GEORGETOWN UNIVERSITY

ANALYTICS 561

PROJECT REPORT

League of Legend Result Prediction Using Gradient Boosted Tree

Authors:
Tianyu YANG
Wen CUI
Ze ZHENG
Cheng ZHONG

December 10, 2017

Abstract

In this project, we try to explore the influence of the in-game features on predicting the match result. Specifically, we constructed two variable groups, one for competitive features, and one for in-game features. And then, we conducted the experiment by trained two predictive models either with or without in-game features. In this case, we use gradient boosting classifier as predictive model. The results showed that in-game features have significant influence on the target variable.

Contents

1 Introduction

Electronic Sports (Esports), as a modern branch of the traditional sports, have received growing attention in recent years thanks to ground-breaking changes and shifts in network technology and media. In response to the increasing fans and their desires to see high-skilled performances in online competitive games, gaming clubs, leagues, and tournaments of various sizes and scopes sprouted all around the world, leading to even deeper engagement of the players and the fans. As a result, the crowd demands better understanding and analysis. League of Legends(LOL) is one of the most popular Multi-player Online Battle Arena(MOBA) game and attracts the most players in the world. Meanwhile, LOL is very competitive and the S-series world championship game is one of the most effective e-sport tournament in the world. In this project, we hope to offer a detailed analysis on LOL through game result prediction using gradient boosted tree classification by pre-game features and in-game features.

During the course of a match, a summoner takes control of one champion and uses skills and basic attacks to fight the champions of other players towards the ultimate goal of destroying the enemy base. A particular ranking game can be separated by several phases: matching phase, ban-pick phase and laning phase. In the matching phase, ten players will be divided into two teams randomly. During the ban-pick phase, each team will ban 5 champions and also pick 5 champions. In the laning phase, players controls different champions and champions can increase their levels, thus enhancing champions' attributes and skills, by gaining experience through killing enemy champions, jungling and killing minions. Also, by killing enemy champions and minions, destroying towers and inhibitors, champion can earn golds to buy items to improve attributes, which help win the game.

Ultimately, we believe that winning of a match occurs through the accumulation of in-game benefits. Such benefits are obtained through line-up selection which results from champions bans and picks, attributes boosts coming from minions kills, neutral monsters kills, champion kills, and turrets kills, all of which either enhance attributes directly via buff, or grant experiences and gold to level champions up and buy better items. Therefore, we only consider relevant features in the dataset for the prediction(classification).

2 Data and Exploratory Data Analysis

We used the LOL match data from Kaggle. The dataset contains records for 25000 ranked EU(Europe) game, as well as json files containing dictionaries for the encoding of champions' names, and summoners' spell names.

For each observation(a ranking match), the relevant features include:

- Game Duration(in seconds)
- Winner (1 = team1(Blue), 2 = team2(Red))
- First Baron, dragon, tower, blood, inhibitor and Rift Herald kills(1 = team1, 2 = team2,0 = none)
- Total Baron, dragon, tower, inhibitor and Rift Herald kills for each team
- Summoner spells (stored as Riot's summoner spell IDs)

- Champions selected by each team(stored as Riot's champion IDs)
- Champions banned by each team(stored as Riot's champion IDs)

The games.csv includes all the ranking information we have in the game and the file includes 23669 rows and 61 columns. Also, we have champion_info.json and champion_info_2.json including all the champion information. Furthermore, we have summoner_spell_info.json that has the summoner skills information.

By screening through the dataset games.csv, we found that in the firstTower column, there are some data missing firstTower Kill in the game.However, it make nonsense that there is no firstTower kill in a ranked game. As we can see in Figure 1, the boxplot indicates that the game duration for those games are less than 300 seconds, which indicates those games are not successfully finished. In that case, we removed all the rows with zero firstTower kill in order to make the data more reasonable. As we can see from the bar chart(Figure 2), there are slightly difference between blue team and red team, but overall, two teams have similar result.

3 Optimization Problem

For classification, we engage the gradient boosted tree method. The method consists of several processes, gradient descent and boosting. The method takes an initial classifier, a weak learner. In the process of boosting, the classifier is applied on the data and residuals are obtained. Then, the residuals are considered to build a slightly improved classifier. Such process is repeated indefinitely until a strong classifier is built. Gradient descent is employed to find the optimal classifier at each boosting step.

Specifically, if we have a sample space $\mathscr{D}_n = \{(\mathbf{X}_1, Y_1), \cdots, (\mathbf{X}_n, Y_n)\}$ of independent observations, where each $\mathbf{X}_i \in \mathbb{R}^d$, and $Y_i \in \{0,1\}$. Suppose we have the weak learner,$F : \mathbb{R}^d \to \mathbb{R}$ and perform M gradient boosting step, then we will have $F_1, F_2, \cdots, F_M$, where each F_i for $i \in \{1,2,\cdots,M\}$ is obtained by minimizing the expected value of some loss function

$$\min_{F_i} E_{\mathbf{x},y}[L(F_i(\mathbf{X}), Y)]$$

and F_{i+1} is improved upon F_i.

4 Technical Details

In the case of classification, we follow [2] and use the negative binomial log-likelihood(deviation), $L(F,Y) = log(1+e^{-2yF})$, as the loss function. Normally, one would restrict $F(\mathbf{X})$ to be a parametric function $F(\mathbf{x};\mathbf{P})$, where $\mathbf{P} = \{P_1, P_2, \cdots\}$ is a finite set of parameters that eventually determine the individual class member. However, when engaging boosting, we emphasize on the additive expansions of the form

$$F(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{1}$$

. The classifier $h(\mathbf{x}; \mathbf{a}_m)$ usually will be a simple parametric function of $\mathbf{x}$ with $\mathbf{a}_m = \{a_1, a_2, \cdots\}$ and βs are the weights of the classifiers obtained from the boosting steps. To solve the optimization, we uses numerical optimization method in the context of gradient boosting. With a parametric objective function $F(\mathbf{x};\mathbf{p})$, the original program becomes,

$$\mathbf{P}^* = \min_{\mathbf{P}} \phi(\mathbf{P}) \tag{2}$$

, where

$$\phi(\mathbf{p}) = E_{\mathbf{x},y}[L(F(\mathbf{X};\mathbf{P}),Y)]$$

, and

$$F^* = \min_{F} E_{\mathbf{x},y}[L(F(\mathbf{X};\mathbf{P}),Y)] = F(\mathbf{x};\mathbf{P}^*).$$

Numerical methods must be employed to solve (2), the solution to which often takes on the form

$$\mathbf{P}^* = \sum_{m=0}^{M} \mathbf{p}_m \tag{3}$$

, where $\mathbf{p}_0$ is an random initial guess and $\{\mathbf{p}_m\}_1^M$ are following boosts based on previous steps.

Steepest-descent is a commonly used method to solve (2). [6] proves the convergence of the gradient descent method. The proof is based on four principal conditions. Let f be a real-valued function defined and continuous on $E^n \subseteq \mathbb{R}^n$, (real Euclidean n-space) and bounded below. For fixed $\mathbf{x_0} \in E^n$ define $S(\mathbf{x_0}) = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$. We define $|.|$ as the Euclidean norm. The function satisfies:

- Condition 1: if there exists a unique point $\mathbf{x}^* \in E^n$ such that $f(\mathbf{x}^*) = \inf_{\mathbf{x} \in E^n} f(\mathbf{x})$.
- Condition 2: at $\mathbf{x}_0$ if f is continuously differentiable on $S(\mathbf{x}_0)$ and $\nabla f(\mathbf{x}) = \mathbf{0}$ for $x \in S(\mathbf{x}_0)$ if and only if $\mathbf{x} = \mathbf{x}^*$
- Condition 3: at $\mathbf{x}_0$ if f is continuously differentiable on $S(\mathbf{x}_0)$ and there exists a constant $K > 0$ such that $|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x}) \leq K|\mathbf{y} - \mathbf{x}|$ for every pair of $x, y \in S(\mathbf{x}_0)$.
- Condition 4: at $\mathbf{x}_0$ if f is continuously differentiable on $S(\mathbf{x}_0)$ and if $r > 0$ implies $m(r) > 0$ where $m(r) = \inf_{\mathbf{x} \in S_r(\mathbf{x}_0)} |\nabla f(\mathbf{x})|$, $S_r(\mathbf{x}_0 = S_r \cap S(\mathbf{x}_0)$, $S_r = \{\mathbf{x} : |\mathbf{x} - \mathbf{x}^* \geq r\}$, and $\mathbf{x}^*$ is any point for which $f(\mathbf{x}^*) = \inf_{\mathbf{x} inE^n} f(\mathbf{x})$. (if $S_r(\mathbf{x}_0)$ is void, we define $m(r) = \infty$.

It follows that Condition 4 implies Condition 1 and 2, and if $S(\mathbf{x}_0)$ is bounded, then Condition 4 is logically equivalent to Conditions 1 and 2.

Theorem 1 (The convergence theorem)**.** *If $0 < \delta \leq 1/4K$, then for any $\mathbf{x} \in S(\mathbf{x}_0$, the set*

$$S^*(\mathbf{x}, \delta) = \{\mathbf{x}_\lambda : \mathbf{x}_\lambda = \mathbf{x} - \lambda \nabla f(\mathbf{x}), \lambda > 0, f(\mathbf{x}_\lambda) - f(\mathbf{x}) \leq -\delta |\nabla f(\mathbf{x})|^2\} \tag{4}$$

is a nonempty subset of $S(\mathbf{x}_0)$ and any sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ such that $\mathbf{x}_{k+1} \in S^*(\mathbf{x}_k, \delta)$, $k = 0, 1, 2, \cdots$, converges to the point $\mathbf{x}^*$ which minimizes f.

Proof. If $\mathbf{x} \in S(\mathbf{x}_0)$, $\mathbf{x}_\lambda = \mathbf{x} - \lambda \nabla f(\mathbf{x})$ and $0 \leq \lambda \leq 1/K$, Condition 3 and the mean value theorem imply the inequality $f(\mathbf{x}_\lambda) - f(\mathbf{x}) \leq -(\lambda - \lambda^2 K)|\nabla f(\mathbf{x})|^2$ which then implies that $\mathbf{x}_\lambda \in S^*(\mathbf{x}, \delta)$ for

$$\lambda_1 \leq \lambda \leq \lambda_2, \lambda_i = \frac{1}{2K}(1 + (-1)^i \sqrt{1 - 4\delta K}),$$

so that $S^*(\mathbf{x}, \delta)$ is a nonempty subset of $S(\mathbf{x}_0)$. If $\{\mathbf{x}\}_{k=0}^{\infty}$ is any sequence for which $x_{k+1} \in S^*(\mathbf{x}_k, \delta)$, $k = 0, 1, 2, \cdots$, then (4) implies that sequence $\{f(\mathbf{x}_k)\}_{k=0}^{\infty}$, bounded below, is decreasing. Thus, $|\nabla f(\mathbf{x}_k)| \to 0$ as $k \to \infty$. The remainder of the theorem follows from Condition 4. □

Corollary 1.1 (The Steepest Descent Algorithm)**.** *If*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{2k}\nabla f(\mathbf{x}_k), k = 0,1,2,\cdots$$

then the sequence $\{\mathbf{x}\}_{k=0}^{\infty}$ *converges to the point* $\mathbf{x}^*$ *which minimizes* f*.*

Proof. It follows the proof of the convergence theorem that the sequence $\{\mathbf{x_k}\}_{k=0}^{\infty}$ is $\mathbf{x}_{k+1} \in S^*(\mathbf{x}_k, 1/4K), k = 0,1,2,\cdots$. □

In the process steepest descent, $\{\mathbf{p}_m\}_1^M$, the increments, as follows. To start with, the current gradient $\mathbf{g}_m$ is computed as:

$$\mathbf{g}_m = \{g_{jm}\} = \{[\frac{\partial \phi(\mathbf{P})}{\partial P_j}]_{\mathbf{P}=\mathbf{P}_{m-1}}\}$$

, where

$$\mathbf{P}_{m-1} = \sum_{i=0}^{m-1} \mathbf{p}_i.$$

The step is,

$$\mathbf{p}_m = \rho_m \mathbf{g}_m$$

where

$$\rho_m = \min_{\rho} \phi(\mathbf{P}_{m-1} - \rho \mathbf{g}_m). \tag{5}$$

Then, $-\mathbf{g}_m$ is the steepest-descent direction and (4) is called the line search along the direction.

Since the loss function is defined on the distribution of $(y, \mathbf{x})$ yet we only have a finite set of data, $\{y_i, \mathbf{x}_i\}_1^N$, we only optimize the data based estimate of expected loss,

$$\{\beta_m, \mathbf{a}_m\}_1^M = \min_{\{\beta_m', \mathbf{a}_m'\}_1^M} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m' h(\mathbf{x}_i; \mathbf{a}_m')).$$

In situations where this is infeasible one can try a greedy stagewise approach. For $m = 1,2,\cdots,M$,

$$(\beta_m, \mathbf{a}_m) = \min_{\beta, \mathbf{a}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a})) \tag{6}$$

and

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{7}$$

In the case of regression tree,the technique we employ for classification, each weak learner is an J-terminal node regression tree [3]. Each regression tree model has the additive form

$$h(\mathbf{x} : \{b_j, R_j\}_i^J) = \sum_{j=1}^{J} b_j \mathbf{1}(\mathbf{x} \in R_j). \tag{8}$$

The above $\{R_j\}_1^J$ are disjoint regions that together cover the space of all joint values of the predictor variables $\mathbf{x}$. These regions are represented by the terminal nodes of the corresponding tree. The function $\mathbf{1}(.)$ has the value 1 if its argument is true, 0 otherwise. The parameters of (8) are $\{b_j\}_1^J$ and the boundaries of the regions $\{R_j\}_1^J$. Since the regions are disjoint, (8) is equivalent to the prediction rule: if $\mathbf{x} \in R_j$,$h(\mathbf{x}) = b_j$. Then, the update at each boosting step is

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m \sum_{j=1}^{J} b_{jm}\mathbf{1}(\mathbf{x} \in R_{jm}). \tag{9}$$

Here $\{R_{jm}\}_1^J$ are the regions defined by the terminal nodes of the tree at the mth iteration. They are constructed to predict $\{\tilde{y}_i\}_1^N$ by least-squares. The $\{b_{jm}\}$ are the least-squares coefficients,

$$b_{jm} = ave_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i$$

. The scaling factor(weight), β_m is

$$\beta_m = \min_{\beta} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h)$$

. Then the update is

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm}\mathbf{1}(\mathbf{x} \in R_{jm})$$

with $\gamma_{jm} = \beta_m b_{jm}$. Then, one can further improve the quality of the fit by using the optimal coefficients fo reach of these separate basis function (9). These optimal coefficients are the solution to

$$\{\gamma_{jm}\}_i^J = \min_{\{\gamma_j\}_i^J} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i + \sum_{j=1}^{J} \gamma_j \mathbf{1}(\mathbf{x} \in R_{jm})).$$

Since the regions are disjoint, the above equation simplifies to

$$\gamma_{jm} = \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$$

through line search (steepest descent).

Algorithm:

```
  $F_0(\mathbf{x}) \leftarrow \mathbf{median\{y_i\}_i^N}$
  for $m = 1$ to $M$ do: do
    $\tilde{y}_i \leftarrow sign(y_i - F_{m-1}(\mathbf{x_i})), i = 1, N$
    $\{R_{jm}\}_1^J \leftarrow$ J-terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
    $\gamma_{jm} \leftarrow \min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    $F_m(\mathbf{x}) \leftarrow F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm}\mathbf{1}(\mathbf{x} \in R_{jm})$
  end for
```

For the application of the algorithm, we used the GradientBoostingClassifier from sklearn.ensemble. To successfully run the function, we converted all categorical variables to dummy variables with the number equal to the total number of categories in the original categories. To implement this, we read the data file to a pandas data frame and apply the pandas.getdummies function. To study the roles of pre-game and in-game features, we subset pre-game features from the dataset and make comparison on the classification result.

5 Results and Conclusion

We first use gradient boosting classifier to test the accuracy score for the data only including pre-game features (line-up, summoners spell for each team member) and it performed a nearly 60% accuracy score. After adding more aspects of all - time features, the prediction accuracy has achieved 93% in contrast to 60% accuracy when we only use pre-game feature to predict the match result. As the figure 3, the ROC score of all-game feature is significantly higher than the score of pre-game features. It suggests that the all-game features are more informative than pre-game features when we make the prediction of the game.

The results suggest several interesting points. First of all, classification using only pre-game features yields an accuracy score better than random guessing. It confirms our hypothesis that line-up selection generates benefits that eventually affects the match result. Therefore, study on champions cooperation and champions' relative advantages and disadvantages over other champions has to be conducted to improve overall gaming skills. However, the accuracy(60%) is only slightly better than random guessing, which is reasonable enough. Since LOL is a game of both strategy and controls, a 60% accuracy indicates that most factors are from in-game controls, where most of the competitiveness and fun come from. After all, if we want a pure strategic game, we would be playing chess. Next, adding in-game features improve the accuracy to over 90%. This suggests that benefits accumulated during the game play an extremely significant role in winning.

The next step on the topic could focus on what in-game features are more important over others and analysis on real-time data to find out significant feature while playing the game. Moreover, comparisons can be made between lower rank matches and higher rank matches to extract insights on what ultimately distinguishes the skill level of players.

References

[1] G.Biau,B.Cadre, *Optimization by Gradient Boosting*, 2017

[2] J.Friedman, *Greedy Function Approximation: A Gradient Boosting Machine*, Institute of Mathematical Statistics, 2001

[3] L.Devroye, L.Gyorfi, G.Lugosi, *A Probabilstic Theory of Pattern Recognition*, Chapter 20, 1996

[4] L.Lin, *League of Legends Match Outcome Prediction*, Stanford University

[5] Z.Li, D.Cui,C.Li, *Dota2 Outcome Prediction*, 2017

[6] L. Armijo, *Minimization of functions having lipschitz continuous first partial derivatives*, Pacific Journal of Mathematics, Vol. 16,No.1,1966

6 Appendix

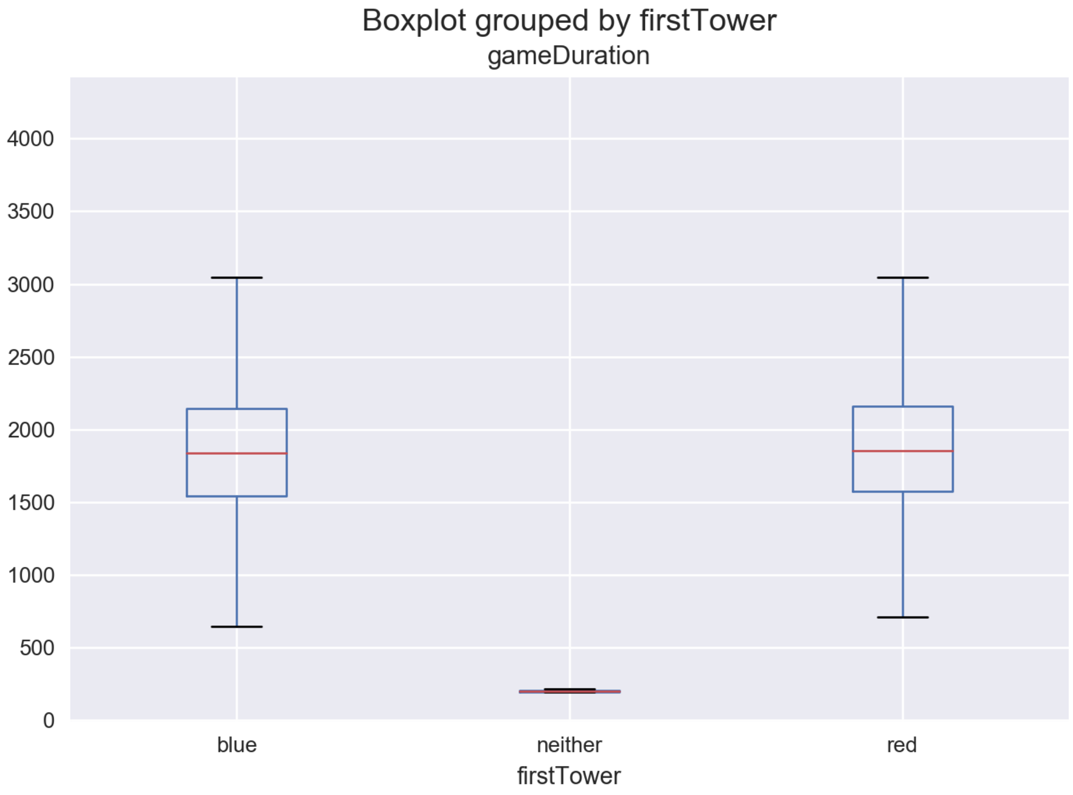


Figure 1: The game duration time for neither team got first tower is extremely low

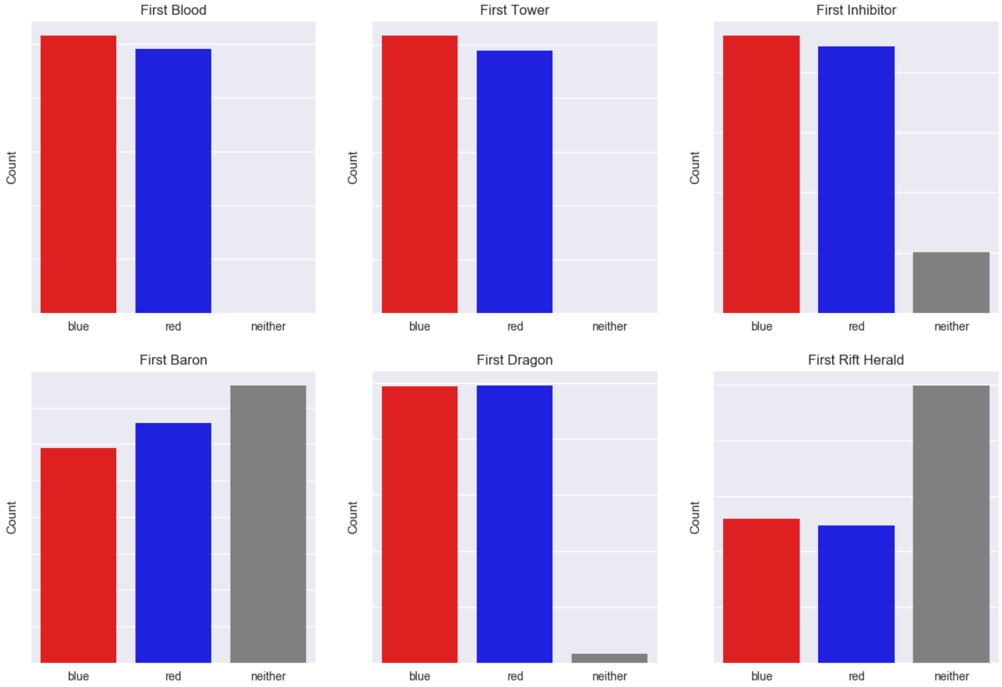


Figure 2: This is the barchart comparison for First Blood, First Tower, First Inhibitor, First Baron, First Dragon, and First Rift Harald

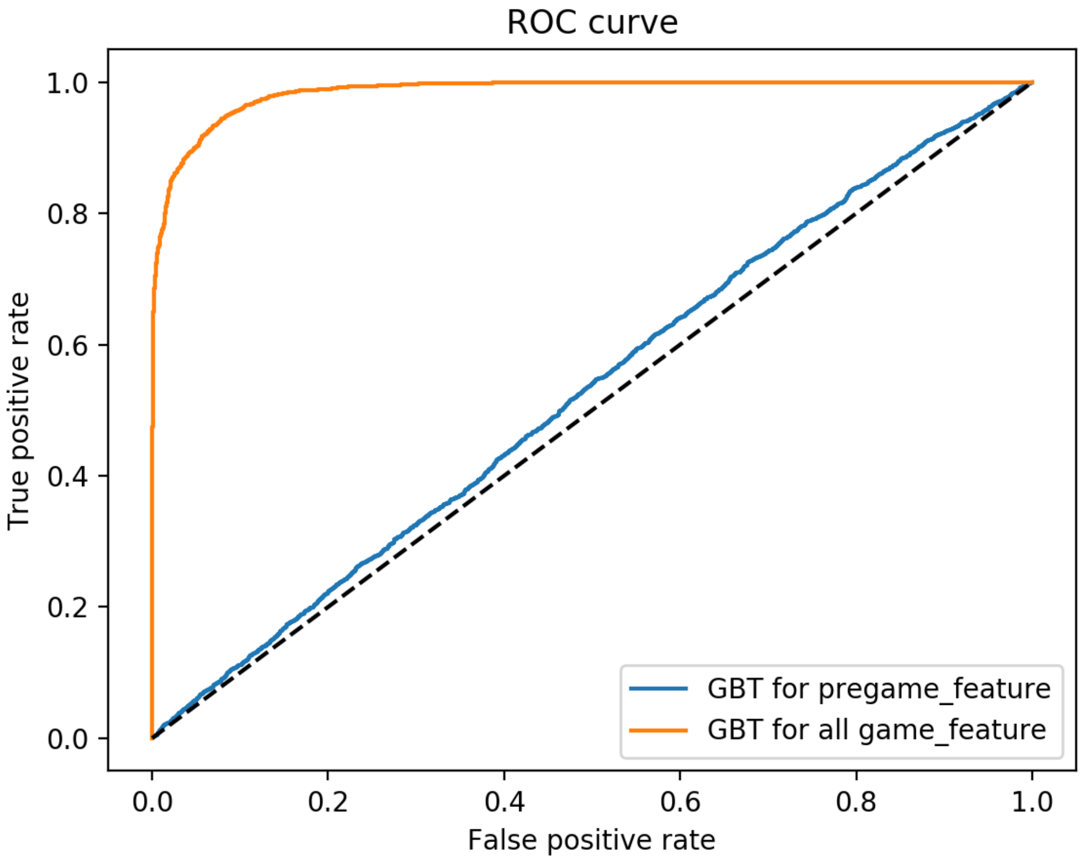
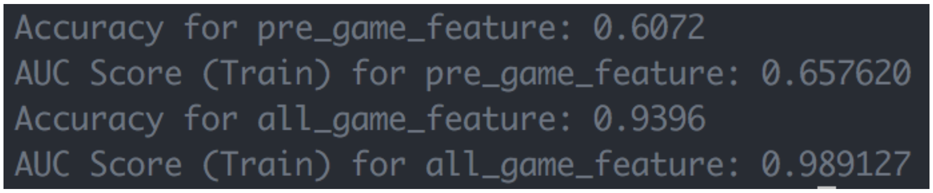


Figure 3: As we can see,when including all game features, the result is more accurate



```
Accuracy for pre_game_feature: 0.6072
AUC Score (Train) for pre_game_feature: 0.657620
Accuracy for all_game_feature: 0.9396
AUC Score (Train) for all_game_feature: 0.989127
```

Figure 4: As we can see,the accuracy score for all game feature is significantly higher than only including pregame feature