CS179: Intro to Graphical Models

Final Project

06/14/2023

1. Describe Problem/Analyze Dataset

Our group decided to work with the Skill estimation dataset for the project. The dataset itself consists of textual rows which represent the Starcraft game stats information played by users separated by comma.

```
…
9/19/2016,MC,[loser],0–2,Stats,[winner],P,P,LotV,online
09/19/2016,MC,[loser],1–2,Dark,[winner],P,Z,LotV,online
…
```

This is the visualization of the dataset.

2. Model Tuning

We used pyGM where we could model skill as a discrete variable and construct a table of win probabilities given the two players' skills.

Steps:

- First we loaded data from appropriate .csv files. We wrote a function, and passed appropriate arguments to load the data.
- After we get nplayers, nplays, nwins from load data we created a get_game function which return a list of games and outcomes.
- Then, we created the function called get_factors where it defines a function that creates factors and variables for a graphical model representing the win probability of a game. It iterates over the games and calculates win probabilities based on skill differences, then adds them as factors to the graphical model along with uniform belief factors for each player's skill level.
- Furthermore, the model_inference function estimates the skill level in a given graphical model by performing either brute force inference or approximate inference using loopy belief propagation or mean field methods. It returns the estimated beliefs about the skill levels and the original model.

Cem Emir Senyurt (45064076, csenyurt)
Orhan Ozbasaran (63039729, oozbasar)

- The function display_ability compares the ability of a model to estimate game outcomes based on the provided beliefs, variables, win probabilities, and a validation set of games. It iterates over the validation games, calculates the probability of the observed outcome using the model's beliefs and win probabilities, and counts the number of correct estimations.

- Lastly, the function display_game_prediction takes beliefs, variables, win probabilities, a specific game, and player names as inputs. It calculates and displays the estimated probability of a specified player beating another player in the next game based on their skill levels, using the model's beliefs and win probabilities. The function considers the order of players, calculates the probability accordingly, and prints the result.

3. Sample Run:

The output shows that the total number of games played, the number of correct estimations, the number of wrong estimations, and the fraction, represents the accuracy or success rate of the model in predicting the outcomes of the validation games, of correct estimations as a percentage. In this sample run, out of 4696 games, the model made 3839 correct estimations, resulting in a 81.75% accuracy.

4. Evaluating the Required Games to Accurately Predict Players' Skills:

In this section, the focus is on evaluating the effect of the amount of training data on the accuracy of player skill prediction. The experiment is conducted for each value of pKeep to track changes in prediction accuracy. The variable pKeep is modified to use various portions of the training data.

The load_data function is first used to load the validation data. Then, a loop is run across each entry in the pKeep_values list for pKeep. The relevant pKeep value is put into the training data on each iteration, and the get_games function is then used to extract a list of games from the training data.

Next, the get_factors function is called to create factors and variables for a graphical model representing the win probability of a game.
The skill levels are ranked by predicted skill, and the display_ability function is called to compare the ability of the model to estimate game outcomes using the obtained beliefs, variables, win probabilities, and the validation set of games. The total number of games played, the number of correct estimations, the number of wrong estimations, and the fraction (accuracy) of correct estimations are printed and stored in the accuracy_results list.

Finally, a plot is generated to visualize the relationship between the fraction of training data used (pKeep_values) and the prediction accuracy (accuracy_results). The x-axis represents the fraction of training data used, and the y-axis represents the prediction accuracy. The plot helps in understanding how the prediction accuracy is affected by the amount of training data available.

Overall, this part of the report demonstrates the evaluation of the model's performance by varying the training data size and analyzing the resulting prediction accuracy. Based on the prediction accuracy values and the corresponding fractions of training data, we can observe the following outcomes:

When using 10% (0.1) of the training data (pKeep = 0.1), the prediction accuracy is 79.94%.

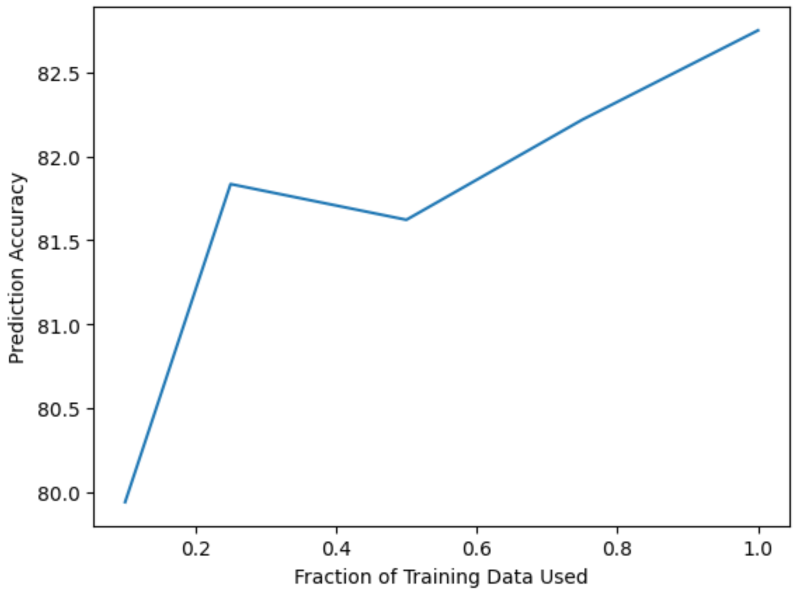
When using 25% (0.25) of the training data (pKeep = 0.25), the prediction accuracy is 81.83%.

When using 50% (0.5) of the training data (pKeep = 0.5), the prediction accuracy is 81.62%.

When using 75% (0.75) of the training data (pKeep = 0.75), the prediction accuracy is 82.21%.

When using 100% (1.0) of the training data (pKeep = 1.0), the prediction accuracy is 82.75%.

These outcomes suggest that as more training data is used, the prediction accuracy generally improves. The highest accuracy of 82.75% is achieved when using the entire training data. However, it is important to note that the accuracy improvement becomes relatively smaller as more data is added, indicating diminishing returns in terms of accuracy gains.



5. Evaluating How Quickly We Can Determine a New Player's Skill:

In this part of the code, we are evaluating how quickly we can determine a new player's skill by gradually adding their games into the dataset and observing how the skill estimation accuracy changes. First, we load the existing player data from the 'train.csv' file and obtain the games dataset. Then, we randomly choose a player, referred to as 'playerOut,' whose games will be excluded from the dataset. This player will serve as the new player whose skill we want to estimate. We divide the games dataset into two subsets: 'games_without_pOut' and 'games_with_pOut.' The former contains all the games except the ones played by 'playerOut,' while the latter contains only the games played by 'playerOut.'

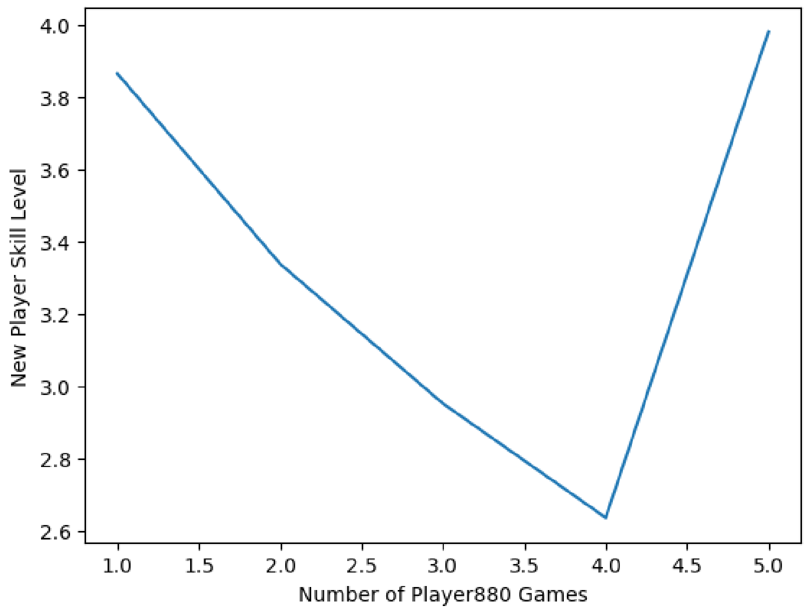
Next, we iterate over the games in 'games_with_pOut' one by one, gradually adding more games to the combined dataset. For each iteration, we update the factors, X, and Pwin based on the combined dataset. We then create a graph model, perform model inference, and obtain the skill estimates for all players in the dataset. The current skill estimate for 'playerOut' is stored in the 'mean_skill_estimates_new_player' list. After each iteration, we print the current skill level estimate for 'playerOut.' Finally, we plot the skill level estimates against the number of games added for the new player.

This process allows us to observe how the skill estimation for the new player converges as more of their games are included in the dataset. The plot shows the trend of the new player's skill estimation as the number of games increases.

Skill Level Estimates: The skill level estimates for the excluded player ('playerOut') are calculated and printed after each iteration. The skill levels obtained are as follows:

- After the first game: Skill level estimate = 3.86
- After the second game: Skill level estimate = 3.33
- After the third game: Skill level estimate = 2.95
- After the fourth game: Skill level estimate = 2.63
- After the fifth game: Skill level estimate = 3.97

These values represent the estimated skill level of the excluded player based on the gradually increasing number of games added to the dataset.



Cem Emir Senyurt (45064076, csenyurt)
Orhan Ozbasaran (63039729, oozbasar)