Faustina Nyaung (91248764), Jonathan Huang (22516795), Yerline Herrera (60559762)

CS 179 Project Report

Introduction

Our group chose to study and explore Skill Estimation to estimate players' Starcraft II skill level and predict game outcomes between two players. To achieve that, we used pyGM to model skill as a discrete variable based on win/loss data. We assumed that higher skilled players in Starcraft II had a better chance of winning. We also constructed a table of win probabilities given players' skill levels in different matchups. To study the best technique for skill estimation, we compared our algorithm with a simple skill estimation approach that calculates a player's skill level as a fraction of games they have won in the past, in comparison to the amount of games they have played. This allowed us to study how much more accurate skill estimation can be if we rank users' skill level and compute win probabilities for each player. To further test the accuracy of our skill estimation algorithm, we also conducted different tests using different amounts of training data scores to see how much training data is actually necessary for us to calculate a similar amount of correct predictions as when we utilize the full dataset. We chose to study this problem because we noticed it takes a really long time for our model to factor the full dataset and wanted to see if we could minimize this problem because we would ideally not like to spend a lot of time running estimation code on big data.

Resources

To work on this project, our group utilized the example dataset provided in class (a subset of Kaggle's dataset) and the professor's example pyGM model code. We used the professor's example as a foundation for our project by also creating a graphical model, ranking players by predicted skill, and predicting match outcomes. We referenced the Kaggle dataset page's description to understand the data we were given, and adapted the professor's code to explore more ideas and different algorithms. We utilized resources such as YouTube and the professor's LoopyBP demo to further learn about belief propagation, mean field, and loopy belief propagation. Then, our group adapted the professor's code by testing different scale values to see which one best fits our data. We also wrote code for a simple skill estimation approach to compare our results with. This simple skill estimation approach calculates the fraction of games a player wins in relation to all the games they have played (according to the training data). Our group also wrote code to test how the amount of training data we use will affect the accuracy of our skill estimation and we observed any differences in performance.

Evaluation Techniques

PyGM vs. Simple skill prediction approach:

Our group came up with multiple ways to evaluate our skill estimation technique. First we decided to use the entire dataset provided to us and we extracted the game conditions for each entry. Specifically, for each row in the csv file, we got the user's ID (which we made ourselves based on their username), their opponent's ID, and the game outcome. While there is other important information in the dataset that could have been utilized, such as addons, our group was more focused on studying the accuracy of pyGM models in comparison to simpler estimation approaches and the impact different amounts of training data had. To evaluate such, we followed the professor's code example to make a graph model, approximate inference, and rank players based on their estimated skill. To predict match outcomes we went through the validation dataset and extracted the two players IDs and used our algorithm to calculate the predicted match outcome using singleton marginals. While the professor suggested that using a joint belief between two players may provide better results, we wanted to focus our tests on how the accuracy of our skill estimation model changed given different scale parameters and available training data in comparison to a simple estimation approach. After we had predicted an outcome for every match in the validation csv file, we compared it to the actual outcome reported in the dataset and discovered that our algorithm estimated the correct outcome 67.89% (63826/ 94007) of the time using the default scale parameter the professor provided (0.3). We then wrote a simple skill estimation algorithm that went through the training dataset and calculated the fraction of games each player won, and associated that fraction with the user's estimated skill. Then for each matchup in the validation dataset, we compared the two players' estimated skill and predicted that the one with the higher associated skill would win. This approach estimated the correct game outcome 65.81% (61868/94007) of the time. From these results, we were able to deduce that using pyGM to model skill as a discrete variable and creating a table of win probabilities for each player made a positive impact on the correct amount of game outcomes predicted. We arrived at this result without even changing the scale value, which is responsible for translating how skill translates to win probability.

Testing different scale values:

Before testing the accuracy and running time for different amounts of training data, our group decided to briefly experiment using different scale values, rather than 0.3 which the professor used in his example. We thought it would be a good idea to change the scale value since it is responsible for scaling how the difference in player skill translates to win probability. Since Starcraft II is a strategy game, we believed that a player's skill level would lead to victory more often. To test our theory, we decided to alter the code and try three new scale values 0.4, 0.5, and 0.6. This allowed us to have quantitative results we could compare. When we used 0.4 as the scale value, our algorithm estimated the correct validation set's game outcome 68.17% (64089/ 94007) of the time. When we used 0.5 as the scale value, our algorithm estimated the correct validation set's game outcome 68.22% (64133/ 94007) of the time. Lastly, when we used 0.6 as the scale value, our algorithm estimated the correct validation set's game outcome 68.27% (64175/ 94007) of the time. From these results we noticed that there was a small increase in accuracy (around 0.05%) as we increased the scaling factor, but it took a lot longer for our code to finish running. Since the difference between accuracy is very small and we do not want to overfit the validation data, we decided to use a scaling value of 0.5 for the next set of tests we conduct.

Testing accuracy and running time for different amounts of training data:

Our group was curious to see if the amount of training data we used had an effect on the accuracy of our model. We decided to explore this question since it took a really long time to run our code on the full training dataset and if we were doing a similar skill estimation technique at a job with big data, we cannot afford for our running time to be unnecessarily long. We ran three different trials where we shrunk the amount of training data used by 10%, 20%, and 70%. To make sure we still had a variety of different players in the training data, we made sure to pick a few rows per user ID. We then compared the accuracy from each of these predictions with the results we obtained from learning on the full dataset. We also compared the time it took for each of these tests to run, to see if there was an overall benefit to using a smaller set of training data. From these tests we observed that by using 10% less training data (173767/193074 rows in the train.csv file), the correct validation set's match outcome was predicted 68.12% of the time in 29 minutes (1746.253 seconds). By using 20% less training data (154459/193074 rows in the train.csv file), the correct validation set's match outcome was predicted 68.11% of the time in around 23 minutes (1362.578 seconds). Lastly, by using 70% less training data (57922/193074 rows in the train.csv file), the correct validation set's match outcome was predicted 67.49% of the time in 4 minutes (253.393 seconds). In comparison, it took around 49 minutes (2911.937 seconds) to run our skill estimation/ match outcome predictor algorithm on the full dataset with 68.22% accuracy. These results suggest that having larger amounts of data does not significantly increase the accuracy. Since Starcraft is a strategy game that is largely skill based, we may not even need many games from a player to determine their skill, so long as they play people from a variety of different skill levels. If a skilled player plays many lower rated players, the results do not provide a good upper bound on their skill, and vice versa.

Conclusion

The dataset we were given allowed us to experiment with different scaling factors, different estimation algorithms, and different training data sizes to gain an understanding of how all these factors play a role in forming predictions on big data in a workplace. After running our tests, we came to the conclusion that the scaling factor is very important to express how much impact "skill" has on a game's outcome. We came to this conclusion after increasing the professor's default scaling value from 0.3 to 0.5 since the game we were analyzing, Starcraft II, requires a lot more skill than the mock game the professor was analyzing in this example. After increasing the scale factor, we noticed that our algorithm predicted the validation data's game outcomes more accurately since skill played a much larger role in Starcraft than we initially gave it credit for. Our team also created a simple skill estimation algorithm to compare against our discrete variable skill model and noticed that our model was a lot more accurate when we created a graphical model and ranked players by predicted skill rather than the fraction of games a player won. This is because the game is highly skill-based. Lastly, we experimented with different training dataset amounts to see if it was necessary to use the full training set we were given since we want to limit unnecessary running times when working with large data in the future. We also wanted to make sure that the accuracy of our skill estimator and outcome predictor was not compromised. From our experiments, we discovered that using the scale value of 0.5 would be good enough to prevent overfitting our test data. We also realized that using less training data would not make a significant difference in accuracy since cutting off 70% of the training data would only lead to 1% difference in accuracy in comparison to using the entire training data set. As a result of using less training data, our run time would shorten by 92%.