Capstone Project 2 Milestone Report 2

Moving on from the last Milestone Report, the next objective was to create a model that could predict the desired outcomes. Since there were different features and different outcome classes, this was broken down into four problems each with their own basic model which can be found in the “Machine_Learning_Models” Jupyter Notebook. I was unsure that a model would be able to handle the player ID numbers so the first set of models didn’t include them. Instead the handedness of the players involved were taken into account. That means that only information about which hand the pitcher threw with and what side of the plate the batter was hitting from was accounted for.in the first set of models. The other thing I was uncertain about was if the model was able to reliably predict multiple outcomes hence the reason to split this initial step into four problems.

A motivation for this project was to get some exposure to creating and tuning neural networks so that was the model of choice in this situation. I felt a neural network was necessary for this kind of project because I wanted a network to pick up on patterns given the size of the data. The dataset consists of more than 1,5 million observations and deep learning algorithms work better on larger datasets. Another reason is that the data has a lot of dummy variables and if a random forest algorithm were used, it might trees that don’t select the necessary columns needed to pick up on certain patterns. With that being said, the models should be able to outperform straight guessing so it’s accuracy should be greater than the average number of occurrences of the majority outcome. In the situation of the first model which predicts outs, it should have an accuracy better than roughly 0.8264 since an out occurred around 0.1736 of the time in the data.

The first model used the situational data (inning, count, pitch type, pitch speed, and pitch location) along with the player handedness to predict if a pitch resulted in an out. The model had a single dense layer comprising of 40 nodes and the rectified linear unit activation function and an output layer of one with the sigmoid activation function. When compiling the model, the Adam optimizer was used along with the binary cross entropy loss function for this specific problem. The model was then fit and saved on 20 epochs of the data with a 75:25 validation split. An early stopping monitor was set up in case the model converged before the 20 epochs. After the model had finished training it had a loss of roughly 0.3862 on the validation set. Various metrics and graphs were then computed from the trained model and printed out. The first being the graphical representation of the accuracy and loss of the model on the training and validation sets as it was learning the data. Next, metrics like the test loss, test accuracy, and AUC were calculated to be 0.3896, 0.8253, 0.5305 respectively. These metrics aren't indicative of a good model because it's performance resembles pure guessing but it's encouraging for a baseline model. The next step was to increase the model capacity and use it to evaluate how well each of the four models compared to each other.

The four models followed the same rigors as the previous baseline case but with a different model architecture and their respective metrics. Since these four of models were baseline cases that would be further tuned, arbitrary hyper-parameters were chosen to see how they perform relative to each other. All four of these baseline models will have similar model architectures and see the same data. The only differences between each will be columns representing pitcher/batter handedness or player ID numbers and either all six outcomes or just outs. The models were built with 3 hidden, dense layers with all ReLU activation functions. The first layer had 300 nodes while the other two had 100 each. This bottlenecked at the output layer where the number of nodes equaled the number of possible outcomes (either 1 or 6) with

a sigmoid activation function. The models were then evaluated on test accuracy, AUC, and their confusion matrices.

The first model again didn't account for player IDs to predict if a pitch resulted in an out. The results were a 0.3872 test loss, 0.8264 test accuracy, and 0.5188 AUC. The only difference in the second model were the target variables which added the five other possible outcomes of a pitch and because of this, the model metric had to be changed to categorical accuracy. This vastly improved the performance of the model producing a 0.3410 test loss, 0.5105 test accuracy, and 0.7880 AUC. Although the accuracy dramatically decreased, it performs arguably better than guessing unlike the previous cases. In this multi-class problem, the best you could do by randomly guessing is roughly the same as how often the most common class occurred and in this case it's called balls at around 0.3601.

The next two models replaced the columns representing player handedness with player ID numbers. Since most players, especially pitchers, have specific handedness, it would be redundant to have both sets of information and would probably result in some type of collinearity. Also since player ID numbers, which were numbers on the scale of 100,000, it was important to scale the data so that the player IDs don't influence how it gets processed through the neural network so a standard scaler transformation was fit and applied to the data. After making the necessary modifications towards preprocessing the data, it was fit with with the same rigors as the previous models and the results were similar. When predicting outs with player IDs, the model produced a test loss of 0.3881, test accuracy of 0.8262, and AUC of 0.5304. Using player IDs to predict calls resulted in a test loss of 0.3443, test accuracy of 0.5012, and AUC of 0.7825. These results are similar to the models the didn't account for player IDs which is encouraging because hopefully these models could pick up on player matchup

patterns with fine tuning. In other words, improving upon the last model that uses player IDs to predict outcomes will be the focus when moving forward to fine tune the parameters.

The process of fine tuning the neural network can be found in the “model_calls_id” Jupyter Notebook. As in the previous cases, the same preprocessing of the data was done once again in this notebook. A standard scaler was fit and applied to the data so that the player ID numbers don’t influence the neurons drastically. An early stopping monitor was set up along with a 75:25 train-test split of the data. After, a few functions were created to help streamline the process.

The first function takes an instantiated, untrained model and trains that model on specified training data and saves is in a subfolder under a given name. That will save training time when the model needs to be loaded repeatedly in the future. The function will then output where the model is saved and the training history. It's worth mentioning that a few parameters were changed from the models in the “Machine_Learning_Models” notebook. The random seed was changed so the models would see new data and more epochs were added in case more training time was needed.

The second function then takes the training history of the fitted model and plots the accuracy and loss of both the training data and validation set. The plots will then be saved in the same folder as the trained models in case they need to be accessed later.

The last function prints out various statistics as a result of the model in a number of steps and saves it to a dataframe for easier comparisons. The first is that the model will predict the probabilities of each outcome for each pitch and assign them to predictions. Given a threshold value, the predictions will then be assigned to a 1 or 0 with the exception of “call_H” predictions. These predictions are given their own separate threshold value due to the class imbalances. These values will then be saved as a dataframe where they will be compared to the

actual test data and each other. The first statistic that is outputted is the loss in the validation set. In theory, this is the value that should be minimized as it's an indication of the models performance on unseen data. The next statistic is the AUC-ROC which is an indication of how well the model fits the data at various threshold values. Following that is the Hamming loss which is calculated as the fraction of labels that are incorrectly labeled, so the smaller this number is the better. The last set of metrics that are the per class precision, recall, and F1 scores. This is to get knowledge on how well the model is doing at predicting the different types of outcomes.

Consequently, 25 models were constructed with varying degrees of complexity starting off with simple models and gradually getting more complex. Various numbers of layers and nodes per layer were tested along with three activation functions: rectified linear unit, tanh, and exponential linear unit. In the end, all the models performed similar to each other in terms of the metrics used to compare them. They all hovered around a 0.3441 loss on the validation set, 0.6908 AUC-ROC score, and 0.2313 Hamming loss. It’s difficult to single out a best model because if a model as one outstanding metric, another model performs better in a different metric. However, if a model had to be chosen, model_final8 produced a 0.3437 loss on the validation set and a 0.2229 Hamming loss which are encouraging metrics on unseen data. The flaw in all the models are their per class predictions and that’s no different for model_final8. The following are the F1 scores for each of the classes predicted by the model:

- Called balls: 0.7532
- Called strikes: 0.5408
- Fouled balls: 0.3971
- Hits: 0.2137
- Swinging strikes: 0.1781

- Outs: 0.4610

The closer these values are to 1, the better the model is at predicting that class so like all other models, this one had an exceptionally hard time predicting hits and swinging strikes.

Clearly, improvements need to be done on this project to develop better models and that could be done in a number of ways. One way is to include who is the umpire calling balls and strikes. Like batters and pitchers, umpires have their own different approach to calling balls and strikes and it's ultimately up to them to call balls and strikes that the players have to adjust to. Another thing that could be done to the data to help model performance is to handle player ID numbers in a different manner, perhaps by vectorizing them. The sheer scale of the player ID numbers posed a problem when compared to the scale of the other predictors. The last two improvements is a way to handle the class imbalances which include down/up sampling the classes and/or reducing the machine learning problem to multi-class classification. Downsampling and upsampling the most common and least common classes respectively helps the model learn about what makes those classes unique. Then reducing the machine learning problem to a simple multi-class classification problem, instead of multi-label, could have the model focus on predicting one specific class since the only possible multi-label situation is the case of a strikeout where either a called/swinging strike with 2 strikes produces an out.