A Predictor of DOTA2
Machine Learning Project Report

WANG XIAOHUI, YANG GUANG, YANG YI, ZHUO YIWEI
1140339072
June 13, 2015

Abstract

In this work, we collected the practical match data of DOTA2, and based on the data, we introduced two research on predicting the match result based on Support Vector Machine (SVM) and Convolutional Neural Networks (CNN). In order to find which parameter determines the result of the match, we exploit SVM to analyze all 11 parameters and find their weights in deciding the match result. And we find out several key parameters that players should focus on since these parameters have an outstanding influence on winning the game. In order to give an advice for players whether should they give up the ongoing game based on historical match data, we exploit CNN to make a prediction based on all 11 data items. We reached a 98.33% prediction accuracy based on our trained CNN classifier model.

Keywords: SVM , CNN , Predictor, DOTA2

1 Background and Motivation

Each match of Dota 2 is independent and involves two teams, the Radiant and the Dire, both containing five players and each occupying a stronghold at either end of the map. Located in each stronghold is a building called the "Ancient"; to win, a team must destroy the enemy's Ancient. Each player controls a "Hero" character and focuses on leveling up, collecting gold, acquiring items and fighting against the other team to achieve victory.

There are nine game modes and around 100 "Heroes" in Dota 2. Heroes are strategically powerful player-controlled units with unique special abilities; though many heroes fill similar roles as others, each confers different benefits and limitations to a team. These Heroes start off very weak early in the game, but level up their abilities and statistics as they accumulate experience, up to a maximum level of twenty-five. The Heroes' methods of combat are heavily influenced by their primary property, which can be Strength, Agility, or Intelligence. Most game modes provide teams with some preparation time before the game begins so that they can balance their hero selections, as the composition of the team can significantly affect their performance throughout the match. Because Dota 2 is highly team-oriented, the players must coordinate and plan with each other in order to achieve victory. [2]

We start our work in order to give reliable advice for players to concentrate on certain indexes. Also, for many circumstances, although continuing the game and struggling for victory would be an active attitude, however we find that for most cases these strives rest in vain and this process would make great harm to players' sentiment. So we researched on the actual match data and wanted to derive a high-performance predictor to make a reliable recommendation for players on whether they should give up the remaining game.

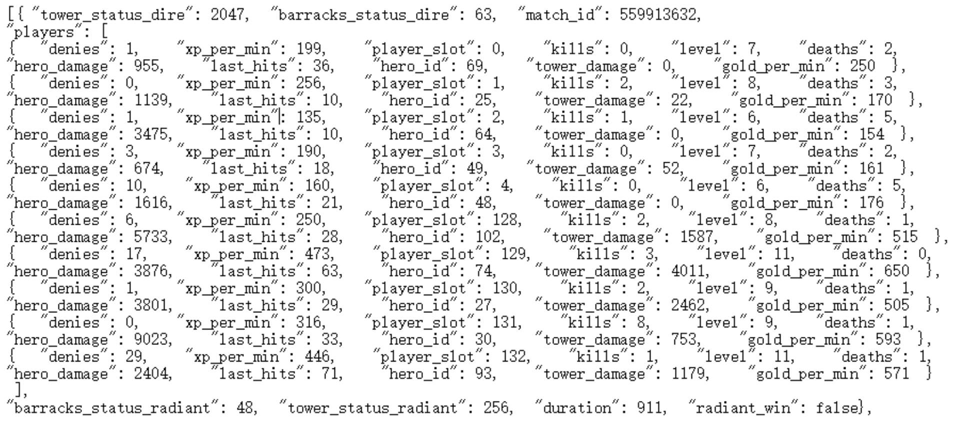
2 Retrieve Data

We collected 100K randomly chosen pieces of actual match data for our research, and choose 90K data as our training set and 10K as the testing set. Each piece of dataset is a log of one match, and listed ten players who attends the match with 11 items for each player describing certain match parameter listed in Table 1.

denies	the number of times a player denied a creep
xp_per_min	the player's total xp/min
player_slot	8-bit, top bit true is team dire and last 2 bits means player slot
kills	the number of kills the player got
level	the player's final level
deaths	the number of times the player died
hero_damage	the amount of damage the player dealt to heroes
last_hits	the number of times a player last-hit a creep
hero_id	the numeric ID of the hero that the player used
tower_damage	the amount of damage the player dealt to towers
gold_per_min	the player's total gold/min

Table 1: 11 Data items for each player in the dataset

Figure 1 shows an sample of dataset from one match. We consider only the ten players' data items. All 100K matches are stored in plain text. We build a 3-Dimensional matrix of $10 \times 11 \times 100000$, storing all data items in the sequence of Table 1 above as our input. And we build a vector of 100000 elements tagged the labels of the inputs in the same sequence.



```
[{ "tower_status_dire": 2047,  "barracks_status_dire": 63,  "match_id": 559913632,
"players": [
{   "denies": 1,    "xp_per_min": 199,    "player_slot": 0,    "kills": 0,    "level": 7,    "deaths": 2,
"hero_damage": 955,    "last_hits": 36,    "hero_id": 69,    "tower_damage": 0,    "gold_per_min": 250  },
{   "denies": 0,    "xp_per_min": 256,    "player_slot": 1,    "kills": 2,    "level": 8,    "deaths": 3,
"hero_damage": 1139,    "last_hits": 10,    "hero_id": 25,    "tower_damage": 22,    "gold_per_min": 170  },
{   "denies": 1,    "xp_per_min"|: 135,    "player_slot": 2,    "kills": 1,    "level": 6,    "deaths": 5,
"hero_damage": 3475,    "last_hits": 10,    "hero_id": 64,    "tower_damage": 0,    "gold_per_min": 154  },
{   "denies": 3,    "xp_per_min": 190,    "player_slot": 3,    "kills": 0,    "level": 7,    "deaths": 2,
"hero_damage": 674,    "last_hits": 18,    "hero_id": 49,    "tower_damage": 52,    "gold_per_min": 161  },
{   "denies": 10,    "xp_per_min": 160,    "player_slot": 4,    "kills": 0,    "level": 6,    "deaths": 5,
"hero_damage": 1616,    "last_hits": 21,    "hero_id": 48,    "tower_damage": 0,    "gold_per_min": 176  },
{   "denies": 6,    "xp_per_min": 250,    "player_slot": 128,    "kills": 2,    "level": 8,    "deaths": 1,
"hero_damage": 5733,    "last_hits": 28,    "hero_id": 102,    "tower_damage": 1587,    "gold_per_min": 515  },
{   "denies": 17,    "xp_per_min": 473,    "player_slot": 129,    "kills": 3,    "level": 11,    "deaths": 0,
"hero_damage": 3876,    "last_hits": 63,    "hero_id": 74,    "tower_damage": 4011,    "gold_per_min": 650  },
{   "denies": 1,    "xp_per_min": 300,    "player_slot": 130,    "kills": 2,    "level": 9,    "deaths": 1,
"hero_damage": 3801,    "last_hits": 29,    "hero_id": 27,    "tower_damage": 2462,    "gold_per_min": 505  },
{   "denies": 0,    "xp_per_min": 316,    "player_slot": 131,    "kills": 8,    "level": 9,    "deaths": 1,
"hero_damage": 9023,    "last_hits": 33,    "hero_id": 30,    "tower_damage": 753,    "gold_per_min": 593  },
{   "denies": 29,    "xp_per_min": 446,    "player_slot": 132,    "kills": 1,    "level": 11,    "deaths": 1,
"hero_damage": 2404,    "last_hits": 71,    "hero_id": 93,    "tower_damage": 1179,    "gold_per_min": 571  }
 ],
"barracks_status_radiant": 48,  "tower_status_radiant": 256,  "duration": 911,  "radiant_win": false},
```

Figure 1: An example of one data sample of a match

3 Exploit SVMs to verify the decisive factors

3.1 Necessity to verify individually

We are all the youth who have played or have heard some video games. As to the issue of finding out a good enough prediction of Dota2, our experiences and intuitions naturally tell us: There are many factors that may affect the result differently, such as the hero familiarity of its controller, the skill of this match, the mistake or accident during the match. From the data set, actually we found there are 11 items for each player in a match, which are identified as *denies*, *xp_per_min*, *player_slot*, *kills*, *level*, *deaths*, *hero_damage*, *last_hits*, *hero_id*, *tower_damage* and *gold_per_min* respectively. They are all the information we can make use of. We are asked by the proposer to use the items of "hero-id" and "level" only to predict the winner originally. But our group doubt whether or not the hero composition and the level of players truly are the most decisive factors. If the answer is negative, the accuracy of some machine learning method can be poor and meaningless. So one of our key points is that we need to figure out how much the single item associate with the accuracy of prediction result individually.

3.2 Apply SVMs efficiently

Our data set, with the size of 100,000 matches, can be regarded as a 10*11*10,000 three-dimensional matrix. It's a mid-size data! Fortunately, through the course of Machine Learning, we know: 1) SVMs, which are among the best (and many believe is indeed the best) "off-the-shelf" supervised learning algorithm; 2) Kernel trick, which give a way to apply SVMs efficiently in very high dimensional feature spaces. So we choose to use the algorithm of kernel trick which should be suitable to solve this problem. Since the story of SVMs has been told in detail in our class, here we just simply summarize our idea as follows: Firstly, we divide 100,000 pieces of match data into two parts, 90,000 pieces for training, 10,000 pieces for testing. Secondly, execute like following pseudo-code.

```
1. iterator i = 1 : 11
2. SvmTrainData(i, 1:90000)
3. SvmTestData(i,90001:100000)
```

3.3 Reveal the truly most decisive factors

We make a bar graph to conclude the accuracy of prediction results using 11 kinds of items individually as follows.

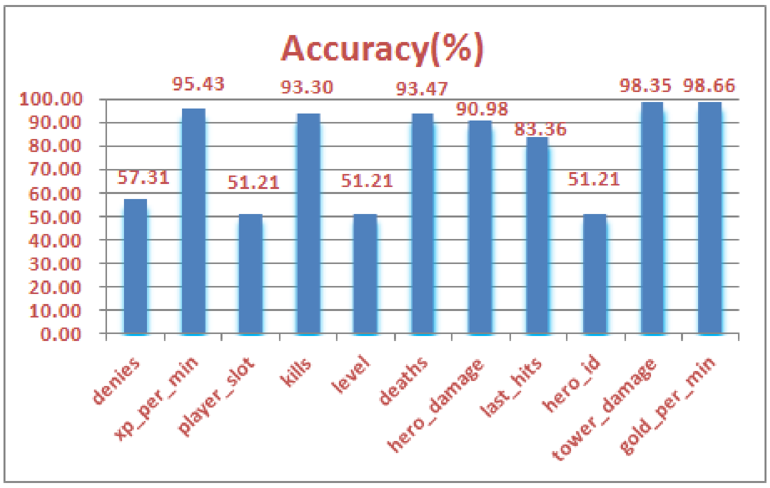


Figure 2: CNN Reduces Parameters

As we can see, two most decisive factors are "gold_per_min" and "tower_damage" rather than "hero_damage" and "level".We further infer the accuracy of prediction results may be just so so if we only use the items of "hero-id" and "level".What we discover is coincident with the baseline, given by the proposer, which gets 61% of accuracy. It seems to draw a conclusion the more money you earn and the more damage you bring to your enemy's tower, the higher probability you will win.

4 Exploiting CNN to make a prediction on whether should we give up

4.1 Convolutional Neural Networks

Convolutional Neural Networks is a type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field. [3] A CNN is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. [4] For each layer of the network, The benefits of CNN is that it can dramatically reduce the amount of parameters that should be trained. As the figure shows below, if we obtain 1000×1000 pixels, with 1M hidden neurons, and for each neuron it should connect each pixel of the image, then there exists 10^{12} connections, which needs to train 10^{12} parameters. However, the spatial connection of the image is localized, so one neuron doesn't have to comprehend the total details of the image, and only need to read the local part of the image. And in upper layer we can synthesis different parts so that we get the global information. By this we can dramatically reduce the amount of connections. Suppose one nueron need only to connect with 10×10 pixels, the total amount of connection is 10^8.

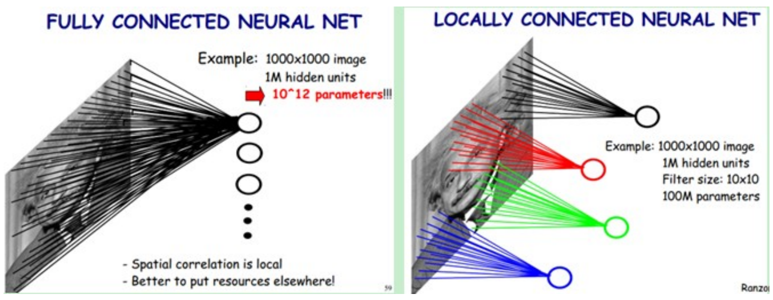


Figure 3: CNN Reduces Parameters

In fact, we can apply same parameters for every neuron, so the parameters we need to train draws down to 10×10.

4.2 Network Design

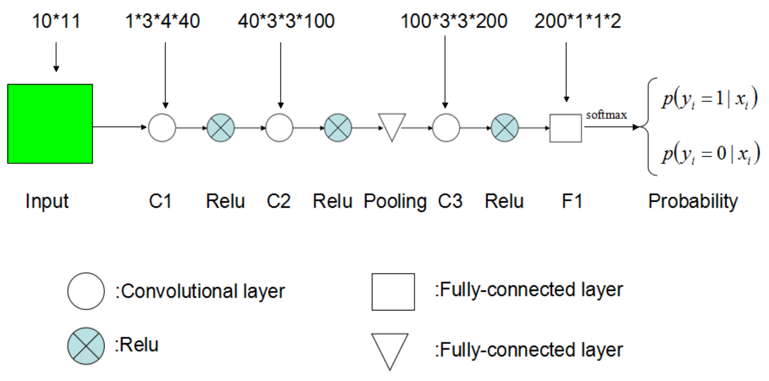


Figure 4: the overall architecture of our CNN

The architecture of our network is summarized in Figure 4. It contains four learned layers ąlthere convolutional and one fully-connected.Each of these learned layer owns a filter bank of size s1*s2*s3*s4,where s1 is the number of this layer′s input feature maps,s4 is the number of it's output feature maps,s2*s3 is the filter size. We adopt the ReLU. [5] as the activation function, which can makes convergence much faster while still presents good quality. [6] and the easy access to an abundance of data for training larger models. Our network learn a direct map from the game input feature map to it's label.The training is implemented using the MatConvNet package. [7] This toolbox borrows its convolution algorithms from Caffe. It may be somewhat slower than Caffe, but it is flexible to design a new CNN architecture.

4.3 Train the Network

Before we set the structured data discussed in section 2, we need to initialize the data by using each match data items subtract the average value of all matches.

We need to train a classifier with an output label with "insist on" or "give up". The training set consists of m labeled samples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$, and the input feature vector $x^{(i)} \in \mathbb{R}^{\mathbb{N}+\mathbb{1}}$ with $x_0 = 1$ corresponding to the interception item and $\mathbb{N}$ here refer to 11. We exploit softmax regression for this binary-classifier. Because label $y^{(i)} \in \{0, 1\}$, so in this condition Softmax degenerated into logistic regression. The hypothesis function listed as below:

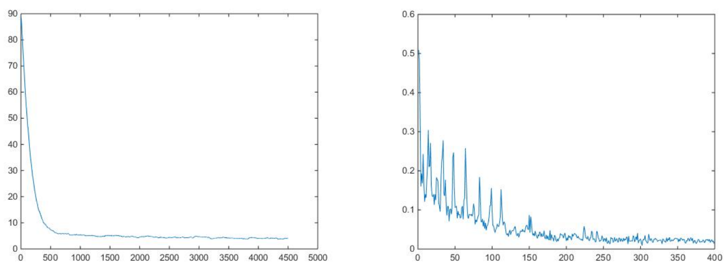
$$h_\theta(x) = \frac{1}{1 + exp(-\theta^T x)}$$

We will train the parameter θ in order to minimize the cost function below, and we adopted gradient descent to minimize $J(\theta)$.

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

4.4 Prediction Evaluation

We performed the experiment using Matlab, and reached an accuracy of 98.33% prediction accuracy rate. Figure 4(a) shows the change of loss function $J(\theta)$ during the training process. Figure 4(b) shows the error prediction rate of the testing set. We perform the testing process once after the network updates its parameters by randomly choose 500 matches from the testing set.



(a) loss function (b) test error rate

Figure 5: The results of CNN

5 Evaluation and Conclusion

We draw the conclusion that one should focus on earning money and bring damage to towers of the foes so that they would be more possible to win a match. We exploits SVM to implement this feature. We also trained a reliable model that can provide reliable recommendation of whether you should give up or insist on based on the retrieved data of the ongoing match. We exploits CNN to implement this feature.

References

1. P.Bahl, V.N. Padmanabhan, "Radar: An in-building RF-Based User Location and Tracking System." *Proc. of IEEE INFOCOM*, volumn 2, pp. 775-784, 2000.
2. "*www.en.wikipedia.org/wiki/Dota_2*" *WikiPedia*
3. "*www.en.wikipedia.org/wiki/Convolutional_neural_network*" *WikiPedi-a*
4. "*www.ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/*" *Stanford*
5. Nair.V,Hinton,G.E, "Rectified linear units improve restricted Boltzmann machines." *ICML*,pp. 807-814, 2010.
6. Krizhevsky.A, Sutskever.I, Hinton, "ImageNet classification with deep convolutional neural networks." *NIPS*,pp. 1097-1105, 2012.
7. "*https : //github.com/vlfeat/matconvnet/*" *Github*