Cmpe 492 Final Project

Midterm Report

Player Rating & Matchmaking via Bayesian Inference

Mustafa Onur Eken
advised by A. Taylan Cemgil

2016

Contents

1 Introduction & Motivation

We can safely say that player rating systems are essential nearly to any game in different contexts (video games, sports etc). This need arises from the fact that an *unbalanced* match is not enjoyable to any of it's competitors. In an *unbalanced* match, there is a high variation among the skills of the competitors, therefore a remarkable challenge is not achieved and inadequate amount of fun is generated.

Assuming the fact that the skill levels of the players in a match is the dominant factor affecting the result of that match, if we could figure out skill levels of players of a particular game then we could arrange *balanced* matches that are entertaining for both highly and poorly skilled players.

With this motivation, Arpad Elo developed a rating system for the board game Chess [1]. According to his mathematical modeling, rating of a player is a unit-less positive number. System has an expected result for a match between two players and when the actual result is observed the ratings of the players are updated. Winner's rating always goes up and loser's rating goes down but unexpected (*upsetting*) results magnify the amount of change in the ratings.

In this project we are going to study and implement a generalized and improved version of Elo rating system, developed by Microsoft Research , *TrueSkill*. It is generalized because matches are not constrained as to be one versus one and unlike Elo, *TrueSkill* learns more from the stalemates. In addition, it converges on players' ratings faster.

From a rating system three major functionalities are expected: 1) Estimating skills 2) Suggesting *balanced* matches and 3) Publishing skills

2 State of the Art

Currently, many commercial online multi-player video games use Elo rating system where the system is slightly modified according to specific needs of the game. A rating system called Glicko, newer than Elo older than TrueSkill, is also used in some video games such as *Counter Strike: Global Offensive* [2]. TrueSkill extends ideas presented in Glicko rating system and it is deployed on *Xbox Live*.

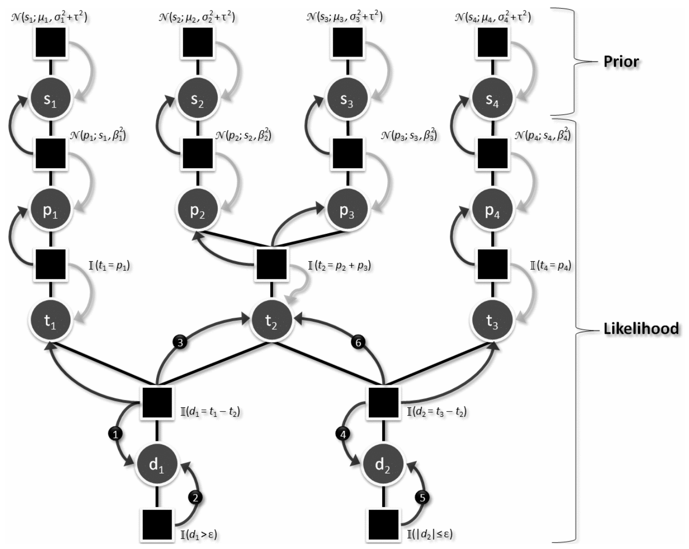
3 Methods

3.1 Theory

TrueSkill can be considered a Bayesian model-based machine learning algorithm [3, 4]. It assumes a graphical model for a match [5] and executes an algorithm on the graph (model) to infer better estimates about the skills of player involved in the match. We will go into details in the subsequent paragraphs.

The model can be described in plain text as following:

1. **Player Model**
    - We denote *estimated* skill of the player i as μ_i
    - System is uncertain about skills of players. The uncertainty associated with a skill, μ_i, of a player is σ_i
    - System thinks the *actual* skill of the player, s_i, is normally distributed with parameters μ_i and σ_i^2
    - A player can generate a performance, p_i, depending on his/her actual skill, s_i. p_i is normally distributed with mean s_i and with variance β^2, where β is a constant depending on chance factor associated with that particular type of game. For instance, one can expect β chosen for *Uno* to be larger than β chosen for *Chess*. See [6] for details.
2. **Team Model**
    - Teams are formed by players and can contain any number of players
    - Similar to the player model, a team can generate a performance, t_i. It is simply obtained by summing up the performances of member players, $\sum p_i$, so t_i is normally distributed as well.
3. **Match Model**
    - Instead of players, only teams compete in a match.
    - A match result is a relative standings of teams involved in that match. Different teams may have same rank in the standings which indicates a draw. The top team in the standings is the winner.
    - In a match, teams' performances are compared, i.e. subtracted from each other, d_i (*team performance difference*). With these d_i system expects some teams rank higher than others and vice versa.



$\mathcal{N}(s_1;\mu_1,\sigma_1^2+\tau^2)$
$\mathcal{N}(s_2;\mu_2,\sigma_2^2+\tau^2)$
$\mathcal{N}(s_3;\mu_3,\sigma_3^2+\tau^2)$
$\mathcal{N}(s_4;\mu_4,\sigma_4^2+\tau^2)$
Prior
s_1
s_2
s_3
s_4
$\mathcal{N}(p_1;s_1,\beta_1^2)$
$\mathcal{N}(p_2;s_2,\beta_2^2)$
$\mathcal{N}(p_3;s_3,\beta_3^2)$
$\mathcal{N}(p_4;s_4,\beta_4^2)$
p_1
p_2
p_3
p_4
$\mathbb{I}(t_1=p_1)$
$\mathbb{I}(t_2=p_2+p_3)$
$\mathbb{I}(t_4=p_4)$
t_1
t_2
t_3
Likelihood
3
6
$\mathbb{I}(d_1=t_1-t_2)$
$\mathbb{I}(d_2=t_3-t_2)$
1
4
d_1
d_2
2
5
$\mathbb{I}(d_1>\epsilon)$
$\mathbb{I}(|d_2|\leq\epsilon)$

Figure 1: A factor graph for a particular match

And finally, for the system to infer new μ_i and σ_i (recall that this is our essential desire: to find better estimates for players' skills) the actual result of the match is compared with the expected outcomes, i.e. d_i. In a multi-team match, we are going to compare only the teams that are neighbors in the standings (*i.e. in the actual results*) of the match. See Figure 1 for a visual representation of the model of an example match where three teams and four players are involved.

A factor graph represents the factorization of a function of several variables [7]. The functions (a.k.a. *functor nodes*) are drawn as square blocks in the example factor graph shown in Figure 1. Most functions can be read and interpreted easily from the Figure 1 such as $\mathcal{N}(p_1; s_1, \beta_1^2)$ but some cannot, e.g. $\mathbb{I}(d_1 > \epsilon)$. See [4] for details of those functions. The whole graph together forms a joint distribution over all variables, in example above it is $f(s_1, s_2, s_3, s_4, p_1, p_2, p_3, p_4, t_1, t_2, t_3, d_1, d_2)$. We are only interested in marginal distributions of actual skill variables: $f(s_1)$, $f(s_2)$, $f(s_3)$, $f(s_4)$. Note that, we expect these distributions to be Gaussian with new expected skill, μ_i^{new}, and new uncertainty σ_i^{new} parameters, refer to [4] for the reasoning. Knowing these, we can use sum-product message passing algorithm [7] to solve for the marginal distributions in the factor graphs that we create for each match.

3.2 Implementation

After we comprehend the theory and math behind the graphical model and belief propagation algorithm described, implementation comes next. We have chosen *Python 2.7.11* as the language and *Sphinx* for documenting the project. The following are the steps we decided to follow in the implementation period:

$*$1. Setup documentation system

$*$2. *Model* module: implement *Player, Team, Match, Environment* classes

$*$3. Implement synthetic data generation

$*$4. *FactorGraph* module: implement *FactorNone, VariableNode, FactorGraph* classes

$*$5. Implement factor graph generation based on a match object

6. Implement sum-product message passing algorithm
7. Test on synthetic data and fix bugs
8. Implement match quality measurer
9. Test on synthetic data and fix bugs
10. Implement skill publishing
11. Test on synthetic data and fix bugs
12. Implement match recommender
13. Test on synthetic data and fix bugs
14. Fetch and format real world data, example dataset: tennis matches [8]
15. Test on real world data
16. Finalize documentation and prepare demonstrations

The steps with $*$ are completed, see the git repository `https://github.com/oeken/reelskill`. Currently we have the backbone classes and we can generate dummy data with a call of a function. By using a particular seed number for our random number generators we ease the debugging. For now, data generator generates a specified number of players, a specified number of teams each sized equally and all matches that could take place between two teams from the team list. I.e. $\binom{T}{2}$ matches where T is the number of the teams.

FactorGraph class is implemented in a doubly linked list manner to ease implementation of future steps such as sum-product algorithm. After generating some data, we can generate the full factor graph for a match object. Graph generator works recursively, for a match it creates sub-graphs for each team and connects the leaf nodes of the sub-graphs. Similarly for a team it creates sub-graphs for each player and connects the leaf nodes of the sub-graphs.

Next we are going to implement the most important part of TrueSkill, *the sum-product algorithm*. After completion of this, we achieve the the first and the most important function of our rating system: 1) Estimating skills. The remaining functionalities are relatively straightforward to implement. Furthermore, we possess a good real world data in our hands: tennis matches. Since it is well organized, we can test TrueSkill on it as soon as the implementation of belief propagation algorithm finishes.

4 Results

This section will be available in the final report.

5 Conclusion & Discussion

This section will be available in the final report.

6 Future Work

This section will be available in the final report.

References

[1] Elo rating system. Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Elo_rating_system`, 2016.

[2] Glicko rating system. Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Glicko_rating_system`, 2016.

[3] Christopher M Bishop. Model-based machine learning. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1984), 2013.

[4] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill: A bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576, 2006.

[5] Ethem Alpaydin. *Introduction to Machine Learning.* MIT Press, third edition, 2014.

[6] Jeff Moser. Computing your skill. `http://www.moserware.com/2010/03/computing-your-skill.html`, 2010.

[7] Frank R Kschischang, Brendan J Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[8] University of California Irvine. Tennis major tournament match statistics data set. `https://archive.ics.uci.edu/ml/machine-learning-databases/00300/`.