Capstone Project 2: PUBG Finish Placement Predictions

Overview:

PUBG is a game where up to 100 players start in match onto an island empty handed and must explore, scavenge and eliminate other players in a free for all until only one player/team is left standing. Kaggle has given us 65,000 games' worth of anonymized data and we're asked to predict the final placement in a game given the in-game stats and initial player ratings. Kaggle has asked that we evaluate our model on Mean Absolute Error between the predicted win placement and the observed win placement.

Target Audience:

Within the last 2 years, Battle Royale (BR) styled video games have exploded in popularity, giving us new IPs such as Fortnite and Realm Royale and even affecting long standing franchises like Black Ops. This surge in numbers can arguably be traced back to PUBG, one of the first BR games to receive a substantial amount of development and sustain a large, consistent player pool. The premise of the game is simple: You and up to total of 100 players are dropped onto a large island empty handed and you must explore, scavenge and eliminate other players in a free-for-all/team vs team first person shooter styled battle until only one player or team is left standing. This project aims to use 65,000 games' worth of anonymized data to predict final placements in a game using over 20 different in-game stats. There are a couple groups of people that would be interested in the results:

- The game's developers: The data analyzed could be used to help conduct balance changes based off of impact of each feature.
- The players: The data analyzed could be used to help pinpoint weaknesses in a player's performance, giving them an objective avenue of improvement to go through.
- Tournament organizers: High impact features could be used when casting tournament games, being able to select the top 5 or 10 most impactful stats that players should keep an eye on.

Data Description:

The data I will be using can be found on Kaggle (here). The file list includes:

- A sample submission file in the correct format
- A training dataset (4.45m rows)
- A test dataset (1.93m rows)

The columns found in our data are as follows:

- DBNOs
- Assists
- Boosts
- damageDealt
- headshotKills
- Heals

- Id
- killPlace
- killPoints
- killStreaks
- Kills
- longestKill
- matchDuration
- matchId
- matchType
- rankPoints
- Revives
- rideDistance
- roadKills
- swimDistance
- teamKills
- vehicleDestroys
- walkDistance
- weaponsAcquired
- winPoints
- groupId
- numGroups
- maxPlace
- winPlacePerc - This is our prediction target

Data Initial Inspection, Cleaning and Wrangling:

After the initial importing of our training and test files into dataframes, the first step I took was to inspect the different game modes available in our data set. Looking at the list, I've split these game modes up as follows:

- Standard Game Modes:
  - Solo Game Modes:
    - solo
    - solo-fpp
  - Multiplayer Game Modes:
    - squad
    - duo
    - squad-fpp
    - duo-fpp
- Non-Standard Game Modes:
  - normal-squad-fpp
  - crashfpp
  - flaretpp
  - normal-solo-fpp
  - flarefpp
  - normal-duo-fpp
  - normal-duo
  - normal-squad
  - crashtpp
  - normal-solo

For reference, Solo games are games where the player is matched solo against up to 100 other solo players. Duo game modes are games where the player and a friend is matched with other partnered teams with a max player count of up to 100 players per game. Squad game modes are games where the player and up to 3 teammates are matched with other similar groups with a max player count of up to 100 players per game. As for FPP vs Non-FPP game types, FPP modes are modes where the players are forced into a first person shooter perspective. However, the remaining mechanics are exactly the same. With these categories in place, the statistics of the number of observations/games for each of the game modes are as follows:

- Game mode : solo, Number of observations - 181943, Number of games - 2297
- Game mode : solo-fpp, Number of observations - 536762, Number of games - 5679
- Game mode : duo, Number of observations - 313591, Number of games - 3356
- Game mode : duo-fpp, Number of observations - 996691, Number of games - 10620
- Game mode : squad, Number of observations - 626526, Number of games - 6658
- Game mode : squad-fpp, Number of observations - 1756186, Number of games - 18576

- Game mode : normal-squad-fpp, Number of observations - 17174, Number of games - 358
- Game mode : crashfpp, Number of observations - 6287, Number of games - 73
- Game mode : flaretpp, Number of observations - 2505, Number of games - 29
- Game mode : normal-solo-fpp, Number of observations - 1682, Number of games - 96
- Game mode : flarefpp, Number of observations - 718, Number of games - 9
- Game mode : normal-duo-fpp, Number of observations - 5489, Number of games - 158
- Game mode : normal-duo, Number of observations - 199, Number of games - 12
- Game mode : normal-squad, Number of observations - 516, Number of games - 16
- Game mode : crashtpp, Number of observations - 371, Number of games - 5
- Game mode : normal-solo, Number of observations - 326, Number of games - 23

EDA (Visualizations and Inferential Statistics):

Following this, the training data was then split into separate dataframes based on the game types (with the non-standard game modes combined into one dataframe). The following procedure was then applied to each of these dataframes:

1. The .describe method was used to take a look at the general base statistics of the dataframe.
2. The .info method was used to confirm that there are no null values in the dataframe.
3. Unnecessary columns were dropped (“DBNOs”/”revives” for solo game mode types).
4. A feature list was populated.
5. A distribution plot was made for each of the features in the feature list.
6. A pearson correlation heatmap was generated.

7. A clustermap was generated.

After completing these, some general conclusions about the datasets are as follows:

- Applicable to all game mode types:
    - For the most part, the statistical distributions plotted all follow an expected non-normal distribution. From these distributions, I do see instances of outliers in our data. My opinion on how these data points came about can be attributed to hacking programs that are available online for PUBG. This gives players an advantage in game that would explain how certain players have boosted values specifically in the “damageDealt”, “headshotKills”, “heals”, “kills”, “killStreaks”, “longestKill”, “rideDistance”, “weaponsAcquired” and “winPoints” features. However, as these instances total for less than .1% of the total number of observations as well as it being relatively difficult to decide an arbitrary line at which point to remove these outliers, these rows of data have been left in. Also, no null values were found in the data set (I believe Kaggle did an initial cleaning of the data before it was uploaded for use in the competition) and so no extra steps to clean the data were needed.
    - Looking at the heatmaps and clustermaps of all the different game modes, I see the number one most correlated value with winning placement is the distance walked by the player. This naturally makes sense: Walking distance is a function of how long the game goes on (the longer the game is the more the player needs to move to keep up with the decreasing area of play), and players that score high winning placements will necessarily have more time played in a given match.
- Solo Game Modes:
    - Along with the walking distance metric, other highly correlated values with the winning placement are:
        - Boosts
        - Kills (Kill Placement, Kill Streaks, Damage Dealt, Longest Kill Streak)
        - Weapons Acquired.
- Duo Game Modes:
    - Along with the walking distance metric, other highly correlated values with the winning placement are:
        - Boosts
        - Kills (Kill Placement, Kill Streaks, Damage Dealt, Longest Kill Streak),
        - DBNOs
        - Heals
        - Weapons Acquired.
- Squad Game Modes:
    - Along with the walking distance metric, other highly correlated values with the winning placement are:

        - Boosts
        - Assists
        - Kills (Kill Placement, Kill Streaks, Damage Dealt, Longest Kill Streak)
        - Revives
        - DBNOs
        - Heals
        - Weapons Acquired

From here, in order to determine how to progress with the machine learning algorithms, I needed to compare and test for any statistically significant differences between the feature distributions for the different game modes. I conducted two comparisons, Non-FPP and FPP modes comparison (choosing the solo game modes data sets to test for this) and solo/duo/squad matches comparison. As done previously with our EDA, for each comparison I made distribution plots of each of the features for each game mode being tested but instead this time overlaid on top of each other. To confirm my results, a Mann-Witney U test was performed as I am working with non-normal distributions. With these comparisons done, I see no statistically significant differences between any of the standard game types. This is important, as this allows me to combined all the data for standard game modes in order to build a highly specific model. Although this hold true, I continued with my ML work by splitting up the multiplayer game modes and the solo game modes as we see important features that are missing in the solo game modes ("DBNOs"/"Assists")

<u>Machine Learning:</u>

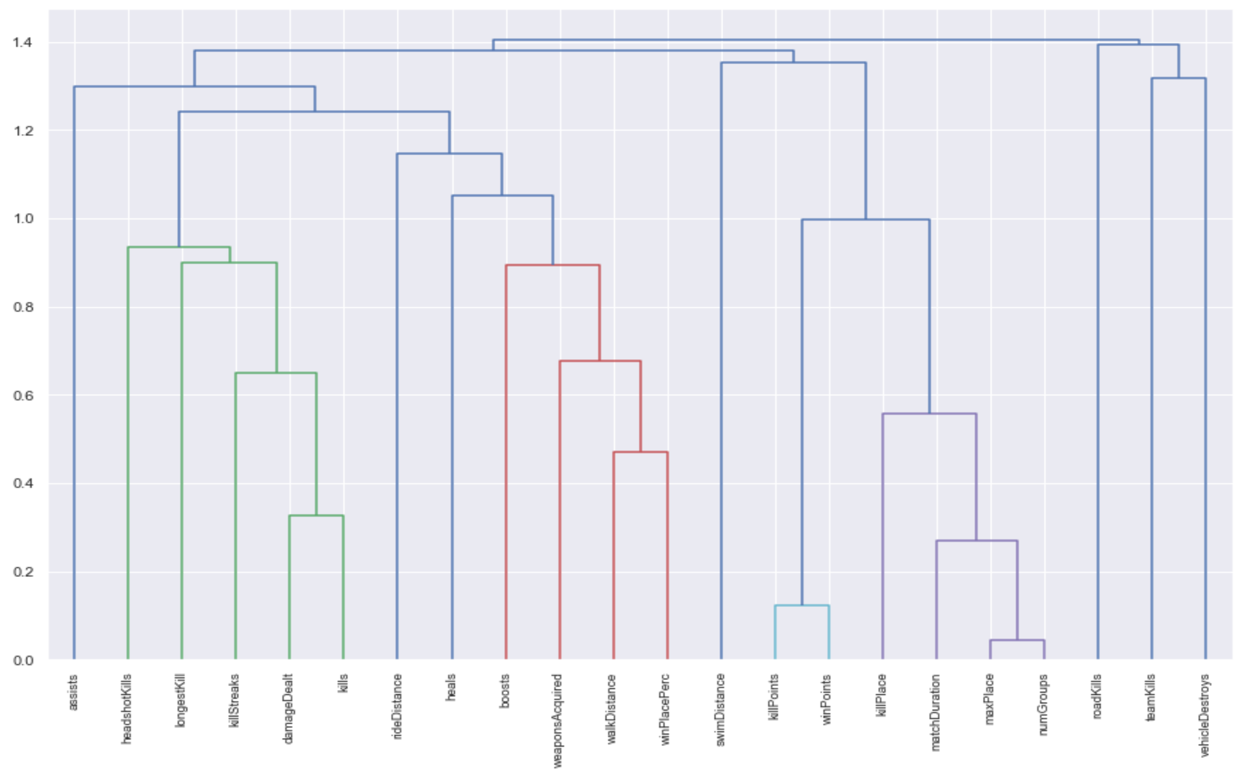
As building a model with 5 million rows of observations to start would greatly increase the time needed to finish my project, I started by building models for specific game modes (Solo, Solo FPP, Duo and Squad). I also tested 3 different ways of building a predictive model to determine win placements: ElasticNet, Random Forest and Keras. Using the conclusion drawn from my comparison tests, I would be able to apply all my parameter tuning and keras model tuning to my final model based off the results from the individual models.

<u>Solo Game Mode Models:</u>

After importing the solo game mode data into a dataframe, I split it up into a training set, a test set and a validation set with these parameters:

- Training Features Shape: (101888, 21)
- Training Labels Shape: (101888,)
- Validation Features Shape: (25472, 21)
- Validation Labels Shape: (25472,)
- Testing Features Shape: (54583, 21)
- Testing Labels Shape: (54583,)

To confirm what I saw in the initial EDA with the cluster maps, I used hierarchical clustering to draw a dendrogram of the solo game features:



ElasticNet Model

Using a crossfold-validation factor of 5, I fit my model to the training data and tested it against the validation and test data giving me values of R^2 = .797, Mean Absolute Error (MEA) = .1034 for the validation data and R^2 = .799, MEA = .1035 for the test data.

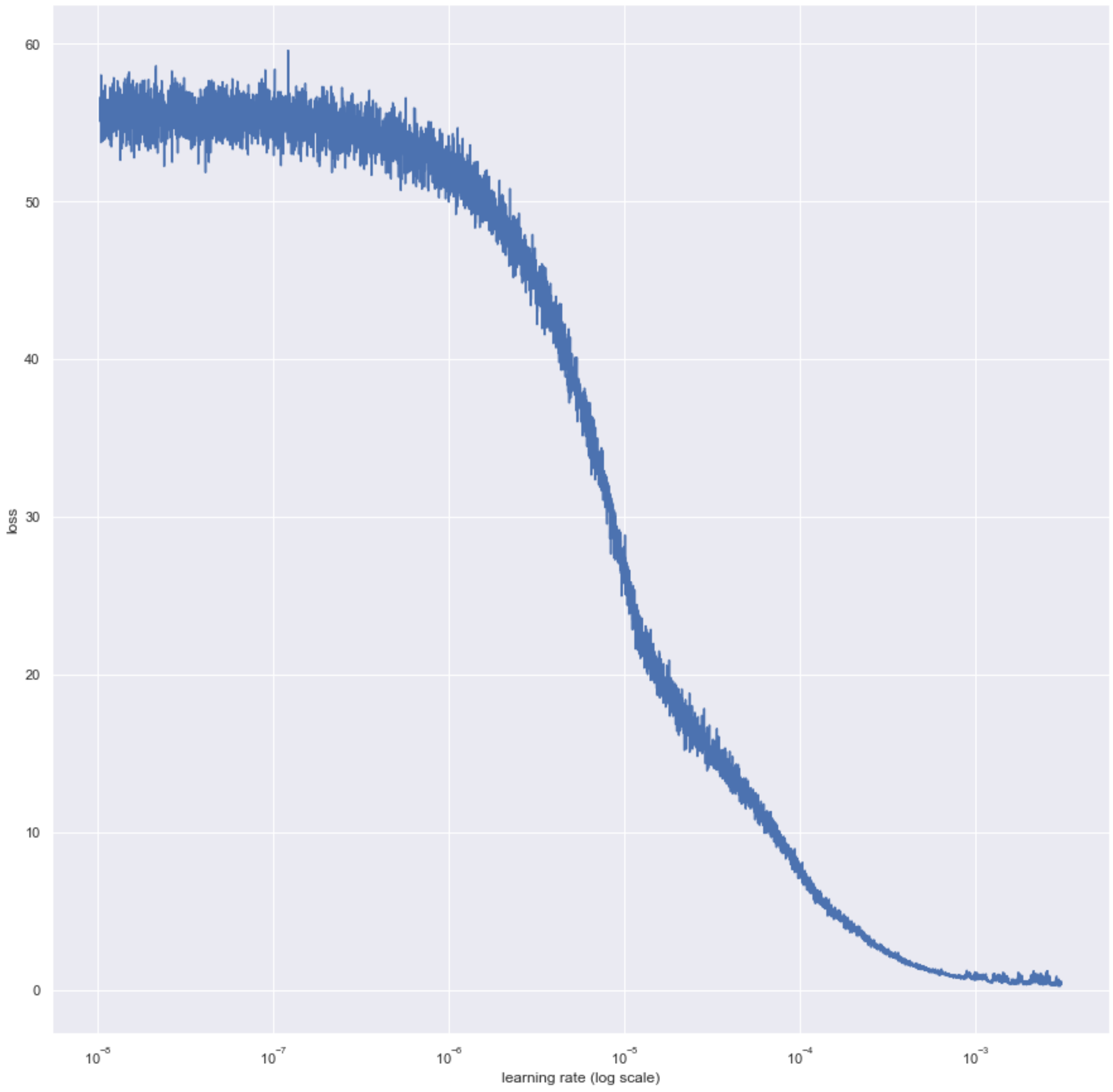
Random Forest:

- I start with parameter tuning. Setting up a for loop with our parameter value ranges I fit a model using the training data and finish by evaluating the OOB score, the R^2 score and the RMSE to determine the best parameters to use for our Random Forest Model.
- Using the results from parameter tuning, I chose the values of n_estimators = 10, max_depth = 15, max_features = 15 to build my Random Forest model. Doing so after combining my validation and training data sets, my model evaluated scores of R^2 = .937 and a Mean Absolute Error of .047 on the test set.
- Going further, I wanted to see if there were any features that were particularly important and also to test if I could refine the model with feature selection. The results are as shown:

- Choosing the 5 most important features, I rebuild my Random Forest model and achieve an R^2 score of .922 and a Mean Absolute Error of .052, indicating that refining the features used in building my model does not provide any benefit to the final model's accuracy.

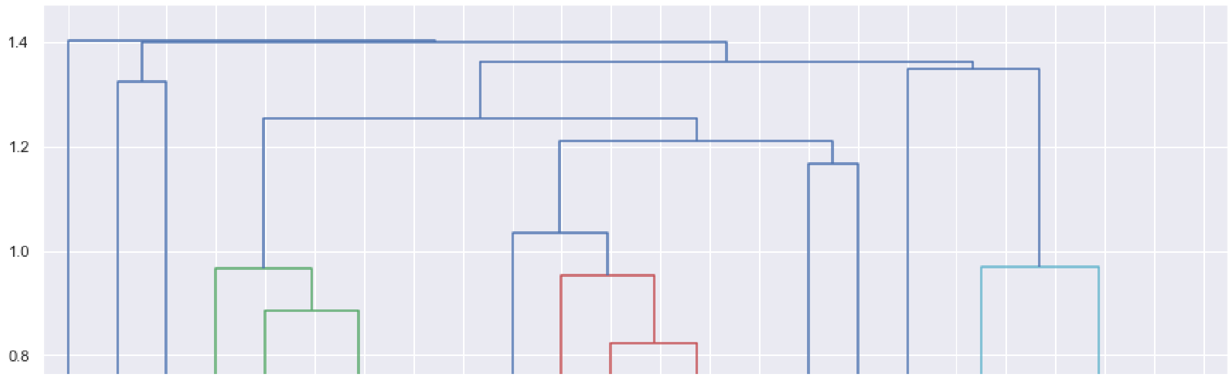
Deep Learning Models Using Keras

- While working with the solo game dataset, I conducted multiple keras tests to try to optimize a deep learning model. Although I'll describe here the different tests conducted, please refer to the code for more specific details:
    - Basic model
    - Higher node count
    - Lower learning rate with higher epochs
    - Increased layers
    - Increased layers and higher node count
    - High epoch (100) count
    - Different optimizers:
        - Adam
        - Adagrad
        - Adadelta
        - Nadam
        - Adamax
- I also implemented an LRFinder class in order to determine the optimal learning rate for my dataset. The specifics of this can be found in the code file, but the corresponding loss plot is as follows:



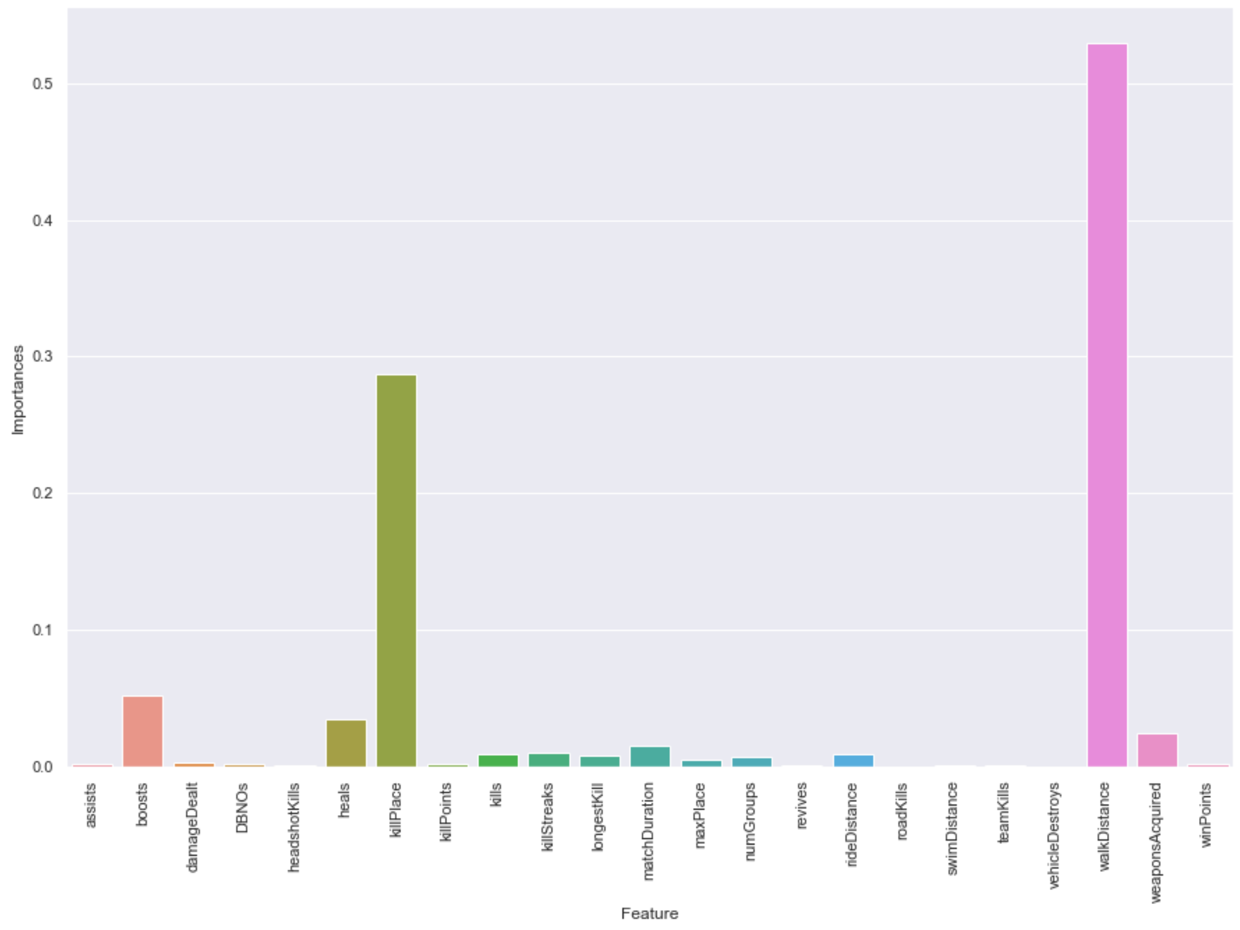
<u>Squad Game Mode Models:</u>

Following the same steps as done with the solo game mode dataset, I used hierarchical clustering to draw a dendrogram of the squad game features:



Random Forest Model

- Data Shape:
    - Training Features Shape: (350854, 23)
    - Training Labels Shape: (350854,)
    - Validation Features Shape: (87714, 23)
    - Validation Labels Shape: (87714,)
    - Testing Features Shape: (187958, 23)
    - Testing Labels Shape: (187958,)
- Once again, tuning our parameters I get n_estimaors = 10, max_depth = 15, max_features = 15. From this point on I used these parameters for all my RF models I built. The model built with these parameters gives a value of R^2 = .904 and MEA of .072.
- Testing for feature selection, I get:



Deep Learning Model Using Keras

- Since the majority of the testing was already done with the solo data, I built only one model using Keras. The parameters are as follows:
    - Model type = Sequential
    - 4 Dense layers of 10 nodes each with "relu" activation with an output layer
    - Compiler = nadam
    - Loss = mean_absolute_error
    - 20% validation split and 30 epochs with a 10 count patience early stopping monitor.
- Using the above model, we achieve a mean absolute error of .078.

<u>Duo Game Mode Models:</u>

Random Forest Model

- Data Shape:
    - Training Features Shape: (219513, 23)
    - Training Labels Shape: (219513,)
    - Testing Features Shape: (94078, 23)
    - Testing Labels Shape: (94078,)
- Model Evaluation Score:
    - R^2: 0.938140613436743
    - Mean Absolute Error: 0.05361313231533958

Deep Learning Model Using Keras

Model Evaluation Score:

- Keras Model's Evaluation: 0.25470508310134
- Mean Absolute Error: 0.06487467905801968

<u>Solo FPP Game Mode Models:</u>

Random Forest Model

- Data Shape:
    - Training Features Shape: (375732, 21)
    - Training Labels Shape: (375732,)
    - Testing Features Shape: (161029, 21)
    - Testing Labels Shape: (161029,)
- Model Evaluation Score:
    - R^2: 0.9617055755293075
    - Mean Absolute Error: 0.04070105669918561

Deep Learning Model Using Keras

Model Evaluation Score:

- Keras Model's Evaluation: 0.2252151389192701
- Mean Absolute Error: 0.05072185873395925

<u>Testing Accuracy of Combined Duo/Squad Game Mode Models:</u>
For this test, I combined the datasets for the Duo and Squad mode games (Non FPP) and proceeded as usual.

Random Forest Model

- Data Shape:
    - Training Features Shape: (658081, 23)
    - Training Labels Shape: (658081,)
    - Testing Features Shape: (282036, 23)
    - Testing Labels Shape: (282036,)
- Model Evaluation Score:
    - R^2: 0.9148389517026513
    - Mean Absolute Error: 0.06579825512792938

Deep Learning Model Using Keras

Model Evaluation Score:

- Keras Model's Evaluation: 0.2759738259169069
- Mean Absolute Error: 0.07616155284389273

<u>Testing Accuracy of Combined Solo/Solo FPP Game Mode Models:</u>
For this test, I combined the datasets for the Solo and Solo FPP mode games and proceeded as usual.

Random Forest Model

- Data Shape:
    - Training Features Shape: (503092, 21)
    - Training Labels Shape: (503092,)
    - Testing Features Shape: (215612, 21)
    - Testing Labels Shape: (215612,)
- Model Evaluation Score:
    - R^2: 0.9562438511432078
    - Mean Absolute Error: 0.04194501403444709

Deep Learning Model Using Keras

Model Evaluation Score:

- Keras Model's Evaluation: 0.22189740920931383
- Mean Absolute Error: 0.04923846020927919

<u>Testing Non Standard Game Mode Models:</u>

Finally, before building my final models I need to build and evaluate the performance of a model built using non-standard game mode's set of data. As the data only has thirty-five thousand lines of observations, a keras model could not be built. Instead, I chose to just use the same parameters as done before and built a Random Forest model. Doing so, I get an R^2 of .626 and a MEA of .139.

<u>Building Final Test Models:</u>
Using the best parameters as tested above, we combine Solo and Solo FPP into one dataset (Solo game modes) and combine Duo/Duo FPP/Squad/Squad FPP into one dataset (Multiplayer game modes) and build an RF model and a Keras model for each.

Solo Game Modes

- Data Shape
    - Training Features Shape: (503092, 21)
    - Training Labels Shape: (503092,)
    - Testing Features Shape: (215612, 21)
    - Testing Labels Shape: (215612,)
- Random Forest
    - R^2: 0.9562438511432078
    - Mean Absolute Error: 0.041945014034447094
- Keras
    - Keras Model's Evaluation: 0.25766196318642476
    - Mean Absolute Error: 0.06638968754153332

Multiplayer Game Modes

- Data Shape
    - Training Features Shape: (2585095, 23)
    - Training Labels Shape: (2585095,)
    - Testing Features Shape: (1107899, 23)
    - Testing Labels Shape: (1107899,)
- Random Forest
    - R^2: 0.9235901834377193
    - Mean Absolute Error: 0.061807421502185954
- Keras
    - Keras Model's Evaluation: 0.26497713117286714
    - Mean Absolute Error: 0.07021288031672876

<u>Kaggle Data Competition Submission:</u>

From the beginning, this project was based off of a kaggle competition (https://www.kaggle.com/c/pubg-finish-placement-prediction) and therefore I completed the project by creating a CSV file for submission. The steps taken to do so are outlined here:

1. Review the sizes of the training and test files:
    a. Training Data's Shape: (4446966, 29)
    b. Testing Data's Shape: (1934174, 28)
2. Split up the training data set into solo game modes, multiplayer game modes and non standard game modes as well as splitting each group of games into the respective training features and labels. The number of games for each mode is found here:
    a. Solo game modes: 718704
    b. Multiplayer game modes: 3692994
    c. Non standard game modes: 35267
3. Construct a Random Forest model for each of these game modes separately using the optimal parameters as tested in the previous sections.
4. From the test data files, I also split up the data into solo/multiplayer/non standard game modes. The respective sizes of each of these dataframes are:
    a. Standard Solo Games Shape: 313767
    b. Standard Multiplayer Games Shape: 1610569
    c. Non Standard Games Shape: 9838
    d. Total Games Shape: 1934174
5. These test dataframes are then pared down to include the necessary features as well as the each row's Id column's value.
6. Using the three different RF models I built in step 3, I found and stored the predicted win place percentages into 3 different arrays.
7. Finally, I concatenated the respective ID value with the corresponding win place percentage for each game mode, then stacked the predictions on top of each other to complete the final prediction dataframe.
8. Using pandas' built in to_csv method, this dataframe is then exported as a CSV file that was used to submit to Kaggle for evaluation.

Conclusions and Possible Future Work:

This concludes the report on my 2nd capstone project, PUBG Final Placement Predictions. Some interesting things to note are:

- Because this project is based off of a Kaggle competition, the data that I used had seemingly already been scrubbed. When it comes to real world projects, this should not be the case and for future projects a heavier emphasis on the initial data wrangling will be necessary.
- During the intial EDA, when looking at the highly correlated features with respect to the win place percentage, the ones that stood out the most did not give very much insight into

the question of “What makes a player good at the game?” Specific features such as “walking distance” and “weapons acquired” do not give any indication of player skill level; they are just byproducts of staying alive longer which necessarily translate to a higher win placement within a game.

- Although far more time was put into tuning the deep learning keras models used in this capstone project, ultimately the random forest models built showed better results. This is a testament to the strength of a Random Forest model and how it can be used as an ensemble method to capture highly non-linear data sets.

Some possible future work that could be done to expand on this project are:

- More dedicated time towards running deep learning keras models. As found in our intial parameter testing for the deep learning models used, I found that the optimal learning rate was at 10^{-4}. In order to test and build models using this far smaller learning rate, a much higher epoch count must be used to compensate. Due to time constraints in this project (and possibly hardware constraints as well), future keras models could be built with these new parameters and left to run without an early stopping monitor to see if a more accurate model can be built.
- Using more data to build the deep learning models. I believe that deep learning models will outperform the standard out of the bag Random Forest models that I’ve used to build my final submission predictions. However, as I add more neurons and layers into the model and increase the features being tested, the data needed to build the model scales up as well.
- Creating a hybrid model: Although the details of how to proceed with this type of model is unclear to me, under my mentor’s suggestion I have looked into publications that combine deep learning neural nets with an ensemble method like Random Forest to build a hybrid model.