Game Result Prediction Through Different Models in Machine Learning

Leo Liu

1. INTRODUCTION

League of Legends is a popular competitive online game, where a total of 10 players are split into blue and red teams. Players will choose unique characters to fight against the opponent's team. The goal of the game is to destroy the other team's Nexus.

In this experiment, we will be going through several methods of machine learning models to try to predict the result of the game and compare their performance.

2. EXPERIMENT

The data set we are using is found on Kaggle. This dataset contains the stats for the game at the 10 minutes mark. Players have similar skill levels due to the game's matchmaking system. There are 19 features indicating the stats per team (38 in total) collected in-game. This includes kills, deaths, gold, experience, etc. The target has a value of 1 or 0, indicating whether the blue team has won at the end of the game. There are around 10,000 samples in total.

In this experiment, we will apply different machine learning models to this data set and compare their performance with each other, and also with the baseline model.

3. METHOD

We will test out 3 machine learning models on this dataset: Linear regression, Logistic regression, and 1-layer Neural Network. For all 3 models, we will be using TensorFlow's neural network framework for ease of implementation.

Before any game has been played, theoretically speaking either side should have an equal chance of winning. The point of the task is to predict as many games correctly as possible, and so, we set the measure of success to the accuracy of the model's prediction comparing to the target.

The dataset is split 60-20-20 into training, validation, and testing set. All the features in the datasets are normalized by the training data. This assumes validation and testing set will have similar distribution when compared to the training data.

During training, we will use the grid search to optimize the hyperparameters using the validation accuracy, then use the best model against the test dataset.

3.0 BASELINE

The baseline model is to simply count which side has more kills, and tie-breaking by counting assist. It has an accuracy of 0.680.

3.1 LINEAR REGRESSION

For linear regression, we set the model to have 1 output node with no activation function and no hidden layers. This way the model only has 1 set of weights and works exactly like a linear regression model.

```
i = Input(shape=(D,))
x = Dense(1, activation="linear")(i)
```

This is a classic linear regression model with prediction using:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{y} = I(\boldsymbol{z} >= 0.5)$$

The learning goal is to minimize the training loss:

$$min_w \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$$

The gradient is:

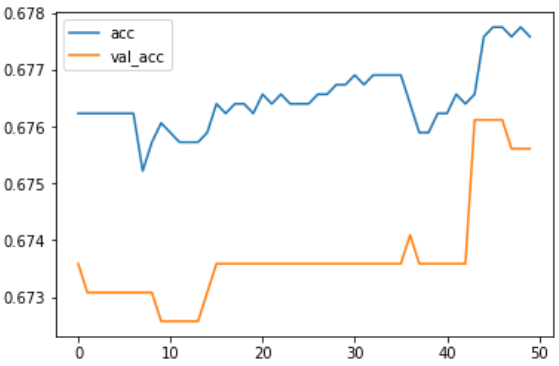
$$\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})$$

We train by moving in the opposite direction of the gradient.

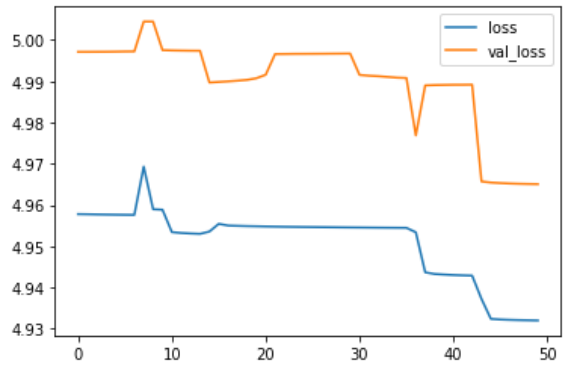
Model output:

Best alpha: 0.025 Val accuracy: 0.676 Test accuracy: 0.676

Accuracy on best performing model:



Loss on best performing model:



We can tell that It doesn't do as well even comparing to the baseline model, it has a high loss score around 5 and accuracy around 0.68.

3.2 LOGISTIC REGRESSION

For logistic regression, we use the same structure as linear regression but add the Sigmoid activation function to the output layer.

```
i = Input(shape=(D,))
x = Dense(1, activation="sigmoid")(i)
```

This model adds a sigmoid activation function on top of the linear regression model to better map out the goodness of the model to its prediction.

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{y} = \frac{1}{1 + e^{-\boldsymbol{z}}}$$

The model follows Bernoulli distribution, so the learning goal is to minimize the training loss:

$$\frac{1}{M}\sum_{m=1}^{M} -(t^m log(y^m) + (1 - t^m) log(1 - y^m))$$

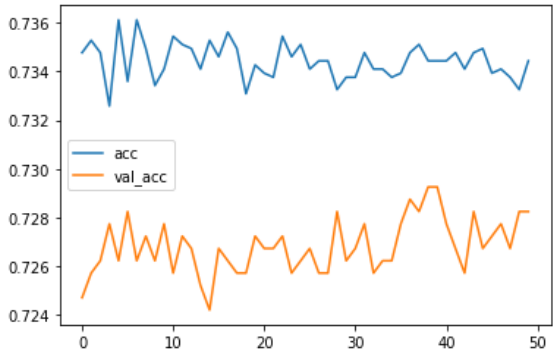
The gradient is very similar to linear regression, with activation function added to calculate y:

$$\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{t})$$

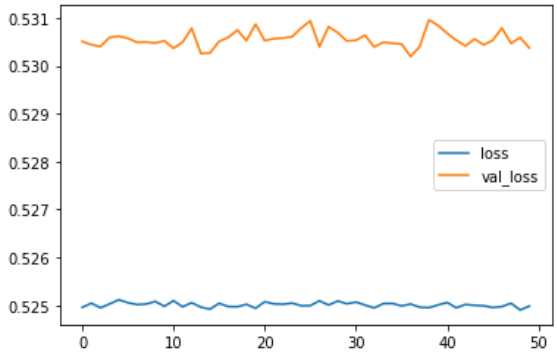
Model output:

Best alpha: 0.01 Val accuracy: 0.728 Test accuracy: 0.730

Accuracy on best performing model:



Loss on best performing model:



This model performs about 7% better than the baseline model for the test dataset. From the loss & accuracy chart, we can also tell it is relatively stable and converges very fast.

3.3 NEURAL NETWORK

For a 1-layer neural network, we add a hidden layer in the model and then use the grid search to get the number of nodes. The search range is [16, 32, 64, 128].

```
def getNNmodel(node):
  i = Input(shape=(D,))
  x = Dense(node, activation="linear")(i)
  x = Dense(1, activation="sigmoid")(x)
  return Model(i,x)
```

This model adds an extra layer of neurons on top of the logistic regression model. Each neuron takes the input layer, go through a set of weight, and propagate the result towards the output layer without activation function. The output node takes them as its features and outputs its prediction.

Each layer forward propagates by:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

With the output node using the Sigmoid activation function:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{y} = \frac{1}{1 + e^{-\boldsymbol{z}}}$$

The model also follows Bernoulli distribution, the loss function is:

$$\frac{1}{M}\sum_{m=1}^{M} -(t^m log(y^m) + (1 - t^m) log(1 - y^m))$$

The gradient is calculated per layer, assuming DJ/Dy is known:

Propagate back gradient:

$$\frac{\partial J}{\partial \boldsymbol{x}} = \boldsymbol{W}^T * \frac{\partial J}{\partial \boldsymbol{y}} \odot (\frac{\partial y_i}{\partial z_i})_{Ny}$$

Weight gradient:

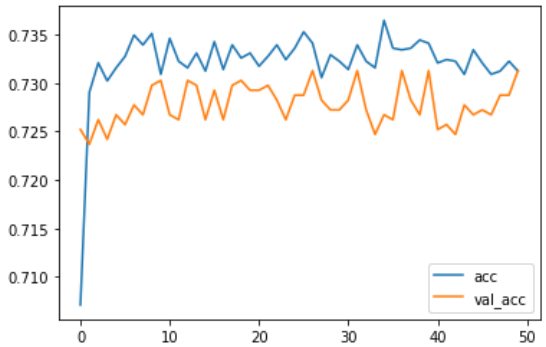
$$\frac{\partial J}{\partial \boldsymbol{W}} = (\frac{\partial J}{\partial \boldsymbol{y}} \odot (\frac{\partial y_i}{\partial z_i})_{Ny})\boldsymbol{X}^T$$

Dy_i/Dz_i is the change of y in relation to z after going through activation function.

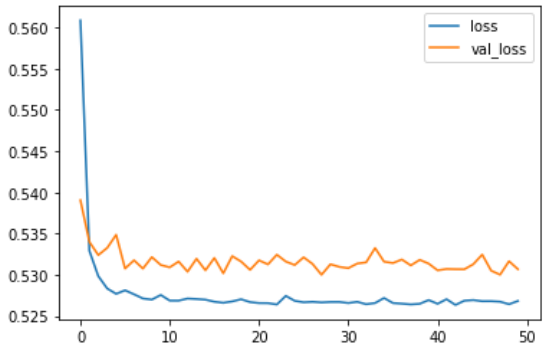
Model output:

Best alpha: 0.025 Best node count: 128 Val accuracy: 0.733 Test accuracy: 0.738

Accuracy on best performing model:



Loss on best performing model:



This model performs similarly to logistic regression. We can see that the model is very stable from the plot, and this time we see a slight upward trend with validation accuracy during the iterations. This model performs about 8.5% better than the baseline model for the test dataset.

4. CONCLUSION & FUTURE WORK

From the experiments on the three models shown above, we can conclude that with appropriate models like logistic regression and neural networks, we can make a better prediction comparing to just counting the kill and assists on both teams.

The 1-layer neural network being the most powerful model, having 73.8% prediction accuracy. That is very good considering each game usually lasts much longer than 10min, with a high chance of either team messing up and giving away the advantage.

The linear regression performed the worst, it has a high loss score and it's highly non-stable, as the model is having a hard time to converge.

For future experiments, we may want to consider the team composition for both teams, as some strong character combos can be an advantage on their own. The task is likely to have patterns within the features, it would be very interesting to see how models with pattern detection like CNN will perform on similar tasks.

5. ACHNOWLEDGMENT

The original data is collected by Kaggle user "michel's fanboi", published in https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

The dataset in this experiment has the column gameId manually taken out.