Imperial College London

Paired Comparison Models on Tennis Matches

A Math Year-2 Research Project
Department of Mathematics

Written by

Young, James M.
mmy20@ic.ac.uk

Fistouris, Christoforos G.
cgf20@ic.ac.uk

Yuan, Christopher
cy520@ic.ac.uk

Dialdestoro, Justin
jvd120@ic.ac.uk

Yu, Chenghao
cy1420@ic.ac.uk

Yagiz, Algan
ay1120@ic.ac.uk

Supervised by
Professor Axel Gandy

Abstract

Paired comparison experiments are found in numerous scenarios in practice. That is, a ranking is often desired pertaining to the results of tabulating preferences between two objects. We focus on tennis matches as it is most notable for utilising this format and being rich in player data. Existing modelling attempts such as the Bradley-Terry (BT) model and both Elo and Glicko rating schemes are introduced with their underlying motivations. Further applications such as evaluating the effects of momentum and appropriate parameter estimates are also explored. In addition, we propose a Markov chain based model, one we refer to as the Fickle Fan (FF) model, and provide relevant background on its processes. A natural Bayesian extension to the BT model is also examined by placing priors on players' strengths and using the MAP estimate obtained from running the Metropolis-Hastings algorithm to predict match outcomes. Subsequently, these models are applied to the 2017 ATP men's tennis data set. Their accuracy is evaluated through rank correlation and likelihood heuristics such as multi-fold cross-validation on the ATP points ranking at the end of 2017. Across the metrics, the BT and FF models outperform their counterparts by small margins. Elo, Glicko, and its derivative Glicko2 scored nearly even. We cite our choice of rating periods to have been substantial to yielding this result. As is the case with most of the models, the novelty FF model is found to have a significant rank correlation. Suggested refinements to it include fine-tuning the proportionality constant and using a different weighting system for transition probabilities. The Bayesian extensions of the BT model have best-performing MAP estimates coming from the gamma prior, suggesting it may be more in line, compared to normal and uniform priors, to the true skill distribution within the chosen tennis data set.

Keywords. paired comparison, Bradley-Terry, Elo, Glicko, Markov chains, Bayesian prior, cross-validation, tennis, momentum.

Contents

Chapter 1

Introduction

In a vast number of disciplines, a ranking is often desired from an available list of subjective comparisons between two objects. Formally, a paired comparison experiment involves presenting stimuli to independent judges, each of which declare their preference between any two of them (David, 1963). From there, a ranking could be constructed. In fact, this process exhibits a more general behaviour, as the margin of preference is kept (Kendall and Smith, 1940) and non-transitivity between stimuli is considered (Kendall, 1955). Typically, these ranking models have been used to decide between subjective, if not, borderline equal stimuli to some extent. Relevant uses include decision making in multi-agent systems (Ito and Shintani, 1997) and psychophysical measurement in social values (Thurstone, 1927). A more popular practice, however, is that of sports analytics.

We shift our focus to the natural inclination of ranking players in sporting disciplines. Tennis, in particular, presents a rich set of data with heterogeneity in tournament formats, surface types, and margins of victory in results. Players are also a form of multi-dimensional stimuli. That is, they have certain traits both physically and those intangible which may affect their match results. Here, paired-comparison is framed as preferring players who have won their matches, and, in this case, the goal is to construct a ranking based on existing match data. As with the subjective nature of ranking skill, models can further take into account pieces of co-variate data to develop a more accurate evaluation in tune with the noticeable trends in practice. We consider tennis data sets in the succeeding chapters for this purpose.

Extensive work has been made in this particular area of research within the past century. With this in mind, we define our objectives of the paper to be as follows: (1) understand and summarize the current state of the field, (2) remark on the strengths and history of its developments, and (3) apply working theory to appropriate real data in hopes of comparing the suitability of our models on the chosen data set. In doing this, throughout the paper, we will also provide noteworthy insights as we see fit along the way in an attempt to motivate the paper and contribute to the field.

The first main focus of the paper will be the **Bradley-Terry** (BT) model, whose inception was in many ways, as we will come to see, integral to establishing the field. Prior to this, we will first formalize the concept of *stochastic transitivity* which will be a fundamental premise in our analysis to follow. Additionally, at this time, we will give an account of the **Thurstonian** model that, as one of our first major results, we will relate to Bradley-Terry via another established framework in the field, Luce's **Choice Axiom**. We end this section with one of the highlights of the paper, using the BT model to statistically examine the effect of *momentum* in professional tennis.

Following this we move our attention to the **Elo** and **Glicko** models, which arise as a result of modelling the *expected wins* of a player in a competitive setting. This differs fundamentally from models discussed prior to this point and makes these models especially powerful tools for the application at hand – modelling professionally tennis

players. Additionally, at this point we propose our own model for rating tennis players, to which we bestow the name the **Fickle Fan** model.

In Chapter 3, we briefly discuss extending ranking models with Bayesian inference, by fitting ratings with a presumed (*prior*) distribution for skill. This is different from the distribution of an individual player's strength from game-to-game that is a fundamental assumption of all competitive game ratings (this is the assumption that allows a player to be beaten by an opponent with lower rating), and instead refers to the distribution of skill across the whole player population. These distributions are implemented to enforce beliefs we may have about the player-base from prior knowledge or intuition. We will see how these priors affect rating by exploring our standard professional tennis data set under a modified BT model with three different prior distributions. To do this, we will employ a numerical means of sampling from otherwise complicated posterior distributions, the **Metropolis-Hastings** algorithm.

As the models have been discussed extensively, we then proceed to evaluate their fits in the 2017 ATP men's professional tennis data set through three different criteria: **Spearman's rank correlation coefficient**, the **Brier score**, and **multi-fold cross-validation**. Chapter 4 introduces the key ideas and areas each test aims to measure as well as their pitfalls and how we overcome them. Specifically, the similarity of ranks produced to the actual ranks at the end of the data set's time period is measured, a test for correlation in a small sample size is performed, and the likelihood of match outcomes under each fit is compared. The set-up for each model is briefly mentioned including how the initial parameters are either arbitrarily set or fine-tuned through maximising the likelihood of observations. We then attempt to explain the results in connection with the theory behind each model and the qualities pertaining to the data set used.

To conclude the introduction, we highlight our key findings and developments from throughout the paper to summarize the areas in which we offer extension (bonus) material. In chronological order, the first of these sections extends our work on the BT model by outlining a means in which we are able to perform a hypothesis test on how momentum affects professional tennis players, looking at real data. To do this, we first carefully develop theory on the asymptotic behaviour of multivariate parameters in our statistical model to be able to construct confidence intervals for our test statistics, before writing custom code to partition our match data and compute a closed form for our log-likelihood function and multidimensional Fisher Information matrix. At the end of chapter 2, we take inspiration from existing models to develop our own system for modelling player rankings, based on Markov chains, accounting for the drawbacks and limitations of our proposal to ultimately achieve a functional system. A Bayesian extension to the Bradley-Terry model is explored in chapter 3, with different choices for prior distributions set on the strengths of players. We outline the background and technical aspects of how we aim to retrieve the desired MAP estimate and related setbacks. To round off our paper, in chapter 4 we study different means of comparing our rating systems, drawing methods from an array of different fields and papers. In addition to performing computational simulations to obtain results, this section includes a comprehensive discussion on the performance of the various models on our data set, offering a natural conclusion to the paper.

Chapter 2

Discussion on Models

In this chapter, we provide a brief introduction to each of the model's theory and derive reasonable conclusions from the ATP men's professional tennis data set. Naturally, the models' strengths and pitfalls lie within the boundaries of its assumptions. For example, a paired-comparison is deemed balanced if all judges are able to cast votes between any two stimuli (David, 1963). In sports, this is akin to a round-robin tournament. However, this not always the case as with single or double elimination formats. In their respective sections, we describe ways in which the models attempt to overcome these problems and other practical considerations. We begin from the ideas of Bradley and Terry.

2.1 Stochastic Transitivity & the Bradley-Terry Model

We begin our discussion into the field of pair-comparison with a look at the history of the subject and the early work forming the basis for one of the main results we will investigate in this paper, the *Bradley-Terry* model (BT model). This section will comprise an account of a few models predating the BT model which although devised separately will provide us with a natural way of establishing this section's primary model. We will establish these models as being characterized by modelling the probability of a player, player i, beating an opponent, player j. We write this as $\mathcal{P}(i \succ j) = p_{ij}$.

In addition to our work on the BT model, we discuss *stochastic transitivity* at the start of the section. This discussion will motivate the work to follow by outlining a theoretical framework that will guide our study into these types of models.

Lastly, to develop the BT model, we investigate the construction of confidence intervals and hypothesis testing using the asymptotic normality properties of our maximum likelihood estimator (MLE) for our model. This will allow us to conclude the section by applying our theory to real data on historical matches between professional tennis players, in order to statistically test the existence of *momentum* in the sport.

2.1.1 Stochastic Transitivity

Before we begin exploring models, we introduce the concept of **Stochastic Transitivity** which will be useful in our work later. Picturing a setting with three players i, j, and k, (in general, we will write J as the set of all players) we consider the binary relation $\succ$ to mean "beats," and therefore we may write "player i *beats* player j" as $i \succ j$. Assuming transitivity, we have

$$i \succ j \text{ and } j \succ k \implies i \succ k$$

However in the case of players in a tournament, it may not necessarily be the case that player i is guaranteed to beat player k given $i \succ j$, and $j \succ k$. Hence we model this with

stochastic transitivity, writing

$$\mathcal{P}(i \succ j) = p_{ij} \text{ and } \mathcal{P}(j \succ k) = p_{jk} \implies \mathcal{P}(i \succ k) = X$$

for X dependent on p_{ij} and p_{jk}. Specifically, we will be interested in stochastic transitivity by *composition*, which states $X = \mathcal{F}(p_{ij}, p_{jk})$, for a symmetric strictly monotone-increasing (in the first element) function $\mathcal{F} : [0,1] \times [0,1] \to [0,1]$. In such a case, we look to identify a pair (F, μ) for a *comparison function*, F, and corresponding scoring function μ (defined below) such that

$$\mathcal{P}(i \succ k) = \mathcal{F}(p_{ij}, p_{jk}) = F(\mu_i - \mu_k)$$

as outlined by Oliveira (2018). Although at first glance this may seem complicated, it in fact yields the following intuitive interpretation:

Given a function $\mu : J \to \mathbf{R}$, $\mu(i) = \mu_i$ can be thought of as the "score" of player i, for $i \in J$. With such a scoring function we define a comparison function to be any strictly increasing continuous function, $F : \mathbf{R} \to [0,1]$, such that $F(d) + F(-d) = 1$, and $p_{ij} = F(\mu_i - \mu_j)$.

Example: Take for example a particular case of a game of rock-paper-scissors between three players. Assuming, each player throws a distinct hand ($J = \{R, P, S\}$) – i.e. player R will only throw rock – we assign the comparison/scoring function pair (F, μ) as

$$\mu(i) = \begin{cases} 0 & i = R \\ 1 & i = P \\ 2 & i = S \end{cases}, \quad F(d) = \begin{cases} 1 & d \in \{-2, 1\} \\ 0 & \text{else} \end{cases}$$

This example also showcases the difference between stochastic and standard transitivity, as $p_{RS} = F(-2) = 1$ (the probability of Rock beating Scissors) and $p_{SP} = F = 1$ (the probability of Scissors beating Paper), however $p_{RP} = 0$.

Note: Although the construction shown in this example is not entirely consistent with the definition provided above as our comparison function, F, is not continuous, this toy example provides insight into the main idea of our pairing (μ, F).

2.1.2 Thurstone's Model & Luce's Choice Axiom

The first model we present predates the field of pair-comparison itself by dating back to 1927 from the field of mathematical psychology, Thurstone's **Law of Comparative Judgement** concerns an individual's subjective preference between two stimuli, so-called *discriminal processes* (Thurstone, 1927). In his original article outlining the principle, he proposes that an individual's perceived "excellence" of a stimulus is modelled by a random variable S_i on a psychological scale given by the form

$$S_i = \mu_i + X_i$$

for a constant μ_i and random variable X_i with mean 0.

In the context of comparing two stimuli, the model deals with two such variables then, namely S_1 and S_2, and we have that stimulus i is preferred to stimulus j when $S_i > S_j$. Considering the quantity $D_{ij} = S_i - S_j$, we have that i is superior to j when $D_{ij} > 0$.

In a strictly psychological setting, Thurstone assumes X_i to take a normal distribution, and hence $S_i \sim N(\mu_i, \sigma_i^2)$. As the difference between two normal distributions, D_{ij} will too follow a normal distribution (as a standard property of the normal distribution)

$$D_{ij} \sim N(\mu_i - \mu_j, \sigma_i^2 + \sigma_j^2)$$

under Thurstone's Law of Comparative Judgement. Hence we conclude the probability of stimulus i winning out over j, $p_{ij} = \mathcal{P}(i \succ j)$, to be

$$\begin{aligned}\mathcal{P}(i \succ j) &= \mathcal{P}(D_{ij} > 0) \\ &= \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}}\right)\end{aligned}$$

To generalize this theory, we question what might happen if we were to select a different distribution for our underlying discriminal process X_i rather than assuming normality. To this end we introduce at this point Luce's **Choice Axiom** (Luce, 1959), writing $P_S(R)$ as the probability of an individual choosing an element belonging to a subset $R \subset S$ of a finite set of alternatives S, the axiom states for $R \subset S \subset T$: $P_T(R) = P_S(R)P_T(S)$ or equivalently

$$P_S(R) = \frac{P_T(R)}{P_T(S)}$$

This is succinctly put by Luce in his reflection, *The Choice Axiom After 20 Years* (Luce, 1977), in the case where R consists of just one element, $R = \{a\}$

$$P_S(a) = \frac{\nu(a)}{\sum_{b \in S} \nu(b)}$$

for $\nu(a)$ (strictly positive) representing the *response strength* of a. Further simplifying to the case where S comprises just two elements a and b, to parallel our work on Thurstone's model with two stimuli,

$$\begin{aligned}\mathcal{P}(a \succ b) = P_S(a) &= \frac{\nu(a)}{\nu(a) + \nu(b)} \\ &= \frac{1}{1 + e^{-(\ln(\nu_a) - \ln(\nu_b))}}\end{aligned}$$

writing $\nu_x = \nu(x)$. Examining this closely, we note that this is nothing more than a stochastic transitivity by composition model (F, μ), with

$$F(d) = \frac{1}{1 + e^{-d}}, \quad \mu(x) = \ln(\nu_x)$$

where $F(d)$ is the logistic function. This result shows that for random variables $(\mu_a + R_a, \mu_b + R_b)$ representing the response strength of elements (a, b) respectively – i.e. $\nu_x \sim (\mu_x + R_x)$ for distribution R_x with mean 0 – the distribution of $D_{ab} = \ln(\mu_a + R_a) - \ln(\mu_b + R_b)$ will have logistic CDF. In their work later cited by Luce & Suppes (Luce and Suppes, 1965), Holman and Marley showed that this will occur when our random variables for response strength have double exponential distribution, i.e.

$$\mathcal{P}(\mu_i + R_i \leq x) = \exp\{-\exp\{\alpha(\mu_i - x) - \beta\}\}$$

Although as noted by Yellot (1977), this is not a necessary condition on the true nature of the distribution of our response strength – that is, the double exponential does not uniquely deduce a logistic form of $(R_a - R_b)$ – for our purposes it will suffice as an illustration of another variation of Thurstone's model.

It is additionally worth noting that the quantitative similarities between the normal and logistic distributions (shown in Figure 2.1) underline the striking similarities between Thurstone's original normal model, and Luce's Choice Axiom.

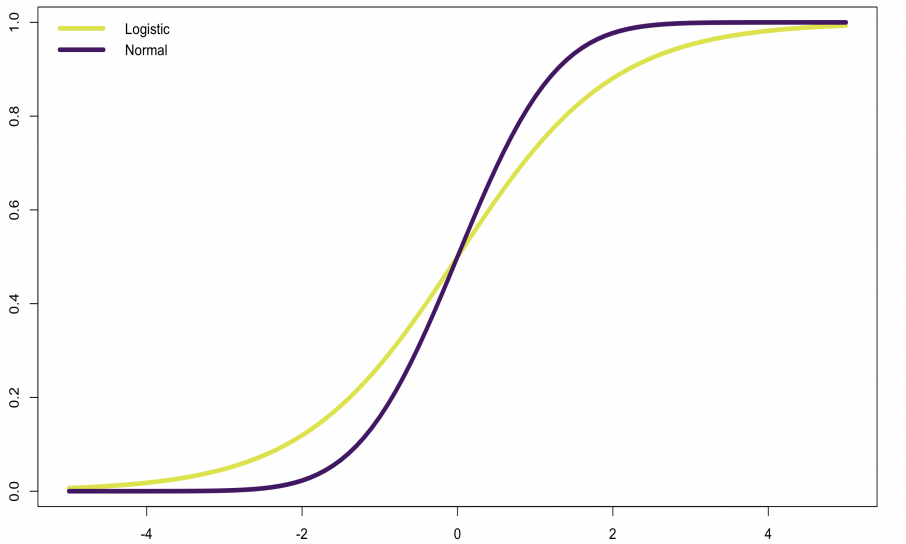

Figure 2.1: Graphical comparison of standard logistic function (yellow) against standard normal CDF (purple)

2.1.3 Bradley-Terry Model

In its original paper (Bradley and Terry, 1952), the Bradley-Terry Model was proposed as a means of determining the probabilities of winners in a pair-comparison taste-tests among a set of roasted pork samples. For our purposes, however, we will instead refer to the more *bland* example of ranking players in a competitive sport, as appropriated from Hunter (2004).

In a setting where n players are repeatedly matched and compete in a drawless game, the BT model gives us the probability of player i beating player j as

$$\mathcal{P}(i \succ j) = \frac{\rho_i}{\rho_i + \rho_j}$$

where $\{\rho_1, ..., \rho_n\}$ represent the *true* skill ratings of players 1 to n. Strikingly, this formulation is identical to that of Luce's Choice Axiom for two elements.

The fact that ratings are arbitrarily defined in this statement means that given a set of real data (i.e. matches played between players), we may in fact use this model to calibrate player ratings, where our estimate for $\rho = \{\rho_1, ..., \rho_n\}$ is given as the maximum likelihood estimate of the model. In this sense, we are working backwards in calculating the ratings that give us the highest probability of obtaining our data given the model. Since, under the model, ρ will yield the same likelihood as $\lambda\rho$ for $\lambda \in \mathbf{R}$, going forward, we define ρ to be normalized such that $\sum_{i=1}^{n} \rho_i = 1$.

To start, we write the log-likelihood function according to our master equation

$$\ell(\boldsymbol{\rho}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(\ln \rho_i - \ln (\rho_i + \rho_j)) \tag{2.1}$$

where w_{ij} denotes the number of times player i has beaten j in our data set. Ford (1957) tells us that existence of a maximizer for our likelihood function requires that for any partition of the players into two sets (S_1, S_2), a player in S_1 to have beaten a player in S_2. Hunter (2004) gives us the graph interpretation of this assumption for a directed graph of n nodes with an edge representing a game between nodes in the direction of the winner as: for all (i, j) there exists a path from node i to j. We will in the future refer to this as **C1** (*Condition 1*).

Given the nature of the process, wherein the more games that are played, the more accurate we expect our vector of skill values to be, it is natural to consider an iterative process in which we successively condition a vector of skill levels, $\boldsymbol{\rho}$, to fit the data. Specifically, starting with an initial guess, $\boldsymbol{\rho}^{(1)}$, we repeatedly feed our vector into an iterative function, $M : \mathbf{R}^n \to \mathbf{R}^n$, such that $\boldsymbol{\rho}^{(k+1)} = M(\boldsymbol{\rho}^{(k)})$ and $\lim_{k\to\infty} \boldsymbol{\rho}^{(k)} = \lim_{k\to\infty} M^k(\boldsymbol{\rho}^{(1)}) = \hat{\boldsymbol{\rho}}$, where $\hat{\boldsymbol{\rho}}$ maximizes $\ell(\boldsymbol{\rho})$.

The MM Algorithm: In finding such an iterative function, we refer to an example in Lange et al. (2000) on the use of surrogate functions to optimize the log-likelihood function. Specifically, by employing so-called *MM functions* (minorizing/maximizing), the group showed that

$$g(\boldsymbol{\rho}|\boldsymbol{\rho}^{(k)}) \leq \ell(\boldsymbol{\rho}) \;\; (\text{equality} \iff \boldsymbol{\rho} = \boldsymbol{\rho}^{(k)}) \tag{2.2}$$

where

$$g(\boldsymbol{\rho}|\boldsymbol{\rho}^{(k)}) = \sum_{i,j} w_{ij} \left(\ln(\rho_i) + \ln\left(\rho_i^{(k)} + \rho_j^{(k)}\right) - \frac{\rho_i + \rho_j}{\rho_i^{(k)} + \rho_j^{(k)}} + 1 \right).$$

Condition (2.2) implies that $g(\boldsymbol{\rho}|\boldsymbol{\rho}^{(k)}) \geq g(\boldsymbol{\rho}^{(k)}|\boldsymbol{\rho}^{(k)}) \implies \ell(\boldsymbol{\rho}) \geq \ell(\boldsymbol{\rho}^{(k)})$, and thus any $\boldsymbol{\rho}$ which maximizes $g(\,\cdot\,|\boldsymbol{\rho}^{(k)})$ will serve to increase the likelihood of our $\boldsymbol{\rho}$ estimate. That is, for $\boldsymbol{\rho}^{(k+1)}$ found to maximize $g(\,\cdot\,|\boldsymbol{\rho}^{(k)})$, the series $(\ell(\boldsymbol{\rho}^{(m)}))_1^\infty$ will be a monotone increasing series.

By separating the components of our parameter vector, $\boldsymbol{\rho}$, $g(\,\cdot\,|\boldsymbol{\rho}^{(k)})$ can be maximized simply by maximizing the function with respect to each of the components independently. This is the advantage of working with $g(\,\cdot\,|\boldsymbol{\rho}^{(k)})$ rather than $\ell(\boldsymbol{\rho})$ directly. Trivially, the maximizer of our surrogate functions becomes

$$\boldsymbol{\rho}_i^{(k+1)} = \frac{\sum_{j\neq i} w_{ij}}{\sum_{j\neq i}(w_{ij} + w_{ji})/(\rho_i^{(k)} + \rho_j^{(k)})} \tag{2.3}$$

where w_{ij} is the number of wins between player i against j, as stated previously (Lange et al., 2000). Promisingly, this result is identical (after reworking) to the form proven by Zermelo (as cited in Hunter (2004))

$$\rho_i^{(k+1)} = W_i \left(\sum_{j\neq i} \frac{N_{ij}}{\rho_i^{(k)} + \rho_j^{(k)}} \right)^{-1}$$

for N_{ij} the number of games between i and j, and W_i the total number of wins for player i – i.e. $N_{ij} = \sum_{j\neq i}(w_{ij} + w_{ji})$, and $W_i = \sum_{j\neq i} w_{ij}$.

Hence, we define the iterating function $M(\boldsymbol{\rho}^{(k)})$ according to (2.3). The properties of this function in the context of an MM algorithm are explored at length in Hunter (2004). Importantly, it is established that under relatively natural regularity conditions, it follows that:

1. The MM algorithm defined by (2.3) will converge in the sense that $\hat{\boldsymbol{\rho}} = \lim_{k\to\infty} \boldsymbol{\rho}^{(k)}$ exists

2. The limit $\hat{\boldsymbol{\rho}}$ will serve to maximize $\ell(\boldsymbol{\rho})$

These properties show the MM algorithm as a valid means of computing our estimated skill levels, $\boldsymbol{\rho}$, for the BT model. Although we have chosen to explore this method specifically in-depth, we note that this is in fact just one of many general purpose optimizers which could be used to compute the MLE of our BT model likelihood function $\ell(\boldsymbol{\rho})$.

2.1.4 Fitting the Tennis Data Set

In this section we apply the Bradley-terry model to real data from the 2017 ATP season (DataHub.io, 2019), to get a sense of how the theory discussed up to this point fares in practice. Computations and maximizations were done in **R**, performed using the `BradleyTerryScalable` package (Kaye et al., 2017). For data sets satisfying **C1**, this package uses an MM algorithm to compute $\hat{\boldsymbol{\rho}}$ as the MLE of 2.1 – as described in the section prior.

The full **R** code can be found in the appendix.

Table 2.1: Top 10 player rankings according to Bradley-Terry model from ATP data.

	ρ	ID	Surname
1	20.81	f324	Federer
2	17.35	n409	Nadal
3	10.43	d643	Djokovic
4	9.39	z355	Zverev
5	8.93	d875	Dimitrov
6	8.62	d683	del Potro
7	7.44	gb88	Goffin
8	7.34	w367	Wawrinka
9	6.91	n552	Nishikori
10	6.70	mc10	Murray

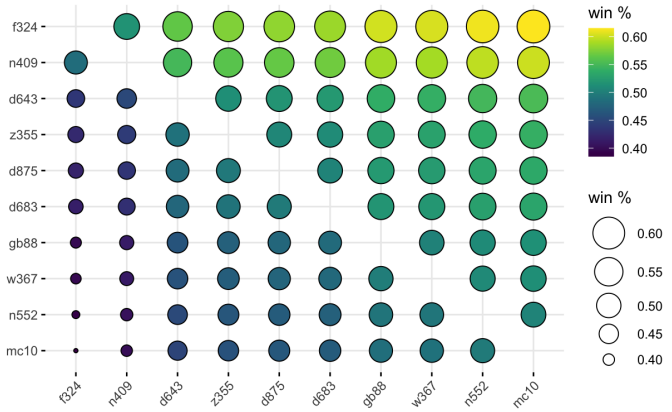

Figure 2.2: Balloon graph for the win probabilities between top 10 players in BT rankings.

The table included displays the top-10-rated players from our data in accordance to the BT model. This ranking follows the natural convention that the higher a player's ρ, the better we assume the player to be. Mathematically, this makes sense as, by the properties discussed of comparison functions (specifically as a symmetry and strictly increasing function) in the section on stochastic transitivity:

$$\rho_i > \rho_j \implies \mathcal{P}(i \succ j) > \mathcal{P}(j \succ i)$$

This translates to the obvious truth that player i's rating is greater than player j's if and only if the probability of player i beating player j is greater than the probability of j beating i.

With this in mind, the system appears to declare two clear leaders in Federer and Nadal, with both players earning a rating more than 0.50 skill units above Djokovic in 3$^{\text{rd}}$. Further down the table it appears to become more competitive, with the last four players all being ranked within 0.11 units of each other.

Looking at 2.2 we see the other interpretation of the BT model, applying our skill ratings of the top 10 players to construct a balloon graph displaying win-probabilities in head-to-heads. A clear gradient moving from left-to-right and bottom-to-top is the most prominent trend in this graph. This is explained by the ordering of the players in the matrix, with players put in descending rank from top-to-bottom and left-to-right. By a similar line of reasoning as expressed above we see that this trend is in fact what we would expect, with the main diagonal straddling the 50% mark.

2.1.5 Constructing Confidence Interval in Bradley-Terry Model

As a quantitative measurement of the performance of players in a sport, the BT model gives us a framework for conducting hypothesis tests on player skill level, or – as we will see later – the effects of covariates on player performance. The methodology investigated

in this section, in the context of competitve sport, finds natural application in the realms of sport betting, sports analysts, as well as tournament organizers by allowing us to better understand the factors and conditions that affect player performance. More generally, however, these techniques could be applied to any system of paired comparisons in a host of different disciplines.

As a maximum likelihood estimator (MLE), we note that our skill parameter, $\boldsymbol{\rho}$, will exhibit key properties associated with MLEs, which we will exploit – namely the asymptotic normality of MLEs – to construct confidence intervals for relevant hypothesis tests. Mathematically, this is stated for the one-dimensional case in the following theorem Vaart (1998)

Theorem 2.1: For m i.i.d. random variables $\{X_1, X_2, ..., X_m\}$, with p.d.f. $f(x|\theta)$ and $\theta \in \Theta$, under certain regularity conditions, given a consistent sequence of MLEs, $(\hat{\theta}_m)$, we have that

$$\sqrt{n}(\hat{\theta}_m - \theta) \to^d N(0, I_f(\theta)^{-1})$$

where $I_f(\theta)$ is the *Fisher Information* of sample size one

$$I_f(\theta) = -E_\theta \left[\frac{\partial^2}{\partial \theta^2} \ln f_\theta(X) \right].$$

Theorem 2.1 tells us our MLE $\hat{\theta}$ will be normally distributed about our true parameter θ as $m \to \infty$.

$$\hat{\theta} \sim N(\theta, (n \cdot I_f(\theta))^{-1})$$

This is generalized for the non-i.i.d. case ($\boldsymbol{Y} = \{Y_1, Y_2, ..., Y_m\}$), with multidimensional parameter $\boldsymbol{\theta} \in \mathbf{R}^n$, as

$$\hat{\boldsymbol{\theta}} \sim N_n(\boldsymbol{\theta}, I_{\boldsymbol{Y}}(\boldsymbol{\theta})^{-1}), \text{ for } I_{\boldsymbol{Y}}(\boldsymbol{\theta}) = -E_{\boldsymbol{\theta}} \left[\nabla^2 \ell(\boldsymbol{\theta}) \right] \tag{2.4}$$

where $\nabla^2 \ell(\boldsymbol{\theta})$ is the Hessian matrix of $\ell(\boldsymbol{\theta}) = \ln f_{\boldsymbol{\theta}}(\boldsymbol{Y})$, and $N_n(\boldsymbol{\mu}, \Sigma)$ is the n-dimensional multivariate normal distribution. (Zhou, 2016)

This form is naturally adapted in the case of our BT model, where we will be estimating n parameters, representing the skill levels of our n players. Hence we attempt to estimate $\boldsymbol{\rho} \in \mathbf{R}^n$, where $\rho_i \in \boldsymbol{\rho}$ is the BT rating of our i^{th} player, as our MLE for the system.

We look to apply this theory in the case of a hypothesis test in the next section.

2.1.6 Investigating Momentum in Tennis

The BT model serves as a simple yet comprehensive theoretical framework for us to model competitive sports played between two opponents, however this simplicity comes at the cost of reducing the depth at which we critically consider the sport, through a number of modelling assumptions. Specifically, the model states that player skill will follow a constant random distribution (the double exponential in this case). In this final section we attempt to improve upon the model by encoding for another parameter, *momentum*, by investigating real data from professional tennis matches and performing an appropriate hypothesis test.

We define momentum in the context of a set consisting of multiple games (most commonly a 3-game set). In such a set, played between player i and j, when i wins the first game, it has been posited that he/she will be experiencing *momentum* going into the next game, which will subsequently affect his/her performance. We model this simply by only considering the game immediately prior: momentum will only affect the winner of the previous game, regardless of the outcome of all games before. Additionally, we assume that momentum affects all players in the same way.

To do this we introduce a new parameter λ to the BT model, and restate the master equation in exponential form

$$p_{ij} = \frac{\exp\{\gamma_i + \mathbf{I}_i\lambda\}}{\exp\{\gamma_i + \mathbf{I}_i\lambda\} + \exp\{\gamma_j + \mathbf{I}_j\lambda\}}, \text{ for } \mathbf{I}_x = \begin{cases} 1 & \text{player } x \text{ won last game} \\ 0 & \text{player } x \text{ lost last game} \end{cases} \tag{2.5}$$

From this, e^λ becomes the momentum coefficient acting on a player's base skill rating $\rho_i = e^{\gamma_i}$. We will estimate these parameters $(\boldsymbol{\gamma}, \lambda)$ by looking at the likelihood function for a set of real data, observed in professional tennis matches, and estimating the MLE for our parameters using an array of computational methods.

Hyptheses: With respect to our master equation 2.5, we define hypotheses for our statistical test as

$$\mathbf{H_0} : \lambda = 0$$
$$\mathbf{H_1} : \lambda \neq 0$$

where $\lambda > 0$ means winning the previous game improves a player's performance, while $\lambda < 0$ corresponds to a drop in strength.

Methodology: To evaluate the likelihood function of our modified BT model given a set of M matches, we must first partition our data set into three categories:

$$S_1 = \{(i,j) | i \succ j,\ \mathbf{I}_i = 1\ \ \&\ \ \mathbf{I}_j = 0\}$$
$$S_2 = \{(i,j) | i \succ j,\ \mathbf{I}_i = 0\ \ \&\ \ \mathbf{I}_j = 1\}$$
$$S_3 = \{(i,j) | i \succ j,\ \mathbf{I}_i = \mathbf{I}_j = 0\}$$

These sets are equivalent to separating the matches where the winner won their past game and the loser didn't (S_1), the ones where the winner lost their past game and the loser won (S_2), and the remainder where represent the first game of the set, where neither player has momentum (S_3). With this we construct our log-likelihood function $\ell(\boldsymbol{\gamma}, \lambda | \mathbf{M})$:

$$\ell(\boldsymbol{\gamma}, \lambda | \mathbf{M}) = \sum_{(i,j)\in S_1} \ln\left(\frac{1}{1+e^{\gamma_j - (\gamma_i + \lambda)}}\right) + \sum_{(i,j)\in S_2} \ln\left(\frac{1}{1+e^{(\gamma_j+\lambda) - \gamma_i}}\right) + \sum_{(i,j)\in S_3} \ln\left(\frac{1}{1+e^{\gamma_j - \gamma_i}}\right)$$

Last, using our MLE, $\hat{\boldsymbol{\theta}}$, we perform our test using the construction outlined above (in section 2.1.5), to evaluate the statistical significance of our result.

Data & Considerations: To test our hypothesis we use data from the 2017 tennis season, looking at matches between the top 8 players, according to the ATP World Rankings. Importantly, our data must fulfil Condition 2.1.3, and hence we look only at the top 8 players to ensure a competitive graph.

Results: Using **R** process our data (see appendix for code), we started by partitioning our matches into the three categories described above. Expressing these sets as matrices (the (i,j) element of each corresponds to the number of times player i beat player j given the appropriate momentum state) we defined our log-likelihood function $\ell(\boldsymbol{\theta}|\boldsymbol{M})$ for $\boldsymbol{\theta} = (\gamma_1, \gamma_2, ..., \gamma_{10}, \lambda) \in \mathbf{R}^{11}$ as a function in **R**.

Next, by minimizing the negative of this function, using the `nlm` function, we were able to obtain the MLE of this parameter vector, $\hat{\boldsymbol{\theta}}$, and find corresponding estimates for our skill parameters $\hat{\boldsymbol{\gamma}} = (\gamma_1, ..., \gamma_{10})$ and momentum coefficient $\hat{\lambda}$ (shown in table 2.2). At this point we wish to establish whether our estimated momentum is in fact a statistically significant result. To do this, we will evaluate our result on an 80% (two-sided) confidence interval.

Table 2.2: Maximum likelihood estimates ($\hat{\boldsymbol{\theta}}$) for player skill parameters and momentum parameter. Calculated using **R** `optim` function with gr='BFGS'.

$\hat{\gamma_1}$	$\hat{\gamma_2}$	$\hat{\gamma_3}$	$\hat{\gamma_4}$	$\hat{\gamma_5}$	$\hat{\gamma_6}$	$\hat{\gamma_7}$	$\hat{\gamma_1}$	$\hat{\lambda}$
1.012	1.967	0.214	-0.532	-0.538	-1.602	0.247	-0.286	-0.218

Since we have shown that

$$\hat{\boldsymbol{\theta}} \sim N_{11}(\boldsymbol{\theta}, I_{\boldsymbol{M}}(\boldsymbol{\theta})^{-1})$$

where $I_{\boldsymbol{M}}(\boldsymbol{\theta})$ is defined in 2.4 as an (11×11) Hessian matrix, we have that the marginal distribution of our estimated momentum coefficient $\hat{\lambda}$ will follow a normal distribution with mean λ and variance equal to the corresponding in the diagonal of our Hessian matrix (the $(11, 11)^{\text{th}}$ element by our construction). That is,

$$\hat{\lambda} \sim N(\lambda, \frac{\partial^2}{\partial \theta^2} \ell(\boldsymbol{\gamma}, \lambda))$$

We estimate this as using our estimates for the MLE, as seen in Zhou (2016) to obtain a functional approximation for our variance

$$\frac{\partial^2}{\partial \theta^2} \ell(\boldsymbol{\gamma}, \lambda) = - \sum_{(i,j) \in S_1} \left(\frac{\exp\left\{\hat{\lambda} + \hat{\gamma_i} + \hat{\gamma_j}\right\}}{(\exp\left\{\hat{\lambda} + \hat{\gamma_i}\right\} + \exp\{\hat{\gamma_j}\})^2} \right) + \sum_{(i,j) \in S_2} \left(\frac{\exp\left\{\hat{\lambda} + \hat{\gamma_i} + \hat{\gamma_j}\right\}}{(\exp\left\{\hat{\lambda} + \hat{\gamma_j}\right\} + \exp\{\hat{\gamma_i}\})^2} \right)$$

Calculating this in **R** yields an estimate on variance of $s^2 = 2.74$. Under our asymptotic distribution, this means that our test statistic for our momentum parameter carries a p-value of $p = 2 \cdot (0.4312471)$ for our two-sided hypothesis test. This figure is well above our $\alpha = 0.2$ significance level, and so we conclude that given our data, we are *not* able to reject the null hypothesis.

2.2 Elo Rating

We introduce our next rating system, the **Elo rating** as developed by Arpad Elo. The rating system has been used in the context of chess tournament rankings since its first use in the *United States Chess Federation* (USCF) in 1960, although its conception dates back earlier to Elo's research in the 1950's (Glickman and Jones, 1999). Unlike the BT-Model, this system assigns each player an explicit rating value to represent their skill. This can be used similarly to BT in order to predict game outcomes and we will discuss how the system does this as well as how this changes after game outcomes.

2.2.1 Motivation

Under this system, each player is given an individual rating between 0 and 3000 which evaluates their strength. The statement of an explicit number for each player allows players to be assigned matches against others of similar strength. Additionally, the rating allows for a definitive ranking of all players participating in a particular event.

We will see that although sharing key similarities with the previously discussed BT model, the Elo model arises as a consequence of accounting for the possibility of ties in matches (an important distinction for chess matches). Fundamentally, the Elo system achieves this by considering *expected score* rather than win probabilities, as evaluated by the BT model.

2.2.2 Winning Probabilities

In a match between players i and j, we have that $E_i + E_j = 1$ for $E_k \in [0, 1]$ for $k = i, j$ as the expected points of i, where we award the points

$$\begin{cases} 1, & \text{player } i \text{ wins} \\ \frac{1}{2}, & \text{player } i \text{ draws} \\ 0, & \text{player } i \text{ loses} \end{cases}$$

Defining the random variable X_j as the number of points awarded to player i in a match against j, we may write E_i as the expectation of X_j, $E[X_j]$, and hence

$$\begin{aligned} E_i &= 1\mathcal{P}(i \succ j) + \frac{1}{2}\mathcal{P}(i \sim j) + 0\mathcal{P}(i \prec j) \\ &= p_{ij} + \frac{1}{2}\mathcal{P}(i \sim j) \end{aligned}$$

for p_{ij} the probability of i beating j, and therefore we see that E_i becomes precisely p_{ij} when we disallow draws.

As was the case in the BT model, our expected score will depend on the underlying skill levels of player i and j, (R_i, R_j) – our function will indeed depend on the difference of these two quantities to be more precise. We may derive the Elo system by assuming a rating difference of 400 should indicate that player i's expected score is 10 times that of j (Berg, 2020). Similarly, a rating difference of 800 indicates player i's expected score is 100 times that of j and so on. This is written as

$$\begin{aligned} E_i = 10^{(R_i - R_j)/400} E_j \implies E_i &= 10^{(R_i - R_j)/400} - 10^{(R_i - R_j)/400} E_i \\ \implies (1 + 10^{(R_i - R_j)/400}) E_i &= 10^{(R_i - R_j)/400} \\ \implies E_i &= \frac{10^{(R_i - R_j)/400}}{1 + 10^{(R_i - R_j)/400}} \end{aligned}$$

Rearranging, we obtain Elo's master equation

$$E_i = \frac{10^{R_i/400}}{10^{R_i/400} + 10^{R_j/400}}$$

which becomes the statement of the Bradley-Terry master equation for $\rho_i = f(R_i) = 10^{R_i/400}$. With this in mind, it is important to stress that despite this core similarity, the two models follow fundamentally different modelling assumptions regarding the distribution of a player's skill level in a given match. Whereas we previously established the BT model – via Luce's Choice Axiom – models a player's skill rating as a double exponential (also referred to as a *Gumbel extreme value distribution*), Elo's model assumes player rating to be normally distributed. In this regard, the Elo model is comparable to the Thurstonian model discussed in the section prior (shown in 2.3). (Glickman, 1995a)

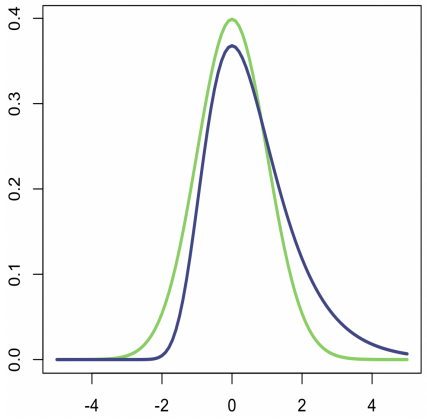

Figure 2.3: Normal p.d.f (green) compared to Gumbel's distribution p.d.f. (purple)

2.2.3 Updating Ratings

The second part of the Elo model concerns generating and updating the estimated skill values in the model R_i. As described by Glickman and Jones (1999), under its original

design, scores for established players are updated following a tournament based on overall performance, with new players (players with less than 20 officially rated games) given a *provisional* rating.

The performance rating formula is the first equation of the Elo system. It follows immediately from the normal probability curve:

$$R_P = R_C + D_P$$

where R_P is the performance rating, R_C is the average competition rating and D_P is to be read as the difference based on the percentage score P, which is obtained from the graph or table.This equation is used to determine ratings on a periodic basis. Also it may be used to determine provisional ratings in systems not on a periodic basis, such as that of the USCF.

Provisional ratings for new players are updated at the end of tournaments, and consider the player's complete career performance. Specifically,

$$R_{\text{new}} = \bar{R}_{opp} + 400\left(\frac{W-L}{N}\right)$$

where $\bar{R}_{opp}$ is the average rating of opponents played, W number of wins, L loses, and N total games. Unfortunately this system has some glaring flaws, most prominently illustrated in the case where winning a match can result in players losing rating, such as the case contrived by Glickman where a new player wins, draws, and loses against opponents rated 1400, 1500, and 1600 in one tournament, generating a provisional rating of 1500. If next he were to defeat a player of rating 700, his updated provisional rating would drop to 1400.

In the case of an established player, with a score generated based on 20 or more tournament games, rating is updated following a tournament by the *scaled* difference in expected and actual performance.

$$R_{\text{new}} = R_{\text{old}} + K(S - S_{\text{exp}})$$

for S and S_{exp} representing the number of points scored and expected points at the start of the tournament, based on their opponents played. K is referred to as an *attenuation* factor, and can be varied based on significance of the tournament and number of games played. In this setting, it is the maximum number of rating points a player can earn or lose from a match. In some sense, it adjusts the sensitivity of the rankings, whereby a large K may result in drastic shifts in placings and a small K may lead to inconsequential changes. A short discussion on our attempts at deriving an appropriate K-factor is discussed below.

2.2.4 Finding an appropriate K-factor

Under the Elo rating scheme, different contexts respond better to different K-values. Thus, one which encapsulates the rating trends found in the tennis scene is presumably able to provide better predictions for a player's skill.

One method for estimating the best K is by maximising the total log-likelihood of the matches under its Elo fit. However, this is likely to over-fit and consequently produce poor performance in unseen data. A similar approach is to instead maximise the log-likelihood resulting from a cross-validation procedure. This aims to find a good estimate for K while accounting for unseen data. This method is discussed in more detail in the next chapter. In doing so, we yielded a value of $\hat{K} = 31.41$. We use this value later for the actual fit with the tennis data set.

An extension to this static version of K is one which takes into account other various factors such as margin of victory, significance of tournament, and the uncertainty in a player's rating as explored further in the Glicko section. An example is the expression

Glickman and Doan (2015) described as part of the standard rating procedures done by the US chess federation, as given by

$$K = \frac{800}{N' + m}$$

where N' refers to the effective number of games played and m is the number of games played in the tournament by the player in question. The idea being we become more certain about a player's rating, the more games they have played. This concept is presented in a different manner in the Glicko section. Although, a similar yet simpler function for K by Kovalchik (2020), as used by the weighted Elo R package (Candila, 2021), is given by

$$K = \frac{250}{(N + 5)^{0.4}}$$

where N is the number of matches won by the player up until that point in time. This convention is explored along with the static K in a tennis data set.

2.2.5 Fitting the Tennis Data Set

The following tables 2.3 and 2.4 tally the rankings derived from the static and dynamic K values as described previously.

Table 2.3: Players with the highest fitted static Elo ratings ($K = 31.41$) from the ATP data set.

	Player	Rating
1	Federer	2597.79
2	Nadal	2544.13
3	Dimitrov	2486.29
4	del Potro	2479.46
5	Goffin	2443.56
6	Djokovic	2439.21
7	Zverev	2397.08
8	Dzumhur	2390.55
9	Sock	2378.39
10	Tsitsipas	2377.07

Table 2.4: Players with the highest fitted dynamic Elo ratings from the ATP data set.

	Player	Rating
1	Federer	2783.52
2	Krajinovic	2690.50
3	Nadal	2668.01
4	Dimitrov	2651.14
5	del Potro	2640.09
6	Tsitsipas	2565.36
7	Djokovic	2560.05
8	Goffin	2557.65
9	Dzumhur	2547.58
10	Benneteau	2537.88

A subtle difference can be found in the two rankings. Notably, a player of Krajinovic's status is ranked as high in the dynamic variant of the rating scheme. In the match data, he has only lost once of which is in the later rating periods. The dynamic K shrinks as the number of games played increases, so this observation is inline with its motivations of being more certain of a players skill over time. However, this model far underestimates the skill improvements of players over long periods of time.

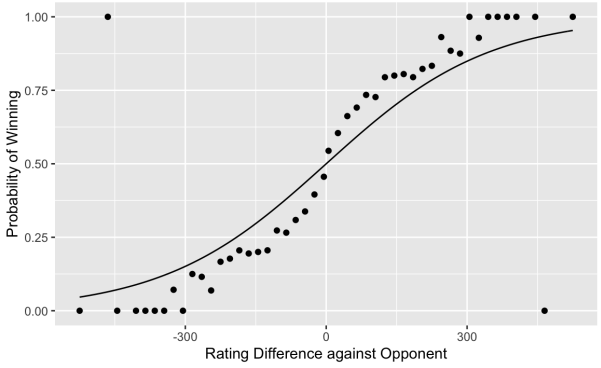

Figure 2.4: Probability of winning while rating difference varies from the ATP data set.

A slightly more sensitive system is the static Elo rating scheme employed. A plot of the probability of winning against the rating difference between players is shown

in Figure 2.4. As K was determined to maximise likelihood, this value for K is supposed to have as little residuals from the theoretical logistic curve shown as possible whilst avoiding the pitfall of over-fitting. In the same graph, there appears what could be deemed as outliers, effectively skewing the value for our best estimate for K. Nevertheless, a trend line proportional to a logistic curve is found, and so players' ratings could be regarded as having almost fully converged according to their true inherent strength. This, of course, is within a certain time epoch – a year to be specific.

2.3 Glicko Rating

We continue by discussing the **Glicko rating** system which was proposed by Glickman (1995b) and extends upon the Elo rating system by considering the reliability of a player's rating. The system adopts a Bayesian approximation method to model ratings as a normal distribution with each player having an individual mean rating and rating deviation Glickman (1999). In particular, this new rating deviation parameter is also assigned to each player and allows us to assess how reliable a rating is and how changes to it and other player's ratings should be adjusted to accommodate this new parameter. To extend further, we will also discuss the Glicko-2 rating system which considers the volatility in a player's performance when updating their ratings. These systems can then be applied to historical data to analyse their effectiveness in predicting results.

2.3.1 Motivation

Glickman (1995b) proposed the following problem - suppose two players, i and j, with identical ratings under the Elo model were to play against one another with player i beating player j. As a result of the Elo rating system, the rating of i increases by an identical amount to the decrease in j's rating. However, this relies on the underlying assumption that the match's outcome gives us equal information on both player's ability - something which is often not the case.

Suppose that both players have played a similar amount of games but player i has just resumed participating in games after a year long absence whilst player j has been consistently participating. In that year long absence, player i's rating has not changed and so would be a less accurate evaluation of their true rating whilst j's rating should be far more reliable due to more recent evidence. The proposed solution in such cases would be to have player i's rating increase by a larger amount than j's decrease. This reflects how the game's outcome gives more evidence to player i's ability. Their rating is already inaccurate so winning against j who has an accurate rating suggests i's should be much higher. Conversely, player j's rating is already quite accurate and so losing to someone with an inaccurate rating may not say much about j's ability,

To account for this, the Glicko model includes a standard deviation parameter associated with a player's rating, referred to as a *rating deviation*, which we denote by σ^2. Higher rating deviation suggests greater uncertainty in a rating and a lower deviation suggests greater reliability in a rating. Adhering to the previously mentioned reasoning, the rating deviation will also passively increase as a player remains inactive, as we become less certain in their ability if they don't participate.

2.3.2 Updating Ratings and Rating Deviations

Under this model, the player's *true* rating, θ is assumed to follow a normal distribution. The mean of this distribution will be the rating the player is normally assigned, μ, and the variance will be the player's rating deviation, σ^2. Hence $\theta \sim N(\mu, \sigma^2)$ Glickman (1999). A 95% confidence interval can then be constructed for θ, given as

$$(\mu - 2\sigma, \mu + 2\sigma)$$

Essentially, if a player has rating μ and rating deviation σ^2, we are not saying that μ is their true rating. It means that if we do this process infinitely many times, 95% of them will lie between $\mu - 2\sigma$ and $\mu + 2\sigma$.

If a player is unrated, we assign them an initial rating $\mu_0 = 2200$ as this is the default for the R package we use. We also need to assign them an initial rating deviation σ_0^2. This value is typically 300 but can also be inferred from the data set to which the model is being applied Glickman (1995b).

Rating Periods

Under the Glicko model rather than updating a player's true rating, we update the *distribution* of their true rating. This is done by updating a player's μ and σ^2 values based on the outcome of games. To begin we must first define a *rating period*. All games within this rating period are assumed to happen simultaneously and we update μ and σ^2 at the end of each period.

The decision of the rating period's length is usually left up to the discretion of those administering the event in which player's are being rated. Glickman (1999), however, suggested a length period where each player participates in around 5-10 games each. This criteria can be addressed when fitting the tennis match data to the model.

After an Outcome of a Game

We proceed to derive expressions for μ' and σ'^2, the updated rating and rating deviation for a player at then end of a rating period. The following derivation is adapted from Glickman (1999).

Let s_{jk}, $1 < j < m$, $1 < k < n_j$, be the kth outcome of the match between the player and opponent j (as the player may play against the same opponent multiple times), with an outcome being either 1, $\frac{1}{2}$, or 0 for a win, draw and loss. Denote the collection of all these outcomes as $\mathbf{s}$. We need to find a new distribution for θ based on the distributions of all the opponents' rating distributions, which we denote as $\theta_1 \sim N(\mu_1, \sigma_1^2), ..., \theta_m \sim N(\mu_m, \sigma_m^2)$. Firstly, we consider the posterior marginal density of θ given the outcomes

$$f(\theta|\mathbf{s}) = \int ... \int f(\theta_1, ..., \theta_m|\mathbf{s}) f(\theta|\theta_1, ..., \theta_m, \mathbf{s})\, d\theta_1 ... \, d\theta_m \tag{2.6}$$

The first term in the integral is the posterior marginal density of $\theta_1, ..., \theta_m$ given the outcomes of the matches. We approximate this using the product of the prior marginal densities as

$$f(\theta_1, ..., \theta_m|\mathbf{s}) \approx \varphi(\theta_1|\mu_1, \sigma_1^2)...\varphi(\theta_m|\mu_m, \sigma_m^2)$$

where $\varphi(x)$ is the Normal density function. By Bayes' theorem we also have the following expression for the second term in (2.6)

$$f(\theta|\theta_1, ..., \theta_m, \mathbf{s}) \propto \varphi(\theta|\mu, \sigma^2) L(\theta, \theta_1, ..., \theta_m|\mathbf{s})$$

Using these expressions, (2.6) becomes

$$\begin{aligned} f(\theta|\mathbf{s}) &\propto \int ... \int L(\theta, \theta_1, ..., \theta_m|\mathbf{s}) \varphi(\theta|\mu, \sigma^2) \varphi(\theta_1|\mu_1, \sigma_1^2)...\varphi(\theta_m|\mu_m, \sigma_m^2)\, d\theta_1 ... \, d\theta_m \\ &= \varphi(\theta|\mu, \sigma^2) \int ... \int L(\theta, \theta_1, ..., \theta_m|\mathbf{s}) \varphi(\theta_1|\mu_1, \sigma_1^2)...\varphi(\theta_m|\mu_m, \sigma_m^2)\, d\theta_1 ... \, d\theta_m \\ &\propto \prod_{j=1}^{m} \int \prod_{k=1}^{n_j} \frac{(10^{(\theta-\theta_j)/400})^{s_{jk}}}{1 + 10^{(\theta-\theta_j)/400}} \varphi(\theta_j|\mu_j, \sigma_j^2)\, d\theta_j \end{aligned} \tag{2.7}$$

The last line comes from considering the terms in $L(\theta, \theta_1, ..., \theta_m|\mathbf{s})$ which are the outcomes of all the games between players. This is the product of the outcomes of all the games played during the rating period but we only require the terms depending on θ. Hence, all the other terms which correspond to games between the other players can be considered constants and taken out the integral. We now proceed to find the maximum likelihood

$$L(\theta|\mathbf{s}) \approx \prod_{j=1}^{m}\prod_{k=1}^{n_j} \int \frac{(10^{(\theta-\theta_j)/400})^{s_{jk}}}{1+10^{(\theta-\theta_j)/400}} \varphi(\theta_j|\mu_j, \sigma_j^2)\, d\theta_j \tag{2.8}$$

where we move the product out of the integral in (2.7) by assuming that an opponent can play at different independent strengths across several games. We approximate further as the term in the integral in (2.8) is a Logistic cumulative function which we approximate as a Normal cumulative function with same mean and variance and then converting back after computing the integral

$$\int \frac{(10^{(\theta-\theta_j)/400})^{s_{jk}}}{1+10^{(\theta-\theta_j)/400}} \varphi(\theta_j|\mu_j, \sigma_j^2)\, d\theta_j \approx \frac{(10^{g(\sigma_j^2)(\theta-\mu_j)/400})^{s_{jk}}}{1+10^{g(\sigma_j^2)(\theta-\mu_j)/400}} \tag{2.9}$$

where

$$g(\sigma^2) = \frac{1}{\sqrt{1+3q^2\sigma^2/\pi^2}} \tag{2.10}$$

and

$$q = \frac{\log(10)}{400} \tag{2.11}$$

Hence, by substituting (2.9) into (2.8) the likelihood can be written as

$$\begin{aligned} L(\theta|\mathbf{s}) &= \prod_{j=1}^{m}\prod_{k=1}^{n_j} \frac{(10^{g(\sigma_j^2)(\theta-\mu_j)/400})^{s_{jk}}}{1+10^{g(\sigma_j^2)(\theta-\mu_j)/400}} \\ \implies \log(L(\theta|\mathbf{s})) &= \sum_{j=1}^{m}\sum_{k=1}^{n_j} [\, q\, g(\sigma_j^2 s_{jk}(\theta-\mu_j) - \log(1+10^{g(\sigma_j^2)(\theta-\mu_j)/400})\,] \\ \implies \frac{\partial \log(L(\theta|\mathbf{s}))}{\partial\theta} &= q\sum_{j=1}^{m}\sum_{k=1}^{n_j} g(\sigma_j^2)(s_{jk} - \frac{1}{1+10^{g(\sigma_j^2)(\theta-\mu_j)/400}}) \end{aligned} \tag{2.12}$$

Define

$$E(s|\theta, \mu_j, \sigma_j^2) = \frac{1}{1+10^{-g(\sigma_j^2)(\theta-\mu_j)/400}} \tag{2.13}$$

which is the expected outcome of a match between the player and opponent j. Since we defined s_{jk} to take values 1, $\frac{1}{2}$, or 0, this result can also be interpreted as the probability that the player wins against j.

From (2.12) we now know that for a maximum likelihood we require

$$\sum_{j=1}^{m}\sum_{k=1}^{n_j} g(\sigma_j^2)(s_{jk} - E(s|\hat{\theta}, \mu_j, \sigma_j^2)) = 0 \tag{2.14}$$

and when substituting $\hat{\theta}$ in the second partial derivative we have

$$\begin{aligned} \left.\frac{\partial^2 \log(L(\theta|\mathbf{s}))}{\partial^2\theta}\right|_{\theta=\hat{\theta}} &= -q^2\sum_{j=1}^{m}\sum_{k=1}^{n_j} g(\sigma_j^2)^2 E(s|\hat{\theta}, \mu_j, \sigma_j^2)(1-E(s|\hat{\theta}, \mu_j, \sigma_j^2)) \\ &= -q^2\sum_{j=1}^{m} n_j g(\sigma_j^2)^2 E(s|\hat{\theta}, \mu_j, \sigma_j^2)(1-E(s|\hat{\theta}, \mu_j, \sigma_j^2)) \\ &< 0 \end{aligned} \tag{2.15}$$

verifying the maximum. We then approximate the likelihood's density function with a Normal density function with mean $\hat{\theta}$ and variance δ^2. We approximate δ^2 by substituting μ in place of $\hat{\theta}$ in (2.15) and taking the reciprocal

$$\delta^2 \approx \left[q^2 \sum_{j=1}^{m} n_j g(\sigma_j^2)^2 E(s|\mu, \mu_j, \sigma_j^2)(1 - E(s|\mu, \mu_j, \sigma_j^2)) \right]^{-1}$$

We can now approximate the posterior distribution of θ as proportional to the product of two distributions we already know

$$f(\theta|\mathbf{s}) \propto \varphi(\theta|\mu, \sigma^2)\varphi(\theta|\hat{\theta}, \delta^2)$$

We arrive at expressions for the mean and variance of the posterior distribution of θ. In other words, the updated rating and ratings deviation at the end of the rating period

$$\sigma'^2 = \left(\frac{1}{\sigma^2} + \frac{1}{\delta^2} \right)^{-1} \tag{2.16}$$

$$\begin{aligned} \mu' &= \sigma'^2 \left(\frac{\mu}{\sigma^2} + \frac{\hat{\theta}}{\delta^2} \right) \\ &= \mu + \frac{1/\delta^2}{1/\sigma^2 + 1/\delta^2}(\hat{\theta} - \mu) \end{aligned} \tag{2.17}$$

Above is the final needed expression for σ'^2 but the expression for μ' still has a $\hat{\theta}$ term. To fix this, we approximate $(\hat{\theta} - \mu)$ using the following method. First define

$$h(\theta) = \sum_{j=1}^{m} \sum_{k=1}^{n_j} \frac{g(\sigma_j^2)}{1 + 10^{-g(\sigma_j^2)(\theta - \mu_j)/400}} \tag{2.18}$$

where $g(\sigma_j^2)$ is defined in (2.10). Substituting for $\hat{\theta}$ and using (2.14) we have

$$h(\hat{\theta}) = \sum_{j=1}^{m} \sum_{k=1}^{n_j} g(\sigma_j^2) s_{jk} \tag{2.19}$$

Expanding this using the Taylor series for $h(\theta)$ around μ gives

$$h(\hat{\theta}) \approx h(\mu) + (\hat{\theta} - \mu)h'(\mu) \tag{2.20}$$

where

$$h'(\mu) = q \sum_{j=1}^{m} \sum_{k=1}^{n_j} g(\sigma_j^2)^2 E(s|\mu, \mu_j, \sigma_j^2)(1 - E(s|\mu, \mu_j, \sigma_j^2)) \tag{2.21}$$

with q defined in (2.11). Using (2.20) we now have

$$(\hat{\theta} - \mu) \approx \frac{h(\hat{\theta}) - h(\mu)}{h'(\mu)} \tag{2.22}$$

Substituting this back into (2.17) and using expressions (2.18), (2.19) and (2.21) we can derive the final expression for μ'

$$\begin{aligned} \mu' &\approx \mu + \frac{1/\delta^2}{1/\sigma^2 + 1/\delta^2} \frac{h(\hat{\theta}) - h(\mu)}{h'(\mu)} \\ &= \mu + \frac{q}{1/\sigma^2 + 1/\delta^2}(h(\hat{\theta}) - h(\mu)) \\ &= \mu + \frac{q}{1/\sigma^2 + 1/\delta^2} \sum_{j=1}^{m} \sum_{k=1}^{n_j} g(\sigma_j^2)(s_{jk} - E(s|\mu, \mu_j, \sigma_j^2)) \end{aligned} \tag{2.23}$$

Hence, after each rating period, we apply expressions (2.16) and (2.23) to each player to update their rating and rating deviation.

After Time Passes

Now suppose that a player does not participate in any matches during the rating period. Their, rating will remain unchanged while their rating deviation will increase. Glickman (1999) showed that after t rating periods have passed, the new rating has the following distribution

$$\theta \sim N(\mu, \sigma^2 + \nu^2 t)$$

where ν^2 is a quantity which represents the increase in uncertainty per unit of time that passes without the player participating. The value for ν^2 will need to be inferred from the data when we come to apply the rating system.

2.3.3 Glicko-2 Ratings

The **Glicko-2** is another rating system which extends the Glicko rating further by introducing an additional rating volatility parameter (Glickman, 2012). Denoted by ρ, this parameter is assigned to each player and governs the consistency in their performance. A player with a high volatility rating may demonstrate erratic performance, playing very well in a few games and poorly in others. Conversely, a player with a low volatility rating will demonstrate consistent performance across games.

Consider the case where two players, i and j play a match, with i winning. Suppose however, that player i has a much lower rating and much higher volatility than player j. In such cases, player i's rating would increase by a smaller amount under the Glicko-2 system compared to Glicko-1. Likewise, player j's rating would decrease by a smaller amount too. This conveys the principle of the volatility parameter as player i is known to have more erratic performances, so the match result is less indicative of their performance compared to a consisten player like j.

Updating Ratings, Rating Deviations and Volatility

The Glicko-2 system also requires defined rating periods, an initial rating for unrated players, and quantities ν^2 and σ_0^2 which we all initialise similarly to Glicko-1. An additional parameter, τ, known as the system constant must also be set. The system constant limits changes in a player's volatility rating. This prevents drastic changes in volatility which in turn result in large rating changes. In essence, this will limit the effect an erratic player's performance has in drastically changing ratings.

We require an expression for the updated rating, rating deviation and volatility for a player, μ', σ'^2 and ρ'. The following procedure is adapted from Glickman (2012).

Let the outcome of the games, s_{jk}, be similarly defined as in Glicko-1. The Glicko-2 rating system operates under a slightly differently scale to Glicko-1. Hence, the initial rating and rating deviation of a player is converted onto the new scale before the calculations begin. We denote the rating and rating deviation under the new scale as

$$\tilde{\mu} = \frac{\mu - 1500}{173.7178}$$
$$\tilde{\sigma} = \frac{\sigma}{173.7178}$$

We first obtain several quantities required to obtain ρ'. The first is the quantity v, an estimate to the variance of the player's rating based on the outcomes. This is given by

$$v = \left[\sum_{j=1}^{m} n_j g(\tilde{\sigma_j}^2)^2 E(s|\tilde{\mu}, \tilde{\mu}_j, \tilde{\sigma}_j^2)(1 - E(s|\tilde{\mu}, \tilde{\mu}_j, \tilde{\sigma}_j^2)) \right]^{-1}$$

where

$$g(\tilde{\sigma}^2) = \frac{1}{\sqrt{1 + 3\tilde{\sigma}^2/\pi^2}}$$
$$E(s|\tilde{\mu}, \tilde{\mu}_j, \tilde{\sigma}_j^2) = \frac{1}{1 + \exp\Big(-g(\tilde{\sigma}_j^2)(\tilde{\mu} - \tilde{\mu}_j)/400\Big)} \tag{2.24}$$

and 2.24 is the Glicko-2 equivalent expression for the expected outcome of a game. We next obtain an expression for Δ, which is an estimate for the improvement of a player's rating based on the game outcomes given by

$$\Delta = v \sum_{j=1}^{m} \sum_{k=1}^{n_j} g(\tilde{\sigma}_j^2)(s_{jk} - E(s|\tilde{\mu}, \tilde{\mu}_j, \tilde{\sigma}_j^2)$$

To obtain ρ' we define

$$f(x) = \frac{e^x(\Delta^2 - \tilde{\sigma}^2 - v - e^x)}{2(\tilde{\sigma}^2 + v + e^x)^2} - \frac{(x - \log(\rho^2))}{\tau^2}$$

We require the root of the above function. This is done numerically via the *Illinois algorithm* which is a version of the *regula falsi* method for obtaining an unknown. Once we obtain the approximate solution, A, we define the updated volatility as

$$\rho' = e^{A/2} \tag{2.25}$$

We may now obtain an expression for a new prior rating deviation

$$\tilde{\sigma}* = \sqrt{\tilde{\sigma}^2 + \rho'^2}$$

Finally we arrive at expressions for the updated rating and rating deviation

$$\tilde{\sigma}' = \frac{1}{\sqrt{1/\tilde{\sigma}*^2 + 1/v}} \tag{2.26}$$
$$\tilde{\mu}' = \tilde{\mu} + \tilde{\sigma}'^2 \sum_{j=1}^{m} \sum_{k=1}^{n_j} g(\tilde{\sigma}_j^2)(s_{jk} - E(s|\tilde{\mu}, \tilde{\mu}_j, \tilde{\sigma}_j^2) \tag{2.27}$$

It remains to convert these back to the Glicko-1 scale

$$\mu' = 173.7178\tilde{\mu}' + 1500$$
$$\sigma' = 173.7178\tilde{\sigma}'$$

Hence, after each rating period, we apply expressions (2.26), (2.27) and (2.25) to each player to update their rating, rating deviation volatility. If the player does not participate in the rating period, we update their rating deviation using the same ν^2 parameter for Glicko-1.

2.3.4 Fitting the Tennis Data Set

To come up with a suitable fit for the model, a reasonable rating period must be set. From the match data, each tournament provides an average of two games played by each player. Hence, to meet the rule of thumb: 5-10 games per player as suggested by Glickman (1995b), we set each rating period to begin every four tournaments.

In a similar manner, appropriate values for σ_0, the initial rating deviation of an unrated player, and ν, the parameter corresponding to the increase in rating deviation per unit time, are often estimated to get better fits for the model. Cross-validatory methods are mainly

able to do so while limiting the effects of over-fitting. Glickman (1999) suggests a simple algorithm which calculates an approximation of the full-likelihood – the total predictive discrepancy of matches – and employs an optimiser in the form of the Nelder-Mead simplex algorithm to minimise such. An expression for a component of this discrepancy is given by the negative binomial log-likelihood

$$d_{ij} = -s_{ij}\log(p_{ij}) - (1 - s_{ij})\log(1 - p_{ij}) \tag{2.28}$$

where s_{ij} is the outcome of the game played between i and j and p_{ij} is the modelled probability that i beats j. An approximation for p_{ij} is given by

$$p_{ij} = \frac{1}{1 + 10^{-g(\sigma_i^2+\sigma_j^2)(\mu_i-\mu_j)/400}} \tag{2.29}$$

At each rating period, 2.28 is calculated and summed for all the relevant matches and the ratings are then updated. This process is continued until a value for the total predictive discrepancy is derived.

In doing so, with a max rating deviation set to 350, we yielded: $\hat{\sigma}_0 = 350$ and $\hat{\nu} = 254.24$. In comparison with the default parameters suggested by the package: $\sigma_0 = 300$ and $\nu = 15$, the cross-validation results were unsatisfactory. A point of mention is that this approximation compromises variance for bias. The relatively small sample size may have caused a significantly large variance for the estimates. Thus, we instead attempt to minimize the total log-likelihood resulting from a 10-fold cross-validation experiment, a method which we explain in further detail in the next chapter.

Subsequently, we acquire the MLEs: $\hat{\sigma}_0 = 146.04$ and $\hat{\nu} = 13.33$. From inspection, these seem to be more in line with estimates from tennis data sets in practice (Glickman, 1999). Though, it is worth mentioning that these estimates are likely to be biased downwards, i.e. underestimating the true parameters. A heat map for the average log-likelihood across each rotation of cross-validation is shown in Figure 2.5, depicting the best combination of parameters as the lightest regions.

The same process is repeated to obtain the equivalent values for the Glicko-2 system. In addition, however, we also vary the value of τ to obtain the best system constant. We acquire the MLEs: $hat\sigma_0 = 151.27$ and $\hat{\rho} = 0..07$. For $\hat{\tau}$ we observed that the MLE was always the minimum value we allowed τ to take and would go as low as 0. This can be viewed as the system trying to minimise the presence of volatility as much as possible. To compensate we decide to set $\hat{\tau} = 0.3$ the lower value of the range Glickman (2012) suggested.

Using these estimates for the parameters, we fit the Glicko and Glicko-2 model onto the match data, under the assumption of a four-tournament rating period. Each player is thus ranked as shown in Table 2.5 and 2.6 along with their appropriate parameters. We note that the final deviation values are also influenced by how players schedule tournament participation. For example, having gaps in participation between rating periods, especially the later ones, could lead to high final deviations.

Table 2.6: Top ten players as per the Glicko2 scheme with initial rating of 2200, starting deviation and volatility of 151.27 and 0.07, and volatility increase per unit time 0.3.

	Player	Rating	Deviation	Volatility
1	Federer	2595.28	60.41	0.07
2	Nadal	2526.23	54.08	0.07
3	Djokovic	2440.06	62.09	0.07
4	del Potro	2420.05	53.20	0.07
5	Dimitrov	2418.67	50.75	0.07
6	Zverev	2403.57	50.85	0.07
7	Goffin	2390.75	48.71	0.07
8	Krajinovic	2364.16	83.81	0.08
9	Agut	2357.28	52.36	0.07
10	Cilic	2357.27	52.01	0.07

Table 2.5: Top ten players as per the Glicko scheme with initial rating of 2200, starting deviation of 146.04 and deviation increase per unit time 13.33.

	Player	Rating	Deviation
1	Federer	2591.10	60.23
2	Nadal	2523.50	54.23
3	Djokovic	2437.06	61.84
4	del Potro	2418.45	53.21
5	Dimitrov	2417.55	50.85
6	Zverev	2401.73	51.03
7	Goffin	2389.55	48.84
8	Krajinovic	2361.26	83.02
9	Agut	2356.27	52.51
10	Cilic	2356.27	52.16

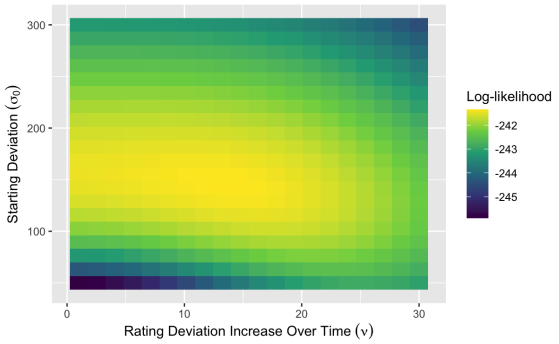

Figure 2.5: Average log-likelihood heat map of the Glicko fit as the initial rating deviation and deviation increase per time are varied.

2.4 The Fickle Fan Rating Model

We round-off our work in this chapter by proposing the use of a stochastic model, the **Fickle Fan Rating** (FFR) model, to produce a ranking from a paired comparison experiment, using Markov Chains. Note that this is a model derived by us by combining existing ideas and our imagination. The stochastic matrix, consisting of the transition probabilities, is configured to reflect the observed preferences between stimuli. Then, obtaining the convergent-state probability vector yields values that can be used to make judgements between any two stimuli. Its advantages and trade-offs are discussed on top of relevant results of Markov chains. In the end, we run a simulation for the top 10 players under the ATP data set and discuss our findings.

2.4.1 Markov Chains

A Markov chain is a stochastic process that describes a system of states and a sequence of random variables. These variables represent possible jumps between each state at every step of time. The underlying property of a Markov chain is that the next state the chain jumps to only depends on its current state and no other before Norris and Norris (1998).

More formally, consider a state space $I = 1, 2, ..., m$ and a collection of I - valued random variables $(X_t)_{t \in T}$ where T is an index set of time. We can interpret this as the random variable changing states at each step in time. Suppose, $X_{n+1} = j, X_n = i, X_{n-1} =$

$i_{n-1}..., X_0 = i_0$. The defining property of a Markov Chain can be described by

$$\mathcal{P}(X_{n+1} = j|X_n = i, X_{n-1} = i_{n-1}, ..., X_0 = i_0) = \mathcal{P}(X_{n+1} = j|X_n = i)$$

where $\mathcal{P}(X_{n+1} = j|X_n = i)$ is the probability of moving to state j when currently on state i on the n^{th} step. Furthermore, all Markov chains we will discuss are assumed to have the property of beging *time homogeneous* meaning

$$\mathcal{P}(X_{n+1} = j|X_n = i) = \mathcal{P}(X_1 = j|X_0 = i)$$

Hence, we denote $\mathcal{P}(X_{n+1} = j|X_n = i)$ as p_{ij}, the probability of moving to state j from state i.

Under this notation, the transition probabilities between the m states can be represented by the *transition matrix* (Grimmett and Stirzaker, 2001) given as

$$P = (p_{ij}) = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ . & . & . & . \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix} \tag{2.30}$$

Structural Properties

We now consider some structural properties of Markov chains which we will use in our model. For two states i and j we say that i *communicates* with j, or $i \to j$, if it is possible to reach state j at some point, given the chain starts at i. If $i \to j$ and $j \to i$ we say that i and j *intercommunicate*, or $i \leftrightarrow j$. If $i \leftrightarrow j$ for all $i, j \in I$ then we say that the Markov chain is called *irreducible*.

Next, we define what it means for a state to be *persistent*. A persistent state i is such that

$$P(X_n = i \text{ for some } n \geq 1|X_0 = i) = 1$$

which means that if the chain starts at i it will eventually return to i at some point in the future.

The *first passage time* of a state i is defined as

$$T_i = \min\{n \geq 1 : X_n = i\}$$

which is first unit in time that the Markov chain reaches i. The expected value of this is defined as the *mean recurrence time* and is given as

$$\mu_i = E(T_i|X_0 = i)$$

If $\mu_i < \infty$ then we say that state i is *non-null persistent*.

Limiting and Stationary Distributions

This matrix can be used to identify the expected distribution of the Markov chain after a certain time. To do so we discuss the *distribution* of such a Markov chain at a time n. This is defined to be an m-dimensional row vector $\lambda^{(n)}$ with entries $\lambda_i^{(n)}$ that describe the probability of the random variable being at state i after n steps. As this describes a distribution of a random variable we must have $\lambda_i^{(n)} \geq 0$ and $\sum_{i \in I} \lambda_i^{(n)} = 1$. By this definition, $\lambda^{(0)}$ is the probability that the chain starts at each state as is referred to as the *initial distribution*.

Consider Applying $\lambda^{(0)}$ to P, as defined in (2.30). Then entry i of the new vector will be given as

$$\lambda_1^{(0)} p_{1i} + \lambda_2^{(0)} p_{2i} + ... + \lambda_m^{(0)} p_{mi} = \sum_{j \in I} \lambda_j^{(0)} p_{ji}$$

which is the probability of being at state i after 1 step. Therefore this new vector is exactly $\lambda^{(1)}$! Applying $\lambda^{(1)}$ to P we thus obtain $\lambda^{(2)}$ and applying this again to P we get $\lambda^{(3)}$ and so on (Grimmett and Stirzaker (2001)). Hence

$$\lambda^{(n)} = \lambda^{(n-1)} P = \lambda^{(n-2)} P^2 = ... = \lambda^{(0)} P^n \tag{2.31}$$

Suppose now that we have a distribution which we denote as π with the defining property

$$\pi = \pi P$$

Then this distribution is defined as a *stationary distribution* of the Markov chain. In fact, if $\lambda^{(n)}$ converges as $n \to \infty$ in the first equality of (2.31), we see that π is exactly $\lambda^{(\infty)}$, the *limiting distribution* of the Markov chain. Denoting each entry in π as π_i then we have the limiting probability of being at state i.

For the rest of the stochastic model we are going to use the fact that an irreducible Markov chain with finite states has a unique stationary distribution π. Details of the proofs for this are given in Grimmett and Stirzaker (2001). The main results required for this are that a finite Markov chain has at least 1 non-null persistent state and that intercommunicating states have the same persistence. Using, this we know that an irreducible finite Markov chain must have all states non-null persistent. The final result comes from the Theorem stating that a Markov chain with all states non-null persistent will have a unique stationary distribution.

2.4.2 Fitting the Tennis Data Set

The idea behind setting up the Markov Chain

Consider a system that depicts the current best tennis player according to a fan. We may encapsulate this onto a Markov chain whose states refer to different players. In this case, the transition probabilities refer to the likelihood of a fan switching between their favourite players. In practice, this could occur when a match concludes in the favor of another player thereby leading to the fan preferring the winner. Thus, we introduce a link from state i to j only if j has beaten i at least once. On a separate note, a link from state i to itself denotes the event in which a fan stays with the same player.

The transition probabilities can be assigned based on the match data between players. The link from state i to itself represents the probability of i winning a match in general. This can be set as the win percentage of a player across all of their matches. The remaining links going out from the player state is weighted based on the number of matches they have lost against other players.

Setting the nodes - players and by adding links between the players, we can construct the transition matrix that is needed for the process of finding the steady state vector. As not all players may have played against each other, we construct ghost links that are all equal in probability from where they extend from. We set their values as a small proportion of that of the existing links. This is to avoid cases when a player has never lost and ends up hoarding all the fans. Also, this fixes the case when players are left with zero fans and a comparison between them may be deemed insignificant. The following algorithm 1 shows the construction of the transition matrix in more detail. We note that the resulting Markov chain is fully-connected, and is thus both aperiodic and irreducible. It then makes sense to solve for the unique steady-state vector.

Algorithm 1: Transition Matrix Construction

```
getTransMatrix (n, r, p)
    Input  : n: no. of players; r: results of games between players
    ; p: proportion of ghost links
    Output : Matrix A with transition probabilities as entries
    A = (a_ij);
    for i ← 1 to n do
        a_ii = games_won(i, r)/games_played(i, r);
    end
    foreach (i, j) in r do
        # where i wins over j
        a_ji+ = 1/games_played(j, r);
    end
    for i ← 1 to n do
        if Σ_{j=1}^{n} a_ij > 0 then
            for j ← 1 to n do
                a_ij = 1/n
            end
        else
            num_no_links = length([a_ij == 0 for j ← i to n])
            for j ← 1 to n do
                if a_ij == 0 then
                    a_ij = p/(num_no_links × (1 + p))
                else
                    a_ij/ = 1 + p
                end
            end
        end
    end
    return A;
```

Deriving the Steady-State Probabilities

We devise the following procedure to approximate the steady-state probability vector.

1. Distribute a number of fans, say 1000, equally among all the player states.
2. At each iteration, a fan can either remain or switch to a different state consistent with the transition probabilities.
3. After a large number, say 10000, of iterations, an approximate distribution of fans is found and used to derive the steady-state probabilities. The best case would be for the number of iterations to approach to infinity but this is not feasible in practise.

A diagram for the Markov chain for the top ten players is illustrated in Figure 2.6. This, though more intuitive and likened to practice, is less efficient of a process. To cross-reference, we also calculate the steady-state vector directly through matrix exponentiation.

Table 2.7: Top ten players according to the final proportion of fans in their state after running the simulation at iteration steps: 0, 1, and 10000.

	Player	% of fans		
		t = 0	t = 1	t = 10000
1	Federer	0.0100	0.0215	0.2118
2	Nadal	0.0100	0.0237	0.0932
3	Zverev	0.0100	0.0235	0.0477
4	Goffin	0.0100	0.0195	0.0351
5	del Potro	0.0100	0.0158	0.0339
6	Dimitrov	0.0100	0.0177	0.0328
7	Sock	0.0100	0.0170	0.0237
8	Djokovic	0.0100	0.0125	0.0211
9	Krajinovic	0.0100	0.0113	0.0208
10	Thiem	0.0100	0.0164	0.0200

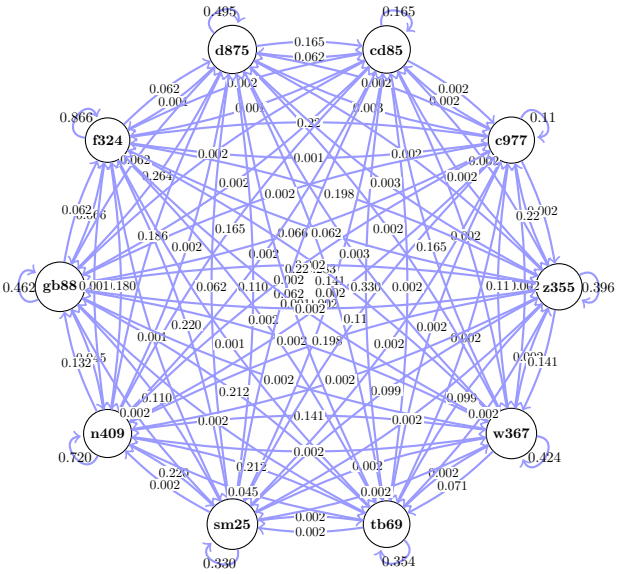

Figure 2.6: Markov State Diagram for the top ten players in the ATP rankings.

As every state is connected to each other with positive probabilities, then the steady-state vector, which we shall denote as $\mathbf{v}$, will not contain a zero entry referring to any of the players. Thus we proceed by defining the probability player i beats j as

$$\mathcal{P}(i \succ j) = \frac{v_i}{v_i + v_j}$$

where v_i and v_j refer to the steady-state entries that correspond to the states of players i and j respectively. This is a rather natural way to define the likelihood of match outcomes as is the case of the Bradley-Terry model.

The following Table 2.7 shows the proportion of fans at each player in varying iteration steps. The players are ranked according to the final proportion of fans found. The steady-state vector is also computed directly and is given in order of players as in Table 2.7 by

$$\mathbf{v} = (0.2113, 0.0933, 0.0470, 0.03497, 0.0343, 0.0331, 0.0242, 0.0214, 0.0200, 0.0198, \ldots)$$

Indeed we observe a similar distribution across the players in both methods.

Chapter 3

Bayesian Statistics & Paired Comparison

Up to this point, we have observed the topic of paired comparison through the lens of traditional frequentist statistics, where we have assumed players to possess underlying skill parameters, ρ whose relative value will determine their chance of winning against an opponent. At this point we pursue an investigation into *Bayesian* statistics in hopes of extending our models to help reckon any initial beliefs or intuitions we might have regarding our system.

Whereas previously we modelled players' skill levels as fixed parameters, with some true quantitative value, we will now consider ratings as realizations of random variables, following some distribution we will attempt to determine. This should come as a fairly natural extension, as we have already seen how some models can analogously be stated by considering a player's performance in a given match as being randomly distributed about his/her true skill level (normally distributed in the case of the Thurstonian and Elo rankings and Gumbel in the case of BT). This analysis will add another layer of control by which we will be able to "tweak" our model.

We begin with an overview on Bayesian inference, outlining theory and highlighting key results which we will use in our application at hand. This will include a brief discussion on the *Metropolis-Hastings* algorithm that will be instrumental in our computational results for the chapter. After this we conclude with an example by using our theory to adapt the BT model with three different hypothesized distributions on player rating, and investigating performance on our standard tennis data set.

3.1 Overview of Bayesian Statistics for Paired Comparison

The following overview is scoped in the context of paired comparison for the convenience of the reader.

The novel step for Bayesian inference is that we first look to select a *prior* distribution to impose on our system, to reflect any initial beliefs we may have about the nature of the distribution of player ratings – i.e. our initial guess as to how players are distributed in skill level. We denote this prior as $p(\theta)$ to be consistent with the work of Gelman et al. (2013). Next, we look to reckon our prior with the data observed to generate an improved estimate on our model distribution for player rating, the *posterior* distribution. Mathematically, this is given by Bayes' rule as

$$p(\theta|\boldsymbol{Y}) = \frac{p(\theta)p(\boldsymbol{Y}|\theta)}{p(\boldsymbol{Y})}$$

This gives us the form for our posterior, $p(\theta|\boldsymbol{Y})$ where $p(\boldsymbol{Y}|\theta)$ – our *sampling distribution* –

is equal to the likelihood of realizing our data $\boldsymbol{Y}$ given θ (for clarity we write this as $L_\theta(\boldsymbol{Y})$ going forward), and $p(\boldsymbol{Y})$ is the marginal likelihood

$$p(\boldsymbol{Y}) = \int_\theta p(\theta)p(\boldsymbol{Y}|\theta)d\theta$$

The fact that this quantity does not depend on θ means that for a fixed $\boldsymbol{Y}$ it can be taken as constant which leads to the commonly used alternative statement of the posterior distribution

$$p(\theta|\boldsymbol{Y}) \propto p(\theta)L_\theta(\boldsymbol{Y}) \tag{3.1}$$

This simply tells us that the posterior is proportional to the prior times the likelihood.

For our purposes, we will be interested in the value of θ that maximizes our form of the posterior distribution, $p(\theta|\boldsymbol{Y})$. This quantity is referred to as the *maximum a posteriori* (MAP) estimate, denoted $\hat{\theta}_{MAP}$, and tells us the value of our distribution parameter that was most likely to realize our data. By proportionality, this will be precisely the value that maximizes 3.1.

3.1.1 Markov Chain Monte Carlo Method (MCMC)

Extending the last idea we proposed before, MCMC methods can widely be used in Bayesian inference. MCMC were first introduced by a physicist and mathematician Nicolas Metropolis and these methods are usually used to generate samples directly from the "unormalized part" of the posterior, instead of dealing with intractable computations. The idea behind MCMC methods, is to construct a Markov Chain in which its equilibrium distribution is the posterior to sample from (it can be a very complicated posterior distribution). Below, we introduce the Metropolis - Hastings Algorithm (MCMC method) one of the most known algorithms developed by Metropolis et al. (1953).

Metropolis - Hastings Algorithm

Metropolis et al. (1953), came up with a very powerful idea which is a computer based algorithm that was used for studying the properties of chemical substances between collisions of individual particles. Hastings (1970) generalized the algorithm such that any probability distribution can be sampled from this algorithm.

The procedure of the algorithm starts by making an initial guess which is used as the starting position of the Markov Chain. Now, let X_t to denote the current state, for $t \geqslant 0$, of the Markov Chain. We draw a (candidate) sample Y, which is sampled from an arbitrary distribution $q(\cdot|X_t)$. Then, the candidate sample can either be rejected or accepted based on the following criterion:

$$a(X, Y) = min\left\{1, \frac{\pi(Y) \cdot q(X_t|Y)}{\pi(X_t) \cdot q(Y|X_t)}\right\}$$

where $a(X, Y)$ is called the Metropolis - Hastings acceptance probability. The **min** function restricts the probability to be no larger than 1. Now, if the above ratio is greater than 1, we can say that the candidate sample Y is accepted which means that the next state is updated to Y meaning $X_{t+1} = Y$. Otherwise, if the sample is rejected, we can say that the state doesn't change, it remains where it was and $X_{t+1} = X_t$

Now, we can proceed in further details by exploring in more depth the two cases of accepting and rejecting and what is really happening in either case:

Algorithm 2: Metropolis-Hastings Algorithm

select initial sample $\mathbf{x}_0 \in \mathbb{R}^n$ and γ **for** $i \leftarrow 1$ *to* $\mathbf{N} - 1$ **do**
  draw $\mathbf{x}_p \in \mathbb{R}^n$ from the proposal distribution $q(\mathbf{x}_p \mid \mathbf{x}_i)$
  calculate the ratio $r(\mathbf{x}_i, \mathbf{x}_p) = \min(1, \frac{\pi(\mathbf{x}_p)q(\mathbf{x}_i|\mathbf{x}_p)}{\pi(\mathbf{x}_i)q(\mathbf{x}_p|\mathbf{x}_i)})$
  draw $u \in [0, 1]$ from uniform probability density
  if $r(\mathbf{x}_i, \mathbf{x}_p) \geq u$ **then**
    $\mathbf{x}_{i+1} = \mathbf{x}_p$
  else
    $\mathbf{x}_{i+1} = \mathbf{x}_i$
  end
end

Case 1: Sample Candidate Accepted and $X_{t+1} = Y$

The probability of accepting the candidate Y equals to

$$P(Y|X_t) = q(Y|X_t) \cdot min\left\{1, \frac{\pi(Y) \cdot q(X_t|Y)}{\pi(X_t) \cdot q(Y|X_t)}\right\}$$

This can be proved by considering the detailed balance equation. The detailed balance equation is defined as

$$\pi(X_t)P(Y|X_t) = \pi(Y)P(X_t|Y)$$

Hauser (2013) proved that the Markov Chain derived from the Metropolis - Hastings algorithm satisfies indeed the detailed balance equation. This guarantees that the stationary distribution of the Metropolis - Hastings algorithm is the target posterior that is needed.

Case 2: Sample Candidate Rejected and $X_{t+1} = X_t$

In the case where we reject the sample candidate, the chain doesn't change state and it simply remains to the current state it is. The probability of rejecting the sample Y is equal to the complementary probability of the acceptance case. By recalling now that every reversible Markov Chain has a stationary distribution, then the Metropolis - Hastings algorithm is satisfied due to reversibility of our Markov Chain and the equation, always holds. Hence, this stationary distribution can be achieved from the Metropolis - Hastings algorithm.

3.1.2 Construction of Transition Matrix

We define the transition matrix as follows,

$$P_{ij} = P(X_n = j|X_{n-1} = i) = \left\{ \begin{array}{ll} q(i,i) \cdot \alpha(i,j), & \text{if } j \neq i \\ q(i,i) + \sum_{k \neq i} q(i,k) \cdot (1 - \alpha(i,k)), & \text{otherwise} \end{array} \right\} \quad (3.2)$$

As with any Transition matrix, the rows have positive entries and their sum equals to 1. If any entry in the steady state vector is 0, then the minimum will be taken to be 1, by definition of the construction matrix.

3.1.3 Symmetric Prior - Proposed Distributions

In the case of a symmetric distribution, the Metropolis - Hastings Algorithm gets simplified to a ratio that is more easily handled. Metropolis et al. (1953) showed that for a symmetric distribution

$$q(X_t|Y) = q(Y|X_t)$$

and the Metropolis - Hastings acceptance probability can be simplified to

$$a(X, Y) = min\left\{1, \frac{\pi(Y)}{\pi(X_t)}\right\}$$

We will next consider, two symmetric distributions, the Normal and the Uniform Distribution. Since both of these two distributions are symmetric, the simplified ratio will be used.

3.2 Bradley-Terry model using Bayesian Inference

The first work done for the Bayesian Inference of the Bradley-Terry model, was proposed by Davidson and Solomon (1973). In their work, they use a family of prior and estimators for calculating the posterior distribution of the log-abilities, and at the end, use this to calculate the rank of the players. In our case, for the BT model, a single match between two players i and j occurs, and as before the model looks to evaluate the probability of i beating j ($\mathcal{P}(i \succ j)$), without loss of generality. Hence, we can separate the likelihood of our simple Bayesian Bradley-Terry model into two considerations:

- **Likelihood of Win:** We write the probability of i beating j in the standard way according to the BT model:

  $$\mathcal{P}(i \succ j) = \frac{\exp(\gamma_i)}{\exp(\gamma_i) + \exp(\gamma_j)}$$

  or equivalently $y_{ij} \sim \text{Bernoulli}(\mathcal{P}(i \succ j))$ where

  $$y_{ij} = \begin{cases} 1, & i \succ j \\ 0, & j \succ i \end{cases}$$

- **Sampling Distribution:** Given our Bayesian prior, we must also take into account the probability of realizing our skill level.

The γ_i's can follow a prior distribution in accordance to the choice of the statistician. For instance, Leonard (1977) suggests a prior on the vector of skill parameters $\boldsymbol{\gamma} = (\gamma_1, , , , \gamma_n)$ in the form of a multivariate normal distribution with mean $\boldsymbol{\mu} = (\mu_1, ..., \mu_2)$, and covariance matrix $\Sigma = \sigma^2 I$: $\boldsymbol{\gamma} \sim N_n(\boldsymbol{\mu}, \Sigma)$. We will take this a step further by saying as we are sampling from a single unified population, we will assume equal mean for all our skill parameters, or $\mu_1 = ... = \mu_n = \mu_0$. By the marginal distributions, we deduce that $\{\gamma_1, ..., \gamma_2\}$ are i.i.d. with

$$\gamma_i \sim N(\mu_0, \sigma^2)$$

By the rescaling property of the BT ratings, we are able to shift the distribution and restate this in the form $\gamma_i \sim N(0, \sigma^2)$.

The mean of the $\gamma_i's$ is set to be zero since it doesn't have an effect on estimating the parameter. The standard deviation, is not set equal to zero since it represents the space, where the model should look at, trying to find the relative differences in the probabilities. Higher standard deviations, imply that probabilities are very close to either 0 or 1 and the, small standard deviations mean that are not close to these two endings, more likely to deviate around the middle.

3.2.1 Choice of Prior Distributions

Choosing a prior distribution is difficult, and many prior distributions have been candidates to be chosen. There is no correct choice of prior, but rather a statistician opinion. Many people that have been researching throughout the Bayesian Statistics applied to the Bradley Terry model, have proposed different priors. This is actually the magic of Statistics, personal opinion and initiative may contribute a lot. As mentioned before, Leonard (1977) proposed a Normal prior, additionally, Caron and Doucet (2012) explored use of the gamma distribution, while Chen and Smith (1984) worked with Dirichlet distribution and Davidson and Solomon (1973) used a Beta distribution. In this paper, we are going to consider three different priors, keeping Leonard's proposal of the Normal distribution and we are also going to consider the Gamma distribution and the Uniform distribution, discussing any advantages or disadvantages we face for each model.

1. **Normal Distribution**

   The case of Normal Distribution is presented, having the $\gamma_i's$ to follow a Normal Distribution with mean 5 and finite variance. An incentive to use unit variance is taken in order to create a Normal Distribution so that by raising e to the normal, we will get the log - normal distribution with mean $e^{5.5}$ and variance $e^{11}(e-1)$. Intuitively the Normal distribution makes a reasonable assumption that the strengths should be distributed according to a Gaussian curve, having less players with very high or low strength (tails) and many players in the middle.

2. **Gamma Distribution**

   The case of Gamma distribution is presented by many authors due to its ease to work with. The use of MCMC methods, extinguishes the difficulty to work with difficult distributions and thus Gamma Distribution could not be used. However, since the prior distribution clearly depicts the ratings - strengths of the players, the curve of the Gamma distribution is a very good candidate, more of a natural intuition, for the actual distribution of the strengths. Hence, the Gamma distribution with parameters $\alpha = 2$ and $b = 1$ is selected to describe the strengths of the players in the tennis data set. Note here that the Gamma distribution is similar to the Normal but is skewed to the left.

3. **Uniform Distribution**

   The case of Uniform distribution is also lastly presented, which intuitively means that all players have common strength. In practise, this defies the abilities of each player, making all the players to have equal strength. Intuitively this case doesn't seem ideal as far as ranking players is concerned, since equal strengths are very rare to occur in a data set, always there will be someone who is better or worst to another player. Another way to quote this is to say that the chances of a player being super skilled and super unskilled are similar.

3.2.2 Fitting the Tennis Data Set

We assign the mentioned priors on the strengths of each player and sample from their posterior distribution using the Metropolis-Hastings algorithm, taking into account the likelihood of each match outcome under the Bradley-Terry model. We find the maximum a posteriori (MAP) estimate and use these values to rank each player. The following tables list the top ten under each prior.

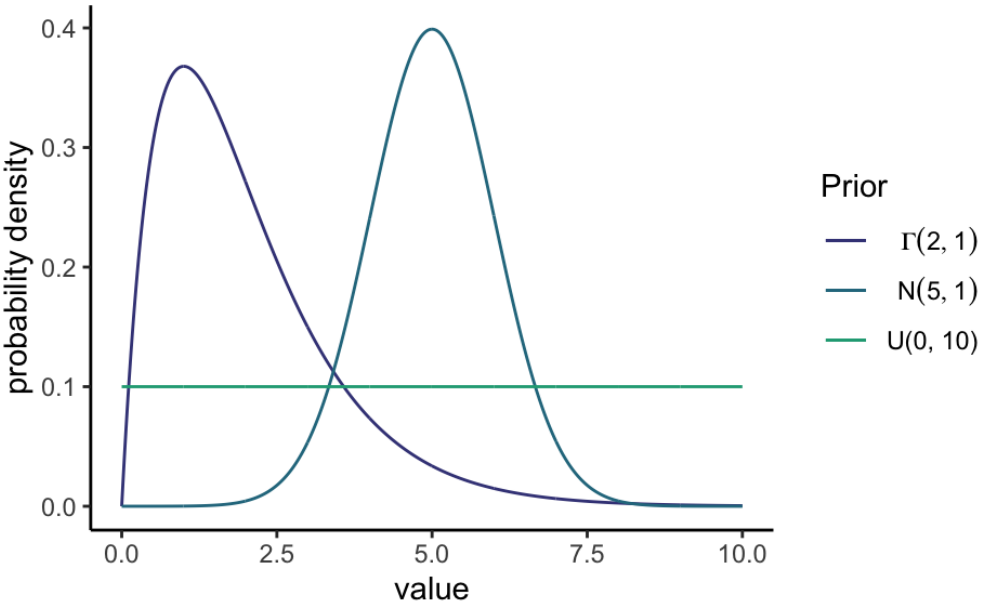

Figure 3.1: Probability densities of a gamma, normal, and uniform distribution shifted horizontally in the bounds $[0, 10]$.

Table 3.1: Real ATP 2017 Rankings

Rank	Player
1	Nadal
2	Federer
3	Dimitrov
4	Zverev
5	Thiem
6	Čilić
7	Goffin
8	Sock
9	Wawrinka
10	Busta

Table 3.2: Top ten players according to their $Gamma(2, 1)$-distributed strengths.

	γ	Player
1	4.7527	Federer
2	4.6979	Krajinovic
3	4.01684	Nadal
4	3.6204	del Potro
5	3.3315	Djokovic
6	3.0896	Dimitrov
7	3.0746	Wawrinka
8	3.0629	Zverev
9	2.9797	Tsonga
10	2.9714	Murray

Table 3.3: Top ten players according to their $N(5, 1)$-distributed strengths.

	γ	Player
1	8.2864	Federer
2	7.5402	Nadal
3	7.3924	Krajinovic
4	6.8351	Wawrinka
5	6.7616	Djokovic
6	6.6578	del Potro
7	6.5384	Dimitrov
8	6.4745	Zverev
9	6.3618	Murray
10	6.2728	Raonic

Table 3.4: Top ten players according to their $U(0, 10)$-distributed strengths.

	γ	Player
1	9.9248	Fabbiano
2	9.8794	Mmarterer
3	9.7158	Delbonis
4	9.0302	Djere
5	8.6304	Darcis
6	7.8345	Chung
7	7.8225	Pella
8	7.8099	Zverev
9	7.6149	Vesely
10	7.5129	Lajovic

3.2.3 Conclusions

In examining the three different rankings we keep their respective prior distributions (Figure 3.1) in mind to explain some of our more prominent trends. Upon inspection, Table 3.4 immediately sets itself apart from both the simulated rankings and real ranking. Indeed, this is perhaps an unsurprising confirmation that the uniform distribution is *not* a good guess for the distribution of skill across players. In this regard, this acts as a "sense-check" for our methodology, pleasantly surprising with the results and confirming our intuition for the uniform distribution by ranking players that don not belong in the top ten. With the exception of Zverev many of the names appearing on the list will likely be new to all but the more avid tennis watchers.

Moving to Table 3.2 and Table 3.3 we see that the normal and gamma priors seem to fare fairly well in comparison to the real rankings. Again, this is perhaps unsurprising considering the similarities between the gamma and normal. Moreover, on the surface, one might expect the normal distribution to be an intuitively better allocation of strengths since many quantities in real life follow normal distributions, from intelligence measures (IQ) to physical ability and income distribution. In many practical occasions, few people live at the ends and many in the center. Intuitively this is clearly described by the normal distribution. In addition to this, the positively skewed gamma distribution appears to be a good fit for the players strengths based on the ranking we obtain. A comprehensive set of quantitative statistical tools and metrics for computing accuracy of each model are described formally in the next section.

Chapter 4

Comparison of Models

In this chapter, we compare the fits of each discussed model on the ATP men's professional tennis data set. Namely, we look at the performance of the Bradley-Terry model and our Bayesian extensions, the Elo rating scheme, the Glicko and Glicko2 rating schemes, and the Fickle Fan model.

As each model has a configurable set of parameters, we find appropriate estimates by maximising the log-likelihood for the cross-validation process in the case of Elo, Glicko, and Glicko2, as explained in their respective sections. Whereas with the Bradley-Terry model, we make use of the `BradleyTerryScalable` package (Kaye et al., 2017) which provides tools to fit the model in sparse scenarios, i.e. a disconnected comparison graph. In particular, it creates a dummy player and places a prior gamma distribution on their skill. This, in turn, allows the graph to be fully connected. We chose an arbitrary shape parameter $a = 1.5$ for this prior gamma distribution. For our Bayesian extensions, the priors on the players' strength coefficients that we consider are: a gamma distribution with shape $a = 2$ and rate $b = 1$, a standard normal distribution shifted horizontally, and a uniform distribution, all within the bounds $[0, 10]$. As for the Fickle Fan model, we arbitrarily set the proportion between the sum of probabilities of existing links to the sum of probabilities of ghost links to 0.01.

Their accuracy is determined through different heuristics, each either comparing against the ATP rankings at the end of the data set's time period, based on tournament points, or the likelihood to some degree between models. Briefly, each heuristic is explained and motivations for using them are outlined. At the end, a summary of the results are shown. We primarily use the following notation below for consistency

- N = total number of players ranked
- $y_i = \begin{cases} 1 & \text{if the higher rated player won} \\ 0 & \text{if the higher rated player lost} \end{cases}$
- p_i = modelled probability that the higher rated player wins

4.1 Spearman's Rank Correlation Coefficient

One approach for determining accuracy is by comparing the model-predicted ranking to the players' actual ranking. A measure for correlation of ranks developed by Spearman (1961) is a well-known example. It is given by

$$\rho = 1 - \frac{6 \cdot \sum_{i=1}^{N} d_i^2}{N \cdot (N^2 - 1)}$$

where d_i denotes the difference between the actual and the predicted ranks. The closer the coefficient is to 1, the better the accuracy. As it is non-parametric, it is not affected

by however the population is distributed. This makes it viable for smaller sample sizes. Outliers are also bounded by the number of ranks, and therefore this measure is not easily affected (Gauthier, 2001). A problem with this approach, however, is that information on the margin of skill between players is lost and is never extensively compared with actual results. In some sense, we assume a constant margin between any two consecutive ranks.

In addition, a significance test for the result can be constructed via the statistic

$$t = \frac{\rho\sqrt{N-2}}{\sqrt{1-\rho^2}} \tag{4.1}$$

which follows a Student t distribution with $N-2$ degrees of freedom (Zar, 1972). For weakly correlated but large sample sizes, a strong significance could still be achieved. Hence, we perform a separate fit for the model where only the top ten players and their pairwise matches are considered. This way, a more useful significance value can be derived.

We fit the match data to the models twice, one containing matches between the top 100 players and one with the top ten. Their coefficients are calculated based on ATP-regulated matches in 2017 and compared against the ATP rankings at the end of 2017. A significance test is done for coefficients under the smaller sample size. Table 4.1 summarises these results.

Table 4.1: The Spearman's rank coefficient for each model based on the ATP data set, where σ_i correspond to the coefficient for a sample of size i.

Model	ρ_{100}	ρ_{10}	p-value (for ρ_{10})
Bradley-Terry			
Regular	0.8558	0.6606	0.0376
$Gamma(2,1)$	0.8185	0.6121	0.0600
$N(5,1)$	0.8151	0.6121	0.0600
$U(0,10)$	0.0349	0.3939	0.2600
Elo	0.7742	0.6485	0.0425
Glicko	0.8156	0.6606	0.0376
Glicko2	0.8171	0.6606	0.0376
Fickle Fan	0.9199	0.8061	0.0049

In both samples, the Fickle Fan model yields the highest coefficient. The p-values, which are calculated from a two-sided hypothesis test for the statistic in 4.1, suggest the ranking capabilities of the models, indeed, are not arbitrary to a certain significance level. Though, the same can not be said with the Bradley-Terry model under a uniform distribution prior, obtaining a p-value of 0.26 and a rather poor coefficient ρ_{100} of 0.0349.

A noticeable sharp decrease in performance from the models, notably the Bradley-Terry model, is seen as the sample size is decreased. This is likely due to the larger contribution of ranks being shifted in smaller sample sizes. Moreover, Nadal is ranked over Federer in the actual rankings at the end of 2017, yet has lost all his games against Federer in the match data. The Bradley-Terry model, with its inherent stochastic transitivity, is thus likely to put Federer over Nadal regardless of their matches with other players.

It is important to note that while a large sample size may not provide meaningful significance results, it is a better measure for the model's accuracy, as it takes into account games played by top players against those outside the top ten.

4.2 Brier Score

Another approach is to compare the actual results from that of the modelled probabilities. An example is the verification score developed by Brier et al. (1950) originally intended for weather forecasts. It is as shown below:

$$\text{Brier score} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2$$

which is the mean-squared error between the predictions and outcomes. The smaller it is, i.e. the closer to zero, the better the accuracy. In contrast to the previous method, this heuristic takes into account not only the rankings, but the degree of which each player is more skillful than another. However, this method is unsuitable for modelling events that either happen too rarely or frequently (Benedetti, 2010).

Table 4.2 tabulates the Brier scores of each model after calculating the individual likelihoods for each match result and finding the mean of the residuals squared.

The Bradley-Terry model outperforms the other models by a narrow margin. This could be due to its underlying mechanism of having assigned players their scores that maximise the likelihood of observing the entire outcome of matches. Naturally, this too should optimise the individual likelihoods of each match to some extent.

Table 4.2: The Brier score for each model based on the ATP data set.

Model	Brier Score
Bradley-Terry	
Regular	0.1865
$Gamma(2,1)$	0.1994*
$N(5,1)$	0.2046*
$U(0,10)$	0.4081*
Elo	0.2107
Glicko	0.2044
Glicko2	0.2039
Fickle Fan	0.1941

Scores marked with stars in Table 4.2 are derived from matches between the top 100 players. This is the case as finding optimal start values for over 500 players with given priors in the Metropolis-Hastings algorithm takes a significant amount of time and computing power. Instead, we may compare the performances of the different priors with respect to each other. The gamma and standard normal priors were considerably better in this setting, which could suggest the assumption of uniform skill is a rather restrictive construct in the tennis scene.

As for the Elo, Glicko, and Glicko2 models, the likelihoods were evaluated after all the ratings have been calculated. Another option would be to do so at each rating period and perform successive updates as an approximation for the full-likelihood (Glickman, 1999).

What is interesting is the performance of the Fickle Fan model in comparison with the others. Nowhere in the model is the likelihood of quantities maximised, but potentially there could exist some latent process stemming from the way in which transition probability weights are distributed that inherently does this.

4.3 Cross-Validation

Another method which builds off the above idea is to compare each model's performance when introduced to unseen data. **Cross-validation**, also known as rotation estimation, is an example often used in machine learning and statistics. It works by isolating a portion of the data set for training the model and testing its predictive accuracy on the rest of the data. A subset of this is **multi-fold cross-validation**. Here, the data set is partitioned, where one is used as test data and the rest as training data. The average performance of the trained model across the different partitions is then evaluated (Refaeilzadeh et al., 2009).

As is the case with most correlation and regression problems, a tendency is to over-fit from the data they are trained on. This is exactly the making of poor predictions in previously unseen data typically caused from oversampling. Cross-validation aims to quantify this (Santos et al., 2018). At the expense of more computationally intensive processes, multi-fold cross validation make use of all pieces of information in the data set and is typically preferred over, say, the traditional half-partition cross-validation process for large sample sizes (Hawkins, 2004).

We perform a 10-fold cross-validation test for the models given the entire match data set. As all models have some form of predicting the probability of a player beating another,

each is judged on their average log-likelihood across all partitions as shown in Table 4.3.

Table 4.3: The average log-likelihood derived throughout a 10-fold cross-validation test for each model based on the ATP data set.

Model	Avg. Log-likelihood
Bradley-Terry	
Regular	-238.2756
$Gamma(2,1)$	-105.7630*
$N(5,1)$	-107.3926*
$U(0,10)$	-162.2748*
Elo	-241.2198
Glicko	-241.3376
Glicko2	-241.3514
Fickle Fan	-254.5593

The Bradley-Terry model once again outperforms the other models. A similar reason from the previous section is suggested to have been the cause. An important remark on the methodology used is there have been instances when the training data bears no relevance to the test data, e.g. all the matches of a player are situated within the test data, and no such probability can be assigned to the likelihood of such match. To overcome this, we disregard these cases, i.e. assign them a likelihood of 1. This still presents an even playing field for the models as these cases are all the same and is present among all of them. After all, the goal of this test is to compare performances relative to each other, rather than interpreting the individual results.

Values starred in Table 4.3 were derived from 10-fold cross-validation of matches played between the top 100 players. This is for a similar reasoning before with the significant computation overheads. Despite this, the gamma prior performs better than the standard normal, which could suggest a non-symmetric distribution of skill may be more in line in practice. Once again, the uniform prior performs substantially worse for a similar reason.

The Elo, Glicko, and Glicko2 models perform relatively the same given their fine-tuned parameters from this very procedure. A possibility is that within the short time-span of the data set, rating deviation and volatility may not have played a huge role. And in estimating for these parameters, indeed we found them to be smaller than typically expected. As an example, the increase in volatility per unit time was estimated to be smaller than the suggested 0.3-1.2 by Glickman (1995b). In some sense, this reduces the Glicko models to their Elo counterpart.

4.4 Conclusions

Three methods were employed to evaluate the accuracy of the models, each with an underlying purpose: testing for correlation and explicit ranking accuracy, testing for individual likelihoods of matches, and testing for their ability to make predictions on previously unseen data. From these, we develop a better understanding for how each model fares in these key areas and how it ties in with their general theory. Table 4.4 below summarises the accuracy of the different models through the methods discussed above.

In general, multiple observations can be made. We do see that the Glicko and Glicko2 ratings outperform the Elo rating by a small margin in the Spearman's coefficient and Brier score measures. This could be indicative of how the rating deviation and volatility parameters allow more accurate assessment of an individual's strength and should therefore result in a better ranking. However, its effects are not as apparent. This may be attributed to the disproportionate distribution of games played by each player in a rating period. The ATP data set mainly consists of single bracket elimination formats, wherein players who are eliminated early play significantly less games than those who progress. Although the average number of games per rating period is within accepted bounds, updates of ratings may be too frequent to register the actual trends in a player's deviation and volatility.

The regular Bradley-Terry model is consistent across the different metrics used and is quite likely a very suitable model for predicting tennis matches. Its Bayesian extensions have varied performances. Both fits by the gamma and normal priors had similar success

Table 4.4: Summary of the models' accuracy based on different heuristics.

Model	Spearman's Rank Coefficient			Brier Score	Cross-Validation (Log-likelihood)
	ρ_{100}	ρ_{10}	p-value (ρ_{10})		
Bradley-Terry					
Regular	0.8558	0.6606	0.0376	0.1865	-238.2756
$Gamma(2,1)$	0.8185	0.6121	0.0600	0.1994*	-105.7630*
$N(5,1)$	0.8151	0.6121	0.0600	0.2046*	-107.3926*
$U(0,10)$	0.0349	0.3939	0.2600	0.4081*	-162.2748*
Elo	0.7742	0.6485	0.0425	0.2107	-241.2198
Glicko	0.8156	0.6606	0.0376	0.2044	-241.3376
Glicko2	0.8171	0.6606	0.0376	0.2039	-241.3514
Fickle Fan	0.9199	0.8061	0.0049	0.1941	-254.5593

with their regular counterpart, whereas the uniform prior failed to meet the same figures. Imprecise priors have the tendency to produce imprecise posterior and therefore MAP estimates in the use of an MCMC algorithm (McDonald and Hodgson, 2018). This is likely the case as to the considerable divergence in rankings and performance across the heuristics.

One of the models which stood out the most was the Fickle Fan model, one which made use of Markov chains. For its simplicity, it effectively condenses the complex relationships between players into a single object. In practice, the Elo rating scheme benefits from this as its simplicity makes scoring and rankings more transparent towards the players. Rather than running a simulation, of course, a direct calculation for the steady-state vector can be made. It excels in predicting actual rankings, but the likelihood it gets from match outcomes is still edged out by the others such as the Bradley-Terry model. A further improvement could be to fine tune the proportionality constant used in setting ghost links between states so as to maximise the likelihood of matches under its fit. At the moment, this is computationally intensive, as steady-state vectors for large transition matrices are calculated at each value the optimiser considers, while searching for a local maximum. Nevertheless, it has some potential and other ways of distributing weights on transition probabilities could be explored.

In summary, subsets amongst the models have different inner-workings that rely on certain conditions for performance: a complete comparison graph or even a precondition of players having played a certain amount of games. Each have their own strengths, simplicity being one of them, and each have their room for improvement, one being the extension from a Frequentist to a Bayesian approach. Most especially, a vast majority of models require some proper estimates for their initial parameters. Their computational costs and improvements vary from one another. In the context of tennis, the models produced relatively strong rankings of players and maximised the likelihoods of matches to a certain extent. Depending on the tournament formats, sample sizes, and consistency of intervals between games, each model is well-suited in one setting or another.

Bibliography

R. Benedetti. Scoring rules for forecast verification. *Monthly Weather Review*, 138(1): 203–211, 2010. doi: 10.1175/2009MWR2945.1.

A. Berg. Statistical analysis of the elo rating system in chess. *Chance (New York)*, 33(3): 31–38, 2020. doi: 10.1080/09332480.2020.1820249.

R. Bradley and M. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. doi: 10.2307/2334029.

G. W. Brier et al. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950. doi: 10.1175/1520-0493(1950)078⟨0001:VOFEIT⟩2.0.CO;2.

V. Candila. *welo: Weighted and Standard Elo Rates*, 2021. URL `https://cran.r-project.org/web/packages/welo/index.html`. R package version 0.1.1.

F. Caron and A. Doucet. Efficient bayesian inference for generalized bradley–terry models. *Journal of Computational and Graphical Statistics*, 21(1):174–196, 2012.

C. Chen and T. M. Smith. A bayes-type estimator for the bradley-terry model for paired comparison. *Journal of Statistical Planning and Inference*, 10(1):9–14, 1984. doi: 10.1080/10618600.2012.638220.

DataHub.io. Atp world tour tennis data [data set]. `https://datahub.io/sports-data/atp-world-tour-tennis-data`, 2019.

H. A. David. *The method of paired comparisons*, volume 12. London, 1963. URL `https://apps.dtic.mil/sti/pdfs/ADA417190.pdf#page=15`.

R. R. Davidson and D. L. Solomon. A bayesian approach to paired comparison experimentation. *Biometrika*, 60(3):477–487, 1973. doi: 10.2307/2334996.

L. R. Ford. Solution of a ranking problem from binary comparisons. *The American Mathematical Monthly*, 64(8):28–33, 1957. doi: 10.2307/2308513.

T. D. Gauthier. Detecting trends using spearman's rank correlation coefficient. *Environmental forensics*, 2(4):359–362, 2001. doi: 10.1006/enfo.2001.0061.

A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin. *Bayesian Data Analysis*. London : CRC Press, third edition edition, 2013. URL `https://ebookcentral.proquest.com/lib/imperial/detail.action?docID=1438153#`.

M. E. Glickman. A comprehensive guide to chess ratings. *American Chess Journal*, 3: 59–102, 1995a. URL `http://www.glicko.net/research/acjpaper.pdf`.

M. E. Glickman. The Glicko system. *Boston University*, pages 1–6, 1995b. URL `http://www.glicko.net/glicko/glicko.pdf`.

M. E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, 48(3):377–394, 1999. doi: 0.1111/1467-9876.00159.

M. E. Glickman. Example of the Glicko-2 system. *Boston University*, pages 1–6, 2012. URL http://www.glicko.net/glicko/glicko2.pdf.

M. E. Glickman and T. Doan. The us chess rating system. 2015. URL https://new.uschess.org/sites/default/files/media/documents/the-us-chess-rating-system-revised-september-2020.pdf.

M. E. Glickman and A. C. Jones. Rating the chess rating system. *Chance (New York)*, 12 (2):21–28, 1999. URL http://glicko.net/research/chance.pdf.

G. Grimmett and D. Stirzaker. *Probability and random processes*. Oxford university press, 2001. URL http://home.ustc.edu.cn/~zt001062/PTmaterials/Grimmett&Stirzaker--Probability%20and%20Random%20Processes%20%20Third%20Ed(2001).pdf.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. doi: 10.1093/biomet/57.1.97.

K. Hauser. Why does the metropolis-hastings procedure satisfy the detailed balance criterion?, 2013. URL https://people.duke.edu/~kh269/teaching/notes/MetropolisExplanation.pdf.

D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004. doi: 10.1021/ci0342472.

D. R. Hunter. MM algorithms for generalized bradley-terry models. *The Annals of Statistics*, 32(1):384–406, 2004. doi: 10.1214/aos/1079120141.

T. Ito and T. Shintani. Persuasion among agents: An approach to implementing a group decision support system based on multi-agent negotiation. In *15th International Joint Conference on Artificial Intelligence*, pages 592–599, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.8541&rep=rep1&type=pdf.

E. Kaye, D. Firth, and D. Selby. Bradley terry scalable. https://github.com/EllaKaye/BradleyTerryScalable, 2017.

M. G. Kendall. Further contributions to the theory of paired comparisons. *Biometrics*, 11 (1):43–62, 1955. doi: 10.2307/3001479.

M. G. Kendall and B. B. Smith. On the method of paired comparisons. *Biometrika*, 31 (3/4):324–345, 1940. doi: 10.2307/2332613.

S. Kovalchik. Extension of the elo rating system to margin of victory. *International Journal of Forecasting*, 36(4):1329–1341, 2020. doi: 10.1016/j.ijforecast.2020.01.006.

K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, 9(1):1–20, 2000. doi: 10.1080/10618600.2000.10474858.

T. Leonard. An alternative bayesian approach to the bradley-terry model for paired comparisons. *Biometrics*, 33:121–132, 1977. doi: 10.2307/2529308.

R. Luce. *Individual Choice Behavior: A Theoretical Analysis*. John Wiley, New York, 1959.

R. Luce. The choice axiom after twenty years. *Journal of Mathematical Psychology*, 15(3): 215–233, 1977. doi: 10.1016/0022-2496(77)90032-3.

R. Luce and P. Suppes. *Preference, utility and subjective probability*, volume 3. Wiley, New York, 1965. URL https://www.imbs.uci.edu/files/personnel/luce/pre1990/1965/LuceSuppes_Book%20Chapter_1965.pdf.

J. L. McDonald and D. J. Hodgson. Prior precision, prior accuracy, and the estimation of disease prevalence using imperfect diagnostic tests. *Frontiers in veterinary science*, 5:83, 2018. doi: 10.3389/fvets.2018.00083.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21 (6):1087–1092, 1953. doi: 10.1063/1.1699114.

J. R. Norris and J. R. Norris. *Markov chains*. Cambridge university press, 1998. URL https://www.cambridge.org/core/books/markov-chains/A3F966B10633A32C8F06F37158031739.

I. Oliveira. Stochastic transitivity: Axioms and models. *Journal of Mathematical Psychology*, 85:25–35, 2018. doi: 10.1016/j.jmp.2018.06.002.

P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. *Encyclopedia of database systems*, 5:532–538, 2009. doi: 10.1007/978-1-4614-8265-9_565.

M. S. Santos, J. P. Soares, P. H. Abreu, H. Araujo, and J. Santos. Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches [research frontier]. *IEEE Computational Intelligence Magazine*, 13(4):59–76, 2018. doi: 10.1109/MCI.2018.2866730.

C. Spearman. The proof and measurement of association between two things. *International journal of epidemiology*, 1961. doi: 10.1093/ije/dyq191.

L. L. Thurstone. A law of comparative judgement. *Psychological Review*, 34(4):273–286, 1927. doi: 10.1037/0033-295X.101.2.266.

A. W. v. d. Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998. doi: 10.1017/CBO9780511802256. URL https://www.cambridge.org/core/books/asymptotic-statistics/A3C7DAD3F7E66A1FA60E9C8FE132EE1D.

J. I. Yellot. The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109–144, 1977. doi: 10.1016/0022-2496(77)90026-8.

J. H. Zar. Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association*, 67(339):578–580, 1972. doi: 10.1080/01621459.1972.10481251.

F. Zhou. Stanford STATS200: Lecture 24 – The Bradley-Terry model. https://web.stanford.edu/class/archive/stats/stats200/stats200.1172/Lecture24.pdf, 2016.

Chapter 5

Appendix

5.1 Bradley-Terry Model

5.1.1 Fitting the Model

The following was used to fit the Bradley-Terry model onto match results of the 2017 ATP tennis data set. This produces the rankings as found in Table 2.1 and the balloon graph in Figure 2.2.

```
# Set-up
## Libraries
library("jsonlite")
library("BradleyTerryScalable")

## Retrieving tennis data set
json_file <- 'https://datahub.io/sports-data/atp-world-tour-tennis-data/
    datapackage.json'
json_data <- fromJSON(paste(readLines(json_file), collapse=""))

path_to_files <- json_data$resources$path
match_data <- na.omit(read.csv(url(path_to_files[34])))
player_data <- na.omit(read.csv(url(path_to_files[38])))

# Fitting the data
## Formatting data for BradleyTerryScalable package
win_lose_tally <- dplyr::count(match_data, winner_player_id, loser_player_id)

tennis_btdata <- btdata(win_lose_tally, return_graph=TRUE)
tennis_btfit <- btfit(tennis_btdata, 1.5)

## Visualising the comparison graph
library(igraph)
comp_graph <- tennis_btdata$graph
par(mar = c(0, 0, 0, 0) + 0.1)
plot.igraph(comp_graph, layout=layout.circle, vertex.size = 1, edge.arrow.size =
    0.05, vertex.label=NA)

# Get list of players in order of coefficients
tennis_id_coef <- coef(tennis_btfit, as_df=TRUE)[c('item', 'coef')]
player_names <- player_data$player_slug[match(tennis_id_coef$item, player_data$
    player_id)]

tennis_player_coef <- tennis_id_coef[, 'coef']
tennis_player_coef$player_name <- player_names
print(tennis_player_coef)

```

```r
## Balloon plot for the top 10 players
library('ggpubr')

# Get only the top players
num_players <- 10
top_players <- tennis_id_coef$item[1:num_players]
top_coef <- tennis_id_coef$coef[1:num_players]

# Get matrix of win percentages
win_pct_matrix <- outer(top_coef, top_coef, function(x, y) x / (x + y))
# win_pct_matrix <- win_pct_matrix - diag(0.5, length(top_players))
diag(win_pct_matrix) <- rep(NA, num_players)
colnames(win_pct_matrix) <- top_players
rownames(win_pct_matrix) <- top_players
win_pct_df <- melt(t(win_pct_matrix))
colnames(win_pct_df)[3] <- 'win␣%'

balloon_plot <- ggballoonplot(win_pct_df, fill="win␣%") + scale_fill_gradientn(
    colors = viridis(10))

balloon_plot
```

5.1.2 Covariate Hypothesis Testing

While results from these hypothesis testing were never used, it offered insight into how we might approach testing momentum in the later sections. The effects of player's co-variate data (dimensions of the stimuli) on their Bradley-Terry coefficients is explored, with the findings that neither age, height, nor weight proved to have a significant effect on men's performance in the ATP data set, whereas both age and height did in women's performance in the WTA data set. An interpretation could be: after a certain level, a player's biology does not offer much improvement to their skill. Whereas in the men's section we hypothesise this level is met, the women's section may not have and so these traits still offer some benefit in some way. A limitation is the usage of age or biological measures directly in the linear model, whereas a more suited method would be to bin them and create separate coefficients for them.

```r
# Perform hypothesis test: does weight, height, or age affect strength?
# Sample from a population
## Get list of players with all the data we need and have played at least once
player_df <- player_data[player_data$weight_kg != 0 & player_data$height_cm != 0
    & player_data$birth_year != 0 & player_data$player_id %in% tennis_id_coef$
    item, ]

## Get random players from this list
n <- min(100, length(player_df[,1]))
player_ids <- sample(player_df$player_id, size=n)

# Build Model
## Observed BT coefficients
Y <- tennis_id_coef$coef[match(player_ids, tennis_id_coef$item)]

## Build design matrix
### Age of players is fixed as data is from 2017
year_data_was_from <- 2017

col_of_ones <- rep(1, n)
col_weight <- player_df$weight_kg[match(player_ids, player_df$player_id)]
col_height <- player_df$height_cm[match(player_ids, player_df$player_id)]
```

```r
col_age <- year_data_was_from - player_df$birth_year[match(player_ids, player_df$player_id)]

## Number of parameters
p <- 4
X_entries <- c(col_of_ones, col_weight, col_height, col_age)

X <- matrix(X_entries, ncol=p, nrow=n)

## Calculate the LSE and RSS
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% Y
Y_hat <- X %*% beta_hat
RSS <- t(Y) %*% Y - t(Y_hat) %*% Y_hat

# Perform hypothesis test on each co-variate
alpha <- 0.05
## For weight, height, and age resp.
c_values <- matrix(c(c(0,1,0,0), c(0,0,1,0), c(0,0,0,1)), ncol=p-1, nrow=p)

## Get 1-alpha statistic based on t_{n-p} distribution
t_stat <- qt(1-alpha / 2, df=n-p)

for(i in seq(1,p - 1)){
  ## Get 1-alpha confidence interval
  c <- c_values[, i]
  beta_mean <- t(c) %*% beta_hat
  denom <- sqrt(t(c) %*% solve(t(X) %*% X) %*% c * RSS / (n - p))
  half_width <- t_stat * denom
  low_bound <- beta_mean - half_width
  high_bound <- beta_mean + half_width

  ## Check if 0 is inside interval
  within_interval <- low_bound < 0 & 0 < high_bound

  ## Alternatively: find p-value
  pivotal_qty <- abs(beta_mean/denom)
  p_value <- 2 * pt(pivotal_qty, df=n-p, lower.tail=FALSE)

  print(paste("Test for H_0: beta_", i, "=0:"))
  print(paste("The ", 1-alpha," CI: ", low_bound, ", ", high_bound, ";"))
  print(paste("p-value: ", p_value, ";"))
  print(paste("Result: ", ifelse(within_interval, "accept H_0.", "reject H_0.")))
  print("")
}

# Women's tennis (WTA) analysis
## Loading the data
```{r}
wta_match_df <- read.csv("https://raw.github.com/JeffSackmann/tennis_wta/master/wta_matches_2017.csv")

wta_players_df <- na.omit(read.csv("https://raw.github.com/JeffSackmann/tennis_wta/master/wta_players.csv"))

wta_rankings_df <- na.omit(read.csv("https://raw.github.com/JeffSackmann/tennis_wta/master/wta_rankings_10s.csv"))
```

Get top 10 players
```

```
rank_rows <- wta_rankings_df[wta_rankings_df$ranking_date == '20171120',][1:10,]
names <- wta_players_df$name_last[match(rank_rows$player, wta_players_df$player_
    id)]
wta_ranking <- cbind(names, rank_rows$points)
colnames(wta_ranking) <- c('player_name', 'points')

## Fit to Bradley Terry Model
match_tally <- dplyr::count(wta_match_df, winner_id, loser_id)
match_tally <- transform(match_tally, winner_id=as.character(winner_id), loser_id
    =as.character(loser_id))

wta_btdata <- btdata(match_tally, return_graph=FALSE)
wta_btfit <- btfit(wta_btdata, a=1)

wta_id_coef <- coef(wta_btfit, as_df=TRUE)[c('item', 'coef')]
player_names <- wta_players_df$name_last[match(wta_id_coef$item, wta_players_df$
    player_id)]

wta_player_coef <- wta_id_coef[, 'coef']
wta_player_coef$player_name <- player_names
print(wta_player_coef)

# Perform hypothesis test: does height, age, or right-handedness affect strength?
# Sample from a population
player_df <- wta_players_df[wta_players_df$player_id %in% wta_id_coef$item, ]

## Get random players from this list
n <- min(300, length(player_df[,1]))
player_ids <- sample(player_df$player_id, size=n)

# Build Model
## Observed BT coefficients
Y <- wta_id_coef$coef[match(player_ids, wta_id_coef$item)]

## Build design matrix
### Age of players is fixed as data is from 2017
year_data_was_from <- 2017

col_of_ones <- rep(1, n)
col_height <- player_df$height[match(player_ids, player_df$player_id)]
col_age <- year_data_was_from - player_df$dob[match(player_ids, player_df$player_
    id)] %/% 10000
col_right <- ifelse(player_df$hand[match(player_ids, player_df$player_id)] != 'R'
    , 0, 1)
# col_left <- ifelse(player_df$hand[match(player_ids, player_df$player_id)] != 'L
    ', 0, 1)

## Number of parameters
p <- 4
X_entries <- c(col_of_ones, col_height, col_age, col_right)

X <- matrix(X_entries, ncol=p, nrow=n)

## Calculate the LSE and RSS
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% Y
Y_hat <- X %*% beta_hat
RSS <- t(Y) %*% Y - t(Y_hat) %*% Y_hat

# Perform hypothesis test on each co-variate
```

```r
alpha <- 0.05
## For weight, height, and age resp.
c_values <- matrix(c(c(0,1,0,0), c(0,0,1,0), c(0,0,0,1)), ncol=p-1, nrow=p)

## Get 1-alpha statistic based on t_{n-p} distribution
t_stat <- qt(1-alpha / 2, df=n-p)

for(i in seq(1,p - 1)){
  ## Get 1-alpha confidence interval
  c <- c_values[, i]
  beta_mean <- t(c) %*% beta_hat
  denom <- sqrt(t(c) %*% solve(t(X) %*% X) %*% c * RSS / (n - p))
  half_width <- t_stat * denom
  low_bound <- beta_mean - half_width
  high_bound <- beta_mean + half_width

  ## Check if 0 is inside interval
  within_interval <- low_bound < 0 & 0 < high_bound

  ## Alternatively: find p-value
  pivotal_qty <- abs(beta_mean/denom)
  p_value <- 2 * pt(pivotal_qty, df=n-p, lower.tail=FALSE)

  print(paste("Test for H_0: beta_", i, "=0:"))
  print(paste("The ", 1-alpha," CI: ", low_bound, ", ", high_bound, ";"))
  print(paste("p-value: ", p_value, ";"))
  print(paste("Result: ", ifelse(within_interval, "accept H_0.", "reject H_0.")))
  print("")
}
```

5.1.3 Momentum Hypothesis Testing

This section displays the code used for the momentum test conducted in section 2.1.6. The process is composed of four steps (1) retrieving and partitioning match data into sets as described in our our methodology (2) constructing our log likelihood function as a sum across the partitioned data (3) optimizing our log likelihood function using built-in R functions (4) estimating our variance of our momentum parameter's marginal distribution as the corresponding diagonal element of the (Hessian) Fisher information matrix.

```r
# Set-up
## Libraries
library("jsonlite")

## Retrieving tennis data set
json_file <- 'https://datahub.io/sports-data/atp-world-tour-tennis-data/
    datapackage.json'
json_data <- fromJSON(paste(readLines(json_file), collapse=""))

path_to_files <- json_data$resources$path
match_data <- na.omit(read.csv(url(path_to_files[34])))
player_data <- na.omit(read.csv(url(path_to_files[38])))
ranking_data <- as.data.frame(read.csv(url(path_to_files[37]), nrows=300))

matches <- match_data[c('winner_player_id', 'loser_player_id', 'match_score_
    tiebreaks', 'tourney_order')]


## Only consider top N players
```

```r

N <- 8
active_players <- unique(c(matches$winner_player_id, matches$loser_player_id))
top_player_ids <- ranking_data[ranking_data$player_id %in% active_players,]$
    player_id[1:N]
top_matches <- matches[matches$winner_player_id %in% top_player_ids & matches$
    loser_player_id %in% top_player_ids,]

# Analysis
## Tally matrices
### Helper functions
winner_sets_won = function(match_score_string){
  scores <- strsplit(match_score_string, "␣")[[1]]
  sets_won <- unlist(lapply(scores, function(x) {
    r <- strsplit(strsplit(x, "\\(")[[1]][1], "")[[1]]
    return(r[1] > r[2])
  }), use.names=FALSE)
  sets_won[is.na(sets_won)] = TRUE
  return(sets_won)
}
v_winner_sets_won = Vectorize(winner_sets_won, vectorize.args = 'match_score_
    string')


### Tally actual matrices
S3 <- matrix(0, nrow=N, ncol=N, dimnames=list(top_player_ids, top_player_ids))
S1 <- S3
S2 <- S3

winner_set_outcomes <- v_winner_sets_won(top_matches$match_score_tiebreaks)
for (i in seq(length(winner_set_outcomes))) {
  scores <- winner_set_outcomes[[i]]
  prev <- scores[-1]
  curr <- scores[-length(scores)]

  w <- top_matches$winner_player_id[i]
  l <- top_matches$loser_player_id[i]

  ifelse(scores[1], S3[w,l] <- S3[w,l] + 1, S3[l,w] <- S3[l,w] + 1)
  ifelse(prev & curr, S1[w,l] <- S1[w,l] + 1, ifelse(prev & !curr, S2[l,w] <- S2[
      l,w] + 1, S2[w,l] <- S2[w,l] + 1))
}


## Process tallies
convert_to_app_df = function(tally){
  new_tally <- as.data.frame(as.table(tally))
  colnames(new_tally) <- c('w', 'l', 'count')
  return(new_tally)
}

S1_df <- convert_to_app_df(S1)
S2_df <- convert_to_app_df(S2)
S3_df <- convert_to_app_df(S3)


## Negative log-likelihood function
bt_likelihood = function(params){
  coefs <- table(params[1:N])
```

```
  names(coefs) <- top_player_ids

  theta <- params[N + 1]

  S1_lk <- sum(S1_df$count * log(1 + exp(coefs[S1_df$l] - (coefs[S1_df$w] + theta
      ))))
  S2_lk <- sum(S2_df$count * log(1 + exp(coefs[S2_df$l] + theta - coefs[S2_df$w])
      ))
  S3_lk <- sum(S3_df$count * log(1 + exp(coefs[S3_df$l] - coefs[S3_df$w])))

  return(S1_lk + S2_lk + S3_lk)
}


## Maximise Negative log-likelihood function
start = start = c(1.06, 1.97, 0.191, -0.594, -0.631, -1.72, 0.104, -0.383, 0)
res_optim <- optim(par=start, fn=bt_likelihood, gr="BFGS", control=list(maxit=100
    0))
mles <- res_optim$par


print(paste("status:", ifelse(!res$convergence, "did␣converge", "did␣not␣converge
    ")))
print(paste("theta_hat:", mles[length(mles)]))

## Get Fisher information
theta <- mles[length(mles)]
variance = 0
for (i in top_player_ids){
  for (j in top_player_ids) {
    if (i != j) {
      gam_i <- mles[match(i, top_player_ids)]
      gam_j <- mles[match(j, top_player_ids)]

      variance <- variance - S1[i,j] * (exp(theta + gam_i + gam_j) / (exp(theta +
          gam_i) + exp(gam_j)) ^ 2) + S2[i,j] * (exp(theta + gam_i + gam_j) / ((
          exp(theta + gam_j) + exp(gam_i)) ^ 2))
    }
  }
}
```

5.2 Elo and Glicko Ratings

5.2.1 Fitting the Model

The following fits the Elo and Glicko models onto match data found in the 2017 ATP tennis data set. This produces the rankings as found in Table 2.3 and Table 2.5.

```
# Setup
## Load data as previously
## Libraries
library("PlayerRatings")

# Simulate calculations of ratings
## Prepare data frame
matches <- match_data[c('tourney_order', 'winner_player_id', 'loser_player_id')]
matches$winner <- rep(1, length(matches[,1]))

```

```r
## Assert four tournaments per rating period
num_tournaments <- length(unique(matches$tourney_order))
rating_periods <- num_tournaments %/% 4
matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_order
    ) %/% rating_periods) + 1)[1:length(matches$tourney_order)]

## Get dataframes of players and their ratings
### Elo ratings
elo_res <- elo(matches, kfac=31.41, as_data_frame=TRUE)
### Glicko ratings
glicko_res <- glicko2(matches, init=c(2200, 146.0352), cval= 13.3277, as_data_
    frame=TRUE)
### Glicko2 ratings
glicko2_res <- glicko2(matches, init=c(2200, 151.27494073, 0.07491802), as_data_
    frame=TRUE)
```

5.2.2 Dynamic Elo K-factor

The following simulates the Elo model under the dynamic K-factor by Kovalchik (2020) under the ATP tennis data set. This produces the ranking in Table 2.4.

```r
# Simulation setup
## Get all relevant players
all_players <- unique(c(match_data$winner_player_id, match_data$loser_player_id))
num_players <- length(all_players)

## Give all players a starting rating
start_rating <- 2200
ratings <- rep(start_rating, num_players)

# Dynamic K simulation
## Helper for counting games
replace_col = function(player, game_count, match_df=matches){
  if (match_df$winner_player_id == player) {
    match_df$winner_count <- game_count
  } else if (match_df$loser_player_id == player) {
    match_df$loser_count <- game_count
  }
}
vreplace_col = Vectorize(replace_col, vectorize.args='match_df')

## Setup for fitting
K <- 800
start_rating <- 2200
matches <- match_data[c('tourney_order', 'winner_player_id', 'loser_player_id')]

all_players <- unique(c(matches$winner_player_id, matches$loser_player_id))
num_players <- length(all_players)
ratings <- rep(start_rating, num_players)

### Count number of games played
total_games_played <- length(matches[,1])
matches$winner_games_played <- rep(0, total_games_played)
matches$loser_games_played <- rep(0, total_games_played)

for (player in all_players){
    relevant_matches <- matches[matches$winner_player_id == player | matches$
        loser_player_id == player,]
    num_games_played <- length(relevant_matches[,1])
```

```
38
39      matches$winner_games_played <- ifelse(matches$winner_player_id == player, seq
            (1, num_games_played), matches$winner_games_played)
40      matches$loser_games_played <- ifelse(matches$loser_player_id == player, seq(1,
             num_games_played), matches$loser_games_played)
41  }
42
43  ### Assert four tournaments per rating period
44  num_tournaments <- length(unique(matches$tourney_order))
45  rating_periods <- num_tournaments %/% 4
46  matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_order
        ) %/% rating_periods) + 1)[1:length(matches$tourney_order)]
47
48  ### Simulate matches and rating fluctuations while updating ratings only at the
        end of each rating period
49  for (i in seq(rating_periods)) {
50      matches_in_period <- matches[matches$tourney_order == i,]
51      updates <- as.data.frame(cbind(all_players, rep(0, length(all_players))))
52      updates[,2] <- sapply(updates[,2], as.numeric)
53      colnames(updates) <- c('player_id', 'score_change')
54      for (i in seq(length(matches_in_period[,1]))){
55        row <- matches_in_period[i,]
56        A <- row$winner_player_id
57        B <- row$loser_player_id
58
59        R_A <- ratings[which(all_players == A)]
60        R_B <- ratings[which(all_players == B)]
61
62        E_A <- 1 / (1 + 10^((R_B - R_A) / 400))
63        E_B <- 1 - E_A
64
65        K_A <- kfac / (row$winner_games_played + 5) ^ 0.4
66        K_B <- kfac / (row$loser_games_played + 5) ^ 0.4
67
68        updates$score_change[updates$player_id == A] <- updates$score_change[
              updates$player_id == A] + K_A * (1 - E_A)
69        updates$score_change[updates$player_id == B] <- updates$score_change[
              updates$player_id == B] + K_B * (0 - E_B)
70      }
71      ratings <- ratings + updates[match(all_players, updates$player_id),]$score_
            change
72  }
73
74  ## Rank players based on their rating
75  player_rankings <- as.data.frame(cbind(all_players, ratings))
76  player_rankings <- player_rankings %>% arrange(desc(ratings))
77  player_names <- player_data$player_slug[match(player_rankings$all_players, player
        _data$player_id)]
78
79  player_rankings <- as.data.frame(cbind(player_names, player_rankings$all_players,
         player_rankings$ratings))
80
81  colnames(player_rankings) <- c('player_slug', 'player_id', 'ELO␣rating')
82  print(player_rankings)
```

5.2.3 Plotting Trends

The following produces the rating differential versus probability of winning trend in Figure 2.4 as well as an unused trend for the Glicko rating scheme.

```
# Libraries
library("ggplot2")
library("dplyr")

# Draw scatter-plot along with predicted curve
## Convert data into a list of points
win_lose_tally <- count(match_data, winner_player_id, loser_player_id)

ordered_partition <- win_lose_tally[win_lose_tally$winner_player_id < win_lose_tally$loser_player_id,]
ordered_partition[,4] <- rep(0, length(ordered_partition[,1]))
colnames(ordered_partition) <- c('p1', 'p2', 'w1', 'w2')

swapped_partition <- win_lose_tally[win_lose_tally$winner_player_id > win_lose_tally$loser_player_id,]
swapped_partition <- swapped_partition[c('loser_player_id', 'winner_player_id', 'n')]
swapped_partition[,4] <- rep(0, length(swapped_partition[,1]))
colnames(swapped_partition) <- c('p1', 'p2', 'w2', 'w1')

tally <- aggregate(.~p1+p2, data=rbind(ordered_partition, swapped_partition), sum)

## Get rating diff vs % win graph
### Helper function
get_rating_win_points = function(res) {
  r_p1 <- res[[1]]$Rating[match(tally$p1, res[[1]]$Player)]
  r_p2 <- res[[1]]$Rating[match(tally$p2, res[[1]]$Player)]

  points_avg <- as.data.frame(cbind(as.numeric(as.vector(abs(r_p1 - r_p2))), ifelse(r_p1 > r_p2, tally$w1, tally$w2), ifelse(r_p1 > r_p2, tally$w2, tally$w1)))

  points_avg$V1 <- cut(points_avg$V1, breaks=seq(0, 1000, 20), labels=seq(5, 995, 20))
  points_avg <- aggregate(.~V1, data=points_avg, sum)
  points_avg <- as.data.frame(cbind(as.numeric(as.vector(points_avg$V1)), points_avg$V2 / (points_avg$V2 + points_avg$V3)))
  colnames(points_avg) <- c('rating_diff', 'p_win')

  neg_points_avg <- as.data.frame(cbind(-points_avg$rating_diff, 1- points_avg$p_win))
  colnames(neg_points_avg) <- c('rating_diff', 'p_win')
  points <- rbind(points_avg, neg_points_avg)

  return(points)
}

## Plot average per rating bin
elo_pred <- function(x) 1 / (1 + 10 ^ (-x / 400))

elo_res <- elo(matches, kfac=31.41, as_data_frame=TRUE)
points <- get_rating_win_points(elo_res)

ggplot(points, aes(x=rating_diff, y=p_win)) + geom_point() + stat_function(fun =
```

```
    elo_pred) + xlab('Rating␣Difference␣against␣Opponent') + ylab('Probability␣of
    ␣Winning')
47
48 ## Plot Glicko scatter-plot
49 points <- get_rating_win_points(glicko_res)
50
51 ggplot(points, aes(x=rating_diff, y=p_win)) + geom_point()
```

5.2.4 Parameter Estimation

Elo Static K

The following calculates the estimate for the static value for K maximising the log-likelihood of the 10-fold cross-validation procedure used in producing Table 2.3 and the results in chapter 4. We yield a value of $K = 31.41$

```
1  # Best Elo K-factor
2  ## Likelihood function
3  library('PlayerRatings')
4  elolikelihood = function(training_data, test_data, kfac=27){
5    data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
       id')]
6    data_df$win <- rep(1, length(data_df[,1]))
7
8    num_tournaments <- length(unique(data_df$tourney_order))
9    rating_periods <- num_tournaments %/% 4
10   data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
       order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]
11
12   elo_res <- elo(data_df, kfac=kfac, as_data_frame=TRUE)
13   r_A <- elo_res[[1]]$Rating[match(test_data[[1]]$winner_player_id, elo_res[[1]]$
       Player)]
14   r_B <- elo_res[[1]]$Rating[match(test_data[[1]]$loser_player_id, elo_res[[1]]$
       Player)]
15
16   l_prob <- sum(log(ifelse(is.na(r_A) | is.na(r_B), 1, 10^(r_A/400) / (10^(r_A/40
       0) + 10^(r_B/400)))))
17   return(l_prob)
18 }
19 elo_cv_l = function(kfac){
20   K <- 10
21   matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')
       ]
22   data_partitions <- split(matches, seq(K))
23
24   num_tournaments <- length(unique(matches$tourney_order))
25   rating_periods <- num_tournaments %/% 4
26   matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_
       order) %/% rating_periods) + 1)[1:length(matches$tourney_order)]
27
28   total_l_prob <- 0
29   for(i in seq(K)){
30     test_data <- data_partitions[i]
31     training_data <- dplyr::bind_rows(data_partitions[-i])
32     total_l_prob <- total_l_prob + elolikelihood(training_data, test_data, kfac=
         kfac)
33   }
34   l_prob <- total_l_prob / K
35   return(-l_prob)
```

```
}

## Optimise CV likelihood
res <- optim(par=c(32), elo_cv_l, method="Nelder-Mead", upper=c(800), lower=c(1))
res
```

Glicko σ and ν

The following calculates estimates for the initial rating deviation σ and rating deviation increase per unit time ν according to the method of minimising the total predictive discrepancy by Glickman (1999). We make use of the Nelder-Mead simplex algorithm as suggested. This yields the aforementioned $\hat{\sigma} = 350$ and $\hat{\nu} = 254$.

```
# Parameter estimation
## Likelihood function
glicko_l = function(matches, status, rating=2200, sigma=300, nu=15) {
  data_df <- matches[c('tourney_order', 'winner_player_id', 'loser_player_id')]
  data_df$win <- rep(1, length(data_df[,1]))
  glicko_res <- glicko(data_df, status=status, init=c(rating, sigma), cval=nu, as_data_frame=TRUE, sort=FALSE)

  mu_i <- glicko_res[[1]]$Rating[match(data_df$winner_player_id, glicko_res[[1]]$Player)]
  mu_j <- glicko_res[[1]]$Rating[match(data_df$loser_player_id, glicko_res[[1]]$Player)]
  sigma_i <- glicko_res[[1]]$Deviation[match(data_df$winner_player_id, glicko_res[[1]]$Player)]
  sigma_j <- glicko_res[[1]]$Deviation[match(data_df$loser_player_id, glicko_res[[1]]$Player)]

  q <- log(10) / 400
  g_RD <- 1 / sqrt(1 + 3 * q^2 * (sigma_i^2 + sigma_j^2) / pi^2)
  p_ij <- 1 / (1 + 10 ^ (- g_RD * (mu_i - mu_j) / 400))

  l_prob <- -sum(log(p_ij))
  return(list(l_prob, glicko_res))
}

## Optim function argument
glicko_param_l = function(param) {
  sigma <- param[1]
  nu <- param[2]
  tournaments_per_period <- 4
  tournaments <- unique(match_data$tourney_order)
  partitions <- split(tournaments, seq(length(tournaments) %/% tournaments_per_period))

  log_l <- 0
  status <- NULL
  for (i in seq(length(partitions))) {
    matches_in_period <- match_data[match_data$tourney_order %in% partitions[[i]],]
    matches_in_period$tourney_order <- i
    res <- glicko_l(matches=matches_in_period, status=status, sigma=sigma, nu=nu)
    log_l <- log_l + res[[1]]
    status <- res[[2]][["ratings"]]
  }
  return(log_l)
}
```

```
40
41 ## Optimise
42 res <- optim(par=c(300, 15), glicko_param_l, method="Nelder-Mead", upper=c(350, 1
       000), lower=c(1, 1))
43 res
```

The following produces the estimates used in fitting the Glicko model as in Table 2.5 and results in chapter 4 by maximising the full log-likelihood in a 10-fold cross-validation procedure. This yields the utilised $\hat{\sigma} = 146.0352$ and $\hat{\nu} = 13.3277$.

```
1  ## Cross validation likelihood
2  glickolikelihood = function(training_data, test_data, sigma, nu, use_glicko2=
       FALSE, starting=2200, volatility=0.15, tau=1.2, rdmax=350){
3    data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
       id')]
4    data_df$win <- rep(1, length(data_df[,1]))
5
6    num_tournaments <- length(unique(data_df$tourney_order))
7    rating_periods <- num_tournaments %/% 4
8    data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
       order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]
9
10   glicko_res <- ifelse(use_glicko2, glicko2(data_df, init=c(starting, sigma,
       volatility), tau=tau), glicko(data_df, init=c(starting, sigma), cval=nu,
       rdmax=rdmax, as_data_frame=TRUE))
11
12   r_A <- glicko_res[[1]]$Rating[match(test_data[[1]]$winner_player_id, glicko_res
       [[1]]$Player)]
13   r_B <- glicko_res[[1]]$Rating[match(test_data[[1]]$loser_player_id, glicko_res
       [[1]]$Player)]
14   RD_B <- glicko_res[[1]]$Deviation[match(test_data[[1]]$loser_player_id, glicko_
       res[[1]]$Player)]
15
16   q <- log(10) / 400
17   g_RD <- 1 / sqrt(1 + 3 * q^2 * RD_B^2 / pi^2)
18   E_A <- 1 / (1 + 10 ^ (- g_RD * (r_A - r_B) / 400))
19
20   l_prob <- sum(log(ifelse(is.na(r_A) | is.na(r_B), 1, E_A)))
21   return(l_prob)
22 }
23 glicko_cv_l = function(sigma, nu){
24   K <- 10
25   matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')
       ]
26   data_partitions <- split(matches, seq(K))
27
28   num_tournaments <- length(unique(matches$tourney_order))
29   rating_periods <- num_tournaments %/% 4
30   matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_
       order) %/% rating_periods) + 1)[1:length(matches$tourney_order)]
31
32   total_l_prob <- 0
33   for(i in seq(K)){
34     test_data <- data_partitions[i]
35     training_data <- dplyr::bind_rows(data_partitions[-i])
36     total_l_prob <- total_l_prob + glickolikelihood(training_data, test_data,
         sigma, nu)
37   }
38   l_prob <- total_l_prob / K
39   return(l_prob)
```

```
}
glicko_cv_param_l = function(param) {
  return(-glicko_cv_l(param[1], param[2]))
}

## Optimise CV likelihood
res <- optim(par=c(300, 15), glicko_cv_param_l, method="Nelder-Mead", upper=c(350
    , 1000), lower=c(1, 1))
res
```

Log-likelihood Heat Map for Glicko Parameters

The following produces the heat map of the log-likelihood as parameters are varied in the Glicko model as in Figure 2.5.

```
# Visualise heat map
## Populate matrix
s_count <- 20
n_count <- 20
sigmas <- seq(50, 300, length=s_count)
nus <- seq(1, 30, length=n_count)

nu_sigma_matrix <- matrix(0, nrow=n_count, ncol=s_count)
for (i in seq(s_count)){
  sigma <- sigmas[i]
  for (j in seq(n_count)){
    nu <- nus[j]
    nu_sigma_matrix[i, j] <- -glicko_cv_param_l(c(sigma, nu))
  }
  print(paste(100 * (i / s_count), '%'))
}

## Create heat map
library('latex2exp')
library('ggplot2')
library('viridis')
vals <- data.frame(x=rep(nus, each=s_count), y=rep(sigmas, length(n_count)), z=as
    .numeric(nu_sigma_matrix))
colnames(vals)[3] <- "Log-likelihood"

my_breaks <- c(-1490, -1495, -1496)
v <- ggplot(vals, aes(x, y, z=`Log-likelihood`)) + geom_raster(aes(fill=`Log-
    likelihood`)) + xlab(TeX('Rating␣Deviation␣Increase␣Over␣Time␣$(\\nu)$')) +
    ylab(TeX('Starting␣Deviation␣$(\\sigma_{0})$')) + scale_fill_continuous(type
    = "viridis")
v
```

5.3 Fickle Fan Model

5.3.1 Fitting the Model

The following sets up the transition matrix of the Markov chain as described and runs either a random walk or a simulation of assigning 1000 players at each state. This produces the rankings and proportion of fans at Table 2.7.

```
# Setup
## Load data as previously
## Actual ranking
```

```r
actual_ranking <- ranking_data[c('player_slug', 'ranking_points')]

# Tally for wins and losses
## Get top players
top_player_ids <- ranking_data$player_id[1:10]
num_players <- length(top_player_ids)

## Filter out only relevant matches
match_data <- match_data[match_data$winner_player_id %in% top_player_ids & match_data$loser_player_id %in% top_player_ids,]
matches <- match_data[,c('winner_player_id', 'loser_player_id')]

## Tally number of games won/lost
tally <- dplyr::count(matches, winner_player_id, loser_player_id)

top_player_ids <- sort(unique(c(as.character(tally$winner_player_id), as.character(tally$loser_player_id))))
#num_players <- length(top_player_ids)

tally_matrix <- matrix(0, nrow=num_players, ncol=num_players, dimnames = list(top_player_ids,top_player_ids))
tally_matrix[as.matrix(tally[,c('winner_player_id', 'loser_player_id')])] <- tally[,'n']

## Get Transition Matrix
prop <- 0.01
games_lost <- colSums(tally_matrix)
  games_won <- rowSums(tally_matrix)
  prob_win <- games_won / (games_won + games_lost)
  trans_matrix <- diag(prob_win) + t(tally_matrix) * (1 - prob_win) / games_lost

  for (i in seq(num_players)) {
    if (sum(is.na(trans_matrix)) > 0) {
      trans_matrix[i,] <- rep(1 / num_players, num_players)
    } else {
      num_havent_lost <- sum(trans_matrix[i,] == 0)
      prob_arb_move <- sum(trans_matrix[i,]) * prop / num_havent_lost
      trans_matrix[i,][trans_matrix[i,] == 0] <- prob_arb_move
      trans_matrix[i,] <- trans_matrix[i,] / sum(trans_matrix[i,])
    }
  }

## Run simulation using random walk of one fan
iter_count <- 1
visit_counts <- table(colnames=top_player_ids) - 1
curr_state <- sample(top_player_ids, 1)
population <- table(colnames=top_player_ids) - 1

for(i in seq(iter_count)){
  next_state <- sample(colnames(trans_matrix), prob=trans_matrix[,curr_state], size=1, replace=TRUE)
  visit_counts[next_state] <- visit_counts[next_state] + 1
}

# Display results
distrib_vector <- as.vector(visit_counts)
order <- order(distrib_vector, decreasing=TRUE)
fan_count <- data.frame(top_player_ids[order], distrib_vector[order])
colnames(fan_count) <- c('colnames', 'Freq')
```

```
player_slugs <- player_data$player_slug[match(fan_count$colnames, player_data$
    player_id)]

pred_ranking <- as.data.frame(cbind(player_slugs, fan_count$Freq, fan_count$Freq
    / sum(population)))
colnames(pred_ranking) <- c('player_slug', 'fans', '% of fans')
print(pred_ranking)

## Run simulation of 1000 players at each state
fans_per_player <- 1000
iter_count <- 10000

# Assign people to each person
population <- table(colnames=top_player_ids) - 1 + fans_per_player
counter <- 0
for(i in seq(iter_count)){
  # Set table to 0
  movement <- table(colnames=top_player_ids) - 1

  for(player in seq(num_players)){
    num_fans <- population[[player]]
    new_player <- sample(colnames(trans_matrix), prob=trans_matrix[player,], size
        =num_fans, replace=TRUE)
    # Tally how many people move to which player
    movement <- movement + table(c(new_player, top_player_ids)) - 1
  }

  # Move the people around
  population <- movement
  if (i %% 1000 == 0) {
    counter <- counter + 1
    print(paste(10 * counter, '%'))
  }
}

# Display results
fan_count <- as.data.frame(sort(population, decreasing=TRUE))
player_slugs <- player_data$player_slug[match(fan_count$colnames, player_data$
    player_id)]

tpred_ranking <- as.data.frame(cbind(player_slugs, fan_count$Freq, fan_count$Freq
     / sum(population)))
colnames(tpred_ranking) <- c('player_slug', 'fans', '% of fans')
print(tpred_ranking)

## Get Steady state
library('expm')
steady_state <- (trans_matrix %^% 1000)[1,]
```

5.3.2 Visualising States Graph

The following creates an adjacency list used to encode the Markov state diagram in Figure 2.6. Unused is the a state diagram created in R.

```
# Get latex adjacency lists
top_matrix <- trans_matrix

output <- "{"
for(row in seq(length(top_matrix[,1]))){
```

```
  row_str <- paste("{", format(round(top_matrix[1, row], 3), nsmall=2))
  for (col in seq(length(top_matrix[row,]) - 1)){
    row_str <- paste(row_str, ",", format(round(top_matrix[col+1,row], 3), nsmall
        =3))
  }
  output <- paste(output, row_str, "},")
}
output <- paste(output, "}")

# Plot Markov state graph in R
## Libraries
library('igraph')
library('ggplot2')
library('viridis')

## Plotting the graph
n <- num_players
g <- graph_from_adjacency_matrix(tally_matrix + diag(n), weighted=TRUE, diag=TRUE
    )
la <- layout.circle(g)

col_n <- 12
colors <- viridis(col_n)

min_vdeg <- min(degree(g))
max_vdeg <- max(degree(g))
V(g)$color <- colors[as.integer(col_n * (degree(g) - min_vdeg)/(max_vdeg - min_
    vdeg))]

min_edeg <- min(E(g)$weight)
max_edeg <- max(E(g)$weight)
E(g)$color <- colors[as.integer(col_n * (E(g)$weight - min_edeg)/(max_edeg - min_
    edeg))]

plot.igraph(g, layout=la,vertex.size=3, vertex.label=NA, edge.arrow.size=0.1,
    edge.width=0.1)
```

5.3.3 Optimising the proportionality constant for ghost links

Briefly, we mentioned the use of an arbitrarily set proportionality constant $p = 0.01$ in deriving the transition probabilities. An attempt to optimise such parameter is given below, though it is noted that it is computationally expensive and was therefore unused.

```
# Setup
## Load cross-validation functions (one including the markovlikelihood function)

# Best Markov Proportionality Constant
markov_cv_l = function(prop){
  K <- 10
  matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')
      ]
  data_partitions <- split(matches, seq(K))

  num_tournaments <- length(unique(matches$tourney_order))
  rating_periods <- num_tournaments %/% 4
  matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_
      order) %/% rating_periods) + 1)[1:length(matches$tourney_order)]

  total_l_prob <- 0
```

```
  for(i in seq(K)){
    test_data <- data_partitions[i]
    training_data <- dplyr::bind_rows(data_partitions[-i])
    total_l_prob <- total_l_prob + markovlikelihood(training_data, test_data,
        prop=prop, iter=20)
  }
  l_prob <- total_l_prob / K
  return(l_prob)
}

## Optimise CV likelihood
res <- optim(par=c(0.01), markov_cv_l, method="BFGS", upper=c(1), lower=c(0))
res
```

5.4 Model Comparisons

5.4.1 Spearman's Rank Correlation Coefficient

The following produces the Spearman's rank correlation coefficient results of each model as in Table 4.1. We note that we vary the parameter N at the start to consider coefficients for sample sizes: ten and 100. Another note is that the Bayesian extensions to the BT model were tested by simply calling the helper functions after their definitions in a separate file.

```
# Setup
## Load data as previously
## Retrieve the actual rankings
actual_ranking <- ranking_data[c('player_id', 'player_slug', 'rank_number')]

# Specify how many players to consider
## Pre-process matches
N <- 10
matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')]
active_players <- unique(c(matches$winner_player_id, matches$loser_player_id))
top_player_ids <- ranking_data[ranking_data$player_id %in% active_players,]$
    player_id[1:N]
top_matches <- matches[matches$winner_player_id %in% top_player_ids & matches$
    loser_player_id %in% top_player_ids,]

# Functions for getting rankings
## Bradley-Terry
library('BradleyTerryScalable')
btranking = function(training_data, a=1.1){
  win_lose_tally <- dplyr::count(training_data, winner_player_id, loser_player_id
      )
  # Fit the BT model
  test_btdata <- btdata(win_lose_tally)
  test_btfit <- btfit(test_btdata, a=a)
  test_id_coef_df <- coef(test_btfit, as_df=TRUE)[c('item', 'coef')]
  return(test_id_coef_df$item)
}

## Elo
library('PlayerRatings')
eloranking = function(training_data, kfac=31.41){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]
  data_df$win <- rep(1, length(data_df[,1]))

```

```
  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  elo_res <- elo(data_df, kfac=kfac, as_data_frame=TRUE)
  return(elo_res[[1]]$Player)
}

## Glicko
library('PlayerRatings')
glickoranking = function(training_data, sigma=146.035, nu= 13.328, starting=2200,
     rdmax=350){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]
  data_df$win <- rep(1, length(data_df[,1]))

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  glicko_res <- glicko(data_df, init=c(starting, sigma), cval=nu, rdmax=rdmax, as
      _data_frame=TRUE)
  return(glicko_res[[1]]$Player)
}

## Glicko2
library('PlayerRatings')
glicko2ranking = function(training_data, sigma=151.27, vol=0.075, tau=0.3,
    starting=2200, rdmax=350){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  data_df$win <- rep(1, length(data_df[,1]))
  glicko_res <- glicko2(data_df, init=c(starting, sigma, vol), tau=tau, rdmax=
      rdmax, as_data_frame=TRUE)
  return(glicko_res[[1]]$Player)
}

## Markov
library('expm')
markovranking = function(training_data, prop=0.01){
  # Tally number of games won/lost
  tally <- dplyr::count(training_data, winner_player_id, loser_player_id)

  player_ids <- sort(unique(c(as.character(tally$winner_player_id), as.character(
      tally$loser_player_id))))
  num_players <- length(player_ids)

  tally_matrix <- matrix(0, nrow=num_players, ncol=num_players, dimnames = list(
      player_ids, player_ids))
  tally_matrix[as.matrix(tally[,c('winner_player_id', 'loser_player_id')])] <-
      tally[,'n']
```

```
79
80    # Get transition matrix
81    games_lost <- colSums(tally_matrix)
82    games_won <- rowSums(tally_matrix)
83    prob_win <- games_won / (games_won + games_lost)
84    trans_matrix <- diag(prob_win) + t(tally_matrix) * (1 - prob_win) / games_lost
85
86    for (i in seq(num_players)) {
87      if (sum(is.na(trans_matrix)) > 0) {
88        trans_matrix[i,] <- rep(1 / num_players, num_players)
89      } else {
90        num_havent_lost <- sum(trans_matrix[i,] == 0)
91        prob_arb_move <- sum(trans_matrix[i,]) * prop / num_havent_lost
92        trans_matrix[i,][trans_matrix[i,] == 0] <- prob_arb_move
93        trans_matrix[i,] <- trans_matrix[i,] / sum(trans_matrix[i,])
94      }
95    }
96
97    # Calculate steady state
98    steady_state <- (trans_matrix %^% 1000)[1,]
99
100   return(player_ids[order(steady_state, decreasing=TRUE)])
101 }
102
103 # Spearman's Rank Coefficient
104 spearmans = function(actual, pred){
105   n <- length(pred)
106   diff_ranks <- c()
107   for(i in seq(n)){
108     player_id <- pred[i]
109     rank <- which(actual == player_id)
110     diff_ranks <- c(diff_ranks, rank - i)
111   }
112   diff_sq <- sum(diff_ranks^2)
113
114   p_coeff <- 1 - 6 * diff_sq / (n * (n^2 - 1))
115   return(p_coeff)
116 }
117
118 comparison_methods = function(actual, ranking, rank_fn){
119   pred <- rank_fn(actual)
120   # Spearmans
121   rho <- spearmans(ranking, pred)
122   n <- length(ranking)
123   rho_p_val <- 2 * pt(q=rho * sqrt(n - 2)/sqrt(1 - rho^2), df = n - 2, lower.tail
        = FALSE)
124   paste("Spearmans:", rho, "p-value:", rho_p_val)
125 }
126
127 print_spearman_results = function(ranking, pred){
128   rho <- spearmans(ranking, pred)
129   n <- length(ranking)
130   rho_p_val <- 2 * pt(q=rho * sqrt(n - 2)/sqrt(1 - rho^2), df = n - 2, lower.tail
        = FALSE)
131   paste("Spearmans:", rho, "p-value:", rho_p_val)
132 }
133
134 rel_matches <- top_matches
135 rel_ranking <- top_player_ids
```

```r
comparison_methods(rel_matches, rel_ranking, rank_fn=btranking)
comparison_methods(rel_matches, rel_ranking, rank_fn=eloranking)
comparison_methods(rel_matches, rel_ranking, rank_fn=glickoranking)
comparison_methods(rel_matches, rel_ranking, rank_fn=glicko2ranking)
comparison_methods(rel_matches, rel_ranking, rank_fn=markovranking)
```

5.4.2 Brier Score

The following produces the Brier score results of each model as in Table 4.2. A note once again is that the Bayesian extension to the BT model with different prior distributions were tested by calling the helper functions written below in their respective file.

```r
# Functions for the likelihood of matches
## Bradley-Terry
library('BradleyTerryScalable')
bt_indv_likelihood = function(training_data, test_data, a=1.5){
  win_lose_tally <- dplyr::count(training_data, winner_player_id, loser_player_id)
  # Fit the BT model
  test_btdata <- btdata(win_lose_tally)
  test_btfit <- btfit(test_btdata, a=a)
  test_id_coef_df <- coef(test_btfit, as_df=TRUE)[c('item', 'coef')]

  ## Evaluate probability
  p_i <- exp(test_id_coef_df$coef[match(test_data$winner_player_id, test_id_coef_df$item)])
  p_j <- exp(test_id_coef_df$coef[match(test_data$loser_player_id, test_id_coef_df$item)])

  l_prob <- ifelse(is.na(p_j), 0.5, p_i / (p_i + p_j))
  return(l_prob)
}

## Elo
library('PlayerRatings')
elo_indv_likelihood = function(training_data, test_data, kfac=27){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_id')]
  data_df$win <- rep(1, length(data_df[,1]))

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  elo_res <- elo(data_df, kfac=kfac, as_data_frame=TRUE)
  r_A <- elo_res[[1]]$Rating[match(test_data$winner_player_id, elo_res[[1]]$Player)]
  r_B <- elo_res[[1]]$Rating[match(test_data$loser_player_id, elo_res[[1]]$Player)]

  l_prob <- ifelse(is.na(r_A) | is.na(r_B), 0.5, 10^(r_A/400) / (10^(r_A/400) + 10^(r_B/400)))
  return(l_prob)
}

## Glicko
library('PlayerRatings')
```

```r
glicko_indv_likelihood = function(training_data, test_data, use_glicko2=FALSE,
    sigma=300, nu=15, starting=2200, rdmax=350, vol=0.15, tau=1.5){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]
  data_df$win <- rep(1, length(data_df[,1]))

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  glicko_res <- ifelse(use_glicko2, glicko2(data_df, init=c(starting, sigma, vol)
      , tau=tau, rdmax=rdmax, as_data_frame=TRUE), glicko(data_df, init=c(
      starting, sigma), cval=nu, rdmax=rdmax, as_data_frame=TRUE))

  r_A <- glicko_res[[1]]$Rating[match(test_data$winner_player_id, glicko_res[[1
      ]]$Player)]
  r_B <- glicko_res[[1]]$Rating[match(test_data$loser_player_id, glicko_res[[1]]$
      Player)]
  RD_B <- glicko_res[[1]]$Deviation[match(test_data$loser_player_id, glicko_res[[
      1]]$Player)]

  q <- log(10) / 400
  g_RD <- 1 / sqrt(1 + 3 * q^2 * RD_B^2 / pi^2)
  E_A <- 1 / (1 + 10 ^ (- g_RD * (r_A - r_B) / 400))

  l_prob <- (ifelse(is.na(r_A) | is.na(r_B), 0.5, E_A))
  return(l_prob)
}

## Markov
library('expm')
markov_indv_likelihood = function(training_data, test_data, rank_data){
  # Tally number of games won/lost
  tally <- dplyr::count(training_data, winner_player_id, loser_player_id)

  player_ids <- sort(unique(c(as.character(tally$winner_player_id), as.character(
      tally$loser_player_id))))
  num_players <- length(player_ids)

  tally_matrix <- matrix(0, nrow=num_players, ncol=num_players, dimnames = list(
      player_ids, player_ids))
  tally_matrix[as.matrix(tally[,c('winner_player_id', 'loser_player_id')])] <-
      tally[,'n']

  # Get transition matrix
  games_lost <- colSums(tally_matrix)
  games_won <- rowSums(tally_matrix)
  prob_win <- games_won / (games_won + games_lost)
  trans_matrix <- diag(prob_win) + t(tally_matrix) * (1 - prob_win) / games_lost

  for (i in seq(num_players)) {
    if (sum(is.na(trans_matrix)) > 0) {
      trans_matrix[i,] <- rep(1 / num_players, num_players)
    } else {
      num_havent_lost <- sum(trans_matrix[i,] == 0)
      prob_arb_move <- sum(trans_matrix[i,]) * prop / num_havent_lost
      trans_matrix[i,][trans_matrix[i,] == 0] <- prob_arb_move
      trans_matrix[i,] <- trans_matrix[i,] / sum(trans_matrix[i,])
```

```
    }
  }

  # Calculate steady state
  steady_state <- (trans_matrix %^% 1000)[1,]

  # Calculate likelihood
  test_matches <- test_data[test_data$winner_player_id %in% player_ids & test_
      data$loser_player_id %in% player_ids,][,c('winner_player_id', 'loser_player
      _id')]

  winner_fan_pct <- steady_state[match(test_matches$winner_player_id, player_ids)
      ]
  loser_fan_pct <- steady_state[match(test_matches$loser_player_id, player_ids)]

  likelihood <- ifelse(winner_fan_pct + loser_fan_pct == 0, 0, winner_fan_pct / (
      winner_fan_pct + loser_fan_pct))
  return(likelihood)
}

# Calculate the Brier score
matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')]

models <- c("Bradley-Terry", "Elo", "Glicko", "Glicko2", "Markov")

bt_l <- bt_indv_likelihood(matches, matches)
elo_l <- elo_indv_likelihood(matches, matches, kfac=31.41)
glicko_l <- glicko_indv_likelihood(matches, matches, sigma=146.03516, nu=13.32772
    )
glicko2_l <- glicko_indv_likelihood(matches, matches, sigma=151.2749, vol= 0.0749
    , tau=0.3, use_glicko2 = TRUE)
markov_l <- markov_indv_likelihood(matches, matches, ranking_data)

bt_brier <- mean((bt_l - 1)^2)
elo_brier <- mean((elo_l - 1)^2)
glicko_brier <- mean((glicko_l - 1)^2)
glicko2_brier <- mean((glicko2_l - 1)^2)
markov_brier <- mean((markov_l - 1)^2)

brier_scores <- c(bt_brier, elo_brier, glicko_brier, glicko2_brier, markov_brier)
print(cbind(models, brier_scores))
```

5.4.3 Cross-Validation

The following produces the 10-fold cross-validation results of each model as in Table 4.3. Again, Bayesian extensions call a helper function in their own respective file.

```
# Setup
## Load data as previously
## Libraries
library('BradleyTerryScalable')
library('PlayerRatings')


# Likelihood functions
## Bradley-Terry
library('BradleyTerryScalable')
btlikelihood = function(training_data, test_data){
```

```
  win_lose_tally <- dplyr::count(training_data, winner_player_id, loser_player_id
      )
  # Fit the BT model
  test_btdata <- btdata(win_lose_tally)
  test_btfit <- btfit(test_btdata, a=1.5)
  test_id_coef_df <- coef(test_btfit, as_df=TRUE)[c('item', 'coef')]

  ## Evaluate probability
  p_i <- exp(test_id_coef_df$coef[match(test_data[[1]]$winner_player_id, test_id_
      coef_df$item)])
  p_j <- exp(test_id_coef_df$coef[match(test_data[[1]]$loser_player_id, test_id_
      coef_df$item)])

  l_prob <- sum(log(ifelse(is.na(p_j), 1, p_i / (p_i + p_j))))
  return(l_prob)
}

## Elo
library('PlayerRatings')
elolikelihood = function(training_data, test_data, kfac=27){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]
  data_df$win <- rep(1, length(data_df[,1]))

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  elo_res <- elo(data_df, kfac=kfac, as_data_frame=TRUE)
  r_A <- elo_res[[1]]$Rating[match(test_data[[1]]$winner_player_id, elo_res[[1]]$
      Player)]
  r_B <- elo_res[[1]]$Rating[match(test_data[[1]]$loser_player_id, elo_res[[1]]$
      Player)]

  l_prob <- sum(log(ifelse(is.na(r_A) | is.na(r_B), 1, 10^(r_A/400) / (10^(r_A/
      400) + 10^(r_B/400)))))
  return(l_prob)
}

## Glicko
library('PlayerRatings')
glickolikelihood = function(training_data, test_data, sigma=300, nu=15, use_
    glicko2=FALSE, starting=2200, volatility=0.15, tau=1.2, rdmax=350){
  data_df <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_
      id')]
  data_df$win <- rep(1, length(data_df[,1]))

  num_tournaments <- length(unique(data_df$tourney_order))
  rating_periods <- num_tournaments %/% 4
  data_df$tourney_order <- rep(1:rating_periods, each=(length(data_df$tourney_
      order) %/% rating_periods) + 1)[1:length(data_df$tourney_order)]

  glicko_res <- ifelse(use_glicko2, glicko2(data_df, init=c(starting, sigma,
      volatility), tau=tau), glicko(data_df, init=c(starting, sigma), cval=nu,
      rdmax=rdmax, as_data_frame=TRUE))

  r_A <- glicko_res[[1]]$Rating[match(test_data[[1]]$winner_player_id, glicko_res
      [[1]]$Player)]
```

```
  r_B <- glicko_res[[1]]$Rating[match(test_data[[1]]$loser_player_id, glicko_res
      [[1]]$Player)]
  RD_B <- glicko_res[[1]]$Deviation[match(test_data[[1]]$loser_player_id, glicko_
      res[[1]]$Player)]

  q <- log(10) / 400
  g_RD <- 1 / sqrt(1 + 3 * q^2 * RD_B^2 / pi^2)
  E_A <- 1 / (1 + 10 ^ (- g_RD * (r_A - r_B) / 400))

  l_prob <- sum(log(ifelse(is.na(r_A) | is.na(r_B), 1, E_A)))
  return(l_prob)
}

## Markov Chain
library('expm')
markovlikelihood = function(training_data, test_data, prop=0.01, iter=1000){
  # Tally number of games won/lost
  tally <- dplyr::count(training_data, winner_player_id, loser_player_id)

  player_ids <- sort(unique(c(as.character(tally$winner_player_id), as.character(
      tally$loser_player_id))))
  num_players <- length(player_ids)

  tally_matrix <- matrix(0, nrow=num_players, ncol=num_players, dimnames = list(
      player_ids, player_ids))
  tally_matrix[as.matrix(tally[,c('winner_player_id', 'loser_player_id')])] <-
      tally[,'n']

  # Get transition matrix
  games_lost <- colSums(tally_matrix)
  games_won <- rowSums(tally_matrix)
  prob_win <- games_won / (games_won + games_lost)
  trans_matrix <- diag(prob_win) + t(tally_matrix) * (1 - prob_win) / games_lost

  for (i in seq(num_players)) {
    if (sum(is.na(trans_matrix)) > 0) {
      trans_matrix[i,] <- rep(1 / num_players, num_players)
    } else {
      num_havent_lost <- sum(trans_matrix[i,] == 0)
      prob_arb_move <- sum(trans_matrix[i,]) * prop / num_havent_lost
      trans_matrix[i,][trans_matrix[i,] == 0] <- prob_arb_move
      trans_matrix[i,] <- trans_matrix[i,] / sum(trans_matrix[i,])
    }
  }

  # Calculate steady state
  steady_state <- (trans_matrix %^% iter)[1,]

  # Calculate likelihood
  winner_fan_pct <- steady_state[match(test_data[[1]]$winner_player_id, player_
      ids)]
  loser_fan_pct <- steady_state[match(test_data[[1]]$loser_player_id, player_ids)
      ]

  likelihood <- ifelse(is.na(winner_fan_pct) | is.na(loser_fan_pct), 1, winner_
      fan_pct / (winner_fan_pct + loser_fan_pct))
  return(sum(log(likelihood)))
}

```

```r
## Dynamic Elo
dynamic_elolikelihood = function(training_data, test_data, kfac=27, start_rating=2200){
  matches <- training_data[c('tourney_order', 'winner_player_id', 'loser_player_id')]

  all_players <- unique(c(matches$winner_player_id, matches$loser_player_id))
  num_players <- length(all_players)
  ratings <- rep(start_rating, num_players)

  total_games_played <- length(matches[,1])
  matches$winner_games_played <- rep(0, total_games_played)
  matches$loser_games_played <- rep(0, total_games_played)

  for (player in all_players){
    relevant_matches <- matches[matches$winner_player_id == player | matches$loser_player_id == player,]
    num_games_played <- length(relevant_matches[,1])

    matches$winner_games_played <- ifelse(matches$winner_player_id == player, seq(1, num_games_played), matches$winner_games_played)
    matches$loser_games_played <- ifelse(matches$loser_player_id == player, seq(1, num_games_played), matches$loser_games_played)
  }

  num_tournaments <- length(unique(matches$tourney_order))
  rating_periods <- num_tournaments %/% 4
  matches$tourney_order <- rep(1:rating_periods, each=(length(matches$tourney_order) %/% rating_periods) + 1)[1:length(matches$tourney_order)]

  for (i in seq(rating_periods)) {
    matches_in_period <- matches[matches$tourney_order == i,]
    updates <- as.data.frame(cbind(all_players, rep(0, length(all_players))))
    updates[,2] <- sapply(updates[,2], as.numeric)
    colnames(updates) <- c('player_id', 'score_change')
    for (i in seq(length(matches_in_period[,1]))){
      row <- matches_in_period[i,]
      A <- row$winner_player_id
      B <- row$loser_player_id

      R_A <- ratings[which(all_players == A)]
      R_B <- ratings[which(all_players == B)]

      E_A <- 1 / (1 + 10^((R_B - R_A) / 400))
      E_B <- 1 - E_A

      K_A <- kfac / (row$winner_games_played + 5) ^ 0.4
      K_B <- kfac / (row$loser_games_played + 5) ^ 0.4

      updates$score_change[updates$player_id == A] <- updates$score_change[updates$player_id == A] + K_A * (1 - E_A)
      updates$score_change[updates$player_id == B] <- updates$score_change[updates$player_id == B] + K_B * (0 - E_B)
    }
    ratings <- ratings + updates[match(all_players, updates$player_id),]$score_change
  }

  r_A <- ratings[match(test_data[[1]]$winner_player_id, all_players)]
```

```
  r_B <- ratings[match(test_data[[1]]$loser_player_id, all_players)]

  l_prob <- sum(log(ifelse(is.na(r_A) | is.na(r_B), 1, 10^(r_A/400) / (10^(r_A/
      400) + 10^(r_B/400)))))
  return(l_prob)
}

# Cross-validation
## Partition match data into K parts
K <- 10
matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')]
data_partitions <- split(matches, seq(K))

## Train and test against partitions
names <- c('Bradley-Terry', 'Elo', 'Glicko', 'Glicko2', 'Dynamic␣Elo', 'Markov')

total_l_prob <- rep(0, 6)
for(i in seq(K)){
  test_data <- data_partitions[i]
  training_data <- dplyr::bind_rows(data_partitions[-i])

  total_l_prob[1] <- total_l_prob[1] + btlikelihood(training_data, test_data)
  total_l_prob[2] <- total_l_prob[2] + elolikelihood(training_data, test_data,
      kfac=31.41)
  total_l_prob[3] <- total_l_prob[3] + glickolikelihood(training_data, test_data,
       sigma=146.035, nu=13.328)
  total_l_prob[4] <- total_l_prob[4] + glickolikelihood(training_data, test_data,
       use_glicko2=FALSE, sigma=151.275, volatility=0.074, tau=0.3)
  total_l_prob[5] <- total_l_prob[5] + dynamic_elolikelihood(training_data, test_
      data, kfac=250)

  total_l_prob[6] <- total_l_prob[6] + markovlikelihood(training_data, test_data)
}
l_prob <- total_l_prob / K
res_df <- as.data.frame(cbind(names, l_prob))
colnames(res_df) <- c('Model', 'Log-likelihood')
print(res_df)
```

5.5 Bayesian Extensions to the Bradley-Terry Model

5.5.1 Calculating Rankings

The following produces the rankings derived from using each of the priors discussed as in Table 3.2, Table 3.3, and Table 3.4.

```
# Setup
## Load data as previously

# Only get top N players
N <- 100
top_players <- ranking_data$player_id[1:N]
match_data <- match_data[match_data$winner_player_id %in% top_players & match_
    data$loser_player_id %in% top_players,]

# Assign Starting Ratings
players <- as.vector(sort(unique(c(match_data$winner_player_id, match_data$loser_
    player_id))))
num_players <- length(players)
```

```r
ratings <- rep(1.000, num_players)
names(ratings) <- players

# Get tally of match sets
win_lose_tally <- dplyr::count(match_data, winner_player_id, loser_player_id)

# Log likelihood function
bt_l = function(ratings){
  winner_r <- ratings[match(win_lose_tally$winner_player_id, players)]
  loser_r <- ratings[match(win_lose_tally$loser_player_id, players)]
  return(sum(log(1 / (1 + exp(loser_r - winner_r)))))
}

## Prior Distributions
### Gamma prior
gamma_prior = function(a=2, low=0, up=10){
  density = function(par) {
    return(prod(dgamma(par, shape=a)))
  }
  sampler = function(n=N) {
    return(rgamma(n, shape=a))
  }
  prior <- createPrior(density=density, sampler=sampler, lower=rep(low,N), upper=
      rep(up,N))
  return(prior)
}

### Beta Prior
beta_prior = function(a=1.5, b=1, low=0, up=10) {
  prior <- createBetaPrior(a=a, b=b, lower = rep(low, N), upper = rep(up, N))
  return(prior)
}

### Normal Prior
normal_prior = function(mu=5, sigma=1, low=0, up=10) {
  density = function(par) {
    return(prod(dnorm(par, mean=mu, sd=sigma)))
  }
  sampler = function(n=N) {
    return(rnorm(n, mean=mu, sd=sigma))
  }
  prior <- createPrior(density=density, sampler=sampler, lower=rep(low,N), upper=
      rep(up,N))
  return(prior)
}

### Uniform Prior
uniform_prior = function(low=0, up=10) {
  prior <- createUniformPrior(lower = rep(low, N), upper = rep(up, N))
  return(prior)
}

## Setup MCMC
library('BayesianTools')

## MCMC Helper
get_mcmc_ratings = function(prior) {
  bayesian_setup <- createBayesianSetup(likelihood=bt_l, names=names(ratings),
      prior=prior)
```

```r

  # Start MCMC
  settings = list(iterations = 10000, nrChains=1, burnin=500)
  bt_mcmc <- runMCMC(bayesianSetup = bayesian_setup,
                     sampler = "Metropolis",
                     settings = settings)

  # Extract MAP
  ratings_mcmc <- MAP(bt_mcmc)
  ratings_mcmc_final <- ratings_mcmc$parametersMAP
  names(ratings_mcmc_final) <- player_data$player_id[match(players, player_data$
      player_id)]
  ratings_output <- sort(ratings_mcmc_final, decreasing=TRUE)
  return(ratings_output)
}

low <- 0
up <- 10
priors <- list(gamma_prior(a=2, low=low, up=up), normal_prior(mu=5, low=low, up=
    up), uniform_prior(low=low, up=up))
prior_ratings <- list()
for (prior in priors) {
  prior_ratings <- append(prior_ratings, list(get_mcmc_ratings(prior)))
}
print(prior_ratings)
```

5.5.2 Accuracy Evaluation

For ease, we perform accuracy evaluation by simply calling helper functions created in other files, as is the case with the Spearman's rank correlation coefficient, or run separate ones in the current file. We do so for the Brier score calculation and 10-fold cross-validation measures.

Brier Score

The following produces the Brier score entries of the Bayesian extensions of the Bradley-Terry model in Table 4.2.

```r
## Brier-Score
### Helper function
```{r}
bt_l_given_rankings = function(ratings, matches=match_data) {
  winner_r <- ratings[matches$winner_player_id]
  loser_r <- ratings[matches$loser_player_id]

  return(1 / (1 + exp(loser_r - winner_r)))
}

### Score calculation
scores <- c(0, 0, 0)
for (i in seq(length(prior_ratings))) {
  scores[i] <- mean((1 - bt_l_given_rankings(prior_ratings[[i]]))^2)
}
print(cbind(priors, scores))
```
```

Cross-Validation

The following produces the average log-likelihood of the 10-fold cross-validation procedure entries of the Bayesian extensions of the Bradley-Terry model in Table 4.3. We note that we set the number of players here to 100, as the search for optimised initial parameters in the Metropolis-Hastings algorithm takes significantly long to run for 500 players, as in the case of considering the entire match data set.

```
## Cross-Validation
K <- 10
cross_validate_mcmc = function(prior) {
  matches <- match_data[c('winner_player_id', 'loser_player_id', 'tourney_order')]
  data_partitions <- split(matches, seq(K))

  total_l_prob <- 0
  for(i in seq(K)){
    test_data <- data_partitions[i]
    training_data <- dplyr::bind_rows(data_partitions[-i])

    players <- as.vector(sort(unique(c(training_data$winner_player_id, training_data$loser_player_id))))
    num_players <- length(players)
    ratings <- rep(1.000, num_players)
    names(ratings) <- players

    l_f <- function(ratings) {
      winner_r <- ratings[match(training_data$winner_player_id, players)]
      loser_r <- ratings[match(training_data$loser_player_id, players)]
      return(sum(log(1 / (1 + exp(loser_r - winner_r)))))
    }
    bayesian_setup <- createBayesianSetup(likelihood=l_f, names=names(ratings), prior=prior)

    # Start MCMC
    settings = list(iterations = 10000, nrChains=1, burnin=500)
    bt_mcmc <- runMCMC(bayesianSetup = bayesian_setup,
                       sampler = "Metropolis",
                       settings = settings)

    # Extract MAP
    ratings_mcmc <- MAP(bt_mcmc)
    ratings_mcmc_final <- ratings_mcmc$parametersMAP
    names(ratings_mcmc_final) <- player_data$player_id[match(players, player_data$player_id)]
    ratings_output <- sort(ratings_mcmc_final, decreasing=TRUE)

    total_l_prob <- total_l_prob + sum(log(bt_l_given_rankings(ratings_output, matches=test_data[[1]])))
  }
  l_prob <- total_l_prob / K
  return(l_prob)
}

## Calculate avg log-likelihood
results <- rep(0, length(priors))
for (i in seq(length(priors))) {
  results[i] <- cross_validate_mcmc(priors[[i]])
}
print(results)
```

5.5.3 Plotting the Prior distributions

The following produces the probability densities of each chosen prior distribution as in Figure 3.1.

```
## Plot priors
library('MASS')
library('ggplot2')
library('latex2exp')
library('viridis')

samples <- seq(0, 10, length=1000)
prior_df <- data.frame(cbind(samples, dgamma(samples, 2), dnorm(samples, mean=5),
     rep(0.1, length(samples))))
colnames(prior_df) <- c('x', 'y_gamma', 'y_norm', 'y_uniform')
ggplot(prior_df) + geom_line(aes(x=x, y=y_gamma, color="a")) + geom_line(aes(x=x,
     y=y_norm, color="b")) + geom_line(aes(x=x, y=y_uniform, color="c")) + scale_
    color_manual(labels = c(TeX("Gamma$(2,␣1)$"), TeX("$N(5,1)$"), TeX("U(0,␣10)"
    )), values=viridis(6)[2:4]) + theme_classic() + xlab('value') + ylab('
    probability␣density') + labs(color="Prior")
```