Using Statistical Learning to Model and Predict Player Performance

Victor Adegbite
Alexander Bedrossian
ECE-UY 4563: Machine Learning
Professor Linda Sellie

May 8, 2019

Contents

1 Abstract

For the project, a dataset for the game of quidditch was provided. The purpose of this project is to predict whether a player makes it to Professional League Quidditch upon graduation from Hogwarts or not based on the features provided in the data set.

Some of the features provided included age, weight, game duration, etc. Half of the dataset for the features and target values were used as training data while the other half was used as test data. A common theme in the dataset were features that contained indeterminate data. To remedy this, this data was taken out. In addition, some of the features could've combined to represent one feature.

In terms of the models used to train the data, three were used – the logistic regression, the Support Vector Machine (SVM), and Neural Networks. For the logistic regression and the SVM models, a penalty matrix was used which contains values of form 10^i, where i ranges from -5 to 5.

Out of the three models, the most effective was the Neural Networks since it yielded the highest accuracy at 92%. A possible reason behind this can possibly be the mathematical nature of the model – a sigmoid function was used to train the data unlike the other two models where some n-degree polynomial function were used to train the data. Another factor could be the optimization methods used on the neural networks which allowed more accurate predictions of the target values.

2 Logistic Regression

The first model used to train the data was the logistic regression. Three types of logistic regressions used were the lasso regularization, the ridge regularization, and the polynomial feature transformation.

Lasso regularization of the logistic regression, as seen below in Figure (1), was first used. The training data for the features and target values were fitted to a logistic regression with a lasso regularization-type penalty. The fitted feature training values were used to predict the target values.

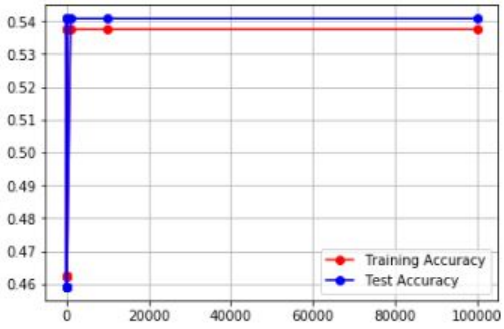


Figure 1: Lasso Regularization

The values obtained for the accuracy for both the testing and training datasets for this regression ranged from 46% minimum to 54% maximum, where the outliers were the values around 46%.

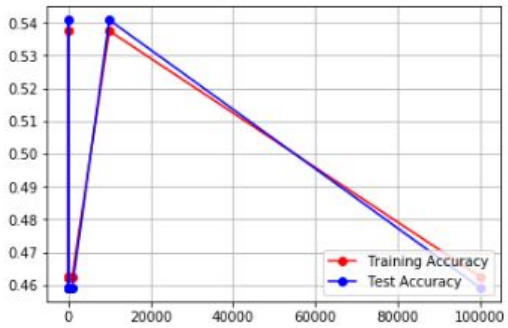


Figure 2: Ridge Regularization

Very similarly to the lasso regularization, a ridge regularization of the logistic regression, as seen in Figure (2), was used to train the data. The training data for the features and target values were fitted to a logistic regression with a ridge regularization-type penalty. In addition, the fitted feature training values were used to predict the target values.

The values obtained for the accuracy for both the testing and training datasets for this regression ranged from 46% minimum to 54% maximum. The difference, however, from the lasso regularization was that there

generally a greater distribution of values around 46% then there were 54%, so we could rule the 54% as outliers to the accuracy dataset.

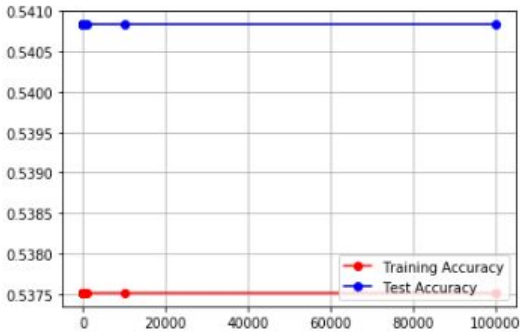


Figure 3: Polynomial Feature Transformation (2^{nd} Degree)

The last model, Figure (3), created used a polynomial feature transformation that mapped the values of the feature matrix to that of a second-degree polynomial. In this process, both the lasso regulated model and the ridge regulated model were used to train the data set under very much the same process aforementioned.

The values obtained for the accuracy for both the testing and training datasets for this regression had a mean of 54% for both the lasso and ridge regularizations. Compared to the non-feature transformed regressions, the feature-transformed counterparts were both better due to higher consistency.

3 Support Vector Machines (SVM)

The next model used was the support vector machines; for this model, three different kernels were used to train the data – the linear, radial basis function, and the polynomial kernel.

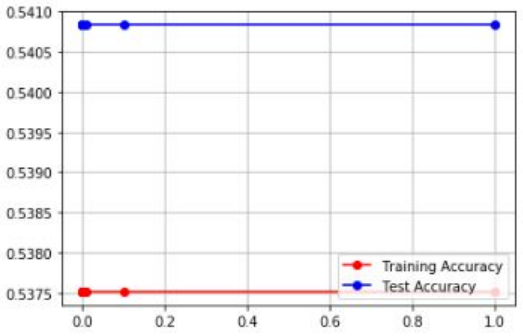


Figure 4: Linear Kernel

The linear kernel of the SVM, as seen above in Figure (4), was first used. The training data for the

features and target values were fitted to the linear SVM. The fitted feature training values were used to predict the target values.

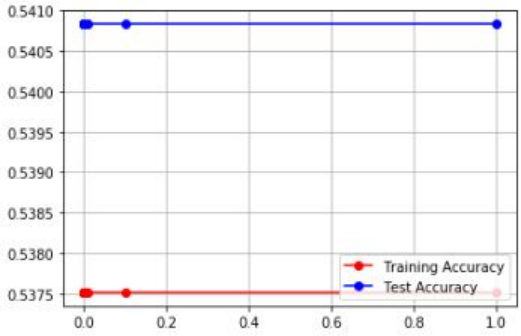


Figure 5: Radial Basis Function (RBF) Kernel

The values obtained for the accuracy for both the testing and training datasets had a mean of 54% with no outliers to the data set.

The next model used to train the data was the RBF kernel of the SVM, as seen above in Figure (5). Like its linear counterpart, the training data for the features and target values were fitted to the RBF SVM, and the fitted feature training values were used to predict the target values.

The values obtained for the accuracy for both the testing and training datasets had a mean of 54% – basically almost identical results to the linear kernel.

The polynomial kernel of the SVM, as seen below in Figure (6), was created. The training data for the features and target values were fitted to the linear SVM. The fitted feature training values were used to predict the target values.

Much like the other two SVM kernels, values obtained for the accuracy for both the testing and training datasets had a mean of 54%.

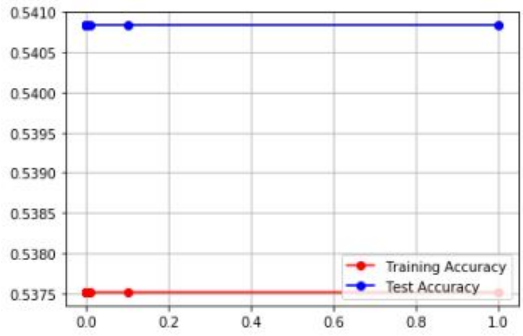
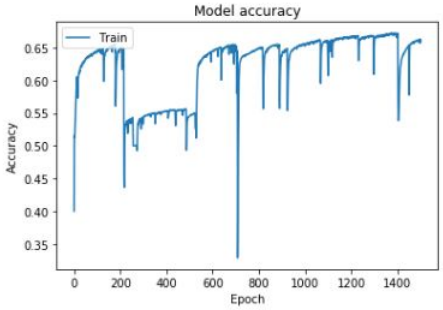
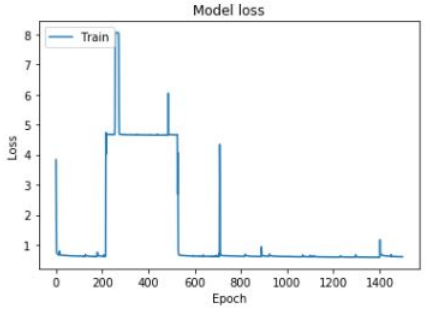


Figure 6: Polynomial Kernel

4 Neural Networks






```
WARNING:tensorflow:From C:\Users\alexb\Anaconda2\lib\site-packages\tensorflow\python
ate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in
Instructions for updating:
Colocations handled automatically by placer.
```

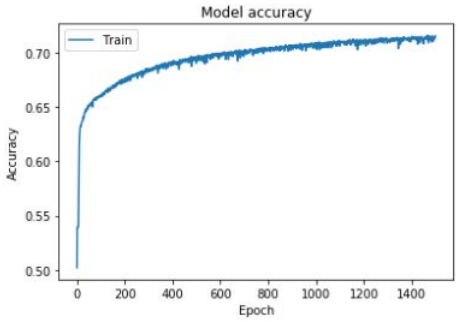
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 132)	17556
dense_2 (Dense)	(None, 42)	5586
dense_3 (Dense)	(None, 42)	1806
dense_4 (Dense)	(None, 42)	1806
dense_5 (Dense)	(None, 2)	86

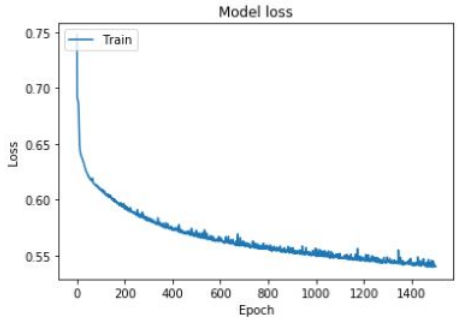
Figure 7:

Our input layer was a size of 132; as that was the total size of the one-hot encoded player stats. We experimented with various hidden layers, activation functions, optimizations, and loss functions. Relu was the first activation function that we mapped and it did produced very interesting results as it repeatedly overfit the training data.

The structure of this NN used 3 hidden layers with size 42; this was decided as the original feature count is 42, where one-hot encoding expanded the inputs to 132. The learning curve seems to mimic a drawing of paint drying.

We experimented with Sigmoid next and our results were slightly better but still not great. There is definitely much more stability and precision while using the sigmoid activation function and that can be seen in difference between the two loss graphs.





Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 132)	17556
dense_2 (Dense)	(None, 42)	5586
dense_3 (Dense)	(None, 42)	1806
dense_4 (Dense)	(None, 42)	1806
dense_5 (Dense)	(None, 2)	86

Figure 8:

This sigmoid NN structure is the same as the Relu NN but we yield 70% accuracy instead of 60% accuracy just by changing the activation function to a more fine grain activation function.

We experimented with adding more hidden layers to see how the AI would react. It appeared to have very similar learning curve with a slight delay (shifted to the right).

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 132)	17556
dense_2 (Dense)	(None, 42)	5586
dense_3 (Dense)	(None, 42)	1806
dense_4 (Dense)	(None, 42)	1806
dense_5 (Dense)	(None, 42)	1806
dense_6 (Dense)	(None, 42)	1806
dense_7 (Dense)	(None, 42)	1806
dense_8 (Dense)	(None, 42)	1806
dense_9 (Dense)	(None, 42)	1806
dense_10 (Dense)	(None, 42)	1806
dense_11 (Dense)	(None, 42)	1806
dense_12 (Dense)	(None, 2)	86

Figure 9:

This 10 layer hidden model gave us a very similar output as the 3 hidden layer models. We will experiment with smaller layer count and vary other parameters now.

We tried mixing activation functions to see the impact. We found Relu to have extreme activation function values and thought that it might help make our input data slightly more influential on the neural network. We wanted the fine grain accuracy of the sigmoid so we produced a NN with Relu Input and 3 Sigmoid layers thereafter (including the output layer). The results were, consequently, much better.

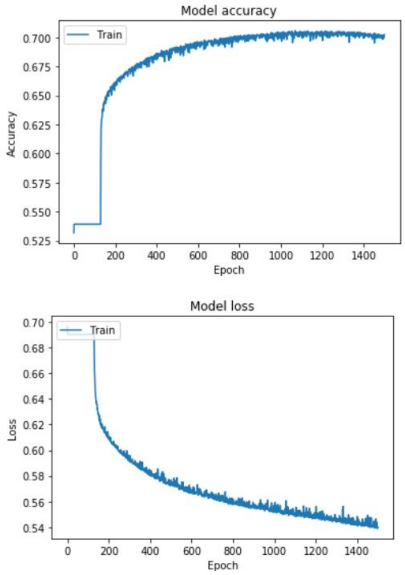


Figure 10:

This 10 layer hidden model gave us a very similar output as the 3 hidden layer models. We will experiment with smaller layer count and vary other parameters now.

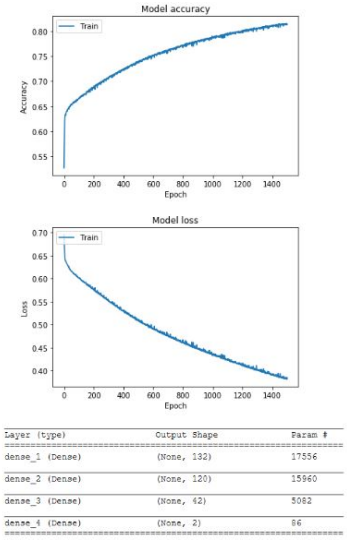


Figure 11:

The NN gave us an accuracy above 80% and was very consistent with loss and accuracy. In an attempt to improve our model, we decided to use adedelta, an optimization used on kernels that are not sensitive to input data.

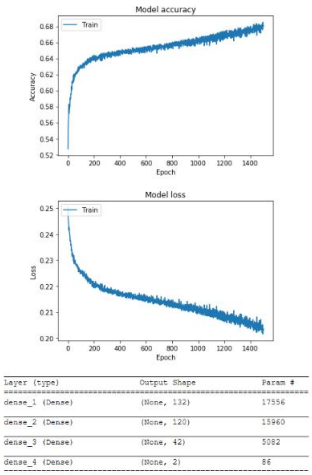


Figure 12:

As shown, the adadelta optimization had produced a substantial decrease in accuracy and an increase for the loss.

Changing the loss function had minor effects, we observed similar accuracy when using similar NNs and only varying the Loss function. As shown below we achieved 84% accuracy using the MSE loss function.

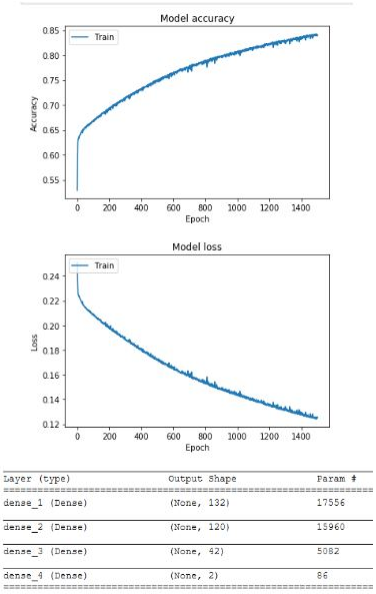
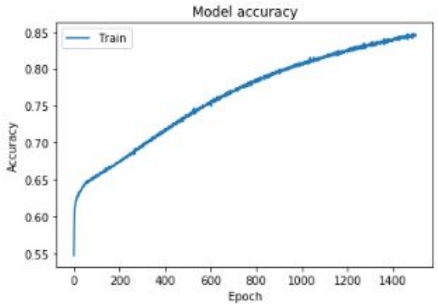


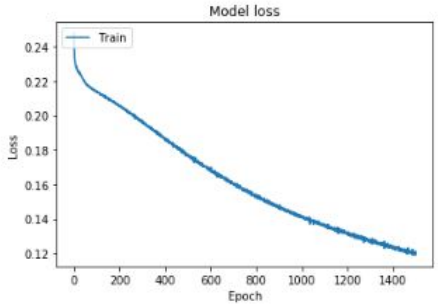
Figure 13:

We can see slightly more variation in the loss as compared to the binary cross-entrophy. We are more familiar with MSE so we decided to stick with it instead of binary cross-entrophy.

Modifying the optimization and using RMS prop instead of adam, gave us 85% accuracy. As shown

below we had a much cleaner learning curve as compared to MSE & adam.



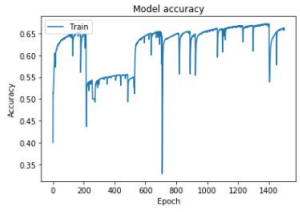


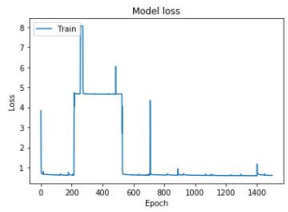
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 132)	17556
dense_2 (Dense)	(None, 120)	15960
dense_3 (Dense)	(None, 42)	5082
dense_4 (Dense)	(None, 2)	86

Figure 14:

The loss variation is very low, which is really nice. We extensively trained this model over the choice of all them and ran it with 10,000 iterations, it asymptotically approached 95% accuracy.

Lastly, we experimented with the infamous, Nadam optimizer, this optimizer adjusts the penalty with momentum, helping the AI learn very quickly when it is not very sensitive to the input data.






```
WARNING:tensorflow:From C:\Users\alexb\Anaconda2\lib\site-packages\tensorflow\python
ate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in
Instructions for updating:
Colocations handled automatically by placer.
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 132)	17556
dense_2 (Dense)	(None, 42)	5586
dense_3 (Dense)	(None, 42)	1806
dense_4 (Dense)	(None, 42)	1806
dense_5 (Dense)	(None, 2)	86

Figure 15:

The loss function for this graph varies a great deal while it is above 85% but we achieved a whopping 92% accuracy with only 1500 iterations.

5 Conclusion

In summary, the statistical learning method that proved to be the most useful was Neural Networks. Looking at it from an analytic standpoint, it is not that hard to understand why; NN are very flexible in that it uses that data to form an optimized representation of the data not limited to a linear model. More specifically, NN characterizes its own optimization through a series of trials which are dependent on the properties of the training data and the set itself.

What makes a regression different is that the model is limited to some hyperplane, ultimately hurting its own accuracy depending on the percentage of the provided dataset are outliers. One may think to use better geometric models instead of a linear object but we must remember the trade-offs in using non-linear models (time, efficiency, correctness, biasing, variance, etc.).