Duc Tran Greenwell

Anders Rathjens

Sam Warutere

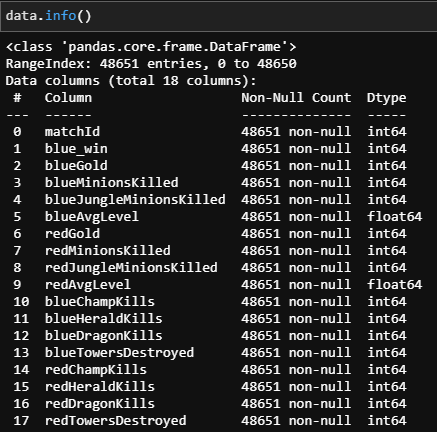
Cohort 11

OPS808

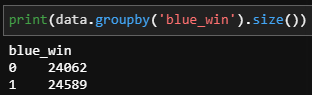
9/14/2021

**Machine Learning Group Report**

1. **Data Background & Description**

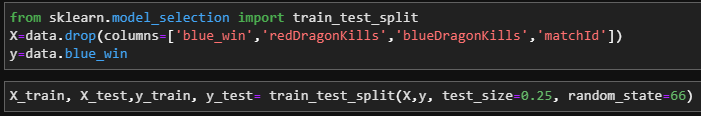
Our goal was to be able to predict the winner of a League of Legends ranked game with data from the 15-minute mark. League of Legends is a 5 vs 5 online team video game. The teams are separated into Blue and Red, and the winner is the team that can destroy the other team's base first. Games can last anywhere from 15 minutes to over an hour, leaving lots of room for comebacks in the game. Our overall goal for this project was to be able to create a model that would predict, with over 75% accuracy, who would win the game based on some quantifiable statistics at the 15-minute mark of the game. A previous dataset with the same goal in mind, that had statistics from the 10-minute mark of the game, was said to have peaked at about 60% accuracy, so we thought we would be able to get over 75% with data from the 15-minute mark. The data itself was rather straightforward. It was taken from Diamond Ranked games (top 2% of all players) last year and it has 18 columns and 48,652 rows. As shown here, before cleaning the data and running summary statistics, we can see that all the columns are even. The data is split up showing both the blue side and the red side. The columns referencing Gold, minions and champion kills all give experience and boost your team's economy for the duration of the 

***Figure 1***

game. Things like Towers Destroyed, Herald Kills and Dragons, or objectives that provide you team instant bonuses such as lots of gold, map control and permanent stat boosts. We ended up removing the “matchId”, “blueDragonKills”, and “redDragonKills” from the dataset since they turned out to be useless. MatchId is only useful for pulling up individual games, and we were not interested in that. On the other hand, both Red and Blue dragon kills had nothing but nulls for their entire columns. The “blue\_win” column is our Y variable since that determines the winner of the match, a 0 being red team won the game, and a 1 meaning blue team won the game. As shown in the figure below, the data is not very uneven, with the blue team slightly winning more games. We proceeded to split the 

***Figure 2***

data 75% to 25% for our models. We then scaled the data to make sure our models were running properly.

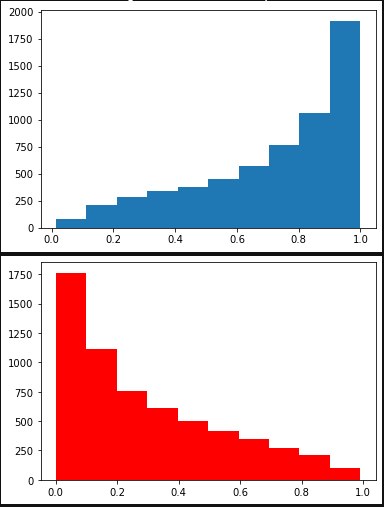
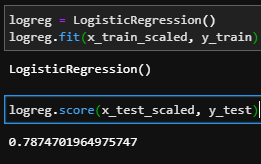
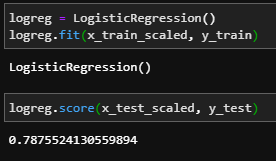


***Figure 3***

1. **Types of Models Used**

**1. Logistic Regression**

The first model we ran was a logistic regression. It scored over the 75% that we were hoping for which was great as well. It had a score of 78.74% (figure1) which was a great baseline to compare the rest of our models to. Then to see if we could improve the model at all, I dropped the 2 lowest coefficients from the model, both of which were below .05. These columns were Blue Jungle Minions and Blue Towers. This model scored barely better than the previous one that we had run, so in the end we felt pretty good with the original model and its score(figure2). We stuck with a threshold of .5 because there was no significant improvement that could be made by moving it either higher or lower. As shown below in the graphs showcasing the prediction probabilities.

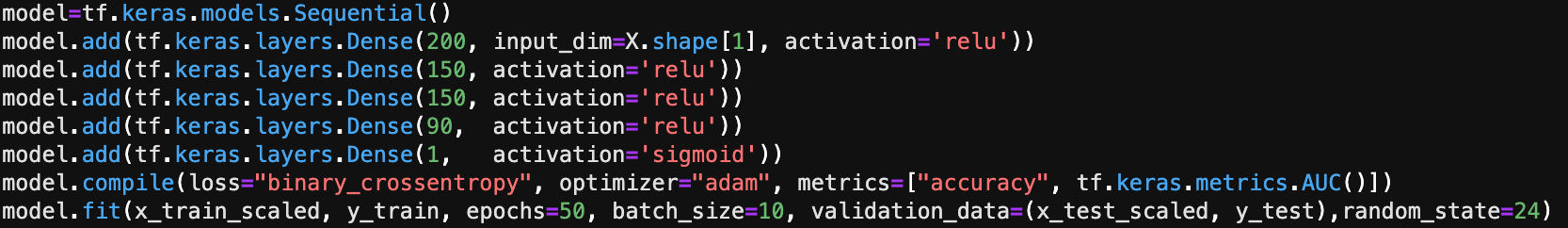


***Figure 4***  ***Figure 5***

***Figure 6***

**2. Deep Neural Network (DNN)**

First, we started the model with 4 HL and one output layer using the relu activation function. Since it was a classification case with 0 - 1 output, we used activation Sigmoid for the output layer. To measure the loss, we used binary cross entropy for the binary classification problem. For optimizers, we used Adam and accuracy for the metrics. We compared the models based on their training and validation accuracies. Before fitting training data into the model, we scaled it. We tried 50 epochs and batch size 10 this very first time.

****

***Figure 7***

After running the model, we got a very low training loss at 0.29 but in contrast, validation loss was high at 0.91 while high training accuracy 0.88 but validation accuracy was only at 0.76

****

***Figure 8***

Because the model had higher test loss than training loss and lower validation accuracy than training accuracy, we took additional steps to hopefully mitigate the overfitting problem with above 75% accuracy.

First step, we tried to improve the model by removing one hidden layer from the original model and running the model again. We found there was not much change in training and validation accuracy, but the validation loss got smaller, from 91% down to 75%. However, there was still evidence of an overfitting problem since test loss was still too high compared to training loss.

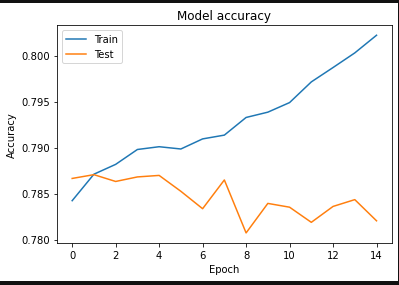
****

***Figure 9***

Taking a close look into the outputs, we noticed that after the first 15 epochs, the training loss and training accuracy started to jump back up. To combat this, we reduced the epochs from 50 to 15 while keeping everything else unchanged. This time we had better results. Our model had very close training and test losses plus, training accuracy scored at 80%, and the validation accuracy scored at 78%.

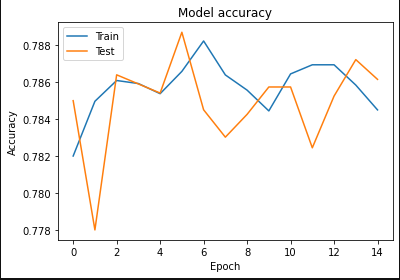
****

***Figure 10***

However, when we plotted the accuracies on the graphs, we noticed that the training accuracy and validation accuracy didn’t behave the same. They seemed to be diverging.

***Figure 11***

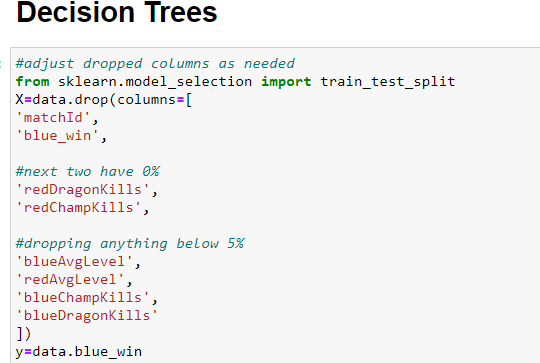
Once again, we fixed it by applying a learning rate of 0.01. The adjustment gave us a desirable outcome which was the same losses and accuracy as before applying the learning rate, yet the training and validation accuracy behaved almost the same trend.

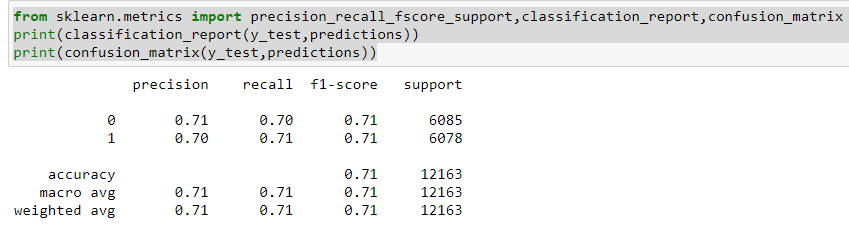


With 78.6% accuracy, we were pleased with this model since it scored higher than our goal initially set at 75%.

***Figure 12***

**3. Decision Trees**

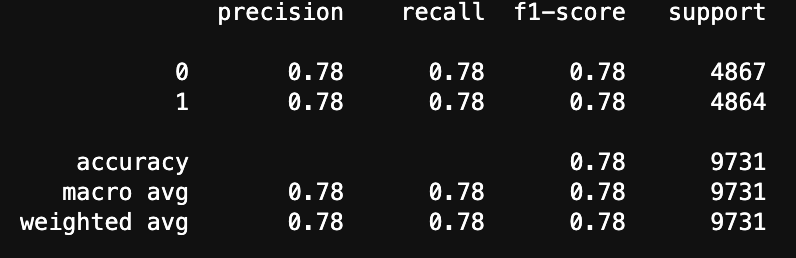
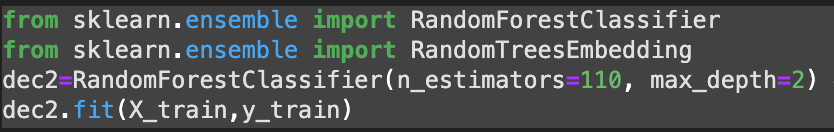
We also used **decision trees** and **random forests besides Logistic Regression and DNN**, because they were gradient boosted. Random forest is especially powerful because it combines individual trees to generate a final output. We see an accuracy and precision of 70%. So, we decided to run it again, but this time we would remove more columns. In this case we removed the 4 columns that had coefficients below .05 from the previous model. The model then improved to 71% accuracy. So overall not much of a difference from the first iteration. We did not have high hopes for the Decision Tree model, but it ended up with a confusion matrix of 4146 true positives, 1939 False positives, 1746 false negatives and 4332 True negatives. A good ***Figure 13***  baseline to compare Random Forest to.



***Figure 14***

**4. Random forest**s:

We use the data in its original form as the method should train well enough to overcome data outliers. For this one we also were getting around 70% accuracy to start, which was not ideal and surprised us. So, we decided to mess with the estimators and saw an increase in all the metrics. We ended up with a model with n\_estimators =110 which gave us an accuracy, precision and recall of 78% just like our Logistic regression model and Deep Neural Network. This was great since it also gave us a confusion matrix of 4275 true positives, 1810 false positives, 1767 false negatives, and 4311 true negatives. This a clear improvement by over 100 in the True positives category, while only losing 21 when it came to the true negatives.

***Figure 15***

***Figure 16***

**III. Findings**

In conclusion, all our models except for the decision tree beat our goal of 75% accuracy since we had models with almost 79% accuracy. Our goal of beating the previous dataset using data from earlier in games was easily done since we clearly surpassed 60% accuracy in all our models, which made us very pleased. All the models also were a testament to the famous saying, “it is not over until the fat lady sings” since in our models we had games that predicted well over 90% probability for the blue or red team to win, and they ended up losing. But this also leaves the door open for many other things we can use this model for. A few possibilities are running this model for games taken from a lower skill bracket where the games are supposedly much more volatile and unstructured, or taking the model and running it vs the pro leagues and seeing how it performs there. Overall, the project was a success and we hit every goal we set for ourselves, and even learned a few new coding tricks along the way.

**Useful Links**

YouTube link: <https://youtu.be/Pj4_XkPrw1I>

Link to Kaggle Dataset for this project: <https://www.kaggle.com/benfattori/league-of-legends-diamond-games-first-15-minutes>

Link to Kaggle 10-minute dataset: <https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min>