Bayesian Analysis of Esports Teams' Perfomances

Aalto University

Ekaterina Marchenko Georgios Karakasidis Rongzhi Liu

02 December 2020

Introduction

In this report, we present a Bayesian Analysis approach for evaluating team performances for the esport *Dota 2*. This topic was chosen due to the personal interest of our team members in esports. We formed the group based on this and all of us were familiar with either Dota 2 or League of Legends.

Before describing the problem thouroughly, we would like to provide a brief insight into the scope. Dota 2 is a MOBA game and an esports discipline. Esports have a similar structure as traditional sports, where there are professional seasons which roughly equal to a year, and these are finished with a main tournament (in our case: *The International (TI)*).

The tournament itself consists of two stages: qualifications and main event. The latter is also splitted into two parts: group stage, and play-offs. During the group stage, teams compete in order to have a better starting position in the play-off stage. For our analysis, we used the data from the last TI where the best teams of the world competed.

The professional Dota 2 scene is divided into different regions (e.g. Europe, CIS, China, etc). Teams from this regions participate in qualifications separately, and the very first goal of the team is to become the best within their region. As a result, teams from one region play with each other much more often, and it can influence the way they perform. Hence, we also used information about teams regions while modelling.

Our aim is to find a way to predict the match result based on the performances of the competing teams (the way performances are computed is discussed later). In other words, we propose a way to make the analysis of the previous matches in which teams participated, and take those results to predict how they would perform in the future.

Data and Preprocessing

In our analysis, we decided to use the results of **The International 2019** Tournament, the results of which are available online through some APIs. The data was gathered with the Python programming language in two steps. At first, match IDs were collected from DotaBuff. Afterwards, we used the obtained IDs in order to acquire detailed information about each match. For the latter we used the OpenDota API.

For our analysis, we used the data from the group stage of the tournament as each team played the same amount of matches. On the other hand, for the posterior predictive checks (which will be discussed in the next section), we made predictions for the play-off stage of TI9.

The next step was to define a new metric in order to quantify the teams' performances. To do this we used a mixture of different measures/scores which we then normalized and used its sum. The initial features can be seen below:

Variable Names	Description
radiant_score	Final score for the Radiant team(number of kills on Radiant)
dire_score	Final score for the Dire team (number of kills on Dire)
radiant_xp_adv	Array of the Radiant experience advantage at each minute in the game. A negative number means that Radiant is behind, and thus it is their experience disadvantage.
radiant_gold_adv	Array of the Radiant gold advantage at each minute in the game. A negative number means that Radiant is behind, and thus it is their gold disadvantage.
hero_damage	Hero Damage Dealt (user specific)
hero_healing	Hero Healing Done (user specific)
obs_placed	Total number of observer wards placed (user specific)
kda	kda (ratio of kills/deaths/assists) (user specific)

These features were extracted for each match and contain information about both of the teams. Since each Dota 2 game consists of 5 players, this means that we end up with 10 data points for each match.

As you can see from the table above, the first 4 features have to do with team statistics while the rest 4 have to do with user-specific statistics since they evaluate the performance of each player separately. Our goal is to find a way to combine these features so that the result will be a performance score for each team, for each match.

Our approach to this can be described by the following equation which assigns a score to each team for each match:

$$R_{score} = score\_gap + exp\_advantage + gold\_advantage +$$
$$\sum_{\forall p \in P_R} hero\_damage(p) + healing\_done(p) + kda(p) + wards\_placed(p)$$

where R_{score} denotes the radiant team score, and P_R is the set of all radiant players on the specific match. The same was also applied for the dire team.

NOTE: This equation is over-simplified since it does not take normalization into account (having all values in the same range).

Each of these functions is described below:

Variable Names	Description
score_gap	Radiant score minus Dire score (so it can also be negative).
exp_advantage	A weighted sum of the experience advantage that Radiant had during the game (can also be negative). We are giving bigger weights to the scores at the end of the game since they are more informative about the final result.
gold_advantage	A weighted sum of the gold advantage that Radiant had during the game (can also be negative). We are giving bigger weights to the scores at the end of the game since they are more informative about the final result.
hero_damage	Simplistic Approach: The difference of the sum of the damage dealt by all Radiant players when compared to Dire players.
healing_done	Simplistic Approach: The difference of the sum of the healing that was done by all Radiant players when compared to Dire players.
kda	Simplistic Approach: The difference of the sum of kdas of all Radiant players when compared to Dire players.

Variable Names	Description
wards_placed	Simplistic Approach: The difference of the sum of the number of observation wards that were placed by all Radiant players when compared to Dire players.

As you can see, a rather simplistic approach was used to combine the data but this was the most effective one. As a normalization technique we are currently using MinMax scaling which will not convert our values to positive numbers and so it will keep the notion of:

- Positive performance scores should probably result in a Radiant win (not always).
- Negative scores should point us to the direction that Dire will most probably win.

For more information about the implementation you may check the data transformation appendix. An important note here is that, for each match, the *score_gap*, *exp_gap* and *gold_advantage* columns for the Dire team are multiplied by -1 so that they will be positive in case the Dire has positive statistics (and negative otherwise).

Problem Solving

By preprocessing our data we managed to get a single scoring measurement for each team. Our next step is to build a model on top of these transformed data and try to predict/sample future scores (future performances). We are going to represent our data as a 2-dimensional array which will contain information about each team. The rows will denote the matches and the columns will denote the teams, while the content of each array cell will be the above performance scores.

Below, we are going to describe two model-architecture approaches which we saw during the course.

Hierarchical Model

The goal here is to create a hierarchical model for the teams of each region. We achieved this by using rstan (Stan Development Team 2020) (see appendix).

The prediction workflow for a match between team X and Y of region is the following:

1. Sample a new performance score for team X.
2. Sample a new performance score for team Y.

Choice of Priors

Separate Model

For comparison, we also built a separate model which "..."

Choice of Priors

Convergence Diagnostics

Posterior Predictive Checks

Model Comparison

Predictive Performance Assessment

Sensitivity Analysis

Discussion of Issues and Possible Improvements

Conclusion

Self-Reflection

Appendix

Data Transformation

Below, we present the python functions that were used for data preprocessing and for computing the performance for a single team of a single match.

```python
def _score_gap(match):
    # Simple score difference
    score = match[['radiant_score', 'dire_score']].values[0]
    return score[0] - score[1]

def _xp_gap(match, use_weights=True):
    # Take the sum of the xp advantages (over all minutes)
    xp_list = np.array(ast.literal_eval(pd.unique(match.radiant_xp_adv)[0]))
    # A negative sum would mean that the radiant team was on a disadvantage
    # most of the times
    if use_weights:
        # The last few minutes are more important and so we should give a
        # greater weight to these
        weights = np.linspace(0.1, 1, num=len(xp_list))  # avoid zero weights
        xp_list_weighted = weights * xp_list
        return np.sum(xp_list_weighted)
    return np.sum(xp_list)

def _gold_advantage(match, use_weights=True):
    gold_list = np.array(ast.literal_eval(pd.unique(match.radiant_gold_adv)[0]))
    # A negative sum would mean that the radiant team had less gold for
    # most of the game
    if use_weights:
        weights = np.linspace(0.1, 1, num=len(gold_list))
        gold_list_weighted = weights * gold_list
        return np.sum(gold_list_weighted)
    return np.sum(xp_list)
```

```
def _hero_specific_scores_v2(match):  # hero damage and hero healing
    stats = {
        "damage": match['hero_damage'].values,
        "healing": match['hero_healing'].values,
        "kda": match['kda'].values,
        "wards": match['obs_placed'].values,
    }  # will contain the damage and healing scores for the radiant players
    # Simply use the sum
    dmg_diff = np.sum(stats['damage'])
    heal_diff = np.sum(stats['healing'])
    kda_diff = np.sum(stats['kda'])
    wards_diff = np.sum(stats['wards'])
    return dmg_diff, heal_diff, kda_diff, wards_diff

def performance(match):
    # match should be a dataframe that contains 5 rows
    #    (a match entry for a certain team)
    score_gap = _score_gap(match)
    xp_gap = _xp_gap(match, use_weights=True)
    gold_adv = _gold_advantage(match, use_weights=True)
    # case 3 (simplistic approach)
    total_dmg, total_heal, total_kda, total_wards = _hero_specific_scores_v2(match)
    return score_gap, xp_gap, gold_adv, total_dmg, total_heal, total_kda, total_wards
```

By using the performance function above for both teams of every match we can get a new dataset which will contain all of the 7 characteristics returned by *performance*. We can then normalize these values with the method mentioned in the Data and Preprocessing section and then simply sum the values of each row in order to get the final performance score.

Stan code

TODO: ADD STAN CODE HERE

References

Stan Development Team. 2020. "RStan: The R Interface to Stan." http://mc-stan.org/.