly predicting the outcome of the game using the "pseudo-game" approach to handling ties. This is cruder because whether a win was predicted with 100% probability or 60% probability, the effect on the accuracy is the same: either the prediction was correct or not.

Another approach to three-way classification is to use a draw threshold. When the expected score is within some threshold of 50%, then we call the game a draw, the intuition being that the players' skills are too close for one of them to beat the other. Otherwise there is enough of a strength difference, and it is a win or a loss depending on whether the expected score is high or low. For example, if we use a threshold

of 5%, we would predict a draw if the probability of the white player winning is in between 45% and 55%. If the probability of the white player winning is outside of this range, then the skill gap is high enough to predict a win or loss.

It turns out that setting a threshold of 1/6 is exactly the same as the pseudo-game approach described above. This is because given an expected score of p, which is the same thing as the probability of winning a pseudo-game, the probability of winning is p^2 and the probability of drawing is $2p(1-p)$. Equating these two probabilities and solving for p gives $p = 2/3$, which is the cutoff at which p is high enough that a win is more likely than a draw. A similar argument shows the lower cutoff is 1/3. Thus, the region for a draw is $1/2 \pm 1/6$. There is a nice symmetry here where an expected score between 0 and 1/3 is predicted to be a loss, an expected score between 1/3 and 2/3 is predicted to be a draw, and an expected score of 2/3 to 1 is predicted to be a win. Since there is such a natural candidate for the threshold, we will just stick to the pseudo-game approach, and not discuss this threshold method any further.

For both of our metrics, deviance and classification accuracy, we produce not only a point estimate but a 95% confidence interval for the expected value of the metric conditional on a fixed model. To be clear, we are not attempting to measure the uncertainty that results from refitting the model on different random training sets. This is generally a good thing to account for, but we did not have the computational budget to refit the models many times. The confidence intervals merely give us some insight into how much the metrics depend on the particular test set we happened to use.

For the Kaggle data, we reported the metric on the overall test set as the point estimate, and we produced a basic bootstrap confidence interval. For the synthetic data, we had access to the data generation mechanism so we just generated many large independent test sets. Then we reported the mean of the metric across the test sets as the point estimate, and used a quantile-based confidence interval. In

particular, for the second synthetic data set, the test set generator included games between players from different groups, even in the first 100 time periods. In each case, the test set was the same size as the validation set, and it was large enough to give very precise estimates of the prediction metrics.

Fitting the Models

All three models were implemented in the Python programming language using the Pandas and NumPy libraries for numerical computing.

For all of the models we used a scale of $c = 1$ since it was convenient to do so and made the ratings comparable. This means that the ratings are not on the usual Elo scale. Also for each model we initialized all the parameters at zero. This has the effect of forcing zero to be the average rating.

For all three data sets, multiple games with the same player happened in the same time period. For example, in the Kaggle data, games were binned by month. Our earlier descriptions of Elo claimed a definite order to all the games, and that one update would be made after each game. Clearly this is not possible with time-binned data. The standard practice is to use a player's current ratings to calculate the updates for every game in the month, then apply all the updates at once, at the end of the month.

For Elo we tried various complicated initialization procedures involving provisional ratings and adaptive K-factors, and we found that nothing worked very much better than a well-tuned fixed global K-factor. Since more complicated procedures were harder to implement and optimize, we just used a fixed global K-factor that we selected based on performance on the validation set. While this may weaken our comparison to Elo regression, it is worth noting that we also could not optimize Elo regression as aggressively as we might have liked to. Moreover, Elo as it is usually applied is not necessarily tuned to the data at all.

For Elo regression, we tried various settings of the hyperparameters mostly at random and kept the values that performed best on the validation set. We did not have the computing budget to systematically search out optimal hyperparameters, since each training run took many hours. For both Elo regression and the constant model we used early stopping. That is, we stopped optimizing the log likelihood when it stopped increasing, not on the training data, but on the validation data. This acts as a form of regularization and also saves computing time [16].

CHAPTER VI

RESULTS

The main results appear in two tables below, one for binomial deviance and one for classification accuracy. The tables show a point estimate, as well as the aforementioned 95% confidence intervals, side-by-side. Elo regression outperformed Elo on the Kaggle data and the second synthetic data set in both metrics. The two models were virtually tied on the first synthetic data set. The constant model also outperformed Elo on the real data set and the second synthetic data set, in both metrics. We can also see that the confidence intervals are extremely tight, as we might expect given the large test sets. For reference, the true skill curves used to generate the first synthetic data set achieve a deviance of around 0.45 and an accuracy of around 0.67. For the second synthetic data set, the true skill curves achieve a deviance of around 0.37 and an accuracy of around 0.74.

Deviance

	Kaggle	Synthetic Data 1	Synthetic Data 2
Constant	0.618 (0.617, 0.619)	0.592 (0.590, 0.593)	0.449 (0.447, 0.450)
Elo	0.632 (0.631, 0.632)	0.455 (0.453, 0.457)	0.490 (0.488, 0.492)
Elo Regression	0.611 (0.610, 0.612)	0.454 (0.452, 0.456)	0.394 (0.392, 0.395)

Table 1: Results of each model on each data set in terms of binomial deviance.

Below we report on the final settings for the Elo and Elo regression models for each of the three data sets. There is nothing to report for the constant model, since it has no hyperparameters.

Accuracy			
	Kaggle	Synthetic Data 1	Synthetic Data 2
Constant	0.475 (0.474, 0.477)	0.505 (0.503, 0.508)	0.663 (0.662, 0.665)
Elo	0.432 (0.431, 0.434)	0.670 (0.667, 0.671)	0.639 (0.637, 0.640)
Elo Regression	0.487 (0.485, 0.488)	0.671 (0.669, 0.673)	0.716 (0.714, 0.718)

Table 2: Results of each model on each data set in terms of 3-way classification accuracy.

The Kaggle Chess Data

The final Elo model used a K-factor of 40. The final Elo regression model used 54 basis functions, spaced every 4 months, with 10 basis functions on each end centered beyond the range of the data, in order to ensure that the end ranges of the data had the same treatment as the middle of the data. With 87,987 players in this data set, the model had a total of 4,751,298 parameters. The length scale was set to 25 and the L_2 penalty was $\lambda = 10^{-5}$.

Synthetic Data Set 1

Here the final Elo model used a K-factor 5. The final Elo regression model used the same basis functions centers. The length scale was set to 5. The L_2 penalty was again $\lambda = 10^{-5}$.

Synthetic Data Set 2

The final Elo model used a K-factor of 7. The final Elo regression model again spaced the basis functions over every 4 months with 10 extra basis functions on each end of the data, for a total of 70 basis functions. The length scale was set to 80, and the L_2 penalty was again $\lambda = 10^{-5}$.

CHAPTER VII

DISCUSSION

General Comments

Elo regression performed at least as well as Elo on each of the three data sets. Looking at the classification accuracy, since it is more easily interpreted, we see that Elo Regression achieved around 5 more percentage points of accuracy on the Kaggle data and around 7 more percentage points of accuracy on the second synthetic data set, which is what we expected given the resources at Elo regression's disposal. We also see that the two methods basically tied on the first synthetic data set, which again makes sense, because there was so much information to learn from, even a cruder method like Elo could produce excellent results. Both models essentially achieved the deviance and accuracy of the true underlying skill curves. Elo regression also came close to the best possible deviance and accuracy on the second synthetic data set.

It is of interest that the constant model also outperformed Elo on the Kaggle data and the second synthetic data set. For the synthetic data, this is probably because we set the skills to be changing slowly over time, and we made the data hard to learn without using information from the future to predict the past, which the constant model implicitly does by using the same rating for all months. It is perhaps more surprising that on the real chess data, a constant model performs better than Elo. As we noted earlier, the Kaggle data had very few games per player per month, and thus little information about how skills changed over time. Even the Elo regression model

did not perform that much better than the constant model. Thus, perhaps most of the temporal patterns Elo displayed in the Kaggle data were just noise.

Now we show some of the estimated ratings from the three models on each of the data sets.

The Kaggle Chess Data

Below we show the rating curves produced by each the of the three models for four players. The top two graphs show situations where there is very little data. Elo produces extreme, sharply changing estimates, while Elo regression smooths out the change to occur more gradually. On the top right graph, the player lost several games, lowering his or her rating, and then stopped playing games altogether. When there are no more games to affect the rating, Elo maintains the player's last rating, while Elo regression shrinks the player's rating back to the mean over time. We think this behavior is generally preferable. On the bottom two graphs, the players had more games to learn from, and we see that Elo and Elo regression generally trace out the same curve, though the Elo regression ratings are much less noisy. We also see on the bottom left graph that the Elo regression rating is generally higher than the Elo rating. We believe the underlying skill is probably closer to the Elo regression rating, but the Elo rating was constrained in its updating procedure and could not push the rating for this player high enough without using a K-factor that compromised its performance on other players.

Perhaps the most interesting finding of this whole work is that the constant model had better predictive accuracy than Elo on this data set. This speaks to how noisy the real data were and how in need of regularization the models were. The constant model probably did well because the true skill levels of players were not changing very quickly, and many players only had games within a small time window anyway.

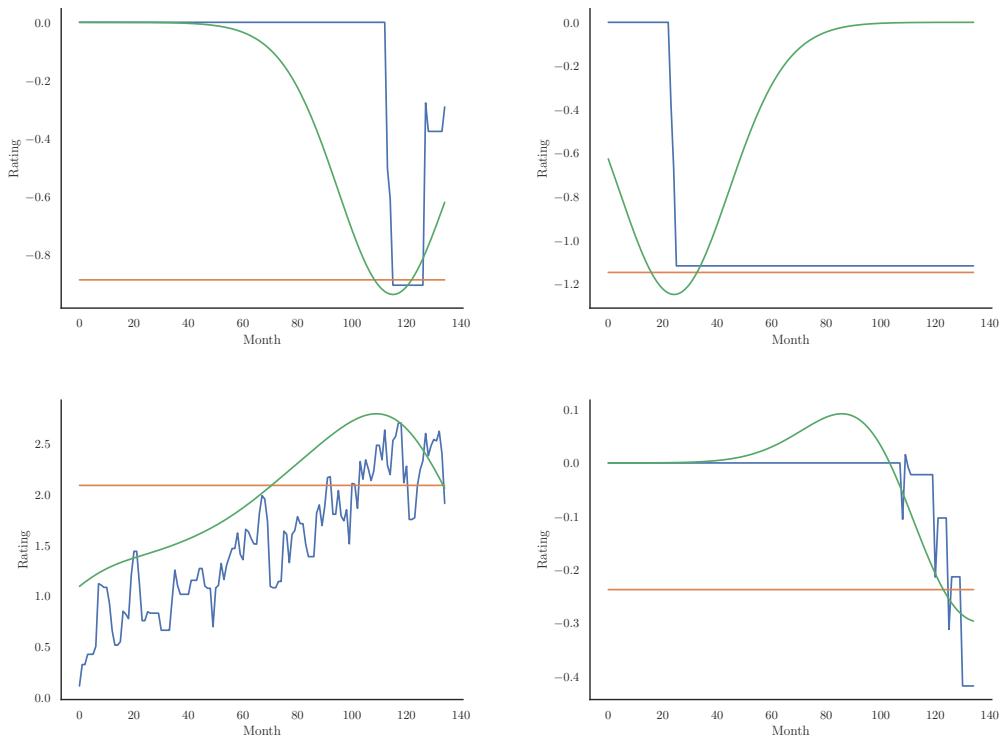

Figure 7: Some example rating curves for the Kaggle data. Blue is Elo, green is Elo regression, and yellow is constant.

Synthetic Data Set 1

With this data set, there were so many games to learn from that Elo and Elo regression performed extremely similarly. We can see below that the curves are almost identical, except that Elo is noisier of course. Also, since the skills were changing relatively quickly and players had games evenly dispersed throughout the months, the constant model did not fare so well.

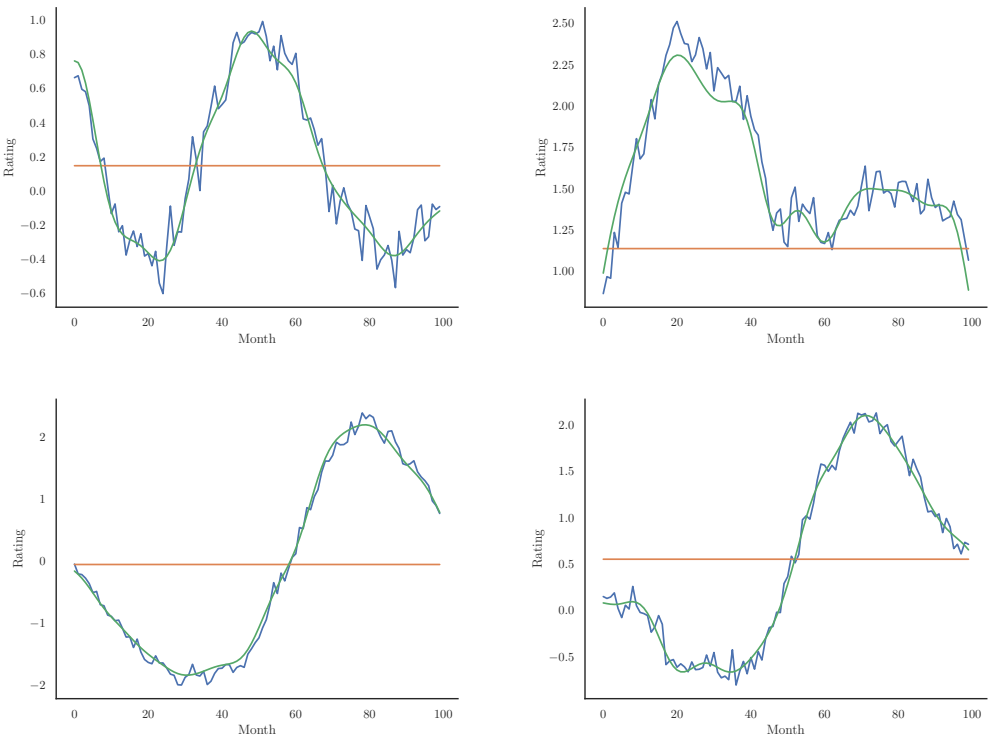

Figure 8: Some example rating curves for the first synthetic data set. Blue is Elo, green is Elo regression, and yellow is constant.

Synthetic Data Set 2

As expected, Elo regression strongly outperformed Elo on this data set. Below, the top two graphs plot the ratings of two players from the weaker group, and the bottom two graphs plot the ratings of two players from the stronger group. We can see that Elo suddenly realizes in month 100, when the two groups of players first compete

with one another, that it had vastly overrated the weaker players and underrated the stronger players. Since Elo regression is privy to all the data in forming its ratings for each month, it does not exhibit this pathological behavior. It is worth noting that a human observer of the Elo ratings, even without any knowledge of the true data generating process, can surmise that the ratings right before month 100 are probably not correct. The sharp change in rating right around month 100 violates our intuition that players change in skill only gradually. The smoothing parameter L in Elo regression models this intuition.

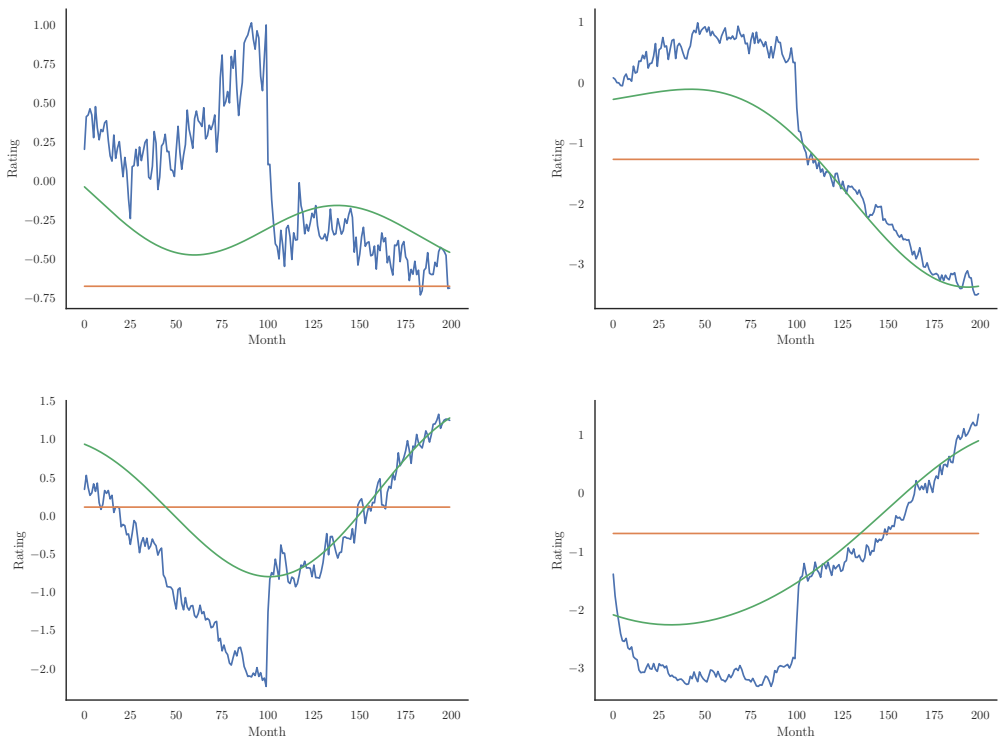

Figure 9: Some example rating curves for the second synthetic data set. Blue is Elo, green is Elo regression, and yellow is constant.

CHAPTER VIII

CONCLUSION

Despite Elo regression's superior predictive performance, Elo still has the strong advantage of scaling remarkably well to large data sets. On the Kaggle data, Elo regression had over 4.5 million parameters. It took many hours to train the model, though we were only using a single laptop CPU. With more powerful computers, especially those with GPUs, we could achieve a drastic speed up. Still, Elo regression would never be as fast as Elo. Thus, in information-rich scenarios, like the first synthetic data set, it makes much more sense to use Elo. However, the Kaggle data showed that real chess data tends to be less than ideal. In that case, when performing historical or retrospective ratings, it could make sense to put the time in to train an Elo regression model. In addition to superior predictive performance, Elo regression is capable of modeling different assumptions about how quickly skills can change over time, which in turn informs how much ratings should be adjusted retroactively. We find this independently interesting.

For use as an actual general-purpose rating system, Elo regression is far from ideal, as we have previously stated. It takes too long to train and it produces ratings that are likely to frustrate players, since a player's rating can change in a way that feels outside of his or her control.

Also, as we previously mentioned, we did not push either model to the limit, in terms of optimizing hyperparameters on the validation data. For Elo, after some

initial unsuccessful experimentation, we stuck to a single fixed K-factor. For Elo regression we did not have the computing power to systematically try out different combinations of basis function number and position, length scale, and L_2 regularization amount. In the future, with better computing resources, we could perform more thorough experimentation.

Further extensions of the Elo regression idea are possible. We could easily add more covariates such as which color pieces the player is using. It is well-known that the white player has an advantage, since they always have the first move. Modeling this advantage would further improve ratings. We could use a different basis, or even different bases for different players. We could use a different distribution, instead of the logistic distribution. We could change the estimation procedure to use Bayesian ideas, so that we could more easily estimate the uncertainties in our ratings. The possibilities are literally endless, since the fundamental idea behind Elo regression is that once you understand an algorithm as a fitting procedure for a probabilistic model, then you can freely change the different parts of the algorithm in a sensible way.

REFERENCES

[1] Strongest chess players through time. https://www.cleanchess.com/great-grand-masters/strongest-players-through-time, . Accessed: 2019-03-18.

[2] The history of the top chess players over time. https://www.youtube.com/watch?v=z2DHpW79w0Y, . Accessed: 2019-03-18.

[3] Binomial deviance. https://www.kaggle.com/c/ChessRatings2/discussion/285. Accessed: 2019-03-18.

[4] Fide rating regulations effective from 1 july 2014. https://www.fide.com/fide/handbook.html?id=172&view=article, . Accessed: 2019-03-18.

[5] Fide title regulations effective from 1 july 2017. https://www.fide.com/component/handbook/?id=198&view=article, . Accessed: 2019-03-18.

[6] Deloitte/fide chess rating challenge. https://www.kaggle.com/c/ChessRatings2. Accessed: 2019-03-18.

[7] World football elo ratings. www.eloratings.net. Accessed: 2019-03-18.

[8] Pierre Dangauthier, Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill through time: Revisiting the history of chess. In *Advances in neural information processing systems*, pages 337–344, 2008.

[9] Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[10] Mark E Glickman. The glicko system. *Boston University*, 1995.

[11] Mark E Glickman. The glicko-2 system for rating players in head-to-head competition, jul. 2000. *Retrieved from the Internet on Oct*, 23, 2003.

[12] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576, 2007.

[13] Krzysztof C Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

[14] Andrew Y Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[15] Nate Silver and Reuben Fischer-Baum. How we calculate nba elo ratings. https://fivethirtyeight.com/features/how-we-calculate-nba-elo-ratings/. Accessed: 2019-03-18.

[16] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.