A Truly Bayesian Ranking Algorithm

Alex Ozdemir

May 2, 2014

1 Introduction

In most sports, teams perform differently in each game, due to causes from conscious changes in strategy to variations in the players' attitudes. In addition, teams which play in the same league may not all play each other, or even play the same number of games. Because of these irregularities, it is quite challenging to determine how "good" a team is.

Various methods have been used to quantify how "good" a team is, and by extension, what one team's probability of victory is against another team. Among the most successful of these is the TrueSkill^{TM}algorithm, developed by Microsoft. While the algorithm is Bayesian in that it updates a belief about the skill of a player or team, the updating rules themselves are not Bayesian methods.

This paper seeks to explore the possibility of a more rigorous Bayesian ranking algorithm. Specifically, we would like to leverage Bayesian statistical methods in order to design an algorithm which uses the record of a team to build a posterior belief distribution for how "good" a team is. Then, we would like to use this predict how likely one team is to beat another team. Using this, we can also construct a ranking of teams, based on how many other teams they're more likely to beat than to lose to.

2 Data

The first choice in designing such an algorithm is what type of data to handle. We chose to look at only games in which 1 team plays 1 other team, and the only result we chose to look at was who won the game. While the argument could be made to look at other indicators of a teams performance, it is better to not do so. Most teams play to win, and by taking into account other factors, such as some sort of score, the algorithm (whose primary job is to predict future victories) becomes biased towards particular strategies. Focusing only on the outcome of a game preserves the algorithms focus on predicting victory and defeat, independent of play style. Furthermore, other indicators of performance may not be universal, for example, the batting average of a baseball team's hitters might be useful when analyzing baseball games, but not for other sports. Avoiding such specification allows our algorithm to be applied to a larger set of games.

One dataset we analyzed was the League of Legends Championship Series Season 4 Spring Split (abbreviated ”LCS Dataset”), which consisted of 8 teams playing 4 games against every other team over a 3 months period, for a total of 112 games (28 games per team). This data set is clearly very regular, which means that simpler indicators, such as the numbers of wins a team has, are probably good estimators of how good they are. The dataset is also small, enabling us to do fairly precise calculations in a short period of time. Both factors made this dataset ideal for testing our algorithm as we developed it.

The next dataset we analyzed was the set of basketball games played during the 2012-2013 National Basketball Association Regular Season (abbreviate ”NBA Dataset”). This dataset consisted of 30 teams playing games over a 6 month period. A total of 1229 games were played, or approximately 82 per team. While the structure of the NBA regular season isn’t completely chaotic, their are some irregularities in which other teams each team plays. This give our algorithm an opportunity to shine in a slightly more challenging dataset.

It was also important to develop some way of testing our algorithm. Both datasets culminate in a playoffs phase in which the teams participate in a bracket structure tournament, composed of series. We omitted these final games, in order to use them to evaluate the inferences of our algorithm. This process will be discussed at greater depth in the results section.

3 Analysis

3.1 Model

Any mathematical analysis of data begins by modeling the source of the data. We modeled a game as an event in which the two participating teams demonstrate a certain *performance*, and the team which has a higher performance wins. Of course, a given team doesn’t always perform at the same level, so performance is a random variable. We also assumed that these performances are approximately independent - that a team’s performance doesn’t depend on the team they’re playing. These assumptions allowed us to model performance as a normal random variable whose mean and standard deviation are fixed for each team, but unknown. That is, for some team A,

$$P_A \sim \text{Norm}(\mu_A, \sigma_A^2)$$

where P_A is the performance of team A, and μ_A and σ_A are their mean and standard deviation, respectively. This particular model of performance is taken from Microsoft’s TrueSkill algorithm.

3.2 Likelihood Function

A likelihood function for one team beating another follows naturally from this model. When two teams play each other, the probability that team A beats team B is the probability that team A’a performance is higher than B’s: $\text{P}(P_A > P_B)$ By writing performance as a normal random variable, this expression simplifies to:

$$\mathrm{P}(P_A > P_B) = \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)$$

which is a likelihood function dependent on 4 unknown parameters: the mean and standard deviation of both the winner's and loser's performance.

Unfortunately, turning this likelihood function into an updating rule for our belief about the parameters is more challenging. The likelihood function is a cumulative normal distribution, which has no closed form, let alone a clean conjugate family for any of its parameters. This mandates we take a numerical approach to designing an updating rule for each of the 4 unknown parameters in the likelihood function. We begin with the general form of an updating rule, derived from Bayes Theorem:

$$g' = gf$$

where g is the prior belief, g' is the posterior, and f is the likelihood function. In this context the likelihood function is a function of 4 variables (If the teams are A and B, the variables are μ_A, μ_B, σ_A, σ_B), whereas the prior and posterior beliefs are both probability density functions of one variable. In order to use the above updating rule we must convert of four variable likelihood function to a function of one variable.

One way to reduce the number of variables in a likelihood function is through integration. In order to update one of the variables in our likelihood function, we can integrate out the other three. By doing this for each of the winner's mean, winner's standard deviation, loser's mean, and loser's standard deviation, we determine the following 4 updating rules:

$$g'_{\mu_A} = g_{\mu_A} \int\limits_{\mu_B} \int\limits_{\sigma_A} \int\limits_{\sigma_B} \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)\, g_{\sigma_B}(\sigma_B)\, g_{\sigma_A}(\sigma_A)\, g_{\mu_B}(\mu_B)\, \mathrm{d}\sigma_B\, \mathrm{d}\sigma_A\, \mathrm{d}\mu_B$$

$$g'_{\mu_B} = g_{\mu_B} \int\limits_{\mu_A} \int\limits_{\sigma_A} \int\limits_{\sigma_B} \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)\, g_{\sigma_B}(\sigma_B)\, g_{\sigma_A}(\sigma_A)\, g_{\mu_A}(\mu_A)\, \mathrm{d}\sigma_B\, \mathrm{d}\sigma_A\, \mathrm{d}\mu_A$$

$$g'_{\sigma_A} = g_{\sigma_A} \int\limits_{\mu_B} \int\limits_{\mu_A} \int\limits_{\sigma_B} \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)\, g_{\sigma_B}(\sigma_B)\, g_{\mu_A}(\mu_A)\, g_{\mu_B}(\mu_B)\, \mathrm{d}\sigma_B\, \mathrm{d}\mu_A\, \mathrm{d}\mu_B$$

$$g'_{\sigma_B} = g_{\sigma_B} \int\limits_{\mu_A} \int\limits_{\sigma_A} \int\limits_{\mu_B} \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)\, g_{\mu_B}(\mu_B)\, g_{\sigma_A}(\sigma_A)\, g_{\mu_A}(\mu_A)\, \mathrm{d}\mu_B\, \mathrm{d}\sigma_A\, \mathrm{d}\mu_A$$

given that A is the winning team, and B is the losing team.

From these updating rules we derive a simple updating algorithm. This algorithm, which is shown in Figure 1, is modeled off of Gibbs sampling in that it updates some of the parameters first, and then updates the others using the parameters which have already been updated. We chose to update both the mean distributions based off of their original distributions (and not update one using the other, after it had been updated) because we wanted to preserve the win/loss symmetry of updating. If we didn't do this, then weird behavior would arise such as two teams distributions changing by different amounts, even if they began the same. The algorithm was programmed in Java, which was chosen for its statistical libraries and fast runtime. The code is attached in Appendix A.

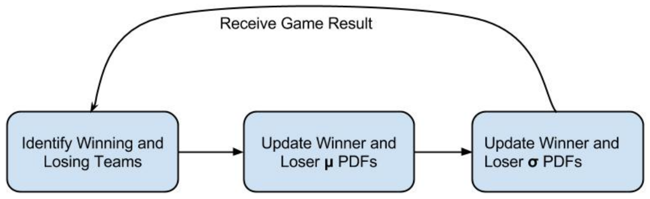
Receive Game Result
Identify Winning and Losing Teams
Update Winner and Loser μ PDFs
Update Winner and Loser σ PDFs

Figure 1: Updating Algorithm

3.3 Prior Distributions

In choosing a prior distribution, an important, but often omitted step is considering the units of your parameters. Both the mean and standard deviation of performance have the same units as performance itself since it is a normal random variable, but what are the units of performance? Given that our end goal is to determine the probability of one team beating another, any interest in performance is actually an interest in the performance *difference* between teams, relative to their standard deviations. Because this difference is judged relative to the standard deviation of the teams, and standard deviations of the teams have the same units as performance, the units don't really matter because they drop out anyway.

Furthermore, if we choose a normal prior for our teams' means, the standard deviation and mean of that prior aren't particularly important either. Because we're only interested in the difference between the performance of teams, the performance values themselves don't matter. While the standard deviation we chose will impact how much our team's mean performances can spread out, because we're measuring the difference between teams *relative to their combined variances*, the standard deviation doesn't end up making a difference either. Consequently we choose a $\text{Norm}(10, 6^2)$ prior for the mean performance of each team, rather arbitrarily, so our team's mean performances will fall roughly between 0 and 20.

However, we do make one significant simplification. After some experimentation we realized that the distributions for the teams' means really did stay between 0 and 20 for the most part, so we actually confined the prior to that interval to reduce the size of our belief distributions and save computational time.

For the standard deviation of performance we used a uniform distribution from 0 to 5 as the prior for all of our teams. This range was also based on some experimentation, and balanced completeness with computational time.

3.4 Example Update

For the reader's convenience we've included a simple demonstration of updating the mean and standard deviation distributions of two teams. We began with two teams, A and B, each having a $\text{Norm}(10, 6^2)$ prior for their mean performance (confined to $[0, 20]$), and a uniform prior for their standard deviation, between 0 and 5. The two teams played a game, and A won. The prior and posterior distributions for the teams' mean performance are shown in Figure 2, and their standard deviations are shown in

Figure 3. For both graphs, the x-axis hold possible parameter values, and the y axis holds the probability that that parameter value is the true one, according to the belief distribution. Note that in Figure 3 the posterior distributions for A and B's standard deviations are identical, so A's is covering B's on the graph.

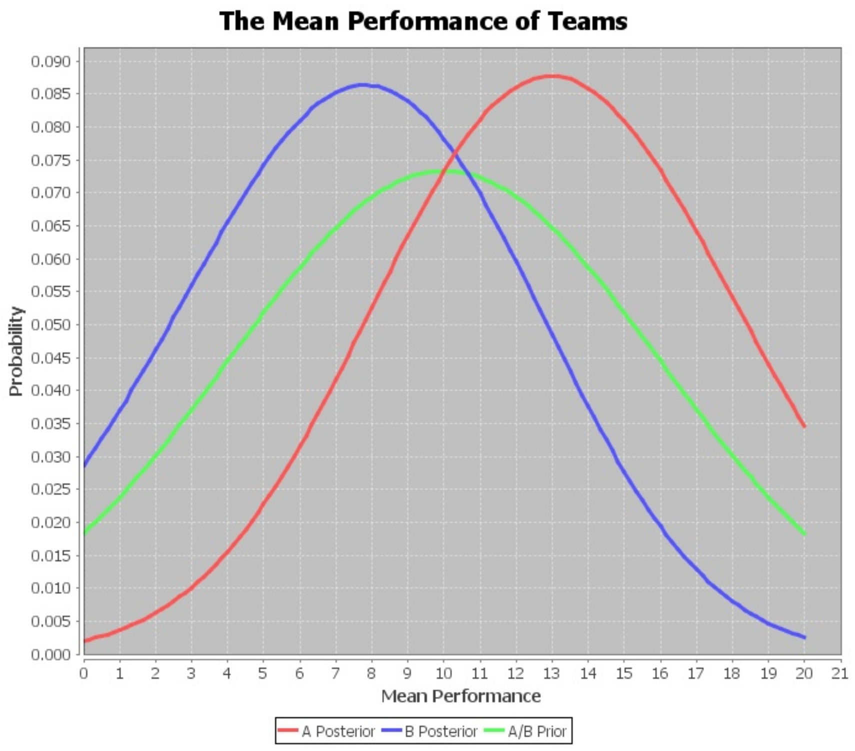

Figure 2: Example Mean Update: Before and After

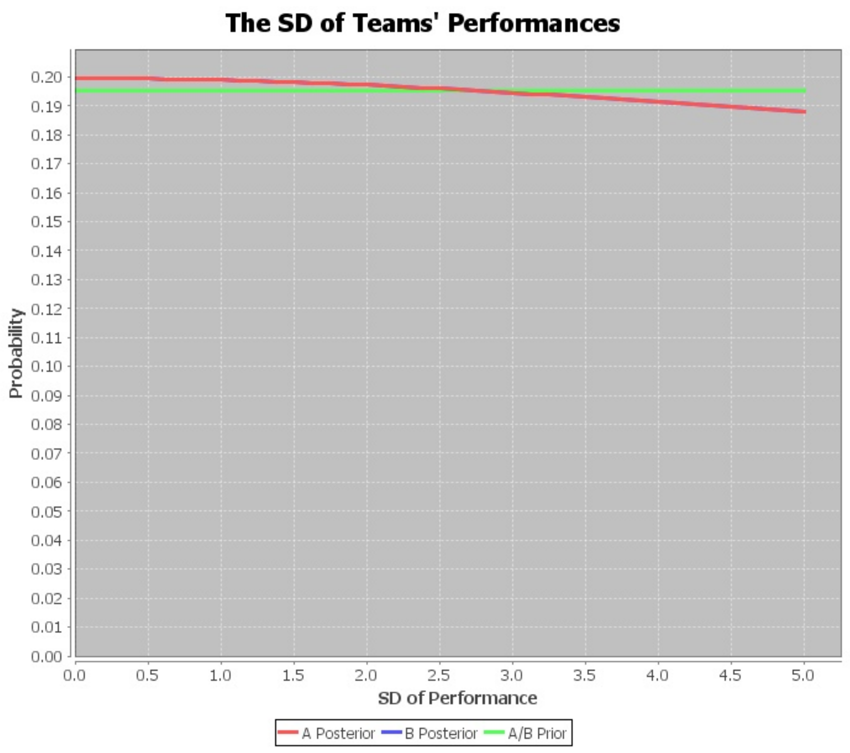

Figure 3: Example SD Update: Before and After

By examining the example graphs 2 and 3, we can confirm some of our intuition about how ”good” the teams are. Notice that the posterior for the winner’s mean cover larger values than the prior, and the posterior for the loser mean covers smaller values than the prior. Both posterior distributions are tighter, reflecting the additional information from the data we updated with. The standard deviation results are slightly harder to interpret this early on, but the fact that A and B have identical posteriors distributions here (A’s is actually covering B’s) reflects the symmetry of the game with respect to the variability in the teams’ performances.

4 Results

4.1 LCS Data

We begin our analysis of the LCS Data by setting priors for each of the eight teams in the league. As discussed previously, we assigned each a $\text{Norm}(10, 6^2)$ prior, confined to [0,20] for their mean performance, and a uniform prior on [0,5] for their standard deviation of performance. These prior are shown in Figures 4 and 5. Notice that only one of the teams’ priors are visible because they’re stacked on each other.

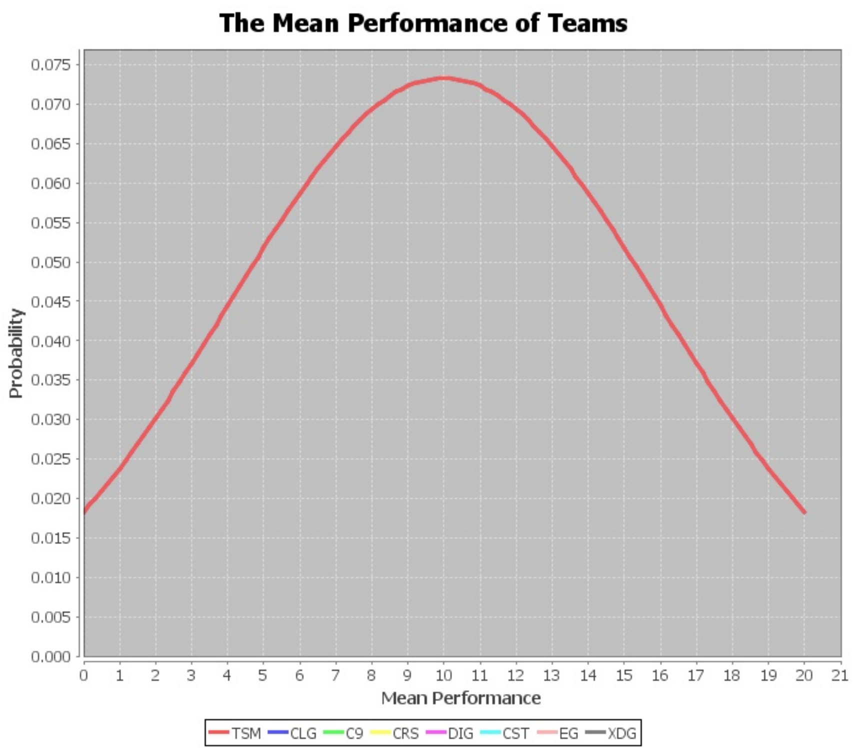

Figure 4: Prior Distribution for the Mean Performance of LCS Teams

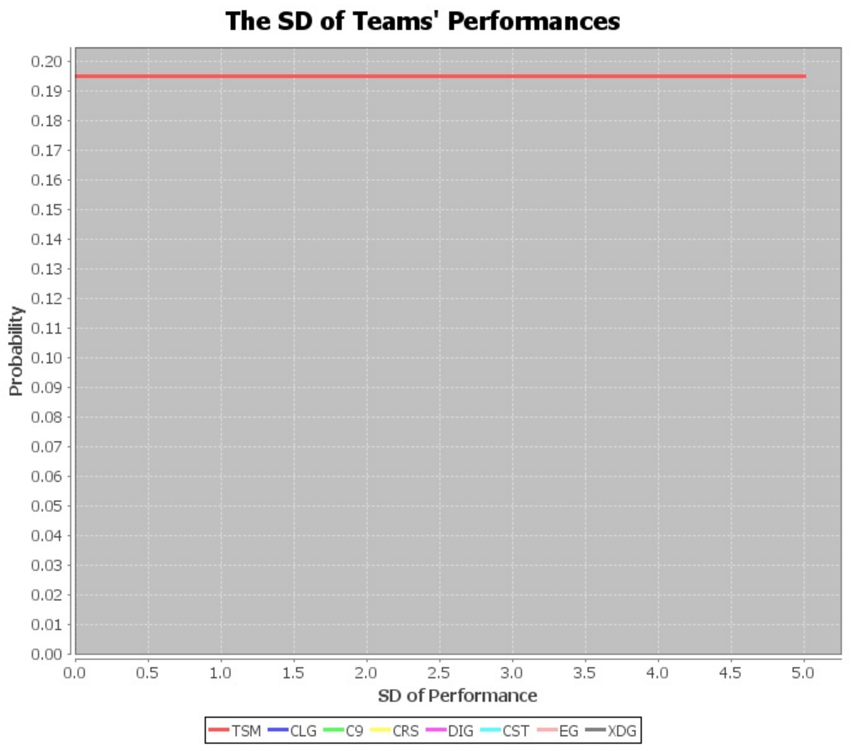

Figure 5: Prior Distribution for the SD of Performance of LCS Teams

Running the algorithm on the LCS data yielded posterior distributions for the mean and standard deviation of performance of each team. The data processed can be found in Appendix A. The posterior distributions for the mean and standard deviations for each team are shown visually in Figures 6 and 7, respectively. The x axis in these graphs holds potential values for the parameter in question (mean or standard deviation), and the y-axis holds the probability of that value being the true value of the parameter, according to the posterior distribution for the parameter.

By examining the posterior distributions, we can gain insight into how good the teams are, relative to one another. From looking at the graph of posterior mean performance we see that some pairs of team have clearly distinct means performances, such as C9 and XDG. However, others, such as CRS and DIG seem quite similar. The standard deviation graph shows some teams are more consistent in their performance than others (for better or worse, depending on their mean).

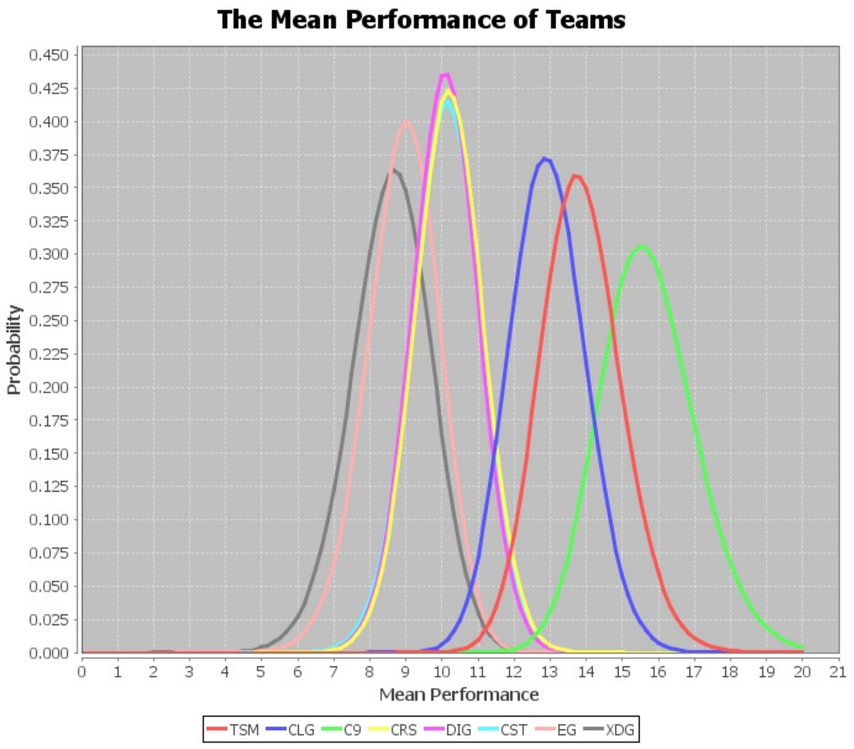

Figure 6: Posterior Distribution for the Mean Performance of LCS Teams

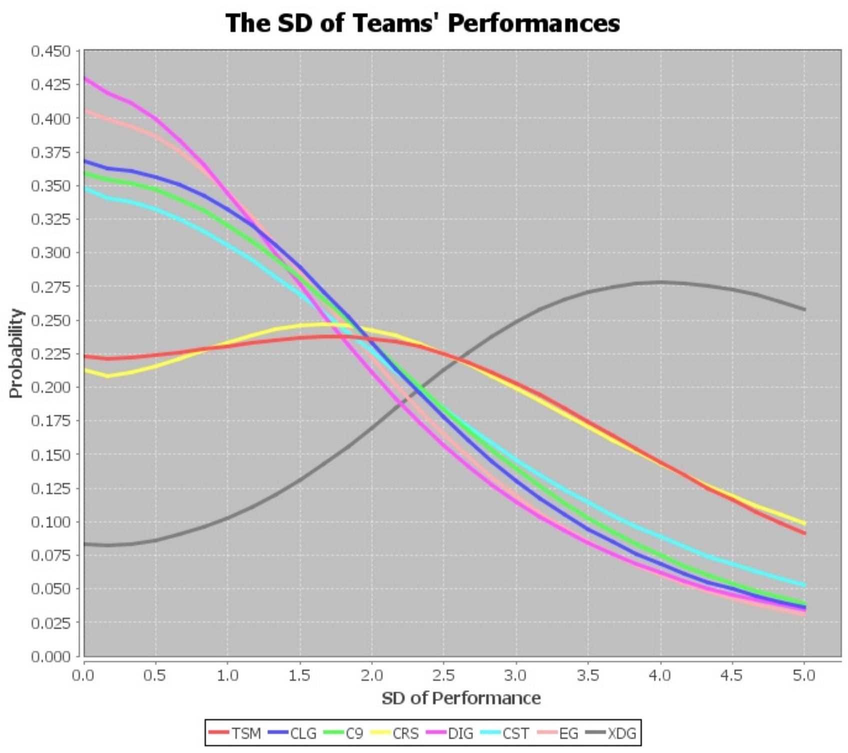

Figure 7: Posterior Distribution for the SD of Performance of LCS Teams

Using these posterior distributions, we can answer some of our initial questions such as, "How likely is one team to beat another". The probability of one team winning in a match is governed by the likelihood function from the Analysis section. We can compute the value of that function using the posteriors for any two teams means and standard deviations be evaluating:

$$\iiiint \mathrm{P}(\mathrm{Norm}(\mu_B - \mu_A, \sigma_A^2 + \sigma_B^2) < 0)\, g_{\sigma_B}(\sigma_B)\, g_{\sigma_A}(\sigma_A)\, g_{\mu_B}(\mu_B)\, g_{\mu_A}(\mu_A)\, \mathrm{d}\sigma_B\, \mathrm{d}\sigma_A\, \mathrm{d}\mu_B\, \mathrm{d}\mu_A$$

which gives the probability of team A beating team B. As an example, we've compute the probability of DIG beating each potential opponent, and included them in Table 1.

Opponent	Probability DIG Wins
C9	0.0418
TSM	0.1214
CLG	0.1505
CRS	0.4861
CST	0.4952
XDG	0.6619
EG	0.6794

Table 1: Probability of DIG winning against other teams

We can also determine a ranking of the teams. By computing the probability of each team beating each other team, we can determine how many others a given team would be expected to win against. We can rank the teams by the number of teams they're expected to beat, as we did in Table 2. A Win / Loss record based ranking has also been included in Table 2, for comparison. Notice that the rankings are identical in extreme cases, as would be expected. If a team consistently wins game against all opponents, we'd expect them to be ranked highly by any reasonable system. However, in the middle ranks, the two ranking differs slightly. This is also consistent with expectations, because when two teams win and lose about the same number of games, the Bayesian algorithm begins to diverge from other methods, because it factors in *who* those victories and losses were against.

Rank	Ranking System	
	W / L	Bayesian
1	C9	C9
2	TSM	TSM
3	CLG	CLG
4	DIG	CRS
5	CRS	CST
6	CST	DIG
7	EG	EG
8	XDG	XDG

Table 2: LCS Ranking Comparisons

Our final task is to evaluate our inferences against the playoffs data alluded to before. The playoffs consisted of a number of 3 or 5 game series played in a bracket structure, after the normal season (which provided all previous data used). Each series ended up resulting in 2 or 3 games played, such that the number of games won by each team varied binomially as the probability of one team beating the other. We decide to judge our algorithm to be fairly successful if the breakdown of the series' matches closely mirrored the expected results if the teams played the number of games they did, with the win probabilities we predict.

For example, during the tournament, DIG played CRS, and lost to them, 1-2. Given that DIG and CRS play 3 times, and the probability that DIG wins each game is 0.4861, the most likely outcome (mode) is that DIG wins one of the games (the expected value is 1.458). That is exactly what happened, so we'd say our algorithm was very successful on this test. Using this method we correctly predicted the number of games that each team would win or lose (see the explanation above) 5 out of 6 times, and were off by only 1 game in the last match. Overall, we consider these test result to be very successful.

4.2 NBA Data

We also ran or algorithm on the 2012-2013 NBA dataset. This data set was much larger, with over 1000 games played between 30 different teams over the course of over 6 months. The posterior distributions for both the mean and standard deviation of each teams performance are shown in Figures 8 and 9. The prior distributions have been omitted because we used the same prior distributions for the mean and standard deviation of NBA teams as we used for the LCS teams. For reference these are a $\mathrm{Norm}(10, 6^2)$ prior for μ and a $\mathrm{U}(0, 5)$ prior for σ.

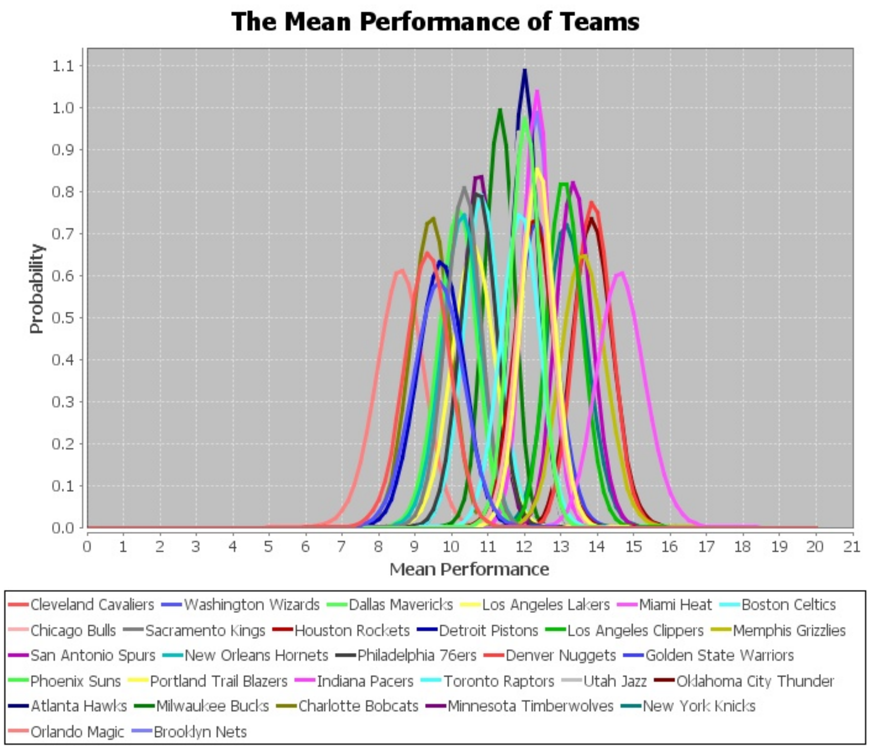

Figure 8: Posterior Distribution for the Mean Performance of NBA Teams

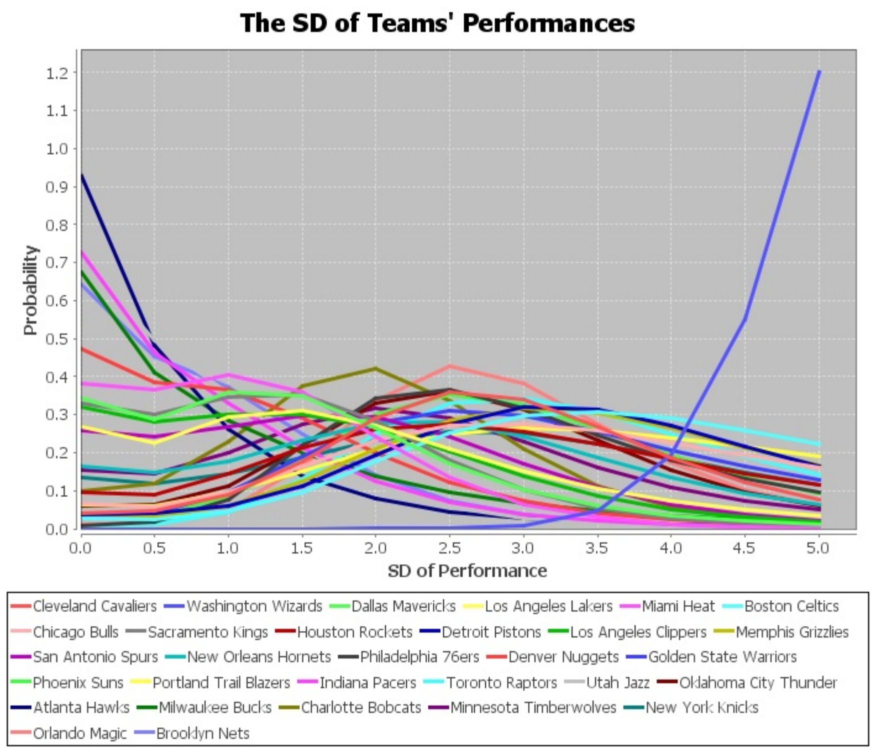

Figure 9: Posterior Distribution for the SD of Performance of NBA Teams

We once again want to evaluate how well our algorithm has performed, this time as it has been used on the NBA dataset. The NBA Regular Season also culminates in a play-offs bracket in which each match-up is a best of 7 series. In order to evaluate how well our algorithm performed in predicting the results of each series, we used the same method as before: computing the expected value of the number of games the teams would win according to our predicted probabilities, and comparing that expected value with what actually happened.

An example of this computation is in the Miami vs. Milwaukee playoffs series. The actual result was (4-0), in favor of Miami, meaning that 4 total games were played. Our algorithm tells us Miami has a 94.1% chance of beating Milwaukee in any given game, so the we'd expect Miami to win 3.76 out of 4 games, which is closest to 4 out of 4. They actually won all 4 games, so our prediction was perfect. The results of these calculations are shown in Table 3.

Algorithm Prediction Performance	Number of Series
Perfect	6
Off by 1 game	4
Off by 2 games	5
Off by 3+ games	0

Table 3: The Accuracy of our Algorithm's Predictions in the NBA Playoffs

Although the predictions weren't as accurate as they were in the LCS playoffs, they're still acceptable, especially considering the historic unpredictability of NBA playoffs.

5 Conclusion

The goal of this project was to create a Bayesian algorithm which could use teams' past victories and losses to build a belief distribution for how "good" that team is, relative to the teams they play against.

By modeling "goodness" as a random variable, *performance* (P). We were successfully able to construct a set of posteriors for the performance of each team, from which we could numerically estimate the probability of any team beating any other team, and generate a ranking of the teams. Our algorithm performed well under testing in our regular dataset, but performed only acceptably in the NBA dataset.

6 Recommendations

One simplification we made in this project was confining the priors for both the mean performances and the standard deviations of performance. Doing so meant we could have missed significant portions of the posterior distributions for these parameters. By using more computational power, we could avoid this simplification.

A different solution to the same problem is to use a system of variable step size for the parameter. Currently our belief distributions consider discrete values of the parameter, separated by a constant step size. It would be interesting to consider a belief distribution which had variable step size - that is the separation between each pair of adjacent parameter values is *not* constant. The parameter values could be closer together around "interesting" parts of the distribution, and further apart elsewhere (My first thought is that an "interesting" part of the distribution is where probability is higher). This could be implemented dynamically - that is that the step size changes every update to reflect a changing points of interest, using interpolation to find probability density for the newly determined parameter values. This would be a really interesting investigation to do.

Additionally, our posteriors for standard deviation increased slightly at 0, which seems very counter-intuitive given that we expect that teams do not always perform at *exactly* the same level. This phenomenon could be the result of a number of factors, such as a low number of games per team. It could also be a symptom of our initial assumption that when two teams play each other their performances are independent. A third explanation is that it is compensation for the upper bound of 5 placed upon our prior distributions for standard deviation. In any case it would be an interesting topic to explore it in the future.

Appendix A Data and Code

Both the data and the code can be found at:

`https://github.com/alex-ozdemir/bayesian-ranking-math153`.