Big Data Problem with Machine Intelligence Stuff

Anders Roland Nielsen[1], Andreas Berre Eriksen[1], Kent Munthe Caspersen[1], Mathias Melgaard Andersen[1], Mikkel Alexander Madsen[1], and Sander Jespersen[1]

[1]*Department of Computer Science, Aalborg University*

Abstract

Here be dragons.

1 Introduction

Online multiplayer games have the possibiltiy of generating a lot of data, this only increases with the number of players and possible options for affecting the power of the player or the team. League of Legends, created by Riot games, was the most played online game in the beginning of 2015 [4], mustering 27 million people playing it daily, in the beginning of 2014 [2]. When playing the game, the players are divided into two teams of 5 players each, from a selection of more than 100 different champions, each player picks a single one. Each team then starts at their respective base, either as the red team in the upper right or as the blue team in the lower left corner. The goal is now to destroy the nexus of the opposing team and protect your own nexus. There are three paths connecting the two bases and both bases with a total of four towers, two belonging to each team. Each base will also send out small packs of minions to assist the champions in the destruction of the towers and eventually the nexus. As the game advances experience and money is earned depending on the performance of the players ans the team as a whole. The experience is used to improve the skills of the champion while money is spend purchasing items that will help the player or the team. The data Riot games gather from the game is extensive with many parameters [3], which gives it many possible interesting applications.

This paper will use the data to create a model capable of predicting the winning team based on the composition of champions, or even predicting which champion should be chosen to increase the probability of winning, given the opponents selection of champions. Since there are so many different champions and purchasable items, the variations of possible options the player can take is large, this means that we will require a great deal of data, more than can be stored optimally in any given database. We will require a cluster to handle this newly created big data problem and all its computations.

2 Related Work

Kevin C. and Daniel P have trained a classifier to predict the winning team of a Dota 2 match[1], by only considering the heroes chosen at the beginning of the game. Dota 2 is a PC game very similar to LoL, also with 5 players on each team. Each feature used represented the presence or absence of a particular hero on one of the teams. When training on 18,000 matches they could correctly predict the outcome of 69.8 % of the 5,669 matches in their test set. With 50,000 training samples, they almost reached 70 % correct predictions.

Only matches between high skilled players were considered.

We think the method can they used can be further improved, since they only tried to capture the strength of each hero individually. They did not consider that some combinations of heroes are stronger when fighting together compared to other combinations. We believe these synergies are important to consider, which are discussed further in section 3.

LoL includes 123 different champions in Patch 5.3.0.291. If we want to take account for synergies between that many champions, we may need to train on a huge amount of data, in order to have all combinations of champions represented an adequately number of times in the training samples. We will therefore approach this problem as a big data problem. Hadoop MapReduce and different variants have successfully been used for big data problems, by distributing the work load between many nodes in a cluster[5]. In cases where it is only necessary to read from data, the Apache Spark framework seems to be much faster. For iterative machine learning jobs, it outperforms MapReduce by a factor of 10.

3 Choosing features

Throughout this article, we will by $T = \{0, 1\}$ denote the two teams *blue* and *red*, respectively, and by $C = \{0, 1, \ddots, m-1\}$ denote the set of all chammpions in LoL. When referring to features, we will by t denote an element in T and by c denote an element in C.

Some champions in LoL may in general be better than other champions. To capture the strength of individual champions, we will need a feature that represent the presence or absence of a particular champion on each team. We define a set of features $X_1 = \{x_1(t, c)\}$, where

$$x_1(t,c) = \begin{cases} 1 & \textit{if } c \textit{ is present on team } t \\ 0 & \textit{otherwise} \end{cases}$$

Some champions in LoL deal very high damage (the damage dealer), but die very fast when taking damage. Other champions deal very little damage, but can be almost impossible to kill (the tank). Let alone, the two types of champions are not that strong, but when playing together they can pose a serious threat, since the tank can be used by the damage dealer as a shield, allowing him to stay alive for much longer and thus deal much more damage. To capture the synergy between two champions on the same team, we will for each team need a feature that represents the presence or absence of every 2-combination of champions on that team. We therefore define a set of features $X_2 = \{x_2(t, c_1, c_2\}$, where $c_1 < c_2$, and

$$x_2(t,c_1,c_2) = \begin{cases} 1 & \textit{if both } c_1 \textit{ and } c_2 \textit{ are present on team } t \\ 0 & \textit{otherwise} \end{cases}$$

We make the restriction $c_1 < c_2$ because we want to ignore permutations. This is because the two features $x_2(t, c_1, c_2)$ and $x_2(t, c_2, c_1)$ represent the same thing, namely that the two champions c_1 and c_2 both are present on team t.

Some champions in LoL may have an advantage when fighting against a particular enemy champion. For instance, a champion that is good at dodging ranged attacks is generally good against an enemy that only has ranged attacks. We say that the better suited champion *counters* the other. To capture that one champion may counter another champion, we will for each champion on team t need a feature that represents the presence or absence of every possible champion on the other team. For all $c_1, c_2 \in C$, we define a set of features $X_3(c_1, c_2)$, where

$$x_3(c_1, c_2) = \begin{cases} 1 & \textit{if } c_1 \textit{ is present on team blue and } c_2 \textit{ is present on team red} \\ 0 & \textit{otherwise} \end{cases}$$

In this case, we do not have the restriction c_1, c_2 and thus consider permutations instead of combinations. To understand why, consider that c_1 counters c_2. In this case, the feature $x_3(c_1, c_2) = 1$ is favorable to the blue team, while $x_3(c_2, c_1) = 1$ is favorable to the red team. Note that in some game modes, $x_3(c_1, c_2) = x_3(c_2, c_1) = 1$ is allowed. That is, the same champions may appear on both teams. For this type of feature, we may also have that $c_1 = c_2$. In that way we can capture if a champion can counter itself due to some asymmetries in the map layout.

4 Representation of features

From the definition of features in section 3, we can calculate that

$$|X_1| = |T| \cdot m$$

$$|X_2| = |T| \cdot \frac{m(m-1)}{2}$$

$$|X_3| = m^2$$

where $m = |C|$ is the total number of champions in LoL.

With 123 total champions in LoL, we have that for each match: $2 \cdot 5 = 10$ X_1 features are present out of 246 possible. $2 \cdot (5 \cdot 4)/2 = 20$ X_2 features are present out of 15006 possible. $5 \cdot 5 = 25$ X_3 features are present out of 15129 possible. By present, we mean the features which value is 1. It is clear that only a sparse number of features are present in each game.

For optimizing the machine learning methods used, we will for each match only represent the features that are present in that game. Before applying a machine learning method, each of the features introduced in section 3 are assigned a unique ID. For each match in the training data, the machine learning method is then given the IDs of the features present in that match. We will now show how each feature in X_1, X_2, and X_3 is assigned a unique id:

Every feature $x_1(t, c) \in X_1$ is given the ID:

$$c + t\frac{|X_1|}{|T|}$$

It is easy to see that this assign unique IDs to all features in X_1. Assigning unique IDs to features in X_2 is a bit more tricky since we are dealing with combinations and thus have the restriction $c_1 < c_2$. If we enumerate all features $x_2(t, c_1, c_2) \in X_2$ for a single t by (c_1, c_2), we get the following enumeration:

$$\begin{array}{r} (0,1),(0,2),(0,3),\ldots,(0,m-1) \\ (1,2),(1,3),\ldots,(1,m-1) \\ \vdots \\ (m-2,m-1) \end{array}$$

If using row and column indexes starting at 0, we see that row 0 contains $m-1$ elements, row 1 contains $m-2$ elements, and so on. That is, any row i contains $m-1-i$ elements. It is clear that the combination (c_1, c_2) lies in row c_1 as element c_2-1-c_1. Thus, any combination (c_1, c_2) is preceded by all elements in row 0 through c_1-1, as well as c_2-1-c_1 elements in its own row. The number of elements in row 0 through c_1-1 is an arithmetic series, and we can calculate the total number of elements using the formula $n(a_0+a_{n-1}/2$, where n is the number of rows and a_i is the number of elements in row i. Since we have $m-1$ elements in row 0 and $m-1-(c_1-1)$ elements in row c_1-1, we get that $c_1(m-1+m-1-(c_1-1))/2 = c_1(2m-1-c_1)/2$ elements are contained in the rows 0 through c_1-1. By adding the number of elements preceding (c_1, c_2) in its own row, we get that (c_1, c_2) is preceded by a total of $c_1(2m-1-c_1)/2+c_2-1-c_1 = c_1(2m-3-c_1)/2+c_2-1$ elements. We can now use the number of elements that precede (c_1, c_2) as the ID for the feature $x_3(t, c_1, c_2)$. However, we must remember to add the offset $2m$, to not clash with the $x_t(c)$ features, and we must also add the offset $t \cdot m(m-1)/2$ since we for each $t \in T$ have $m(m-1)/2$ of the $x_3(t, c_1, c_2)$ features. We finally get that:

Every feature $x_2(t, c_1, c_2) \in X_2$, is given the ID:

$$|X_1| + t \cdot \frac{|X_2|}{|T|} + \frac{c_1(2m-3-c_1)}{2} + c_2 - 1$$

Assigning an ID to the features in X_3 is simple since we are dealing with permutations. If we start enumerating all those features in the same way we did for the X_2 features, we will quickly see that the feature $x_3(c_1, c_2)$ appears as the c_1m+c_2'th element in the enumeration. Now, we only need to add the correct offset of $|X_1|+|X_2|+|X_3|$, to get that:

Every feature $x_3(c_1, c_2) \in X_4$ is given the ID:

$$|X_1| + |X_2| + c_1m + c_2$$

References

[1] Riot games. How does he saw me? a recommendation engine for picking heroes in dota 2. http://cs229.stanford.edu/proj2013/PerryConley-HowDoesHeSawMeARecommendationEngineForPickingHeroesInDota2.pdf, 2013. Accessed: 22nd of Februray, 2015. 1

[2] Riot games. Our games. http://www.forbes.com/sites/insertcoin/2014/01/27/riots-league-of-legends-reveals-astonishing-27-million-daily-players-67-million-monthly/ 2015. Accessed: 22nd of Februray, 2015. 1

[3] Riot games. Api documentation - full api reference. https://developer.riotgames.com/api/methods, 2015. Accessed: 22^{nd} of Februray, 2015. 1

[4] statista.com. Most played pc games on gaming platform raptr in january 2015, by share of playing time. http://www.statista.com/statistics/251222/most-played-pc-games/, 2015. Accessed: 22^{nd} of Februray, 2015. 1

[5] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1863103.1863113. 2