Rating systems with multiple factors

Marius Stănescu

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2011

Abstract

Rating systems have been receiving increasing attention recently, especially after TrueSkill$^{\text{TM}}$ was introduced (Herbrich et al., 2007). Most existing models are based upon one latent variable associated with each player; the purpose of my project is to construct a multiple-feature model for rating players. Such a model associates more characteristics to a competitor and could – besides telling your skill and being used for matching players – provide insight into the characteristics of one's play and strategy.

We found that simply fitting the models through maximum likelihood has low generalising capacity, and also requires massive amounts of data in order to yield high accuracy. We turned towards a Bayesian approach and used Assumed density filtering and Expectation Propagation algorithms (Minka, 2001). They bring a significant accuracy bonus, even without a time series model to keep track of how players' skills evolve. We have also implemented a version of TrueSkill$^{\text{TM}}$ adapted to our problem (game of Go) and use it for comparing our models.

We present experimental evidence on the increased performance of the multiple factors models; they significantly raise the accuracy of Expectation Propagation model, and with enough data, more factors improve also the Assumed density filtering model. On small datasets, we discovered that an iterative method brings a significant advantage for Assumed density filtering, greatly surpassing even the TrueSkill$^{\text{TM}}$ algorithm. There are some additional benefits for the multiple factors approach, including higher accuracy for predicting results of balanced games.

Acknowledgements

Save your explanations, I got some questions for you first and you'd better answer them! slurred Hellian.

With what? Banaschar sneered. *Explanations?*

No. Answers. There's a difference–

Really? How? What difference?

Explanations are what people use when they need to lie. Y'can always tell those,'cause those don't explain nothing and then they look at you like they just cleared things up when really they did the opposite and they know it and you know it and they know you know and you know they know that you know and they know you and you know them and maybe you go out for a pitcher later but who picks up the tab? That's what I want to know.

Right, and answers?

Answers is what I get when I ask questions. Answers is when you got no choice. I ask, you tell. I ask again, you tell some more. Then I break your fingers, 'cause I don't like what you're telling me, because those answers don't explain nothing!

Steven Erikson (The Bonehunters)

I would like to express my gratitude to my supervisor Iain Murray for always having the right *answer* at hand (at which, alas, he gets to keep all his fingers). However, I'll keep the many insightful comments on all stages of my work, the excellent advice and a significant slice of his time and ideas. I thank him for his continued guidance, which he offered in an always helpful attitude and with seemingly endless supplies of patience. It has truly been a pleasure.

I am unbearably in debt to my mother, Monica, for her invaluable effort in raising me. She and my cat have been providing support and encouragement throughout the entire MSc.

Thanks a lot to my cousins Răzvan and Florin for their support and joint funding of my MSc course.

This dissertation came to light despite the total lack of help from durere, loctarar, Tibos and y4nk, who refused to acknowledge my increasingly desperate pleas against playing every night. Let them be forgiven.

Thanks to my *Girlfriend* for being so patient, loving and caring.

I would like to also thank my flatmates for helping me discover my latent ability of listening to myself. The eerie silence has worked wonders for getting my thoughts in order and making this project realizable.

Special thanks to Steven Erickson who – along with Matthew Stover and Joe Abercrombie – has saved me again and again from my hard times and kept me sane. Nothing better than epic fantasy to keep gradients out of your dreams.

Thanks also go to all my new friends, for being great company this past year. They are too many to mention, but the special prize goes to Mark. You can admire his 'best buddies' smile in the picture.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Marius Stănescu*)

To *Miau*.

Table of Contents

1 Introduction **1**

1.1 Rating systems . . . 1

1.2 Purpose . . . 1

1.3 Motivation . . . 2

1.4 Objectives . . . 2

1.5 Project outline . . . 3

2 Background **5**

2.1 Evolution of rating systems . . . 5

2.2 Related work . . . 6

2.2.1 Recommender systems . . . 6

3 Data **8**

3.1 Data format . . . 8

3.2 Data restrictions . . . 9

3.3 Komi and handicaps . . . 10

4 Maximum Likelihood fitting of models **12**

4.1 Standard Bradley–Terry and Elo models . . . 12

4.2 Fitting the models . . . 13

4.3 Regularization . . . 14

4.4 Extending the system with more factors . . . 15

4.5 Evaluation and results . . . 16

4.5.1 Number of factors . . . 17

4.5.2 Time variability . . . 18

4.5.3 Generalising capacity . . . 19

4.5.4 Calibration . . . 21

5 Bayesian Approach **23**

5.1 True Skill^{TM} 23

5.1.1 TrueSkill^{TM} description 24

5.1.2 Modifications 27

5.2 Assumed-density filtering 30

5.2.1 Algorithm description 31

5.2.2 Applying ADF to rating 32

5.2.3 One factor ADF model 34

5.2.4 ADF with more factors 35

5.2.5 Problems 36

5.3 Expectation Propagation 37

5.3.1 Implementation 38

5.4 Evaluation 41

5.4.1 Making predictions 41

5.4.2 Model initialisation 41

5.4.3 Dealing with negative variances 42

5.4.4 Factor restrictions 43

5.5 Accuracy experiments and results 44

5.5.1 Comparing ADF models with different number of factors . 44

5.5.2 Extra iterations of ADF algorithms 45

5.5.3 Combining multiple factors and extra iterations 46

5.5.4 Expectation Propagation 47

5.5.5 Increasing the size of the dataset 49

5.6 One factor vs. more factors 50

5.6.1 Metrics 50

5.6.2 Combining the metrics 53

5.6.3 Benefit of extra term in prediction performance 54

5.6.4 Players with close skill values 55

5.6.5 Highly skilled players 57

6 Conclusions **60**

6.1 Goals and results 60

6.2 Future work 62

A **64**

A.1 Reducing the ADF update to two players 64

A.2 Reducing the EP update to two players 65

Bibliography **66**

Chapter 1

Introduction

1.1 Rating systems

Generally speaking, *rating* can be defined as the task of assigning some properties to a subject. Additionally, *ranking* deals with arranging the subjects in the right order, consistent with some of these properties.

In some circumstances, the term *rating* is encountered when working with systems that have individual users' choices and preferences as input. A good example can be found in movie rating systems. These are not addressed in our project, and we will use the term *rating system* for denoting a framework that processes a number of recorded performances into rankings and ratings of the antagonists' skill.

1.2 Purpose

During the past years many have aimed at defining and devising the concept of a rating system. Competition is embedded in our character, and as long as we strive to do better than others, the necessity of rating and ranking contenders emerges. We need to be able to establish who is the best, and to contrast.

Usually, direct measurements of the abilities are unavailable. The only information we possess is structured as a quantity of outcomes – who won – which are affected by these underlying abilities. A rating system is designed to use this available information and to make valid estimations about the unknown parameters – the skills of the competitors.

Rating systems are also used for predicting future encounters, and are profitable when correct predictions are rewarded – like betting on the winner of a tennis match. An accurate rating system that can foresee better than other parties can surely make profit. This need for rating is present in the tennis system, for example, and is often encountered in many other areas, from educational systems to ranking web pages.

1.3 Motivation

A rating system should provide fun, exciting matches for the players. And a game is most fun when the outcome is most uncertain. Why is this the case? Because if the apriori chances of winning are close to 50%, the player has to go for that extra mile, has to exert some extra effort, in order to really have a favourable impact upon the game outcome. Succeeding in a challenging game feels very rewarding when playing, and some individuals may even find joy in playing against slightly better opponents.

If we would have a situation where one player dominates the other, it will probably be boring for both of them, because the outcome would be clear from the beginning. In other words, from the players' perspective, the perceived quality of the system is better explained by their winning ratio. If this is too high, then the player was assigned to fight against opponents that are too weak, and the other way around. Ideally we would like the number of wins to be close to the number of losses, and a good ranking system is able to make this possible.

We also aim to try to understand and give a meaningful explanation for the features that emerge from the use of the ranking system. Current systems associate a single latent variable to each person; we are interested in the possible advantages of learning multiple latent features about every player. Such a more complex system could – besides telling your skill and being used for matching players – provide insight into the characteristics of your play and strategy. Each feature could have a meaningful interpretation, and one can see why, for example, one would be likely to be defeated by a lower-ranking opponent and improve his strategies accordingly.

1.4 Objectives

In this project we are interested in fulfilling the following goals:

Model Choosing a simple existing model for expressing the skills of the players (such as the ones employed in chess) and extending it to include different numbers of factors.

Data Acquiring a suitable dataset for evaluation; it should contain a large number of matches, and for simplicity we would like to chose a two player game (as team support would lead to more complicated models). Ideally only wins and losses should be possible, without possibility of draw. Basic understanding of the game and any specific mechanisms (such as handicap systems) is required.

Rating systems Implementing appropriate algorithms for fitting the parameters in the models devised, in order to obtain comparable (preferably better) performance than current rating systems. Choosing a suitable and representative system to compare against.

More factors Identifying what advantages an approach with more factors brings, and what disadvantages. Contrasting predictions made using a system with one or more parameters. More factors might increase accuracy for very skilled players, or for balanced matches, for example.

1.5 Project outline

The structure of the project comes from the list of goals presented above, and the thesis is organised in chapters as follows:

Chapter 2 The next chapter briefly mentions the evolution or rating systems, and related work in the recommender system areas which helps us come up with ideas for extending a model to multiple factors.

Chapter 3 The game of Go is introduced, and we describe the format of the dataset, along with its analysis (cleaning, handicap system, choosing appropriate subsets for experiments).

Chapter 4 The standard model is presented, and how we extend it with additional factors. We then proceed to fit it using maximum likelihood, and make concluding remarks from the results obtained.

Chapter 5 We switch to a Bayesian approach, and we start with describing TrueSkill$^{\text{TM}}$ (Herbrich et al., 2007) which stands as our source of inspiration. It will be used in the evaluation to compare against the two rating systems we implemented using Assumed density filtering and Expectation Propagation. Both algorithms are presented in this chapter, along with all the ensuing evaluation procedures and results' discussions. We thoroughly investigate the benefits of using more factors.

Chapter 6 The conclusion of this thesis, where the progress made during the project is summarised and assessed.

Chapter 2

Background

2.1 Evolution of rating systems

Counting the number of wins and losses is a simple metric for rating, but the percent of games won does not tell anything about the opponents against one played. One can get a very high percentage just by choosing weak players, easily defeated. Another problem is that skills change in time; players that have been beaten repeatedly but learnt from their losses and improved their game would be disadvantaged by a simple win–loss system.

Naturally, we aim to win most games, but too many victories probably denote a lack of challenge. Ideally, winning about half of the games would bring balance, and if most of the players would be near a 50% ratio, the ranking by win percentage would not be informative.

The most basic rating systems are based on rewards: every win provides a prize consisting of a number of points proportional to the importance of the match. The computations are very easy, simple to understand and it is this simplicity that makes them popular, such systems being encountered in many sports, for example tennis (ATP - Association of Tennis Professionals, 2011). Usually, in these areas the focus is on the high-level competition between top players.

However, even if their inherent structure makes them able to easily differentiate between the top players, their ability to compare and contrast lower-ranked competitors drops steeply. Also the number of points awarded do not have any solid statistic reasoning behind, making these systems inefficient from a statistical standpoint.

The first sport to turn towards a more scientific approach is probably chess. This happened in the forties, with the emergence of the Chess Federation (USCF). They started by using Harkness (1967), a system which employed rewards that scaled with the difference between the current ratings of the players. This amount was fixed beforehand, and computed using a simple function.

The next important system was proposed by Elo (1978). He based the system on a statistical model, which makes the underlying assumptions explicit and open to criticism and refinement. Elo's two main assumptions were:

- a player's performance is a normally distributed variable, with the same standard deviation as other players';
- the skill of a player should change very slowly, according to the belief that a player cannot improve or lose his skill easily.

The ensuing systems tried, besides computing ratings, to also measure the uncertainty of the skill's estimation for every player. One such system was proposed by Glickman, which was specifically constructed for dealing with matches between two players. More background information about pairwise comparison and other models can be found in (Glickman, 1999).

During the last decade, these rating systems have been extended for events that include more players, and probably the most significant results were obtained by Microsoft's approach, TrueSkill^{TM} (Herbrich et al., 2007) and TopCoder's ranking algorithm (TopCoder Inc, 2008). All these are based on a single feature (skill of the player) and Bayesian inference. We should note that the performance of the team is modelled as a (non weighted) sum with the components being the skills of the participants, and this is not necessarily a valid assumption.

2.2 Related work

2.2.1 Recommender systems

The data that we wish to analyse is best modelled as a result of a pairwise interaction between items of the same set (*players*). A related problem is when the items belong to different sets – e.g. *users* and *movies*.

Recommender systems analyse available data in order to make predictions about what TV series, movies or music a particular customer would enjoy. One movie will be seen by many, and most customers watch a large number of movies. Moreover, people are inclined to provide feedback about what they enjoy or dislike, so very large datasets are procurable. The Netflix competition has recently sparked interest for this subject, and new *collaborative filtering* (Zhou and Luo, 2009) methods have been developed (Bell and Koren, 2007; Takács et al., 2008). They model relationships between users and interdependencies among products, in order to identify new user-item associations (Koren et al., 2009).

These methods can be divided into *neighbourhood methods*, which try to infer relationships between movies or between customers (Sarwar et al., 2001), and *latent factor models*. These aim to give an explanation for the ratings given by the users by describing them and the movies each with a number of factors (usually under a hundred).

Matrix factorization implementations of latent factor models have been favoured recently and are considered to be scalable and to provide good accuracy (Cai et al., 2008). The data is characterised by a matrix with one dimension containing users and the other rated items (eg. movies). Only a fraction of the entries are known, and the remainder are unavailable. The goal would be to complete these values, by considering that the rows and columns of the matrix interact in an useful way.

The items and users are each associated with a vector of f features, in a latent factor space where synergies are quantified by the inner product of said vectors. The predicted interest of one user in a particular item is expressed by $\underline{u}^T \cdot \underline{i}$, where $\underline{u}$ and $\underline{i}$ have the same length, f. These features are very similar with what we propose for modelling players' skills.

Chapter 3

Data

As mentioned throughout the document, in this project we are developing a multiple-featured ranking system which will be able to analyse games' outcomes and assign appropriate ratings to players. A suitable dataset for the experiments is the data from the *KGS GO* [1] server.

This data has been collected since the start of the KGS GO community [2], and represents all the games played on the server within a ten-year period. There are a total of over 40 million matches, featuring more than 180 thousand players. As part of fitting the models, we will split this into several smaller datasets, containing training and validation data. The final tests were done at the end of the project, on games from the last year of the dataset. This part was withheld and never used during the project, in order to reduce any unintentional overfitting that may occur. Matches towards the end of the period have been preferred because the number of games per day is larger.

3.1 Data format

The data was received as a text file, 2.5 GB long. We have built a C# parser in order to port the information to Matlab. We store it as a matrix, with a line for each game. It contains the following information (columns):

- **date of the game**: originally $yyyy-dd-mm$, we converted it to a natural number.
- **name of the white player**

[1] `http://www.gokgs.com/`
[2] Many thanks to KGS administrator **William M. Shubert**, who kindly provided the data

- **name of the black player**: these were account names, we built a hashtable and assigned natural number id for every player. This was saved as a text file for eventual further referencing.
- **white and black ranks**: the ranks for the two players, as assigned by the KGS system. This information was discarded.
- **game type**: some matches are not ranked by the KGS rating system; a game is either ranked or free.
- **size**: the dimension of the board, for example 9 (9×9), 13 or 19.
- **handicap**: black is the weaker player, and has a number of handicap stones to play before white's first move, as an advantage.
- **komi**: points added to the score of the player with the white stones as compensation for playing second.
- **game duration**: this information was discarded.
- **score**: the difference in number of points, with some specific values for win by time, win by resign, no result, unfinished and forfeit games. Games without result, or unfinished, were removed. For the rest, we have simply used 1 for white win, and 0 for black win.

3.2 Data restrictions

There are some issues that may arise. Firstly, skills of the players may change in time. For a model that does not take this assumption into account, we should use data within a narrow time range. Skills are not likely to modify in a short amount of time. For testing, we have extracted 10 completely disjunct datasets from the last year, of 20 days each. Non-ranked matches and games on board sizes other than 19×19 have been discarded.

Secondly, we would like to discard players that have played only a few matches. Having little information about a player is likely to cause diminished performance. Dealing with new players is one of the challenges a rating system has to solve, but is outside the scope of this project. Our main interest is extending a rating system to use more than one latent feature for each player.

Only games between players with more at least 12 matches have been kept. Note that the game count is *only* between these (remaining) players, and does not feature subjects that are not in the set or that are outside the time limits. Thus, every player has at least 12 games with other players present in the dataset. Moreover, when splitting the data into training, validation and test sets, we have enforced the 12 games restriction on the players in the training part (which consists of the first 80% of the games). Otherwise, it might happen that some competitor has all his matches in the test set, and we wouldn't be able to predict any of them. Naturally, players that do not figure in the training set have been eliminated from validation and test data, too.

The resulting datasets have around 200 000 - 250 000 games each, and in between 7 000 and 10 000 players.

3.3 Komi and handicaps

Usually the weaker player takes black, and places the first stone. To balance this advantage and to prevent ties, he gives away some 'komi' points to the white player. The players are ranked from 30 *kyu* to 1 *kyu*, then from 1 *dan* to 7 *dan* (amateur ranks) and from 1 professional *dan* to 9 professional *dan*, which is separate from amateur ranks. This division would make for $30 + 7 + 9 = 46$ number of ranks. The difference among each amateur rank is one handicap stone. For example, if a 4k (weaker) plays a game with a 2k (stronger), the 4k would need a handicap of two stones to even the odds.

We need to take both komi and handicap into account, when computing the likelihood of one player winning. For a game with H handicap stones, and K komi points, our models update the skill difference in a match between players W and B with a term corresponding to the handicap:

$$\text{Updated difference} = S_W - S_B - (H - \frac{K}{\text{Komi factor}}) * \text{ Handicap factor} \ . \tag{3.1}$$

If the skills of the player were numbers corresponding to their ranks ($S_W, S_B \in [0, 46]$), then the correct handicap factor would be 1. In our algorithms, we aim for skill values roughly between -3 and 3, and we set

$$\text{Handicap factor } = 1/7 \qquad \left(\frac{\text{skill range}}{\text{ranking range}} = \frac{6.5}{46}\right) \tag{3.2}$$

Komi are points given in the opposite direction as the handicap, and have to be converted into an equivalent number of handicap stones. By asking an estimate from an expert player, the following values were proposed. A stone is worth about 15 points in blitz games, 10 points in average speed real time games (such as default KGS timings), and about 6 or 7 in correspondence games. We settled on using

$$\text{Komi factor } = 12, \tag{3.3}$$

though this factor could potentially be learned in future work.

Chapter 4

Maximum Likelihood fitting of models

In this chapter we describe a standard Bradley–Terry model and fit it to the data using maximum likelihood. We then extend the simple model to include mode than one factor per player. In the last section of the chapter we evaluate both models and present the results.

4.1 Standard Bradley–Terry and Elo models

The basic rating framework is the model suggested by Bradley and Terry (1952), which has been proposed in order to deal with repeated comparisons between a set of subjects. In their simplest form, Bradley–Terry models assume that each player has a real rating, and the winning probability of a player in a game is proportional to his rating (in the simplest case of two players competing with no ties). Thus, in this system the probability of *Player A* defeating *Player B* is computed as the *performance* associated with the first player divided by the sum of the performances:

$$P(p_A > p_B) = \frac{\gamma_A}{\gamma_A + \gamma_B} = \frac{1}{1 + \dfrac{\gamma_B}{\gamma_A}}. \tag{4.1}$$

In the Elo ratings, the term γ_i quantifies the skill of *Player i*, and is defined as an exponential function of his strength (skill value): $\gamma_i = e^{\frac{S_i}{\text{Scale}}}$.

Then, using the sigmoid function we can write:

$$P(p_A > p_B) = \frac{1}{1 + e^{-\frac{S_A - S_B}{\text{Scale}}}} = \sigma\Big(\frac{S_A - S_B}{\text{Scale}}\Big). \tag{4.2}$$

In this model it is considered that every player has a true playing strength or – equivalently – *skill*, which is represented by $S_A; S_B$. The scale (denoted onwards as k) determines what the numbers mean in terms of ability, i.e. the significance of 100 rating points. For example, let's consider two players that are 100 rating points apart. When using $k = 100$ the probability of the weakest player to win would be 0.27 whereas if $k = 200$ it would increase to 0.37.

This simple system is known to work reasonably well in practice, and is a good candidate for our model. In the next section we will describe our methodology for fitting a simple Bradley–Terry model to the data, and then we will discuss increasing the number of factors in the model.

4.2 Fitting the models

Let us assume that we possess data from a number of matches between m players, with their respective outputs (i.e., who is the winner). We note with D the outcomes of all games.

The likelihood of S given D is used for approximating the skills of the players; the maximum likelihood value can be then chosen as an estimation. The computation time is usually quite low using conjugate gradients, for example.

In this section we fit the model above to this data and determine the maximum likelihood parameters, i.e. skills, for these subjects. The number of wins of p_i over p_j will be denoted by w_{ij} and by convention $w_{kk} = 0$. We compute the probability of the data given the skills of the players:

$$P(D|S) = \prod_{i,j=1}^{m} \Big(\frac{\gamma_i}{\gamma_i + \gamma_j}\Big)^{w_{ij}}. \tag{4.3}$$

We assumed that the outcomes of pairs involving different players are independent. The log likelihood can be expressed as

$$\mathcal{L}(\gamma) = \sum_i \sum_j (w_{ij} \log \gamma_i - w_{ij} \log(\gamma_i + \gamma_j)) = \sum_{i,j} w_{ij} \log \sigma\Big(\frac{S_i - S_j}{k}\Big). \tag{4.4}$$

Note that $\mathcal{L}(\gamma) = \mathcal{L}(\alpha\gamma)$, for a positive constant α , which indicates that the solutions can be thought of as equivalence classes, and two parameter vectors $\{\gamma_1, \cdots, \gamma_m\}$ and $\{\gamma'_1, \cdots, \gamma'_m\}$ are equivalent if they are linearly dependent (scaled by a constant). Consequently, we can add an additional constraint, for example $\sum_{i=1}^{m} \gamma_i = 1$.

The maximum likelihood value can be reached by starting with some default parameters, and optimizing iteratively until we are satisfied with the results. We can employ a gradient ascent method, by updating $S'_i = S_i + \eta \frac{\partial L}{\partial S_i}$, where *prime* denotes the new values, and

$$\frac{\partial L}{\partial S_i} = \sum_{j=1}^{m} w_{ij} \frac{1}{\cancel{\sigma}} \cancel{\sigma}(1-\sigma)\frac{1}{k} = \sum_{j=1}^{m} \frac{w_{ij}}{k}\Big[1 - \sigma\Big(\frac{S_i - S_j}{k}\Big)\Big]. \tag{4.5}$$

Alternatively, we could use other iterative algorithms (Hunter, 2004), and compute directly

$$\gamma_i^{t+1} = w_i \Big[\sum_{i \neq j} \frac{w_{ij} + w_{ji}}{\gamma_i^t + \gamma_j^t}\Big]^{-1}, \tag{4.6}$$

with the additional restriction that γ_i must be rescaled, such as $\sum_i \gamma_i = 1$. This leads to

$$e^{\frac{S'_i}{k}} = \sum_j w_{ij} \Big[\sum_{i \neq j} \frac{w_{ij} + w_{ji}}{e^{\frac{S_i}{k}} + e^{\frac{S_j}{k}}}\Big]^{-1}. \tag{4.7}$$

4.3 Regularization

In trying to fit the data better, we might impair the capability of the model to generalize, a phenomenon usually known as *overfitting*. This is certainly unwelcome, and there are some ways to avoid it. One solution is to use early stopping, which means halting the optimization (error minimization) process at some point, which is only optimal in its current window.

Regularization comes as another means to solve the problem. Regularization modifies the error function we were minimising (if we consider $Err(S) = -\mathcal{L}(\mathcal{S})$) by including a penalty term for the type of skills that we would like to avoid. For instance, if we want to stay away from large skill values, we can use a regularization term that punishes this sort of instance:

$$Err(S) = -\mathcal{L}(\mathcal{S}) + \alpha R(S), \text{and choosing } R(S) = 0.5 \sum_i S_i^2. \tag{4.8}$$

The term is called a *weight decay regularizer*, and the constant α is the weight decay rate. A good description can be found in MacKay (2003). Now our error function rewards small skill values, and favours less overfitting on the training data.

4.4 Extending the system with more factors

We can extend this model, thinking that in modelling a match between two opponents it would be useful to consider other features besides the skill level. If before we had

$$P(p_A > p_B) = \frac{1}{1 + e^{-\frac{S_A - S_B}{k}}}, \tag{4.9}$$

with S_A and S_B being real values we might want to use extra concepts such as *offensive* and *defensive* characteristics of the players. So, instead of writing

$$P(p_A > p_B) = \sigma(S_A - S_B) \tag{4.10}$$

(let's forget about the scale for now) we would try

$$P(p_A > p_B) = \sigma(S_A - S_B + O_A W_B - O_B W_A). \tag{4.11}$$

This would make the probability for player 1 higher if his *offensive* (O_A) is high, and the opponent's *weakness* (W_B) is also high, while decreasing it if the adversary is likely to succeed when attacking.

This model can be reduced to the first, simpler one, as the baseline model is nested within the three-parameter model. Let us write it using more general terms:

$$P(p_A > p_B) = \sigma(S_A^{(1)} - S_B^{(1)} + S_A^{(2)} S_B^{(3)} - S_B^{(2)} S_A^{(3)}), \tag{4.12}$$

where lower indices stand for player, and the upper indices stand for the parameters (currently three).

If we would like to keep this property, things are likely to become awkward, for example for a two-parameter model. Here we have only $\{S_A^{(1)}; S_A^{(2)}\}\{S_B^{(1)}; S_B^{(2)}\}$ to work with. A solution would be to consider the other parameter a susceptibility against better players, leading to

$$P(p_A > p_B) = \sigma(S_A^{(1)} - S_B^{(1)} + S_A^{(1)} S_B^{(2)} - S_B^{(1)} S_A^{(2)}). \tag{4.13}$$

However the underlying logic upon which the approach is based favours an odd number of parameters, say $2k + 1$. These can be organised as

- (1) the skill, a scalar, S_i
- (k) an *offensive* vector, containing k parameters, $O_i = \underline{v}_i^{(1)}$
- (k) a *weakness / susceptibility* vector, containing k parameters, $W_i = \underline{v}_i^{(2)}$

$$P(p_i > p_j) = \sigma(S_i - S_j + \underline{v}_i^{(1)T} \underline{v}_j^{(2)} - \underline{v}_i^{(2)T} \underline{v}_j^{(1)}) \qquad (4.14)$$

In a similar way to the previous section, we compute the derivatives with respect to the strength and the extra parameters, and use a gradient ascent method to find the maximum likelihood values.

4.5 Evaluation and results

Rating systems are mainly used for making predictions about the results of future encounters, and the quality of these prognoses is a metric for evaluating the quality of the system. Thus, we will evaluate the different algorithms by how effectively they predict the outcomes of contests.

The predictions for individual games will be scored separately and then we will compute the mean across all games. The evaluation function used for scoring (also suggested by Mark Glickman) is the *Log Likelihood* we previously maximise, namely:

$$\text{Log Lik} = \text{Outcome} * \log(w) + (1 - \text{Outcome}) * \log(1 - w), \qquad (4.15)$$

where Outcome is 0 for a black win, and 1 for a white win, while w is the predicted score for white. It is easy to see that predicting an expected score of 0% or 100% has an undefined Log Likelihood, since log(0) is undefined. Furthermore, even if white wins, the marginal benefit for a prediction of slightly above 99% is minimal compared to a 99% prediction. Similarly, we can argue a 1% prediction for a black win. Therefore there is no good reason to predict an expected score above 99% or below 1%. So for purposes of scoring, all predictions above 99% will be treated as 99%, and all predictions below 1% will be capped to 1%.

The method described in the previous section is a point estimate method, since it provides a one-value (in a dimension proportional with the number of players) solution to our problem. We obtain a fixed skill value (or a set of factors) for every player, and no information about the uncertainty of this prediction or about how the skill of the player changes through time. In fact, we assume that his skill is constant in the period over which we train and test the model. To minimise the impact this assumption has over the results, we use for evaluation datasets that are restricted to a period of 20 days. A player's performance should not vary significantly in a short period of time, at least compared to a 200 days interval.

For each dataset, the data is split into three sets:

- first 80% of games are the *training* set, which we use for training the model;
- the next 10% consist of the *validation* set, used for optimising parameters of the model (eg regularization constant, also used for early stopping);
- and the last 10% form the *test* set; we evaluate the prediction performance by the log likelihood on these games.

4.5.1 Number of factors

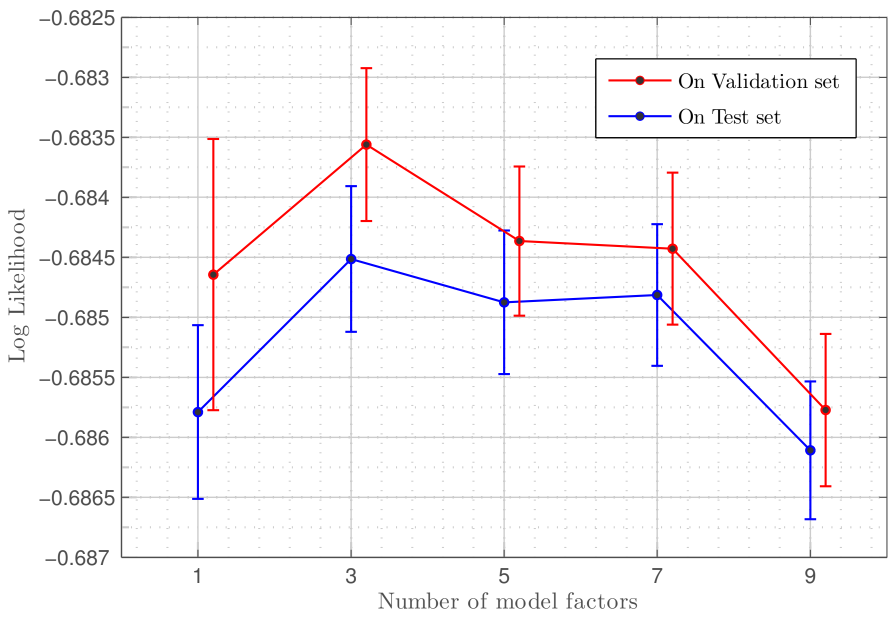


Figure 4.1: **Performance of models with increasing number of factors** Red shows the log likelihood on the validation set, blue on the test set. In this plot and all of the following figures, unless stated otherwise, errorbars show standard error corresponding to the 10 datasets.

The performance of models with different numbers of factors is shown in figure 4.1; results are averaged over 10 datasets, with errorbars corresponding to the standard deviation divided by $\sqrt{10}$. This measure is known as standard error, and has been used for most of our figures; one margin of standard error gives 68% confidence.

We show the results on both validation and test sets.

The three factors model is slightly improving the single factor one, but using more than three factors on this amount of data leads to overfitting and worse performance.

4.5.2 Time variability

We have chosen datasets that span over a relatively small amount of time (20 days) in order to reduce the impact of time variability. However, time still negatively affects the performance, as can be seen in the following experiment. In Figure 4.2 is presented the Log Likelihood for the simple model (left) and the three factors model (right), in two particular cases. The first case (blue) is the standard, with the performance on the validation and test datasets. For the second case (red) we have swapped the two sets, by doing validation on the last 10% of the data and testing on the previous 10%. It can be seen that for both cases, the data closest in time to the training set gives better performance. It should be noted that the validation and test sets contain games that are played over approximatively 2 days each, while the training set is made of matches ranging around 16 days.

We assumed constant skills for the players, which may prove too restrictive an hypothesis, even from a couple of days to the next one. Time variability is certainly a problem, and even though limiting the datasets reduces the influence of time variability, it is desirable that the problem is addressed by adapting the models.

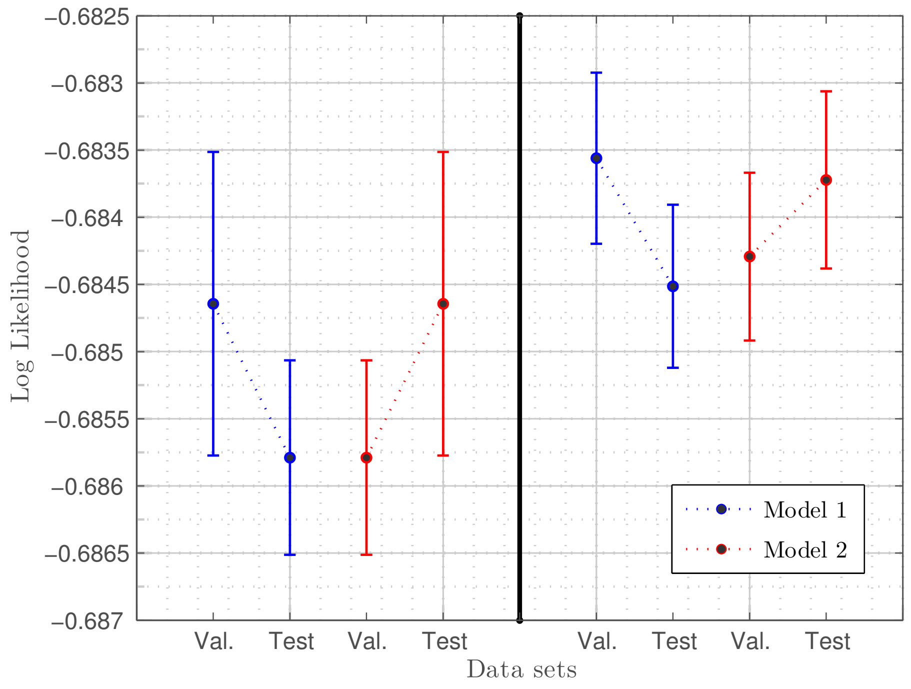


Figure 4.2: **Performance for the simple (left) and 3 factors models (right)** on the validation and test datasets. The standard experiment is depicted in blue, while red has the test set before the validation set, in time. In all cases, the models have better performance on the games that are closest to the training set, time-wise.

4.5.3 Generalising capacity

Since the maximum likelihood solution is a single estimate of the skills, the models may (in principle) suffer in terms of generalization capacity and accuracy of estimations for new test data. Further, such an algorithm does not provide a way to measure the uncertainty we have about the skills of the players.

We investigated how well the models fit and generalize, using the following experiment. The skills of the players are fitted on one of the datasets, and then 10 couples of synthetic datasets are generated, based on these skills. They are identical to the original dataset, except the results which are randomly drawn according to the underlying skills of the players. On the synthetic data, we refit the skills and compare the score on both virtual datasets, like in Figure 4.3.

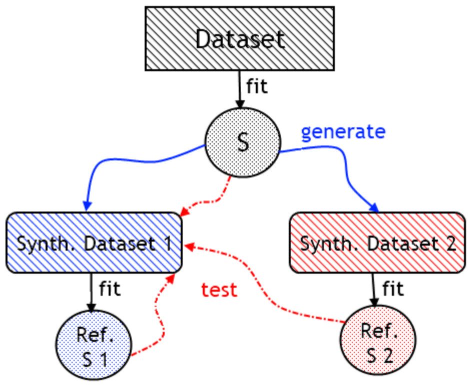


Figure 4.3: **Testing on synthetic datasets**. We start with a normal dataset, fit the skills (S), generate two synthetic datasets, and refit. There are a total of six tests, for the three skill sets over each of the two generated datasets. The 1 factor model is used.

Then, by doing 10 experiments we obtain the following performances, with errorbars (Figure 4.4). The refitted skills are doing almost as good on any synthetic dataset, but they are significantly worse than the *true* set of skills. The maximum likelihood models are not generalizing very well, and they need more data in order to perform near the accuracy of the *true* skills.

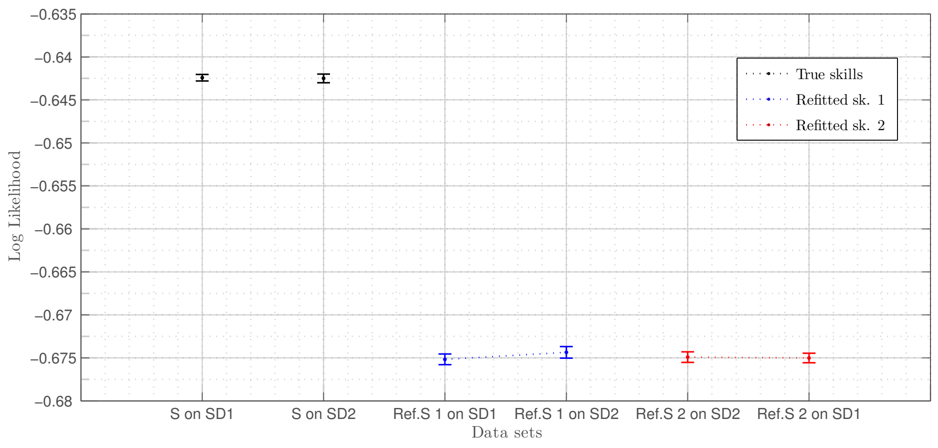


Figure 4.4: **Performance for one factor model**, on synthetic datasets. The prediction accuracy of the true skills used for generating the datasets is shown in black, while blue and red depict the prediction accuracy of refitted skills.

4.5.4 Calibration

We compare the win probabilities our models assign to the matches to the actual win rate of the same games. On one of the datasets, we take the set of predictions for all the matches, and divide it in bins of length 7% (eg. the first bin consists of all the games with predicted probability $p \in [0.50, 0.57)$). The number of games in each bin is shown in Figure 4.5, and it can be observed that the 3 factors model is less confident about its predictions.

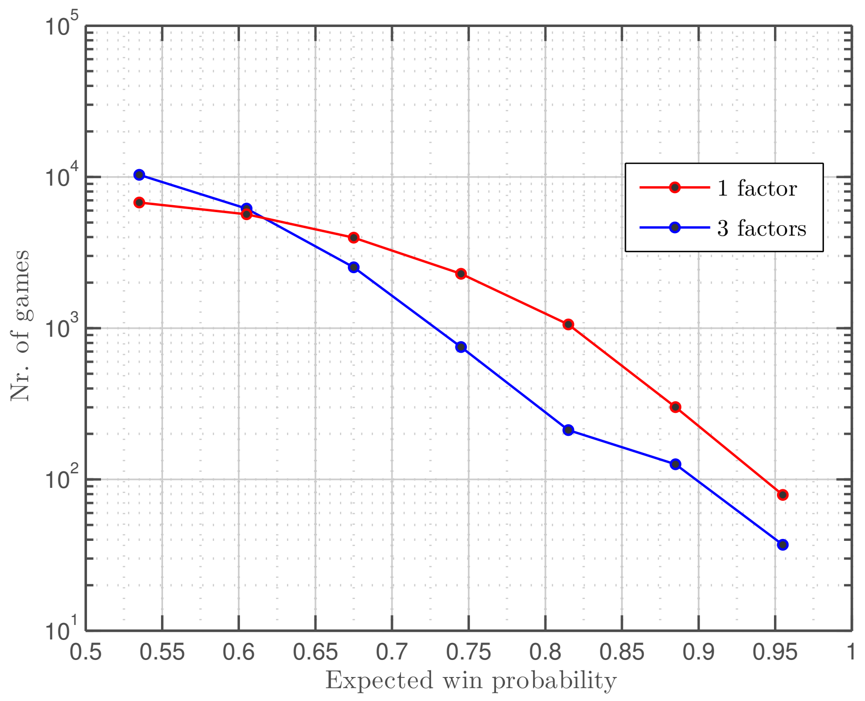


Figure 4.5: **Number of games** predicted to have a specified win probability.

For every bin, we compute the fraction that are real wins and contrast it with the predictions. The results for the simple and three factors models are shown in Figure 4.6.

As expected, both models are rather overconfident. Three factors is slightly better calibrated, however. The result of Maximum Likelihood is a point (in a dimension proportional with the number of players), which maximizes the probability of a given set of outcomes to occur. It should work well, provided we have a lot of information about every player (i.e., many games). But we do not, and it may even be unrealistic to expect more than a couple of games per day for a player. Generally, the solutions fitted by maximum likelihood lead to overconfident predictions. This behaviour is described in more detail in MacKay (2003, chap. 41).

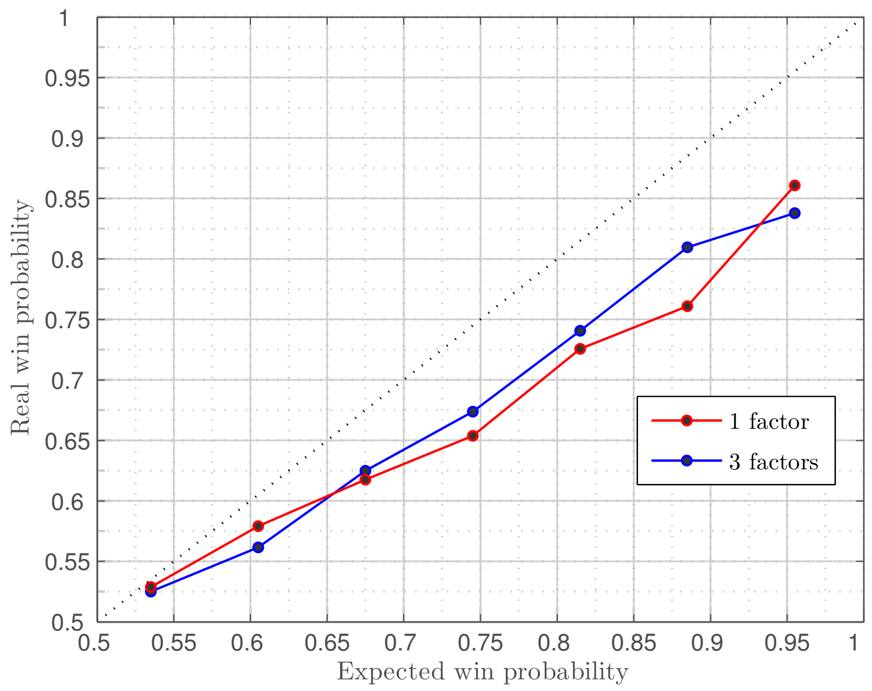


Figure 4.6: **Expected vs. real** win probability for the simple and three factors models.

In this chapter, we have started with a standard Bradley–Terry model and extended it to include more than one factor per player. We have fitted these parameters using maximum likelihood, and a conjugate gradient ascent method. While there is evidence that 3 factors may perform better than only one, the difference is not statistically significant. There is also strong evidence that a Bayesian model would improve our results by handling all the hypotheses for the skills of the players, rather than only a single assumption, be it the most probable or not. Instead of working with a point estimate, we should deal with the full distributions over the players' strengths. The Bayesian approach will be treated in the next chapter.

A potential problem is posed by the impact of time on players' skills. Choosing datasets where this effect is limited may not be enough; Including some way to incorporate time series into our models would be a welcome addition.

Chapter 5

Bayesian Approach

We start this chapter by briefly describing the *TrueSkill^{TM}* algorithm, which is one of the most prominent current rating systems. Using two core techniques also part of TrueSkill$^{\text{TM}}$, we devise models based upon *Assumed density filtering* and *Expectation Propagation*. We build the algorithms from scratch, in order to be able to customize them to working with multiple factors models.

While the *first half* of the chapter describes the algorithms and the underlying theory, in the *second half* the evaluation is set and the models are compared and contrasted using a multitude of experiments. We are interested in finding out how and when more factors improve the accuracy, what are the differences between TrueSkill$^{\text{TM}}$, Assumed density filtering and Expectation Propagation and how they reflect in the results obtained.

5.1 True Skill$^{\text{TM}}$

The Elo model described in the previous section suffers from the following problems:

- It does not support teams and treats draws like half wins, half losses, which is suboptimal. A draw implies similar strength values between players, while wins or losses show that one player is more skilled than the other. However, we do not know how big is the difference.

  So in Elo draws do not convey as much information as they could. We will not discuss these two problems further, because in Go there are neither teams nor draws.

- Its beliefs about the players' performances are normally distributed random variables, with the same standard deviation. Elo fixes this variance as a global constant and does not attempt to infer it from the data, which is restrictive.

- The belief about one player's skill would change very slowly, according to the hypothesis that a player cannot improve or lose his skill easily. Furthermore, the higher one's rating, the less likely one would want to risk it fluctuating much. The system considers ratings for players with less than 20 matches as provisional. During these games players are associated parameters that let the algorithm determine their skill faster. This is a rather high number of games required to infer the skill of a player, and we would like a system with faster convergence properties.

To solve the last two problems, the ensuing systems tried, besides computing ratings, to also measure this deviation for every player. This adds a means to estimate the certainty of the rating associated to a player. The rating system devised by Glickman (1999), and later TrueSkill$^{\text{TM}}$ (Herbrich, Minka, and Graepel, 2007), treat skills as normally distributed variables where the variance denotes the reliability of the estimation.

Even for limited data, exact Bayesian inference is intractable. We are not able to compute the posterior distribution, and we need to approximate it. Consequently, we are facing a marginalization problem which is one of the basic subjects of Bayesian analysis theory. To solve the problem, the TrueSkill$^{\text{TM}}$ system employs a Gaussian density filtering algorithm, which we also used for our system. We will continue by briefly presenting TrueSkill$^{\text{TM}}$, and the ideas we used in constructing our models.

5.1.1 TrueSkill$^{\text{TM}}$ description

The probabilistic model used by TrueSkill$^{\text{TM}}$ is constructed to deal with players that take part in games or enter tournaments, and compete in teams of different sizes. It estimates the skills of these players after each match (or competition). The system is initialised with a prior Gaussian distribution over the skills

$$p(\text{skill}) = N(s; \mu, \sigma^2). \tag{5.1}$$

The 'true skill' for the player is denoted by μ, and σ indicates the uncertainty of the estimation. The uncertainty will usually decrease with the number of matches, as we get more information about the respective player.

In matches, players do not perform according to these exact values, but are characterised by

$$p(\text{performance}|\text{skill}) = N(p; s, \beta^2), \tag{5.2}$$

where β is a fixed value, constant for all players. Integrating out the skills, we obtain

$$p(\text{performance}|\mu, \sigma) = \int_{-\infty}^{\infty} N(p; s, \beta^2) N(s; \mu, \sigma^2)\, ds. \tag{5.3}$$

These individual performances are then combined in order to obtain team performances ($t = \sum_i p_i$, the sum of the performances for all players in a team), and these are compared with the order ω obtained as a match result. We are only interested in one versus one matches, so we will not go into further detail.

We wish to compute the posterior distribution over the skills, given the outcome ω :

$$p(s|\omega) = \frac{p(\omega|s)p(s)}{\int p(\omega|s)p(s)\, ds}. \tag{5.4}$$

One more type of parameter is introduced, d, which relates to the differences in the team performances for any consecutively placed teams. Then the joint probability of the model can be written as:

$$p(\omega, d, t, p, s) = p(\omega|d)\, p(d|t)\, p(t|p)\, p(p|s)\, p(s), \tag{5.5}$$

from which we would like to find out

$$p(\omega|s) = \int\int\int p(\omega, d, t, p, s)\, dd\, dt\, dp. \tag{5.6}$$

This marginalization is solved using message passing (Kschischang et al., 2001), and you can see an example factor graph in the Figure 5.1. Most of the distributions are Gaussian, leading to easy computations, except for the bottom node (black colour). This distribution is approximated with a Gaussian by computing the first and second moments, and the message passing algorithm is run until convergence. The method that specifies this approximation and convergence criteria is Expectation Propagation (Minka, 2001), and is only used within the factor graph corresponding to a single match, although with any number of teams and players. On the other hand, the Gaussian filtering algorithm is employed in the context of updating the players' skills after a match. It approximates the posterior distribution with a Gaussian, and uses it as the prior for the next game (Herbrich et al., 2007). These algorithms will be further explained in the ensuing sections.

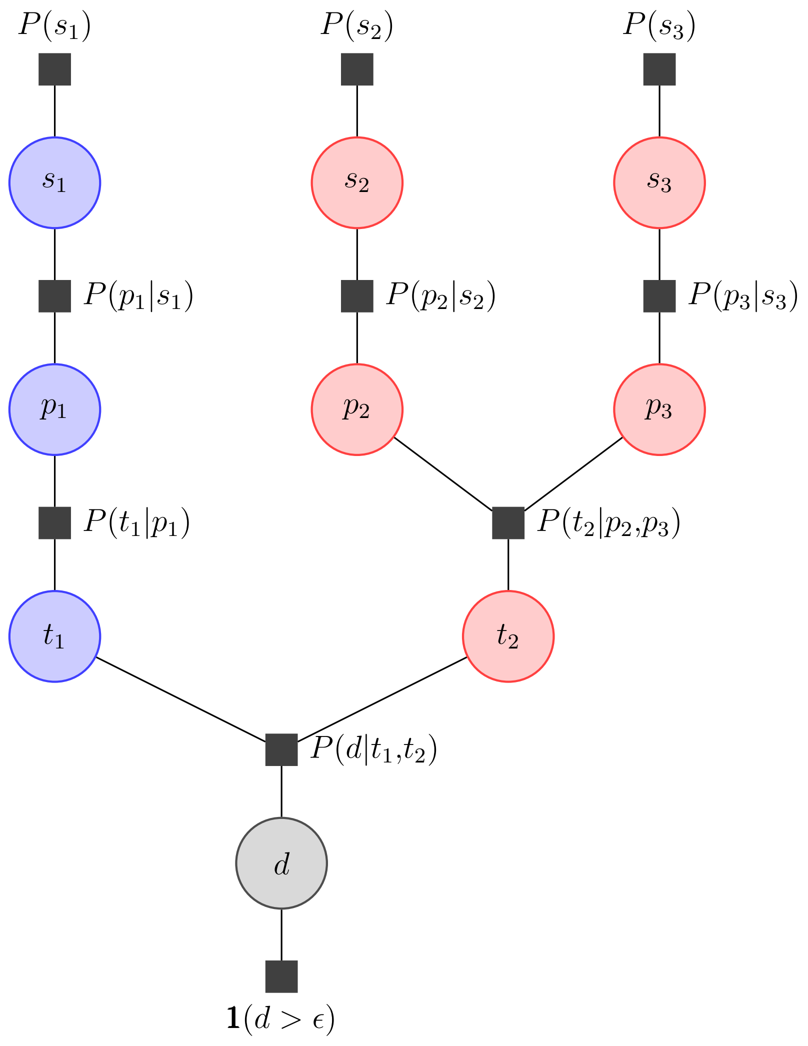


Figure 5.1: **Example of TrueSkill$^{\text{TM}}$ factor graph** for two teams. One player forms the blue team, and the red one has two players. Adapted from Figure 1 from (Herbrich et al., 2007).

For our problem, the factor graph is simplified, as we do not have more than two players, or teams. Also, we did not include the performance level, and assumed that results depend directly on the true skills of the players. For these reasons, we did not employ factor graphs, and will not present the computations in a message passing framework.

5.1.2 Modifications

We would like to compare our models to TrueSkill$^{\text{TM}}$ during evaluation, and we need to implement the algorithm and choose the right parameters, to obtain good results. Fortunately, there is already available a very good C# implementation [1] for TrueSkill$^{\text{TM}}$. However, some changes were needed (such as interfacing with Matlab, integrating the handicap system, building a prediction metric, and choosing the right parameters). We will briefly talk about the most important modifications required in order to adapt TrueSkill$^{\text{TM}}$ to our problem in the next three subsections of 5.1.2.

5.1.2.1 Predictions

We have outlined the basic working principles of TrueSkill$^{\text{TM}}$, and let us assume that we inferred both

- a set of *true* skills and
- variances denoting the system's uncertainty for the skills' estimations.

In TrueSkill$^{\text{TM}}$, the draw probability is used as an evaluation metric, since the focus is on match quality and consequently, the balance level of a match (larger draw probability results in more interesting matches). We, on the other hand, would like to predict the outcome of a future game, using the values inferred by our models.

In Elo, a Cumulative Gaussian or a Sigmoid function of the difference in skills was used, and we can generalise this approach. If we subtract the two Gaussians corresponding to the skills of the players, the result is another normal distribution, with the mean equal to the difference of means and a bit wider than the previous distributions (see Figure 5.2). This new Gaussian represents all the possible game outcomes. One would like to count all the possible outcomes where he is performing better than his opponent.

[1]Open source code by Jeff Moser `https://github.com/moserware/Skills`

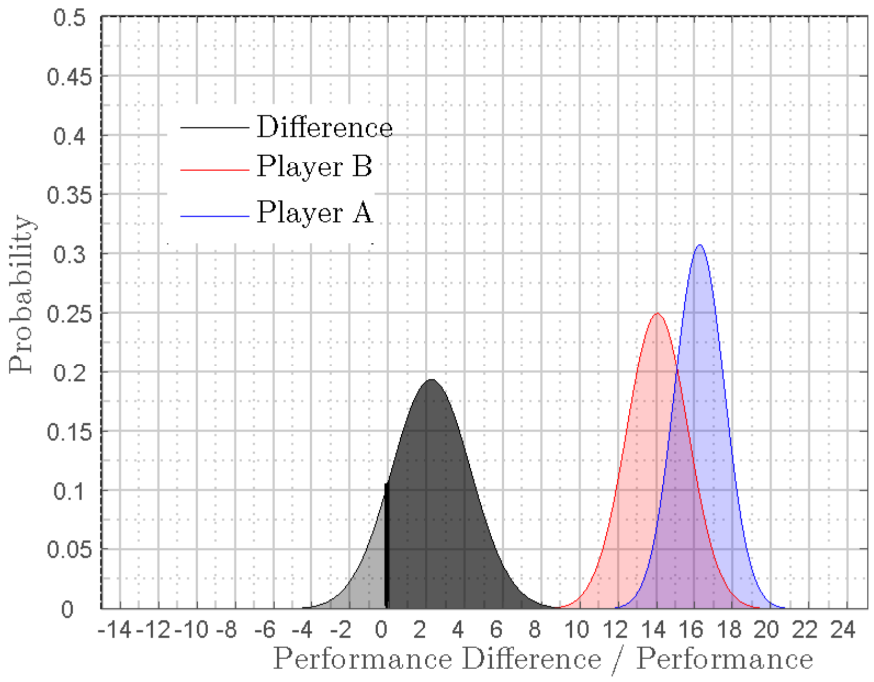


Figure 5.2: **Skill estimates for two players** in blue and red. They represent the probability distribution of their individual performances. The probability of their performances' difference is shown in grey, and is also normal distributed. Darker shade of grey (right side) is associated with the better player winning the game. Blue player skill $\sim N(16.2, 1.3^2)$, red player skill $\sim N(14, 1.6^2)$.

The first player, for example, would be interested in all the positive differences, meaning the area under the Gaussian, to the right of zero:

$$p(p_A > p_B) = \int_{-\infty}^{\infty} \mathbf{1}(S_A - S_B > 0) p(S_A; \mu_A, \sigma_A) p(S_B; \mu_B, \sigma_B)\, dS_A\, dS_B \tag{5.7}$$

$$= \int_{0}^{\infty} N(D; \mu_A - \mu_B, \sigma_D^2) dD \tag{5.8}$$

$$\sigma_D = \sqrt{(\sigma_A^2 + \sigma_B^2)}. \tag{5.9}$$

On the example in Figure 5.2, this method yields a probability of 85.70% for the better player to win (darker shade of black in figure). Since we used the sigmoid function for computing the likelihood of one player beating another, we can also try :

$$p(p_A > p_B) = \int_{-\infty}^{\infty} \sigma(S_A - S_B) p(S_A; \mu_A, \sigma_A) p(S_B; \mu_B, \sigma_B)\, dS_A\, dS_B \tag{5.10}$$

$$\approx \sigma\left(\frac{\mu_A - \mu_B}{\sqrt{1 + (\pi/8)(\sigma_B^2 + \sigma_A^2)}}\right) \tag{5.11}$$

using the approximation which we will later introduce in equation 5.22. On the same example, this method gives a win probability of 79.28% for equation 5.10, computed using a quadrature method. If we use the approximation as in equation 5.11, the resulting probability is 79.36%, very close to the 'exact' version which yields 79.28%. Both of these values are lower than 85.70% obtained with the previous method (5.7), denoting less confident predictions.

5.1.2.2 Handicaps

The handicap system has already been mentioned in section 3.3.The skills used for the players are the values we obtain after doing the handicap and komi subtraction. We update these modified skills according to the game's outcome (which will be a match between players closer in skill than without the handicap modification). Then, we set the final skills as the updated version plus the handicap term we deducted (see Figure 5.3).

5.1.2.3 Choosing the parameters

Since TrueSkill$^{\text{TM}}$ was last used for Chess problems, we had access to parameters optimized for this problem. We started from these, and modified them until we obtained good results.

We picked an initial mean of 1500, and initial standard deviation of 400. These are the values a new player starts with. For the handicaps, we have 10 for Komi factor, and 45 for the Handicap factor. The draw probability is zero.

The β parameter was defined by Herbrich (2007) as the length of the 'skill chain'. In other words, it indicates how wide each skill class is, in terms of skill points. A player with β more skill points than another player has an 80% chance of winning against that person. In high-skill games, where smaller differences in points lead to this 80% : 20% ratio, we need small beta values. In our model a value of 400 was used.

In the TrueSkill$^{\text{TM}}$ system, the uncertainty about players' skills decreases over time, as we acquire more data about the players. It is almost impossible

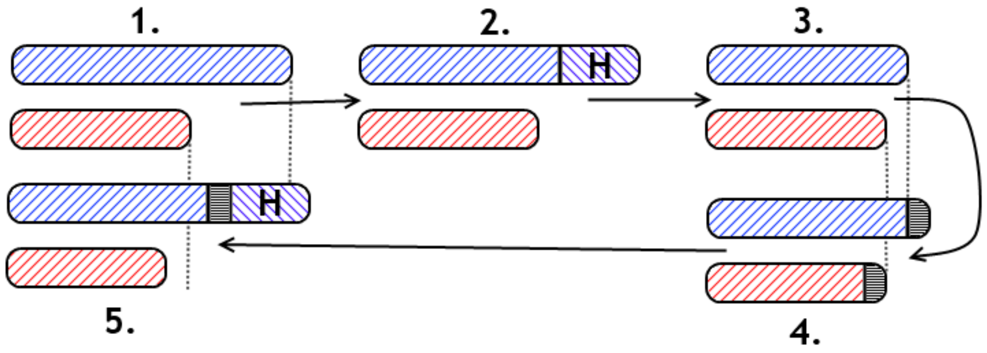


Figure 5.3: **Updating the skills of two players**. The initial values are shown in part 1, blue for one player and red for other. We assume the estimates have equal variances, for simplicity. The handicap value is identified in part 2, and subtracted in part 3. Then the algorithm computes the updates, shown in black horizontal lines. We assume blue player wins, hence the system will increase his estimate and lower red player's estimate. The updates have the same absolute value here because we assumed the variances are equal. Then, in part 5, we add back the handicap factor to obtain the final values for the skills of the two players.

for a player who started poorly to reach a high skill value, because the system becomes very certain about him after a few games. To solve this problem, the dynamic factor τ is introduced. Without it, the TrueSkill$^{\text{TM}}$ algorithm would always cause the standard deviation of players to shrink. Before the updates are computed, τ^2 is added to the variance of the players. Thus, a larger τ would cause players to change skill values faster. The TrueSkill$^{\text{TM}}$ default value is the initial standard deviation divided by 100. We used the algorithm on a dataset with a small number of minimum matches per player, so we picked $\tau = 15$ to avoid slow convergence.

5.2 Assumed-density filtering

Having described the TrueSkill$^{\text{TM}}$ algorithm, and how it was modified to work on our dataset, we will continue by introducing our first Bayesian model. It has similar theoretical background, its core technique being the Assumed-density filtering, from which Herbrich et al. (2007) derived the Gaussian density filtering used in TrueSkill$^{\text{TM}}$. Constructing the algorithm will enable us to customize it, e.g. by adding additional factors.

5.2.1 Algorithm description

Assumed-density filtering, commonly encountered in conjunction with terms such as *moment matching* or *online Bayesian learning*, is a method that can be used to approximate posteriors in Bayesian models. It needs a joint distribution over some observed variables D and hidden parameters S, and it computes an approximation for the posterior $p(S|D)$. The following description of ADF is based on the algorithm presented in Minka (2001).

We note with S the vector of skills we want to find, and D is the data we have observed. The joint distribution of S and the N results of the matches $D = \{r_1, \cdots r_N\}$ is then

$$p(D, S) = p(S) \prod_i p(r_i|S), \tag{5.12}$$

where r_j is the result of match number j.

For ADF, we need to factor this distribution into a product of factors:

$$p(D, S) = f_0 \cdot \prod_i f_i \quad \text{, so we choose} \tag{5.13}$$

$$f_0 = p(S) \qquad \text{as the prior, and} \tag{5.14}$$

$$f_i(S) = p(r_i|S) \qquad \text{as the other factors.} \tag{5.15}$$

Next, a distribution from the exponential family is chosen to approximate the posterior (the *assumed* density). The simplest choice available is the Gaussian distribution, also used by Glickman (1999), TrueSkill$^{\text{TM}}$ (Herbrich et al., 2007) and ultimately TrueSkill through time (Dangauthier, Herbrich, Minka, and Graepel, 2008). Consequently, we pick:

$$q(S) = N(S; m, V). \tag{5.16}$$

We start off by using a prior to initialise the posterior, and sequence through all the factors, updating and incorporating each one into our posterior. In every step, we start with a Gaussian prior $q(S)$, and we use the new observation's likelihood $f_i(S)$ to obtain an approximate posterior $q^{new}(S)$. Updating the prior term is easy, and for the others the exact posterior is

$$\hat{p}(S) = \frac{f_i(S)q(S)}{\int_S f_i(S)q(S)\,dS}. \tag{5.17}$$

q^{new} is found by minimising the KL divergence $KL(\hat{p}(S)||q^{new}(S))$, and considering that the new approximate posterior should also be a Gaussian distribution. The q^{new} obtained after using the last observation from D is the final approximation of the ADF algorithm.

5.2.2 Applying ADF to rating

The first thing we would like to point out is that we can reduce the scope of the updates by considering only the two players directly involved in a match $r_i = \{p_A \text{ vs. } p_B\}$.

The KL divergence can be split into two terms (see Appendix A.1). Minimising them keeps the skills of the players that are not involved constant, and henceforth we are free to update with respect to p_A and p_B only. The first KL divergence in A.8 is solved by matching the mean and covariance of the two two-dimensional distributions, $\hat{p}(S_A, S_B)$ and $q^{new}(S_A, S_B)$.

Minka (2001) computes the mean and covariance of the updated posterior q^{new} using the gradient of

$$Z_i = Z_i(m_{AB}, V_{AB}) = \int_{S_{AB}} f_i(S_A, S_B) q(S_A, S_B) \text{ , resulting in} \tag{5.18}$$

$$m_{AB}^{new} = m_{AB} + V_{AB} \nabla_m \log Z_i \tag{5.19}$$

$$V_{AB}^{new} = V_{AB} - V_{AB}(\nabla_m \nabla_m^T - 2\nabla_v \log Z_i) V_{AB}. \tag{5.20}$$

For computing the integral and the gradients, we employ the following approximation from MacKay (1992)

$$\int_s \sigma(s) N(s; m, v^2)\, ds \approx \sigma\left(\frac{m}{\sqrt{1 + (\pi/8)v^2}}\right). \tag{5.21}$$

The approximation is useful because $q(S_A, S_B)$ is gaussian, and $f_i(S_A, S_B)$ is a sigmoid function. However, f_i does not fit well into equation 5.21 because, in the simplest case, $f_i(S_A, S_B) = \sigma(S_A - S_B)$. We need to generalise and we do so by considering the argument of f_i to be the dot product of:

- x, a vector of constants and
- S_{AB}, a vector containing the skills of players A and B.

Then the approximation 5.21 can be generalised to :

$$\int_S \sigma(x^T S) N(S; m, V)\, dS \approx \sigma \underbrace{\left(\frac{x^T m}{\sqrt{1 + (\pi/8) x^T V x}} \right)}_{z} = \sigma(z). \tag{5.22}$$

Introducing x for writing the argument of the sigmoid is a mathematical trick which helps computations. An example of usage is shown in equation 5.29.

In this case of rank one derivatives, for $Z_i \approx \sigma(z)$, Minka (2008) suggests that computations can be simplified by using α and β as follows:

$$\nabla_m \log Z_i = \alpha_i x_i \tag{5.23}$$

$$\alpha_i = \frac{\sigma(-z_i)}{\sqrt{1 + (\pi/8) x^T V x}} \tag{5.24}$$

$$\nabla_m \nabla_m^T - 2\nabla_v \log Z_i = \beta_i x_i x_i^T \tag{5.25}$$

$$\beta_i = \alpha_i \frac{\alpha_i + (\pi/8) x^T m^{new}}{\sqrt{1 + (\pi/8) x^T V x}}. \tag{5.26}$$

The final ADF algorithm is shown below:

Algorithm 1: ADF algorithm

```
1 Initialise $m = 1$, $V = I$
2 Set constant $x$ such that $\sigma(x^T S_{AB}) = p(p_A > p_B)$
  for each match $r_i \in D$ do
3     Extract $m_{AB}$ and $V_{AB}$, the information for the two players
      (such that $p_A$ is the winner)
4     $z_i = \frac{x^T m_{AB}}{\sqrt{1 + (\pi/8) x^T V_{AB} x}}$
5     Update $m_{AB}, V_{AB}$ according to :
          $\alpha_i = \ldots$    (eq 5.24)    $m_{AB}^{new} = m_{AB} + V_{AB} \cdot \alpha_i \cdot x$
          $\beta_i = \ldots$     (eq 5.26)    $V_{AB}^{new} = V_{AB} - V_{AB} \cdot (\beta_i x x^T) \cdot V_{AB}$
```

Note that because we have swapped players around such that the first player (p_A) is always the winner, we can find x like in equation 5.29. In that example, the update

$$\Delta m_{AB} = m_{AB}^{new} - m_{AB} = V_{AB} \cdot \alpha_i \cdot x \tag{5.27}$$

(α_i is a number) will thus be a 2×1 vector, with the first component a positive number, and the second one negative. Consequently, the skill for the winner will be updated by increasing the previous value, and the loser will have his rating decreased.

5.2.3 One factor ADF model

For the simple model,

$$p(p_A > p_B) = \sigma(S_A - S_B) \tag{5.28}$$

$$S_A - S_B = \begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} S_A \\ S_B \end{pmatrix} \tag{5.29}$$

$$\Rightarrow \quad x = (1 - 1)^T,\ S_{AB} = \begin{pmatrix} S_A \\ S_B \end{pmatrix}. \tag{5.30}$$

The mean of S_{AB} will consist of the means for the skills of the two players, and the covariance matrix will be as below :

$$m_{AB} = \begin{pmatrix} m_A \\ m_B \end{pmatrix} \qquad V_{AB} = \begin{pmatrix} v_A^2 & 0 \\ 0 & v_B^2 \end{pmatrix}. \tag{5.31}$$

We consider only the diagonal terms from the covariance matrix, ignoring the correlations between players. This is a simplification which leads to a faster algorithm, and smaller memory requirements. Keeping track of the correlations might provide higher accuracy (Birlutiu and Heskes, 2007), but is outside the scope of the project, whose main goal is to investigate the benefits of introducing more factors in the vanilla models.

A quick check reveals that, for example, the $x^T V_{AB} x$ term in the approximation is

$$\begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} v_A^2 & 0 \\ 0 & v_B^2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = v_A^2 + v_B^2, \tag{5.32}$$

which can be double checked with the original approximation 5.21, considering the formula for the difference between two Gaussian distributions.

5.2.4 ADF with more factors

For a three factors model, we consider the following terms:

$$\text{Player A} \rightarrow \begin{pmatrix} S_A \\ O_A \\ W_A \end{pmatrix} \text{ with means } m_A = \begin{pmatrix} m_{S_A} \\ m_{O_A} \\ m_{W_A} \end{pmatrix} \tag{5.33}$$

$$\text{and standard deviations } v_A = \begin{pmatrix} v_{S_A} \\ v_{O_A} \\ v_{W_A} \end{pmatrix}; \tag{5.34}$$

The covariance matrix is:

$$V_A = \begin{pmatrix} v_{S_A}^2 & 0 & 0 \\ 0 & v_{O_A}^2 & 0 \\ 0 & 0 & v_{W_A}^2 \end{pmatrix}. \tag{5.35}$$

For simplicity we have again assumed independence, now between each factor. As future work, it would be interesting to use the full covariance matrix and check for anti-correlations between the factors.

The win probability is

$$p(p_A > p_B) = \sigma(S_A - S_B + O_A \cdot W_B - O_B \cdot W_A), \tag{5.36}$$

and we can thus choose

$$x = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} S_{AB} = \begin{pmatrix} S_A \\ S_B \\ O_A \cdot W_B \\ O_B \cdot W_A \end{pmatrix} \rightarrow \tag{5.37}$$

$$m_{AB} = \begin{pmatrix} m_{S_A} \\ m_{S_B} \\ \dfrac{m_{O_A} v_{W_B}^2 + m_{W_B} v_{O_A}^2}{v_{W_B}^2 + v_{O_A}^2} \\ \dfrac{m_{O_B} v_{W_A}^2 + m_{W_A} v_{O_B}^2}{v_{W_A}^2 + v_{O_B}^2} \end{pmatrix} v_{AB} = \begin{pmatrix} v_{S_A} \\ v_{S_B} \\ \dfrac{v_{O_A} v_{W_B}}{\sqrt{v_{W_B}^2 + v_{O_A}^2}} \\ \dfrac{v_{O_B} v_{W_A}}{\sqrt{v_{W_A}^2 + v_{O_B}^2}} \end{pmatrix}. \tag{5.38}$$

This can be used to compute the approximation

$$Z_i = \int \sigma(x^T S_{AB}) N(S_{AB}; m_{AB}, V_{AB}) \approx \sigma(z_i), \tag{5.39}$$

where V_{AB} is the matrix with elements of v_{AB} on the diagonal, and zero for the rest. However, it is not appropriate to use α and β for updating, because we are interested in updating each of the six parameters individually (three for each player), rather than updating m_{AB}. So we compute each gradient separately, for example :

$$\nabla_{m_A} = \sigma(-z) \cdot \begin{pmatrix} 1 \\ \dfrac{v_{W_B}^2}{v_{W_B}^2 + v_{O_A}^2} \\ \dfrac{v_{O_B}^2}{v_{W_A}^2 + v_{O_B}^2} \end{pmatrix} \tag{5.40}$$

and use the updating rules:

$$m_A^{new} = m_A + V_A \nabla_m \log Z_i \tag{5.41}$$

$$V_A^{new} = V_A - V_A(\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)V_A. \tag{5.42}$$

5.2.5 Problems

The first obvious disadvantage of the method described above would be that inference strongly depends on the order chosen for the updates. The ADF model will give different results if we reverse two games, even if they happen in the same day, maybe even a few minutes apart. It is unlikely the skills of the players change that fast. If we do not possess any mechanism for keeping track of the skill evolution in time, it is reasonable to assume that the order of game outcomes in a given day, for example, has a reduced importance. Therefore, the results of inference should be independent of the order of games within a day.

Another restriction is that information is only propagated forward in time. For instance, if A and B are very good friends and decide to enter the system, they may play a lot of matches between them but none against other opponents. After these, updating their skill parameters would certainly provide a good grasp of their relative compared strengths, but we don't know how they relate to the community. Suppose now that player A starts to play against other people, its rating changes but the rating of B does not. The TrueSkill$^{\text{TM}}$ and ADF algorithms presented do not propagate that information backwards in time to correct player A's skill estimate. Possible solutions to this issue would be to make several passes of the algorithm, to extend the model to incorporate and use covariance information, or the algorithm we will present in the next section.

5.3 Expectation Propagation

The two issues mentioned in the previous section can be directly addressed by extending the Assumed density filtering to running full expectation propagation (EP) until convergence. This method, described by Minka (2001), was successfully used by Dangauthier, Herbrich, Minka, and Graepel (2008) for their *TrueSkill through time* algorithm. In this section we will describe how we applied EP for inferring the skills of the players, and the equations for a multiple factors model.

The basic idea of expectation propagation is to update repeatedly on the same game outcomes but making sure that the effect of the previous update on that game outcome is removed before the new effect is added (Dangauthier et al., 2008).

The posterior distribution was not Gaussian; but if the factors involved were normal distributions (scaled by a constant), then we would be able to compute the posterior analytically. One way is to take each factor in our model and approximate them one by one with Gaussians, using the knowledge from the other factors.

One might wonder, how approximating a sigmoid with a normal distribution might give a decent estimate. This happens because the estimation is done in the context of all the other factors: when we multiply all of them together, the posterior distribution that summarizes the other factors will become very compact. Its variance would be very small, compared to any single likelihood term. When we make our approximation for one term, we are interested in its product with the summarizing posterior constructed using information from the other factors. Thus, we do not need to capture all the details in every individual likelihood function, all we really need to get a good approximation is to represent very well the zone that overlays with the posterior. An example is shown in Figure 5.4.

EP repeatedly iterates (in no particular order) over the outcomes and refines the approximations for the terms until convergence. The resulting approximation will be less dependent on the games' order.

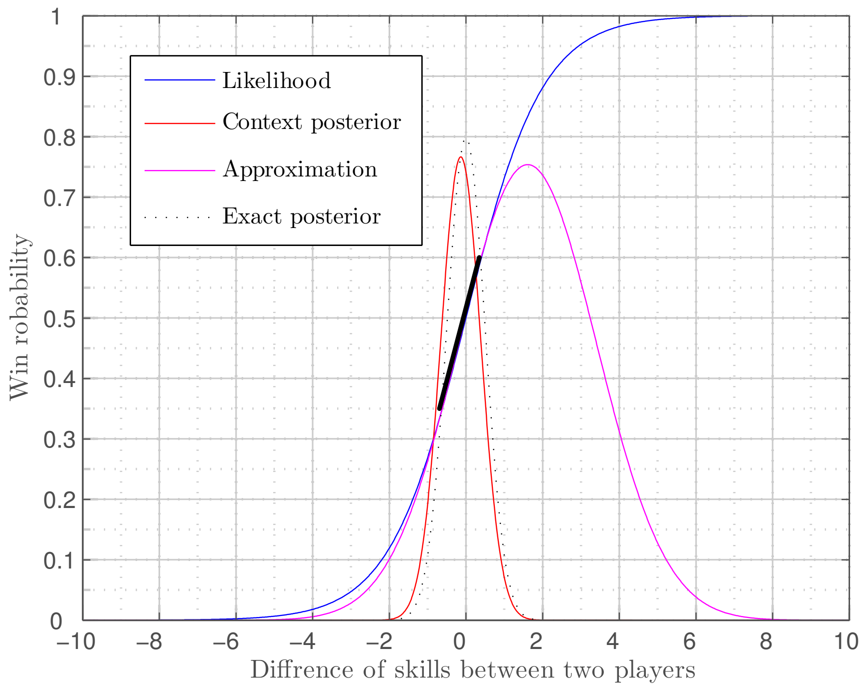


Figure 5.4: Example for approximating the likelihood of an outcome in EP. The approximation will be later noted with $\tilde{f}_i$ (see 5.43), and the context posterior with $q^{\setminus i}$; this $q^{\setminus i}$ is the exact posterior, from which the effect of term i is removed (see 5.45).

5.3.1 Implementation

We continue to use the notation from Minka (2008), as our algorithm is a direct extension of the EP implemented there, and shares similar equations.

Similar to the ADF setting, we have the factors $f_i(S_{AB}) = \sigma(S_A - S_B + O_A \cdot W_B - O_B \cdot W_A)$ (for three factors) which we want to approximate with

$$\tilde{f}_i(S_{AB}) = c_i \exp(-\frac{1}{2}(S_{AB} - m_i)^T V_i^{-1}(S_{AB} - m_i)). \tag{5.43}$$

We are always able to do this, because we can define the approximation of the factor as proportional to the new posterior (we will drop the ^{new} notation from now on) over the old posterior:

$$\tilde{f}_i(S_{AB}) \propto \frac{q^{new}(S_{AB})}{q^{\setminus i}(S_{AB})}. \tag{5.44}$$

$q^{\setminus i}$ is the posterior from which we have removed the effect of the term i. If the posterior over S_{AB} is $q(S_{AB}) \sim N(S_{AB}; m, V)$ then

$$q^{\setminus i}(S_{AB}) \sim N(S_{AB}; m^{\setminus i}, V^{\setminus i}) \propto \frac{q(S_{AB})}{\tilde{f}_i(S_{AB})} \tag{5.45}$$

$$(V^{\setminus i})^{-1} = V^{-1} - V_i^{-1} \tag{5.46}$$

$$m^{\setminus i} = V^{\setminus i}(V^{-1}m - V_i^{-1}m_i). \tag{5.47}$$

The new posterior is computed via ADF, using the exact posterior $\hat{p}$:

$$Z_i(m^{\setminus i}, V^{\setminus i}) = \int_{S_{AB}} f_i(S_{AB}) \cdot q^{\setminus i}(S_{AB})\, dS_{AB} \tag{5.48}$$

$$\hat{p}(S_{AB}) = f_i(S_{AB}) \cdot q^{\setminus i}(S_{AB}) \tag{5.49}$$

$$q(S_{AB}) = \operatorname*{argmin}_q KL(\hat{p}(S_{AB})||q(S_{AB})) \tag{5.50}$$

$$m = m^{\setminus i} + V^{\setminus i}\nabla_m \log Z_i \tag{5.51}$$

$$V = V^{\setminus i} - V^{\setminus i}(\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)V^{\setminus i}. \tag{5.52}$$

In all of the above equations, we have used S_{AB} instead of S, meaning that for every match we update only the information for the two players directly involved. This is correct, because we can prove that minimising the KL divergence for the whole set of players is equivalent to keeping $S_{\setminus AB}$ unmodified and minimising the KL divergence for S_{AB} (see Appendix A.2).

Finally, we update the approximation, using, as in (Minka, 2008):

$$V_i^{-1} = V^{-1} - (V^{\setminus i})^{-1} \tag{5.53}$$

$$V_i = (\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)^{-1} - V^{\setminus i} \tag{5.54}$$

$$m_i = V_i(V^{-1}m - (V^{\setminus i})^{-1}m^{\setminus i}) \tag{5.55}$$

$$= m^{\setminus i} + (V_i + V^{\setminus i})(V^{\setminus i})^{-1}(m - m^{\setminus i}) \tag{5.56}$$

$$= m^{\setminus i} + (\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)^{-1}\nabla_m \log Z_i \tag{5.57}$$

$$c_i = Z_i|I + V^{\setminus i}V_i^{-1}|^{1/2} \exp(\frac{1}{2}\nabla_m^T(\nabla_m \nabla_m^T - 2\nabla_v)^{-1}\nabla_m \log Z_i) \tag{5.58}$$

The final EP algorithm is shown below:

Algorithm 2: EP algorithm

```
1 Initialise the prior term
2 Initialise the other terms to 1 :
      $m_i = 0$                                   (5.59)
      $s_i = 1$                                   (5.60)
      $V_i = 10^{20} \cdot I$                     (5.61)
3 Initialise posterior :
      $m = m_{prior}, V = V_{prior}$              (5.62)
  while $all\ \{m_i, V_i, c_i\}\ not\ converged$ do
      for $i = 1, \dots N$ do
4         Set $m$ and $V$ with only the information for the two players
5         Remove $\tilde{f}_i$ from the posterior, to get an old posterior
              $V^{\setminus i} = (V^{-1} - V_i^{-1})^{-1}$
              $m^{\setminus i} = V^{\setminus i}(V^{-1}m - V_i^{-1}m_i)$
6         Compute the posterior (same as ADF), using the old posterior and $f_i$
              $Z_i(m^{\setminus i}, V^{\setminus i}) = \int_{S_{AB}} f_i(S_{AB}) \cdot q^{\setminus i}(S_{AB})\, dS_{AB}$
              $m = m^{\setminus i} + V^{\setminus i}\nabla_m \log Z_i$
              $V = V^{\setminus i} - V^{\setminus i}(\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)V^{\setminus i}$
7         Update the approximation $\tilde{f}_i$ for $f_i$
              $V_i = (\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)^{-1} - V^{\setminus i}$
              $m_i = m^{\setminus i} + (\nabla_m \nabla_m^T - 2\nabla_v \log Z_i)^{-1}\nabla_m \log Z_i$
              $c_i = Z_i|I + V^{\setminus i}V_i^{-1}|^{1/2} \exp(\frac{1}{2}\nabla_m^T(\nabla_m \nabla_m^T - 2\nabla_v)^{-1}\nabla_m \log Z_i)$
```

In the next section we will continue with evaluating the models we built in this chapter, and presenting the results we have obtained.

5.4 Evaluation

5.4.1 Making predictions

We have mentioned before two different methods for predicting game outcomes, once we inferred the skills and variances of the players:

- $$p(p_A > p_B) = \int_{-\infty}^{\infty} \mathbf{1}(S_A - S_B > 0) p(S_A; \mu_A, \sigma_A) p(S_B; \mu_B, \sigma_B)\, dS_A\, dS_B$$
  $$= \int_0^{\infty} N(D; \mu_A - \mu_B, \sigma_D^2) dD, \text{ and}$$
- $$p(p_A > p_B) = \int_{-\infty}^{\infty} \sigma(S_A - S_B) p(S_A; \mu_A, \sigma_A) p(S_B; \mu_B, \sigma_B)\, dS_A\, dS_B.$$

We found the first one to be more sensitive to the player's deviations, and thus sensible to changes in parameters. The best results were obtained by artificially increasing variances of the players when making predictions. Otherwise, the systems tend to be extremely overconfident.

For example, for the first metric we used:

$$p(p_A > p_B) = \int_0^{\infty} N(D; \mu_A - \mu_B, (x \cdot \sigma_D + y)^2) dD, \tag{5.63}$$

where x and y are two constant we optimised on the validation set, using a conjugate gradient method.

In the experiments, this prediction method provided better results, compared to the one using the sigmoid function. In all of the following tests, unless stated otherwise, the results are obtained using the prediction technique from above.

5.4.2 Model initialisation

For both ADF and EP, the skills and variances are initialised in the beginning of the algorithm. This corresponds to the prior on these parameters, the belief we have about where the skills and variances should lie.

The strengths (skills in the 1 factor models, and the first factor in the other models) are initialised with 1, and have a standard deviation of 1. This choice is not very important, as this parameter is invariant to addition with a constant, since the systems depend only through the difference between players. However, the deviation of 1 reflects our belief about the skills of the players, because the skill range (best player minus worst player) is assumed to be around 6.

We want the product of the two extra factors (i.e., $O_A^i W_B^i$ where O comes from Offence, and W from Weakness) to represent the advantage player A would gain by using 'strategy' i against player B. A positive value would be easy to represent, so we think of O and W both as positive values, between 0 and some constant, 1 for example. Thus they were initialised with random numbers distributed with a mean of 0.5 and deviation 0.15. Having Gaussian beliefs about these factors may cause inconsistencies, as normal distributions may have some mass on negative numbers, which we ignore. However, using positive numbers is easier to interpret, and usually the variance of the Gaussians is small enough not to cause damage when the means are close to zero.

For multiple sweeps of the ADF algorithm, we kept the skill values as found out in the previous sweep, and increased the variances of the strengths to 1, and the variance of the factors to 0.1.

5.4.3 Dealing with negative variances

It is mentioned by Minka (2001) that the EP algorithm may fail on subsequent iterations due to negative variances. This can happen in equation 5.45, where we would end up with a resulting negative variance $V^{\setminus i}$, and we wouldn't be able to compute $Z_i(m^{\setminus i}, V^{\setminus i})$. Also it has been noted that sometimes the approximations may oscillate without reaching convergence, in the presence of many negative variances. Minka (2001) suggests modifying the variance for these terms to a very big number, which would practically ignore the respective observation ($V^{\setminus i} \approx V$).

We have to pay even more attention to this phenomenon, because our approximation of Z_i (5.22) uses a square root of a (weighted) sum of variances. And it might become negative even for a $V^{\setminus i}$ that has mostly positive terms. This is the main stability issue in our algorithm.

As we have mentioned, ignoring the correlations between players (and between factors) is probably diminishing the predictive ability of the algorithm. However, it is easier to control the stability if we assume independence and that the variance for the skills involved is diagonal.

We have decided against artificially increasing the variances like in the suggestion above, because even if it guarantees convergence, the results we obtain are worse than without imposing positivity. This happens particularly for players with few matches, where ignoring some games makes a decent difference on their skill estimations. We have found that if we upper limit the components of the posterior variance (V) to 2, for example, it prevents most of the problems without significantly impairing the performance. In this way, $V^{-1} - V_i^{-1}$ is less likely to be negative.

This works well for the 1 factor EP, but the multiple factors model tends to be less stable. To obtain convergence, we had to use additional constraints, such as:

- limit terms of $V^{\setminus i}$ to less than 2.5;
- before updating the approximations m_i, V_i, limit V to more than 0.1. Otherwise V^{-1} may get large.

5.4.4 Factor restrictions

Another issue concerns the additional factors, which in our model we interpreted as some positive values, between 0 and 1, for example. This is a problem for both ADF and EP models, and we have to keep the factors above zero.

For very set of factors we obtain

$$A \sim \begin{pmatrix} O_A^i \\ W_A^i \end{pmatrix} \qquad B \sim \begin{pmatrix} O_B^i \\ W_B^i \end{pmatrix}; \tag{5.64}$$

the model is dependent on the value of $O_A^i W_B^i - O_B^i W_A^i$, which may be interpreted as the determinant of the 2×2 matrix formed with the respective factors.

After inferring every set of factors, we do the following checks:

- if a factor is negative, make it small (10^{-3}) instead.
- if the determinant is bigger than a constant (5), divide all the factors such that the determinant will be less than that constant. For players distributed roughly between $(-3, 3)$, differences between their skills should not be larger than this value.
- likewise, limit the products across the diagonals (meaning both $O_A^i W_B^i$ and $O_B^i W_A^i$). There might be cases when the two products have big, close values,

resulting in a small difference, which passes unnoticed. No matter how better a player is, and how weak his opponent, $O_A^i W_B^i$ has no reason to be larger than 5.

5.5 Accuracy experiments and results

5.5.1 Comparing ADF models with different number of factors

We used the same datasets as in the previous evaluation part (section 4.5). The three factors model is the best, while five factors give roughly the same performance as the simple model; using more factors is not efficient, as we probably do not have enough data for a large number of factors. As future work, it would be interesting to construct a system which can use a different number of factors depending on the quantity of data it has; for example, a hierarchical model that can use nested models with varying number of factors.

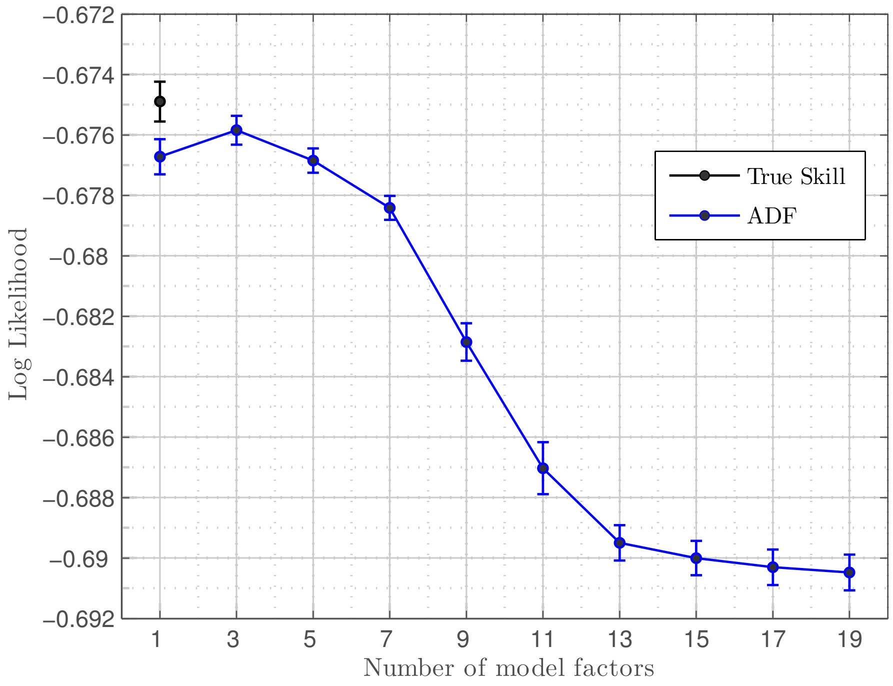


Figure 5.5: **ADF performance** for models with different number of factors. Errorbars on standard deviation corresponding to results averaged on 10 datasets. For reference, the best maximum likelihood performance is at −0.683 (Figure 4.1).

TrueSkill^{TM}, which basically uses ADF 1 factor plus some additional tricks, has a slightly better performance. Probably, the biggest difference is made by increasing the variance before each match, effectively simulating a simple time series model.

We should remember that the three factors maximum likelihood solution, which is the best in the previous chapter, has a log likelihood of about -0.683. On Figure 5.5 this value is around the performance of the 9 factors ADF, lower than both TrueSkill^{TM} and the best ADF model.

5.5.2 Extra iterations of ADF algorithms

Restricting the size of the dataset and using only players with more than 12 games may result in a rather small quantity of data, which affects the ADF algorithms in a negative way. We can emulate a twice as big dataset effectively by doing one more algorithm sweep on the same data. With this idea, we tested how well the ADF models would perform with increasing number of iterations.

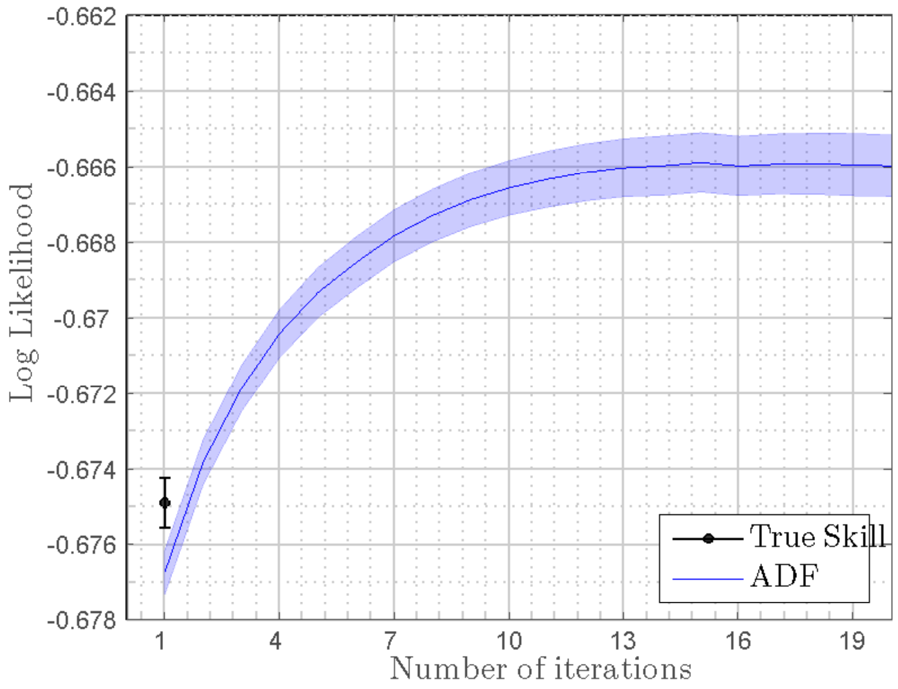


Figure 5.6: **ADF performance** for 1 factor, and different number of iterations. Errorbars on standard error from the 10 datasets, as before.

As we mentioned in section 5.4.2, after doing one sweep, we keep the means obtained but increase the variances, then do another iteration. This is similar to the dynamic factor mechanics in TrueSkill^{TM}, and prevents variances from reaching very small values, causing diminished updates.

As we can see in Figure 5.6, doing more iterations brings a massive benefit. Starting the algorithm again, but roughly knowing where the skills of the players lie (as a prior), makes better use of the information available. We can see the performance we gain over the previous iteration, in Figure 5.7. The biggest increase comes with the second sweep, then every further iteration brings decreasing advantage.

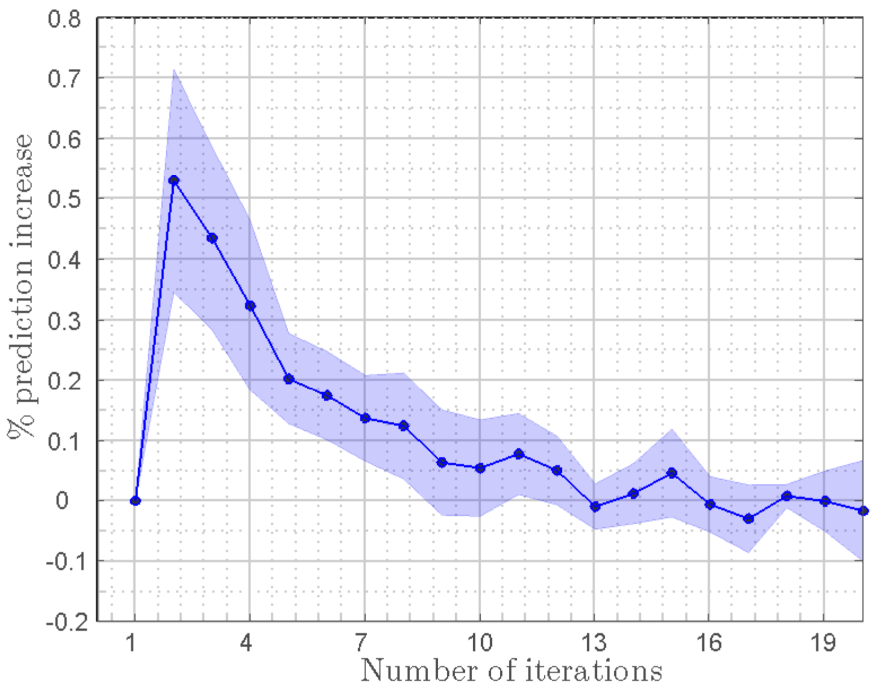


Figure 5.7: **ADF performance** for 1 factor, and different number of iterations. We have plotted the increment of the percentage of correct predictions: how much we gain by running the algorithm one more sweep through the data. This means that on the third sweep ADF gets about 0.43% more matches correctly predicted than on the second sweep, and 0.95% more than on the first iteration.

5.5.3 Combining multiple factors and extra iterations

Running the previous experiment on ADF models with more factors yields the performances in Figure 5.8. We expected the three factors model to be better than the initial model, and there is evidence this might be true, though the improvement is not statistically significant. The performance decreases with increasing the number of factors, but even so, the 11 factors model, for instance, is much better than only one iteration of ADF, or TrueSkill^{TM} (which is at -0.675).

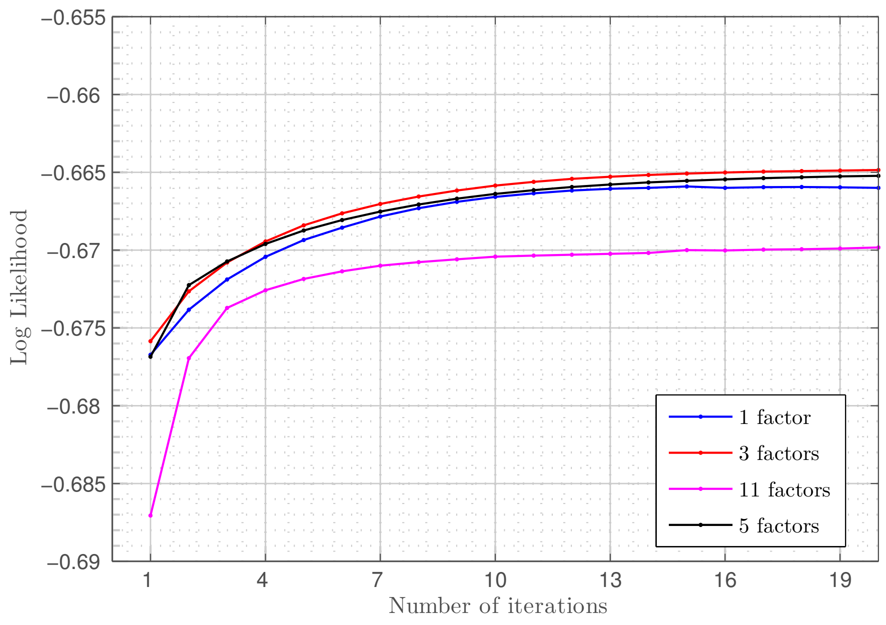


Figure 5.8: **ADF performance** for 1,3,5 and 11 factors, and different number of iterations. Standard error values are mostly less than 0.001. They do not appear in the figure because the closeness of the lines would make it difficult to read.

5.5.4 Expectation Propagation

Running Expectation Propagation on the same datasets as before leads to results in Figure 5.9. The three factors EP does much better than EP with 1 factor. Increasing the number of factors to more than three comes with a drop in performance, as there probably are not enough matches per player in the datasets.

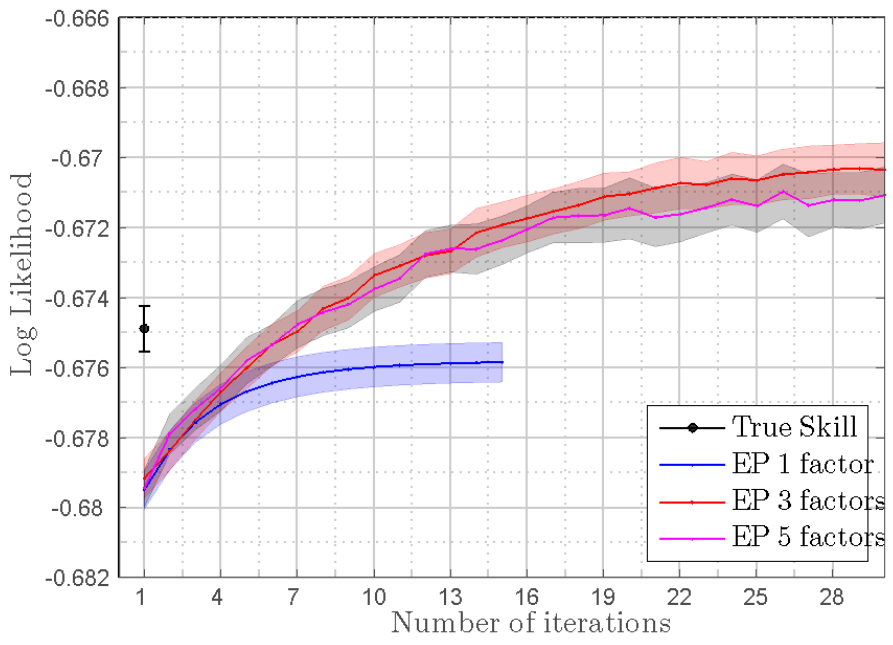


Figure 5.9: **EP performance** for 1,3 and 5 factors, and different number of iterations. Errorbars on standard errors corresponding to running the algorithms on the 10 datasets. EP 1 converges after 15 iterations. The more iterations ADF models are around the upper plot limit of -0.666 (Figure 5.8.)

EP with 1 factor converges faster than the versions with more factors, but also yields lower performance. In fact, it barely reaches TrueSkill$^{\text{TM}}$. This probably happens due to TrueSkill$^{\text{TM}}$'s increasing variance trick. However, when looking again at Figure 5.8 we can see that the best performance for ADF models with more iterations lie around the -0.666 value. This is still significantly better than EP. An explanation for this fact would be that for the outcome of EP, all games have roughly the same importance, while in TrueSkill$^{\text{TM}}$ and the ADF models games towards the end of the dataset have more influence on the resulting players' skills. Consequently, the sequential methods have an advantage at predicting matches that take place just after the training data. Probably including a time series model in the EP algorithm would cancel this advantage.

Nonetheless, we can also look back at Figure 4.1. The best performance obtained from the maximum likelihood methods is around -0.6835 which is a lot lower than the accuracy of all the models we have just described. Going Bayesian is fully justified and brings significant advantages over the maximum likelihood fitting of models.

Having obtained a good idea about how the algorithms relate to each other in terms of accuracy, we will try to provide more insight into how more factors compare to one factor, in the next section.

5.5.5 Increasing the size of the dataset

All the previous experiments have been realised using small datasets. They contain matches played over periods of 20 days, in an effort to minimise the impact of time on players' skills. While a befitting choice when comparing models with more factors to one factor version of themselves, this restriction also brings some disadvantages. Firstly, the relatively small number of games per player is probably a reason why increasing the number of factors for ADF models does not improve the accuracy as we would have expected. Expectation Propagation makes better use of the data available, and we can see a clear advantage for the three factors model. Assuming that the lack of data causes poor performance, especially when increasing the number of factors in ADF case, we test the algorithms on a larger dataset.

We increase the time range to 200 days, and otherwise keep the same restrictions as for the previous datasets. First of all, TrueSkill^{TM} is now better than ADF 1 factor, no matter how many iterations. This is not surprising, since the simple time series model included in TrueSkill becomes very important with increasing the time range of the games.

Secondly, doing more iterations is not always better, as after a point the models lose performance. Probably using the resulting skills as a prior contrasts too much with the actual skills of the players at that point, since the time difference is of 200 days.

In the third place, we can see that increasing the number of factors now brings better accuracy, as we have previously presumed. At its highest point, the ADF 3 factors is comparable to TrueSkill^{TM}, while there is evidence that ADF 7 factors is slightly better. At any rate, they both surpass 1 factor ADF.

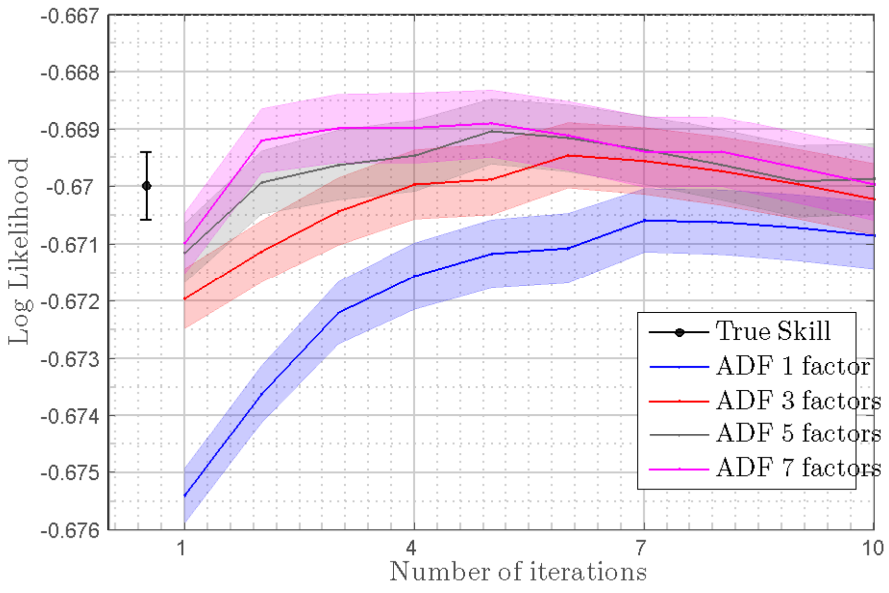


Figure 5.10: **ADF performance** for 1,3,5 and 11 factors, and different number of iterations. Errorbars on standard error corresponding to the games in the test set.

By contrasting how well TrueSkill$^{\mathrm{TM}}$ performs compared to ADF 1 factor on both small and large datasets, we can see that ADF is less accurate when increasing the number of days. From this we can infer the importance of a time series model, and the advantages even a simple method (as increasing the variance after every match – emulating a Gaussian diffusion) brings to the rating systems.

5.6 One factor vs. more factors

In the previous section we have examined the accuracies for the different algorithms implemented, and in particular, how well models with more factors compare to their one factor versions. In this section, we will continue with defining a few extra metrics which we will then use to further contrast the models. More specifically, we will look at ADF and EP with one and three factors.

5.6.1 Metrics

5.6.1.1 Difference between models

In the one factor model the winning probability is $P(p_A > p_B) = \sigma(S_A - S_B)$, while in the three factor models the same concept is expressed as $P(p_A > p_B) =$

$\sigma(S_A - S_B + O_A W_B - O_B W_A)$. After we fit these parameters for both models, the difference between these 'activations' would be

$$D = (S_A^3 - S_B^3 + O_A^3 W_B^3 - O_B^3 W_A^3) - (S_A^1 - S_B^1), \tag{5.65}$$

where upper indices represent the number of the model factors. Henceforth, this metric relates to how much the predictions of the two models will differ.

Histograms for both ADF and EP are provided in Figure 5.11. We can see that doing one iteration of ADF does not lead to great differences, while in EP the one and three factors versions yield more different activations. This is not surprising, since one iteration of ADF does not have enough time and data to fully take advantage of the more complex model.

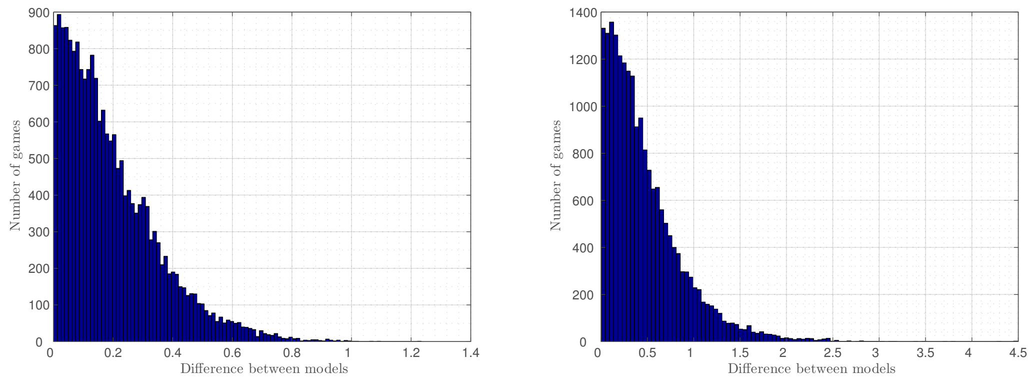


Figure 5.11: **Histogram on number of games**, function of the difference between likelihood of outcomes (D in equation 5.65). ADF in left figure, EP in right figure.

5.6.1.2 Importance of the extra terms

In the same settings as above, it would be interesting to look at how much the term with the factors $O_A^3 W_B^3 - O_B^3 W_A^3$ explains out of the whole 'skill difference' $S_A^3 - S_B^3 + O_A^3 W_B^3 - O_B^3 W_A^3$. For this, we have noted the *'extra term' importance* with

$$I = \frac{|O_A^3 W_B^3 - O_B^3 W_A^3|}{|O_A^3 W_B^3 - O_B^3 W_A^3| + |S_A^3 - S_B^3|} \tag{5.66}$$

Absolute values are mandatory, since sometimes the two term may have opposing signs, and even cancel out. The importance is a real number, $I \in [0, 1]$. Histograms for both ADF and EP are provided in Figure 5.12. They are very similar, and maybe there is small evidence EP has a few more games where the extra term is more important. Overall, in EP both terms have roughly the same importance on the given sets of games; In the ADF model, the extra term is a little less significant.

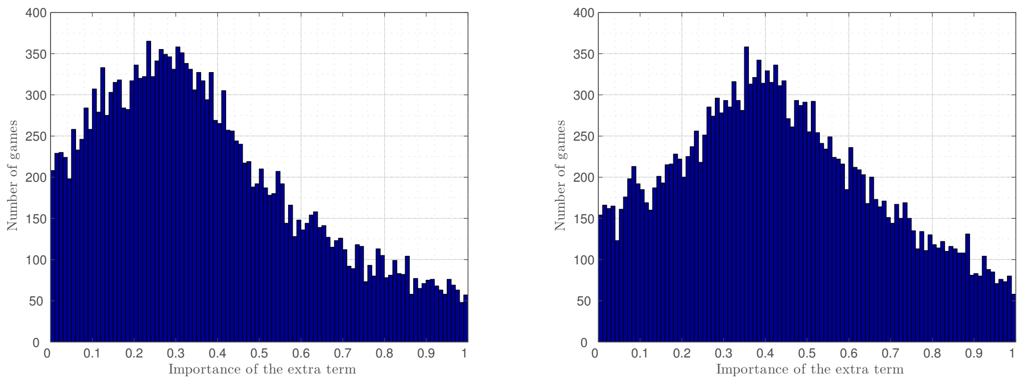


Figure 5.12: **Histogram on number of games**, function of the importance of the extra term provided by the factors (I in equation 5.66). ADF in left figure, EP in right figure.

5.6.1.3 Increase in prediction percentage

We also need a way to quantify how well (or how bad) games are predicted by the models, and for this task an easy choice is to look at the predictions :

$$P^1(p_A > p_B) = \sigma(S_A^1 - S_B^1) \tag{5.67}$$

$$P^3(p_A > p_B) = \sigma(S_A^3 - S_B^3 + O_A^3 W_B^3 - O_B^3 W_A^3) \tag{5.68}$$

$$\%Increase = P^3(p_A > p_B) - P^1(p_A > p_B), \tag{5.69}$$

in a game where p_A is the winner. Upper indices represent the number of factors of the model.

Histograms for both ADF and EP are provided in Figure 5.13. They look like normal distributions, with the mean slightly above zero (meaning that on average there is some small improvement over the 1 factor model). Negative values correspond to games where the more factors model has done worse.

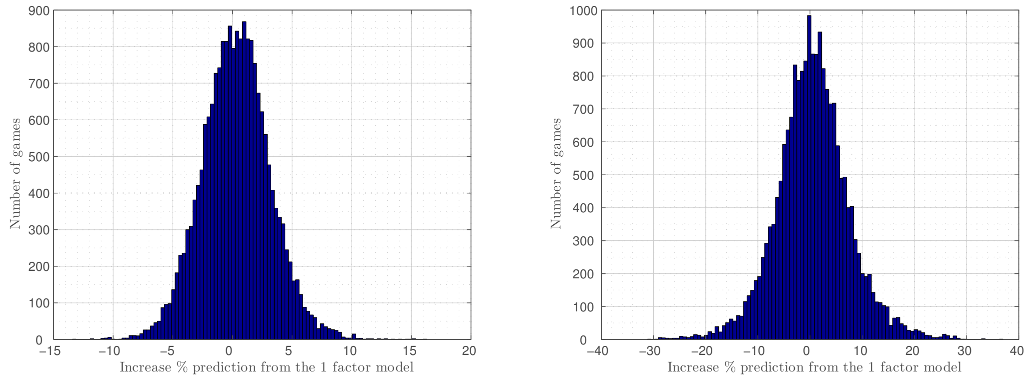
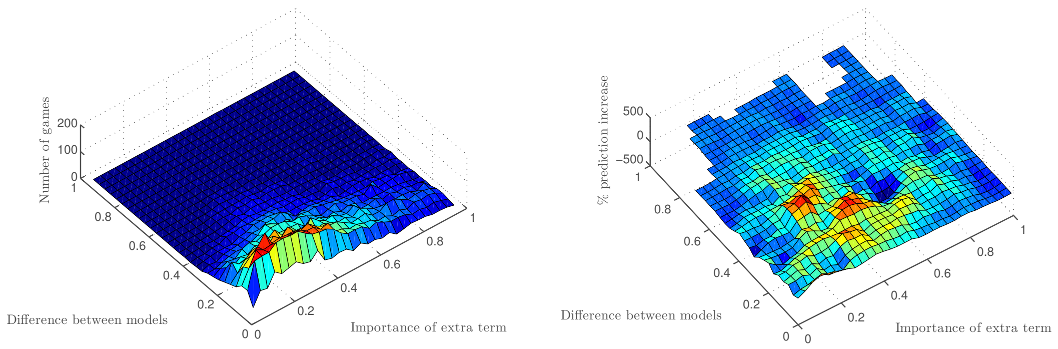


Figure 5.13: **Histogram on number of games**, function of the percentage increase in prediction of the model with 3 factors over the 1 factor (see equation 5.69). ADF in left figure, EP in right figure.

5.6.2 Combining the metrics

A histogram of the number of games for both importance of extra term and the absolute value of the difference between ADF models is shown below. The bulk of the matches do not differ much, and the extra term has a moderate importance.



(a) **Histogram on number of games**, function of both metrics for the ADF model.

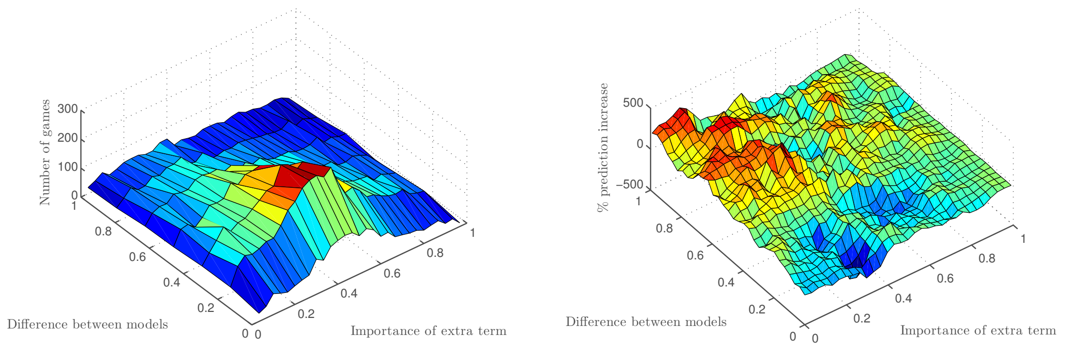
(b) **Percentage prediction increase** as function of the other two metrics. The height represents the sum of the percentage for the games in every bin. ADF model.

Figure 5.14: **ADF model**

Next, we plot the sum of percentage prediction increase for every bin, in Figure 5.14(b). Since in some zones there are very few games (as it can be seen in Figure 5.14(a)), showing just the mean wouldn't be very useful. We would probably need more games per bin to obtain a relevant figure.

We can observe that for bigger differences (about $0.4 - 0.5$ on the Y axis), there are the biggest gains, even if there are significantly fewer games than on the smaller difference zone (also see Figure 5.11). For values higher than 0.5, the number of games is too small to have any impact.

A similar histogram for EP (5.15(a)) shows a distribution of the games which is comparable to the ADF case. The difference would be that there are more games with bigger differences when moving from the 1 factor model to 3 factors. This was easily identifiable in Figure 5.11.



(a) **Histogram on number of games**, function of both metrics for the EP model.

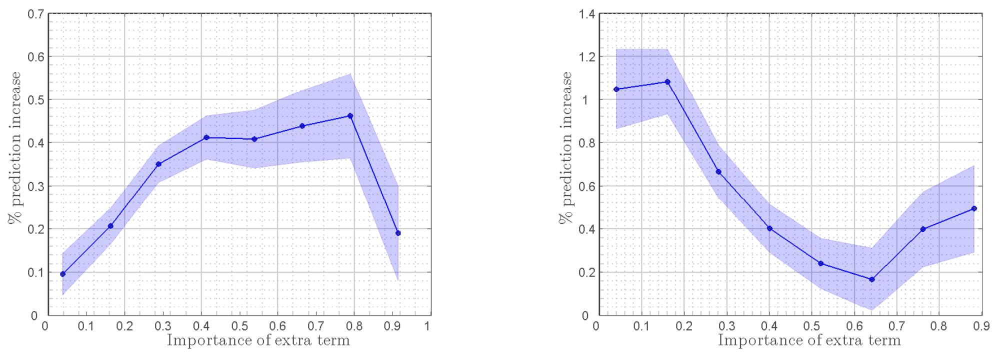
(b) **Percentage prediction increase** as function of the other two metrics. The height represents the sum of the percentage for the games in every bin. EP model.

Figure 5.15: **EP model**

The games that differ most from the 1 factor model bring, again, the biggest gain (Figure 5.15(b)). This behaviour is even more accentuated than for the ADF case, but probably running until convergence brings more differences between models than only one sweep through the data, as in the ADF instance.

5.6.3 Benefit of extra term in prediction performance

The gain in prediction percentage as a function of the importance of the extra term is shown in Figure 5.16(a) for ADF and in Figure 5.16(b) for EP. The result are rather contrasting, with ADF having better predictions as the extra term becomes predominant. On the other hand, EP gains less for match-ups where the extra term represents the majority of the skill difference between players.



(a) **ADF model** (b) **EP model**

Figure 5.16: **Increase of prediction percentage** for the three factors model compared to one factor model, as a function of the importance of the extra terms involved.

5.6.4 Players with close skill values

One hypothesis is that for players very close in skill – as predicted by a one factor model – fitting a number of additional factors would lead to better accuracy. The extra factors might provide information about the players that the rating system could use to better predict otherwise very balanced games.

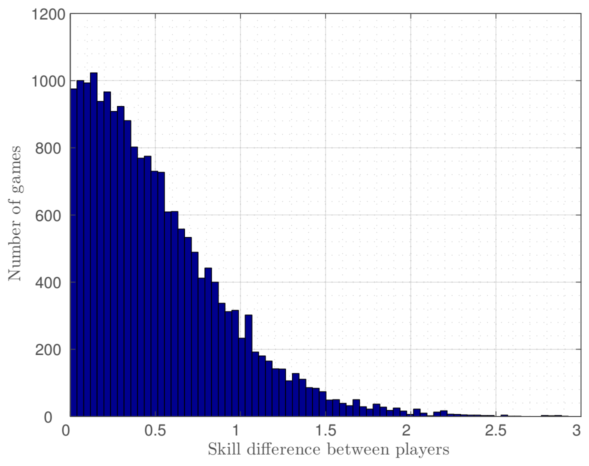


Figure 5.17: **Number of games** between players expected to have a certain difference in skills. Values are computed on one dataset, using an EP model with one factor.

We are interested in matches with players that have similar strengths. A histogram of the difference of skills for all matches in the test set, using an EP 1 factor model, is shown in Figure 5.17.

We fit using an EP model with 3 factors, and then compare the performances. In Figure 5.18 we look at how well they both do on matches between players with similar skill values, as predicted by the first EP model. As expected, the model with more factors is better at judging very close matches, and has similar performance on the rest of the games. We should also note that there are more close games (on which EP with more factors does better), with about half of the games being in the $[0, 0.4]$ interval. These are the balanced games which we would naturally want to predict better.

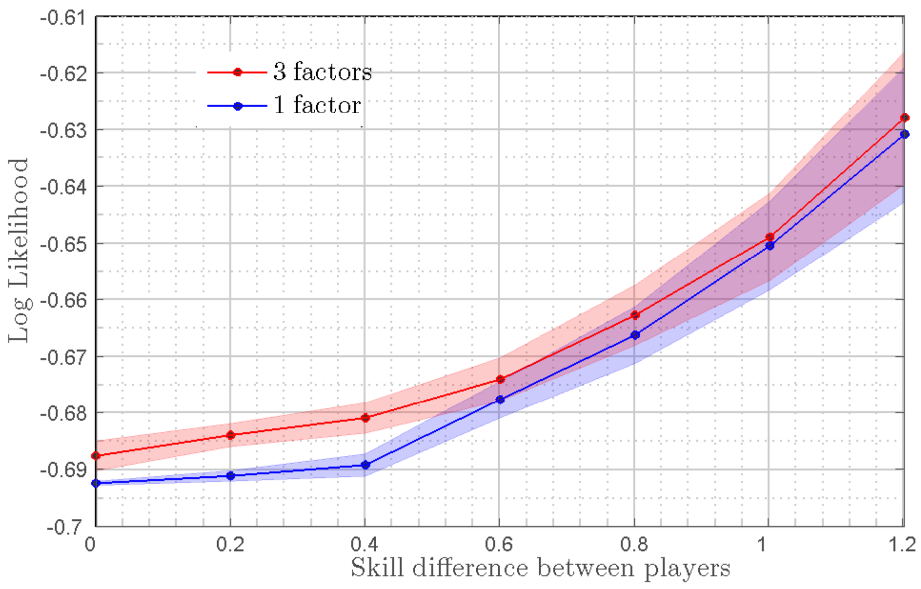


Figure 5.18: **Log Likelihood** for EP 1 factor and EP 3 factors models, as a function of skill difference between players. Experiments were done on a single dataset, with points corresponding to the means of each bin. The bins are equally spaced (0.2 skill difference each), and the errorbars represent the standard error for games in each bin.

The errorbars in Figure 5.18 are done on the games in each bin; some of the games in a bin may be predicted better by the EP with three factors, some by the model with only one factor, as long as on average the three factors model gives higher log likelihood. We would like to confirm the affirmations in the previous paragraph more statistically solid; in the next experiment we measure the percentage prediction increase on each game. Indeed, the gain is bigger for matches with small skill difference between players (Figure 5.19). This proves our hypothesis right, and indeed more factors help predict balanced matches better.

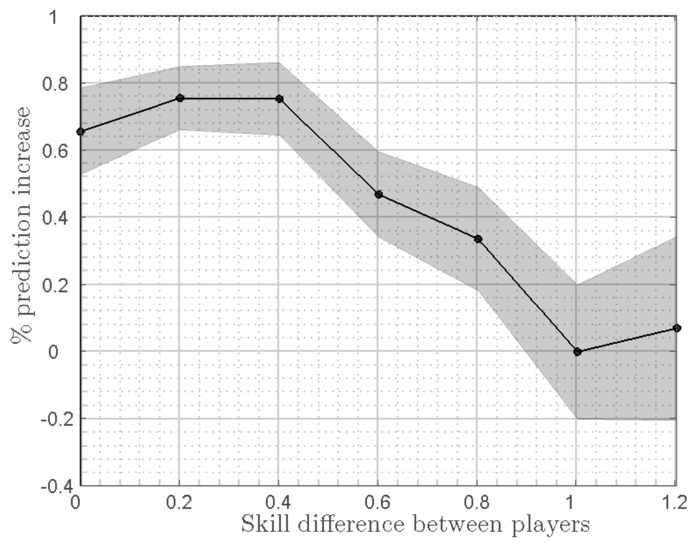


Figure 5.19: **Percentage prediction increase** for EP 3 factors over EP 1 factor model, as a function of skill difference between players. Same conditions as in the previous figure.

5.6.5 Highly skilled players

Another hypothesis is that in high-level matches, where the best players are involved, the difference in skills could be less significant compared to the proficiency of the players for different strategies. It is possible that a model with more factors will perform better on these types of games.

A simple way to pinpoint matches with good players is to compute the average skill for the two competitors, as identified by a model with one factor. The associated histogram is shown in Figure 5.20.

Using this metric, the one factor model identifies the competitive level of the games. Next, for the same matches, we look at the importance of the extra term in a three factor model. Figure 5.21 shows that as players improve, the factors become more important compared to just the skill difference. As we have predicted, the factor model gives more weight to the factors for games with better players. Also, it should be noted that for very poor players, the factors are more relevant than the skill difference, too. Consequently, it seems that just the skill difference is a slightly less relevant for players that are very good, or very bad.

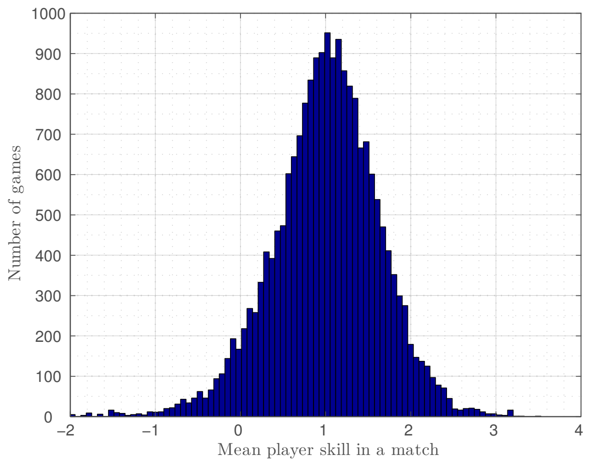


Figure 5.20: **Number of games** corresponding to different levels of average player skill. For each game, the mean of the skills of the two players is calculated. The skills have been computed using EP with one factor.

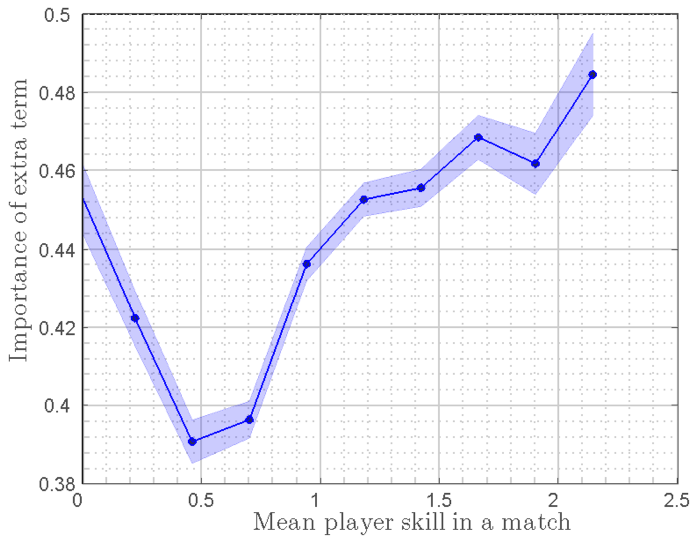


Figure 5.21: **Importance of the extra term** for different skill levels involved in a game, computed with a 3 factors EP model. The matches have been ranked using EP with one factor. Errorbars on the standard error for games in each bin.

However, looking at how well the 3 factors EP and 1 factor EP predict the results of the same games (Figures 5.22 and 5.23), we can see that the 3 factor model is actually better only on the games with weaker players. The accuracy for matches with better players is similar for the two models. We can conclude that possibly for less strong players the strategy and style of play is more important than a small difference in the skill value, and there is evidence that this assumption does not hold with very good players.

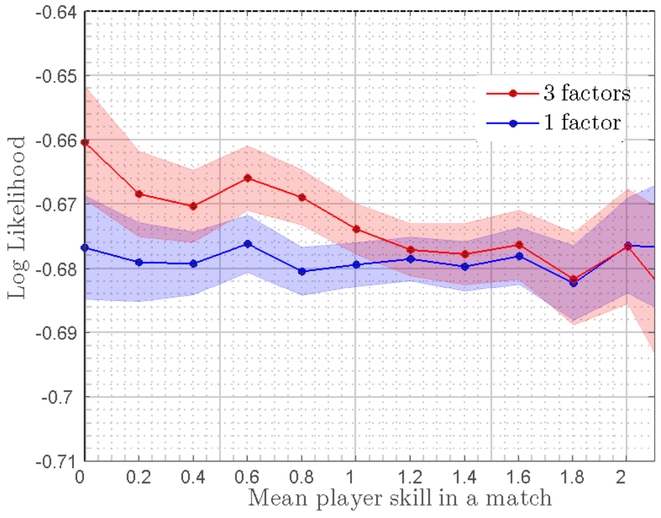


Figure 5.22: **Log Likelihood** for both 3 factors EP (red) and 1 factor EP (blue), on the same data as in previous experiment. Errorbars on standard error for games in each bin.

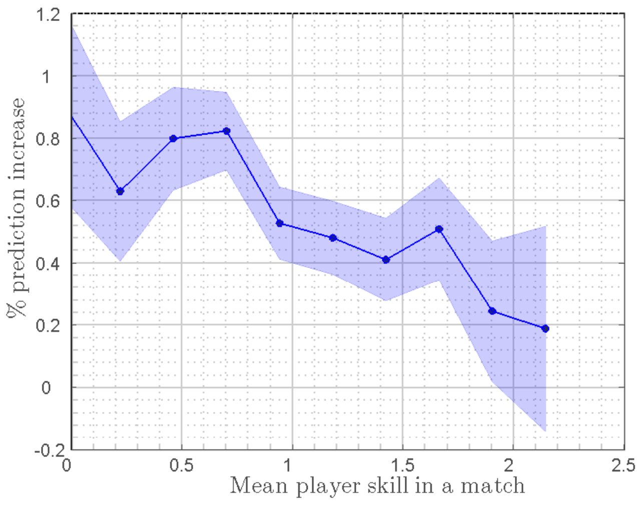


Figure 5.23: **Percentage prediction increase** for the 3 factors EP over 1 factor EP, on the same dataset. Errorbars on standard errors for games in each bin.

Chapter 6

Conclusions

This chapter summarises the work conducted during this project, and makes further comments on the results presented throughout this document. We mention justifications for the research tracks we followed, and how they relate to our initial goals. Having presented all the resulting remarks in the evaluation sections, we will gather here the few key observations and organise them into compact, summarising points. In the end, we will suggest appropriate further work.

6.1 Goals and results

As discussed in Chapter 1 we started with four goals, which we have tried to accomplish during this project. All four have been successfully addressed:

Model The Terry–Bradley model for rating players is used in current algorithms and all the state of the art rating systems investigated build their models upon it. Extending to include different numbers of factors was not difficult, and thus it made a good choice for our research.

Data The game of Go is a two player win–lose game, with an interesting handicap system. Not having to deal with team support and draws prevented unnecessary complications of the models, since our main interest lies in investigating the number of latent skills associated to a player. The handicap system required some effort to incorporate, but we did not experience any other hardships.

The KGS Go dataset has a large number of players and games, and very clean records. We possess enough games per player per day to test our models. A number of features (like time of play, game size, free or ranked game) allows for custom choice of subsets for experiments. It should be noted that there already is a matchmaking system in use on the server, and thus the ranked games are balanced. Differences in prediction percentages of 1% (between two algorithms) are hence quite significant.

Rating systems We fitted the models first by using maximum likelihood, which was not especially successful. A simple means to compare different models, but gives poor results due to data insufficiency. We used periods of 20 days for testing, in order to limit the effect of skills' change in time, and on the resulting datasets models with more factors were not statistically improving as much as we expected. They were indeed slightly better, but the models did not necessarily generalise to data as well as expected, and the Bayesian methods gave much better results.

Choosing TrueSkill$^{\text{TM}}$ as a reference, we continued with developing ADF and EP models for rating. The key results are:

– ADF is slightly better with three factors than only one, but a larger number of factors requires unreasonably large amounts of data. Because TrueSkill$^{\text{TM}}$ contains time series elements, it does better than ADF with any number of factors.

– In contrast, the three factors EP does much better than EP with one factor, as it makes better use of data. But, again, increasing the number of factors to more than three also required a lot of data to work well. TrueSkill$^{\text{TM}}$ works better than EP with one factor, but is now outperformed by EP with three factors, despite the time series advantage.

– We found out that iterating ADF on small datasets gives an unexpectedly bigger advantage, surpassing all previous algorithms. ADF with one factor and ten iterations is much better than both TrueSkill$^{\text{TM}}$ and EP with more factors. Re-running ADF knowing where the skills of the players lie (as a prior) seems to make better use of the information available. Also the games towards the end of the dataset have more influence than with EP, which helps predicting outcomes of matches immediately after.

- On a large dataset with more data available (200 days), we can use more factors and still improve the (iterated) one factor ADF model. However, TrueSkill^{TM} is now superior to the one factor (iterated) ADF (time series more effective). Nevertheless, a 7 factors ADF with one or two extra iterations has a higher accuracy than even TrueSkill^{TM}.

More factors We have compared and contrasted the models' predictions using a multitude of experiments, in order to find out how and when more factors improve the accuracy. The most important results are:

- ADF gives better predictions for games in which the additional factors count more than the initial difference in skill. While this is what one would hope, on the other hand, EP gains less for match-ups where the extra terms represent the majority of the skill difference between players. It would be interesting to further investigate the EP behaviour, as this result may be a sign of problems that could be addressed in future work.
- A model with more factors is better at judging very close matches, and still has similar performance on the rest of the games. This is a very favourable conclusion, as most games set up using a matchmaking system are balanced, and these are also the games everyone is interested in.

6.2 Future work

It is clear from the results described that extending current models to use more than one latent variables for each player has many advantages. However, there is a significant amount of work that we have not had the opportunity to carry out. This is, mainly, related to including some way to incorporate time series into our models. For the ADF algorithm an easy addition would be to emulate TrueSkill^{TM}'s choice, by adding extra variance to the skills after every match, or every day. In such a model the variance increases linearly with time, so the confidence interval grows like the square root of time. We might be worried about drifting to infinity, so we may try for example a process such as:

$$s(t+1) = s(t) \cdot \sqrt{1-\varepsilon^2} + \varepsilon\nu$$
$$\nu \sim N(0, w^2),$$

and control its spread with the w parameter.

Other interesting tasks, suitable for future work would be:

- To fit the posterior over the factors using a full covariance matrix and to check for anti-correlations between the factors. For simplicity, we have assumed that they are independent, which may negatively affect the performance of the algorithms.
- Keeping track of the correlations between players, which has already been proved to increase the accuracy (Birlutiu and Heskes, 2007).
- Constructing a system which can use a different number of factors depending on the quantity of data it has; for example, a hierarchical model that can use nested models with varying number of factors. Our algorithms suffered from the lack of data, and it would be useful to automatically choose the best number of factors.
- Investigate if factors map to some real, identifiable characteristics and provide any insight into the strategies used by players. Obtaining opinions from expert gamers would be particularly valuable.
- Learning the handicap and komi scaling factors directly from the data.
- Investigating why approximate inference using ADF is affected so much by re-iterating on the same (relatively small) data.

Appendix A

A.1 Reducing the ADF update to two players

The factor for the match is

$$f_i(S) = \sigma(S_A - S_B) = f_i(S_A, S_B) \tag{A.1}$$

By separating the players in two sets, one containing p_A and p_B, and the other one the rest of the competitors $p_{\setminus AB}$, we can expand the exact skills' posterior

$$\hat{p}(S) = \frac{f_i(S_A, S_B)q(S_A, S_B)q(S_{\setminus AB}|S_A, S_B)}{\displaystyle\int_S f_i(S_A, S_B)q(S_A, S_B)q(S_{\setminus AB}|S_A, S_B)\, dS} \tag{A.2}$$

The integral from the denominator is

$$\begin{aligned}
&\int_{S_{\setminus AB}} \left[\int_{S_{AB}} f_i(S_A, S_B)q(S_A, S_B)q(S_{\setminus AB}|S_A, S_B)\, dS_{AB}\right] dS_{\setminus AB} \\
&= \int_{S_{AB}} f_i(S_A, S_B)q(S_A, S_B)\, dS_{AB} \int_{S_{\setminus AB}} q(S_{\setminus AB}|S_A, S_B)\, dS_{\setminus AB} \\
&= \int_{S_{AB}} f_i(S_A, S_B)q(S_A, S_B)\, dS_{AB}
\end{aligned} \tag{A.3}$$

We can now write the posterior as

$$\hat{p}(S) = \hat{p}(S_A, S_B)q(S_{\setminus AB}|S_A, S_B) \tag{A.4}$$

Computing the KL divergence :

$$\begin{aligned}
&KL(\hat{p}(S)||q^{new}(S)) \\
&= \int_S \hat{p}(S)\left[\log \hat{p}(S) - \log q^{new}(S)\right] dS && \text{(A.5)} \\
&= \int_S \hat{p}(S_A, S_B) q(S_{\setminus AB}|S_A, S_B)[\log \hat{p}(S_A, S_B) + \log q(S_{\setminus AB}|S_A, S_B) \\
&\quad - \log q^{new}(S_A, S_B) - \log q^{new}(S_{\setminus AB}|S_A, S_B)]\, dS && \text{(A.6)} \\
&= \int_S \hat{p}(S_A, S_B) q(S_{\setminus AB}|S_A, S_B)[\log \hat{p}(S_A, S_B) - \log q^{new}(S_A, S_B)]\, dS + && \text{(A.7)} \\
&\quad \int_S \hat{p}(S_A, S_B) q(S_{\setminus AB}|S_A, S_B)[\log q(S_{\setminus AB}|S_A, S_B) - \log q^{new}(S_{\setminus AB}|S_A, S_B)]\, dS \\
&= KL(\hat{p}(S_A, S_B)||q^{new}(S_A, S_B)) \\
&\quad + E_{\hat{p}(S_A,S_B)}[KL(q(S_{\setminus AB}|S_A, S_B)||q^{new}(S_{\setminus AB}|S_A, S_B))]. && \text{(A.8)}
\end{aligned}$$

These two can be minimised separately, and since the posterior is Gaussian, it is obvious that the second KL divergence will lead to

$$q^{new}(S_{\setminus AB}|S_A, S_B) = q(S_{\setminus AB}|S_A, S_B). \tag{A.9}$$

A.2 Reducing the EP update to two players

$$\begin{aligned}
\hat{p}(S) &= p(p_A > p_B) q^{\setminus i}(S) && \text{(A.10)} \\
&= p(p_A > p_B) q^{\setminus i}(S_{AB}) q^{\setminus i}(S_{\setminus AB}|S_{AB}) && \text{(A.11)} \\
&= \hat{p}(S_{AB}) q^{\setminus i}(S_{\setminus AB}|S_{AB}) && \text{(A.12)} \\
KL(\hat{p}(S)||q(S)) &= KL(\hat{p}(S_{AB}) q^{\setminus i}(S_{\setminus AB}|S_{AB})||q(S_{AB}) q(S_{\setminus AB}|S_{AB})) && \text{(A.13)} \\
&= KL(\hat{p}(S_{AB})||q(S_{AB})) \\
&\quad + E_{\hat{p}(S_{AB})}[KL(q^{\setminus i}(S_{\setminus AB}|S_{AB})||q(S_{\setminus AB}|S_{AB}))] && \text{(A.14)}
\end{aligned}$$

Bibliography

ATP - Association of Tennis Professionals. ATP Rankings Frequently Asked Questions. 2011. URL `http://www.atpworldtour.com/Rankings/Rankings-FAQ.aspx`.

R. M. Bell and Y. Koren. Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75, Dec. 2007. ISSN 19310145. doi: 10.1145/1345448.1345465.

A. Birlutiu and T. Heskes. Expectation propagation for rating players in sports competitions. *Knowledge Discovery in Databases: PKDD 2007*, pages 374–381, 2007.

R. Bradley and M. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. ISSN 0006-3444.

J. Cai, E. Candes, and Z. Shen. A singular value thresholding algorithm for matrix completion. *preprint*, pages 1–28, 2008.

P. Dangauthier, R. Herbrich, T. Minka, and T. Graepel. Trueskill through time: Revisiting the history of chess. *Advances in Neural Information Processing Systems*, 20:337–344, 2008.

A. E. Elo. *The Rating of Chess Players, Past and Present.* Batsford, (London), 1978. ISBN 0923891277.

M. E. Glickman. Parameter Estimation in Large Dynamic Paired Comparison Experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, Aug. 1999. ISSN 0035-9254. doi: 10.1111/1467-9876.00159.

K. Harkness. *Official chess handbook*. David McKay Company, 1967.

R. Herbrich. Ranking and Matchmaking TrueSkill Revealed, 2007. URL `http://www.microsoft.com/downloads/en/details.aspx?id=7367`. Talk: Gamefest 2007.

R. Herbrich, T. Minka, and T. Graepel. TrueSkill(TM): A Bayesian skill rating system. *Advances in Neural Information Processing Systems*, 20:569–576, 2007.

D. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32(1):384–406, 2004. ISSN 0090-5364.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. ISSN 0018-9162.

F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

D. J. C. MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

D. J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge Univ Pr, 2003. ISBN 0521642981.

T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, 2001. Massachusetts Institute of Technology.

T. P. Minka. EP: A quick reference, 2008. URL `http://research.microsoft.com/en-us/um/people/minka/papers/ep/minka-ep-quickref.pdf`.

B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. ISBN 1581133480.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix factorization and neighbor based algorithms for the Netflix prize problem. *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, page 267, 2008. doi: 10.1145/1454008.1454049.

TopCoder Inc. Algorithm Competition Rating System, 2008. URL `http://apps.topcoder.com/wiki/display/tc/Algorithm+Competition+Rating+System`.

J. Zhou and T. Luo. Towards an Introduction to Collaborative Filtering. *2009 International Conference on Computational Science and Engineering*, pages 576–581, 2009. doi: 10.1109/CSE.2009.381.