Springboard - DSCT
Capstone Project 1 - Final Report

David Buehler

October, 2019

Introduction

PlayerUnknown's Battlegrounds (PUBG) is a wildly successful battle royale game on Steam, developed by PUBG Corporation, a subsidiary of Bluehole Studio, Inc. The game at its peak had 3.2 million concurrent players, and was all anyone that I knew talked about. Kaggle had a competition to see who best could make predictions on a set of millions of players data from PUBG, as quoted from their website: "*predict final placement from final in-game stats and initial player ratings.*" This was the problem I set out to solve. I needed to come up with machine learning models to predict the final placement of players in the game based on in-game stats, as well as initial player ratings. Through these models, I could look at what features of the dataset were the most important in making the final prediction, hoping that these importances would reveal an in-game strategy that was best suited to winning any given game. Per Kaggle: "*What's the best strategy to win in PUBG? Should you sit in one spot and hide your way into victory, or do you need to be the top shot?*"

The client in this case is the PUBG Corporation, mainly the developers of the game. This will potentially help the developers of PUBG create a better game if it seems like players are

spending too much time camping and not enough time looking for kills, which the community, including me, finds it to be a boring play style. The developers could find ways to incentivize movement in the game and finding other players to kill, making the game a more fun experience overall.

For the machine learning modeling portion I decided to use regression models as the target variable is a continuous one instead of a categorical variable. Linear, Ridge, Lasso, Random Forest, and XGBoost regressors were used in the machine learning modeling portion. My study reveals some expected and unexpected results. Linear and Ridge regression both had an R^2 value of .88, with Lasso regression falling a bit behind at .86.

Looking at feature importances and coefficients, on all three types of regression, kill streaks had the largest negative effect on the predictions of the model, while road kills had the largest positive effect on the Linear and Ridge models, and weapons acquired having the largest effect on the prediction of the Lasso model. For Random Forest regression and XGBoost, they both performed better than the first three models, with Random Forest having and R^2 value of .95 and XGBoost doing a bit better at .96. Both models relied heavily on walk distance as the largest feature importance. This was unexpected, as I thought something like kills, kill place, or kill streaks would be the most important variable in any model.

Approach to the Problem

Data acquisition and wrangling

For my project, I decided to focus on just the 'solos' game type, as that was the game type I enjoyed and played the most. In my opinion, this style offers that 'true' battle royale experience of being by yourself against 99 other players, and it was a fight to the death to be the last player standing. To extract the data for this game type, I just needed to pull the data that matched "solo", "solo-fpp", or "normal-solo-fpp" and put all of that data into its own Pandas DataFrame. Having done this, I could drop a few columns from the data set right away: down but not outs (DBNOs) and revives specifically. These two are attributes of duo or group play, and will never happen during a solo game. I also dropped matchId and groupId since those only gave categorical information about what match or group you are in. I assume that everyone in a group will have the same group Id, so that was redundant information when dealing with just solo players.

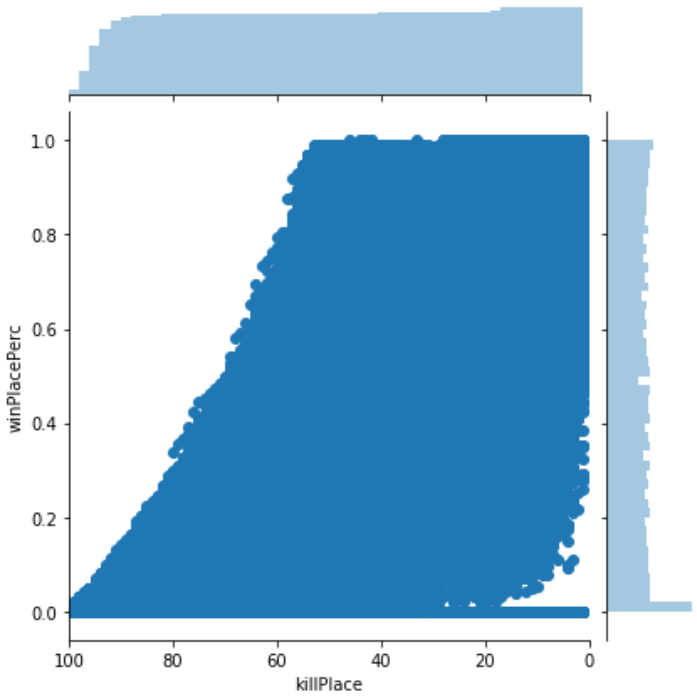
Right from the outset of this project I knew I would have to deal with cheaters in the data set. However, those are hard to find as there is no label that says "This data point represents someone who is a cheater". I did two things that, from my own experience, I thought may remove the obvious cheaters from the data set. First I dropped data points that had more than 50 kills. There were only 9 such entries and while some of them may have been

actual players that just had a great game, it is more likely that these were cheaters from PUBG's rampant cheating problem in early 2018. Another criteria I had for cheaters was the ratio of headshot kills to kills. Anything larger than 80% and with greater than 20 kills I also considered to be a cheater, and subsequently dropped them from the data set.

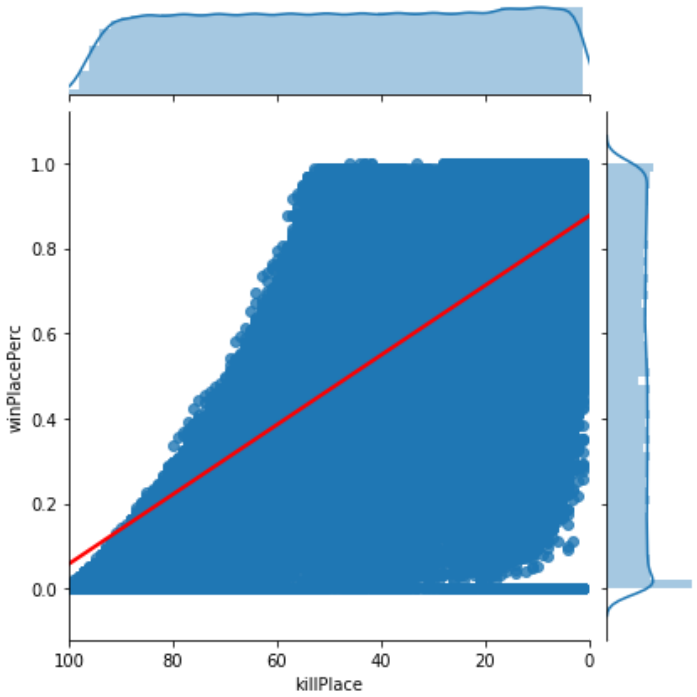
Fortunately this data set came from player data directly from PUBG Corporation and Bluehole, so there were no missing values in the data set at all. However, there were some large outliers in the match duration column in the data set. I noticed that the minimum value in the matchDuration column was 9. That means there was a match listed in the data set that lasted 9 seconds. I dug a little deeper into this and found that there were 246 entries that lasted no longer than 10 minutes. With this information I decided to drop all rows that were greater than 3 standard deviations from the mean in the matchDuration column. This is a standard practice in statistics when trying to find outliers, so I applied that rule of thumb here.

Data Storytelling and Inferential Statistics

Data storytelling mostly consisted of exploring the effect independent variables have on win place percentage. I took a look at a number of the columns that I thought would have a strong effect on it: kills, damage dealt, kill place, heals, match duration, and weapons acquired. Out of all of these graphs, it seemed like kill place will have the largest effect on the outcome of a person's placement.



Shown above is that graph, and the linear relationship can be seen fairly easily. As a persons kill place gets better (lower is better), the more likely they are to have a better win place percentage. Here is that same graph with a linear regression line plotted:



Seen here is a clear correlation between kill place and win place percentage. Using statsmodels ordinary least squares regression (OLS), I made a linear regression model between kill place and winPlacePerc, and got an R^2 value of .576, which was the highest of the five variables tested. The next highest, surprisingly, was weapons acquired, with an R^2 value of .369. Not nearly as strong a correlation, but a noteworthy one.

Before I got into the OLS regression however, I performed a bootstrap test on the data, specifically looking at people with 5 or more kills and how they performed vs. people with less than 5 kills. I came up with a null hypothesis of: "On average, players with 5 or more kills win games as often as players with less than 5 kills". Performing the test, I found that people with 5 or more kills performed better than people with less than 5 kills. Those with 5 or more kills had a 95% confidence interval between 0.8941 and 0.8974, and those with less than 5 kills had a 95% confidence interval between 0.4718 and 0.4732. Just by these numbers alone we can see that the intervals have no overlap and it would be extremely unlikely to observe results like this just by chance. Performing a t-test from scipy.stats ttest_ind() function, I got a p-value of 0, thus rejecting the null hypothesis.

Baseline Modeling

To begin the modeling section, I knew I needed a base regression model against which to compare other models. I started with a base linear regression model with no hyperparameter tuning as a baseline. After fitting the data to the training set, I took a look at

the coefficients of the model and got some interesting information straight away. I thought that kills or kill place would be the most positive coefficient for a linear regression model, but it turned out to be road kills. Kills and kill place actually had negative coefficients which means a player's increase in kills or kill place will contribute to its win place percentage to go down. After this I predicted the test set, and came up with an R^2 value of .88 for both training and testing data, which is pretty good for a baseline model. This model showed no signs of overfitting; however the model had predicted values above 1 and below 0, which do not align with the scale of the target, a percentage, and therefore its values are expected to be between 0 and 1.

There were a couple approaches to this that were thought of, and I decided to take the natural log of the target variable, fit and predict, then exponentiate the predictions to get back to the original problem setup. Taking the score of this log model, it came out with an R^2 value of .77 on both training and testing data, which is almost .11 down from the base linear model.

Seeing as there was no overfitting in the linear regression model, using Lasso and Ridge regression is a moot point. However for practice I decided to try Lasso and Ridge regression and see if those did any better on the data. Both Lasso and Ridge regression take inputs in the form of alpha values, so I needed to find the best suited alpha value for both forms of regression, and I did so using a GridSearchCV over a large range of numbers and 5-fold cross validation. The best alpha for Lasso and Ridge were .01 and 190 respectively. Starting with Lasso regression, I plugged in the alpha value, fit the data then looked at the coefficients and accuracy on the test data. With an R^2 value of .86 on the training and testing data, Lasso regression did not turn out any better than Linear regression, which left Ridge regression as the final type to be tested.

Ridge regression was fitted to the data with an alpha value of 190, and came out with road kills back to being the largest positive coefficient, and kill streaks still remaining as the largest negative coefficient. Ridge regression graphs seen below were very similar to the Linear regression graphs from earlier, even the accuracy score between the two models were the same at .88.

Extended Modeling

After getting all of the baseline modeling done, it was time to look deeper into the data set and see if I could find a more accurate model. The first thing I tried was Random Forest regression, and the first job was to tune some hyperparameters, in this case number of estimators (trees in the forest) and the maximum depth of those estimators. Using the same method I used for finding alpha values in Lasso and Ridge regression, GridSearchCV with 11 different values for the hyperparameters, I let the cross validation run and came out with values of 100 for the best number of estimators and 20 for the best maximum depth. With those figured out, I fit and predicted on the data set, and the test R^2 value came out to .95. Much better than the three other methods I tried earlier. Next I took a look at the feature importances of the model. Fortunately the Random Forest regressor has an attribute feature_importances_ that lets me look at them very easily. Next I wanted to see if max_features had any effect on the accuracy of the model. From scikit-learn's documentation, I found that Random Forest had three types of max features. "Auto" which used the total number of features available to the model, "sqrt" which used the square root of the number of features, and "log2" which used a base 2 logarithm of the number of features.

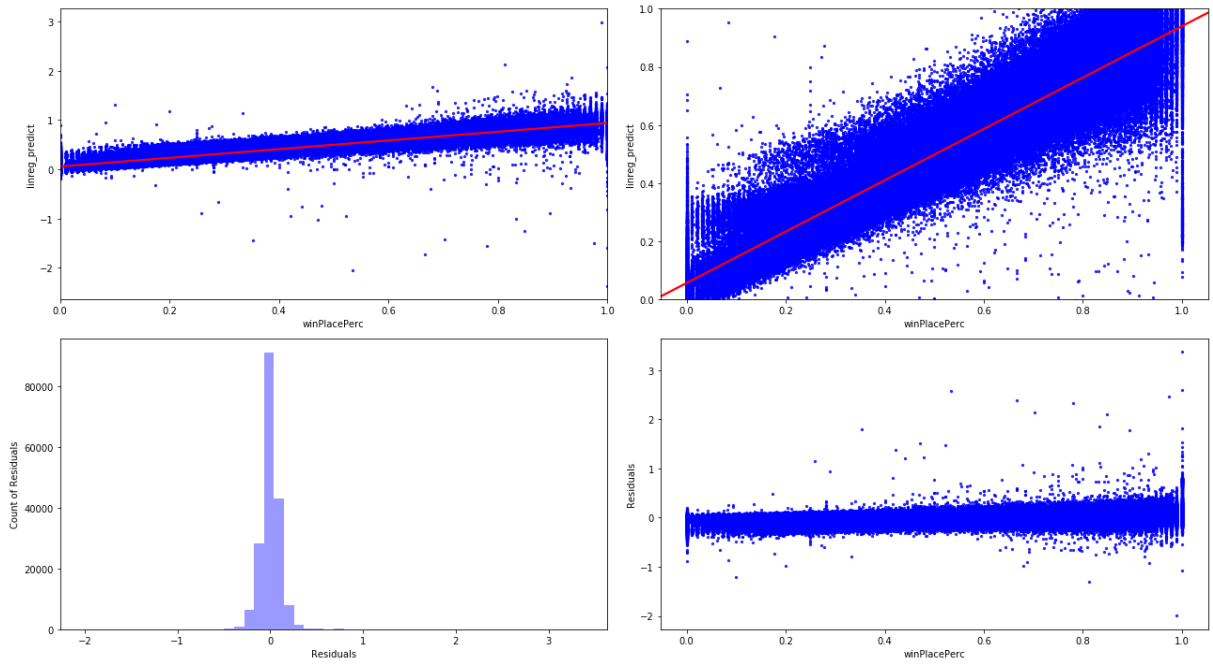
Finally, I wanted to take a look at a package outside of scikit-learn and decided to go with XGBoost. Gradient boosting is another type of random forest approach where each new tree is based off the success or failure of the tree made before it, so it helps correct errors as the model progresses. For this model I decided not to go with the GridSearchCV, purely to save on time and instead just plugged in increasing number of estimators values. I settled on 2000 estimators as that is when I noticed the score beginning to plateau and the model was only getting slightly better as the number of trees went up. After settling on a value for the number of estimators, then fitting and predicting, I again took a look at the feature importances to see if they were any different than the Random Forest regression. Turns out they were not, as one random forest regressor will take the same variables into account mostly the same way as a slightly different random forest regressor.

Findings

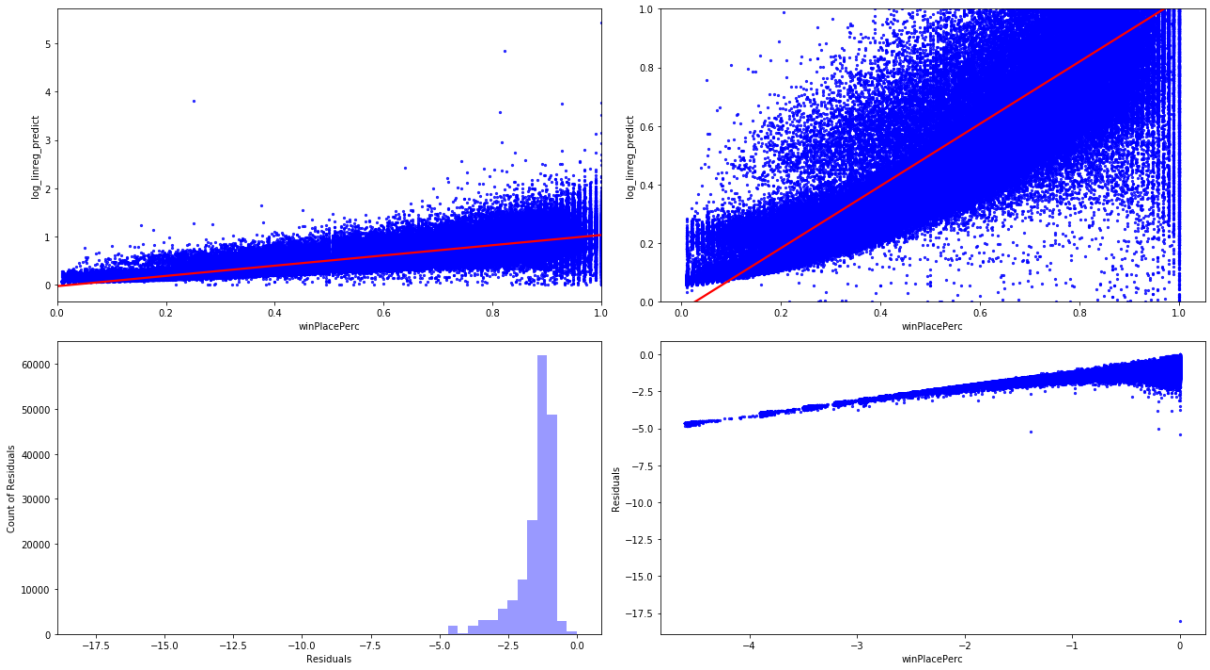
Model	R^2 (test)	RMSE (test)	95% Min Resid	95% Max Resid
Linear	0.88	0.1033	-0.2056	0.2011
Lasso	0.86	0.1098	-0.2611	0.2727
Ridge	0.88	0.1033	-0.2056	0.2011
Random Forest	0.95	0.6233	-0.1226	0.1267
XGBoost	0.96	0.6011	-0.1189	0.1245

Linear Regression

I got some interesting results from the linear regression portion. The linear regression graphs seen below turned out well and had a strong linear correlation between the predicted and actual values with an R^2 value of .88.



However, the model made predictions above 1 and below 0, which is not technically possible in this case, as the win place percentage values are only between 0 and 1. After taking the log of the data and exponentiating the predictions, the model gave me graphs that looked like those below, not great for what we were looking for.

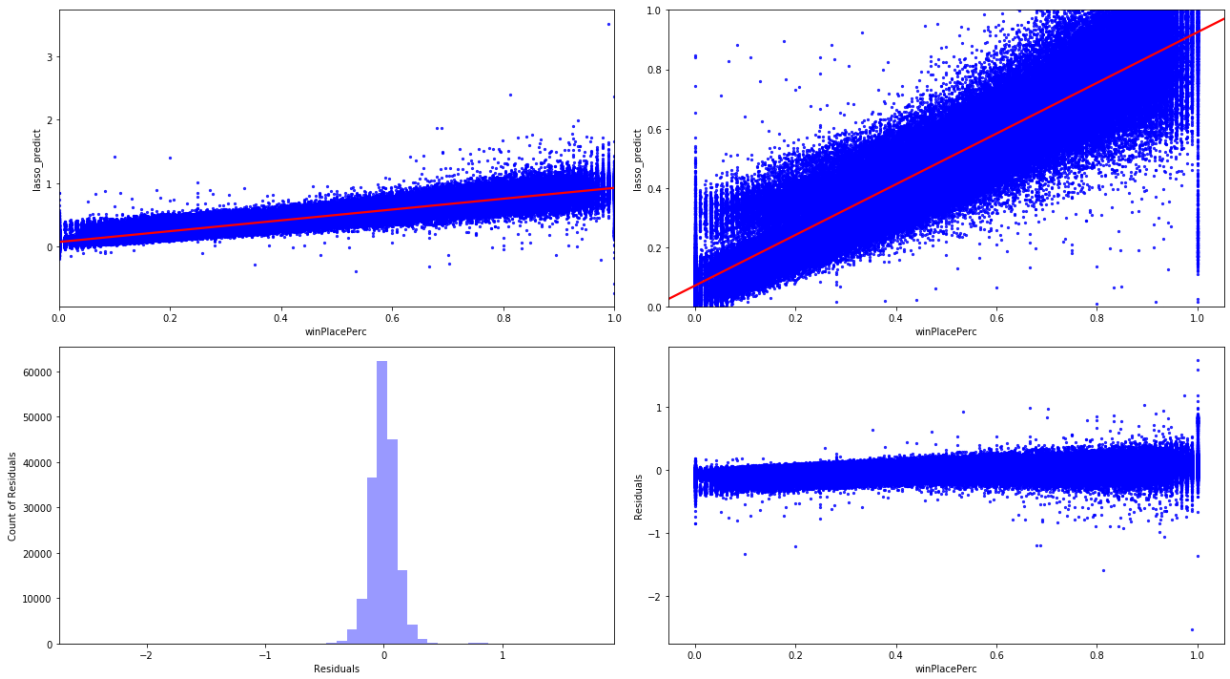


With an R^2 value of only .76, it turned out that was not the right approach. Fortunately I had two other models I wanted to try next.

Also with linear regression I took a look at how each variable was affecting the model through the coefficient attribute of the model. Strangely enough road kills was the coefficient that had the largest positive effect on the predictions. For every kill someone got using a vehicle, their win place percentage was predicted by this model to rise by about .035. On the flip side, kill streaks had the largest negative coefficient at close to -.2, so as a person quickly racked up consecutive kills, their win place percentage was predicted to drop by -.2.

Lasso Regression

Lasso regression turned out pretty similarly to the plain Linear regression model, however performed a bit worse at .86 R^2 compared to .88 R^2 of the Linear model. Looking at the graphs, Lasso regression suffered from the same problem as above, the model was making

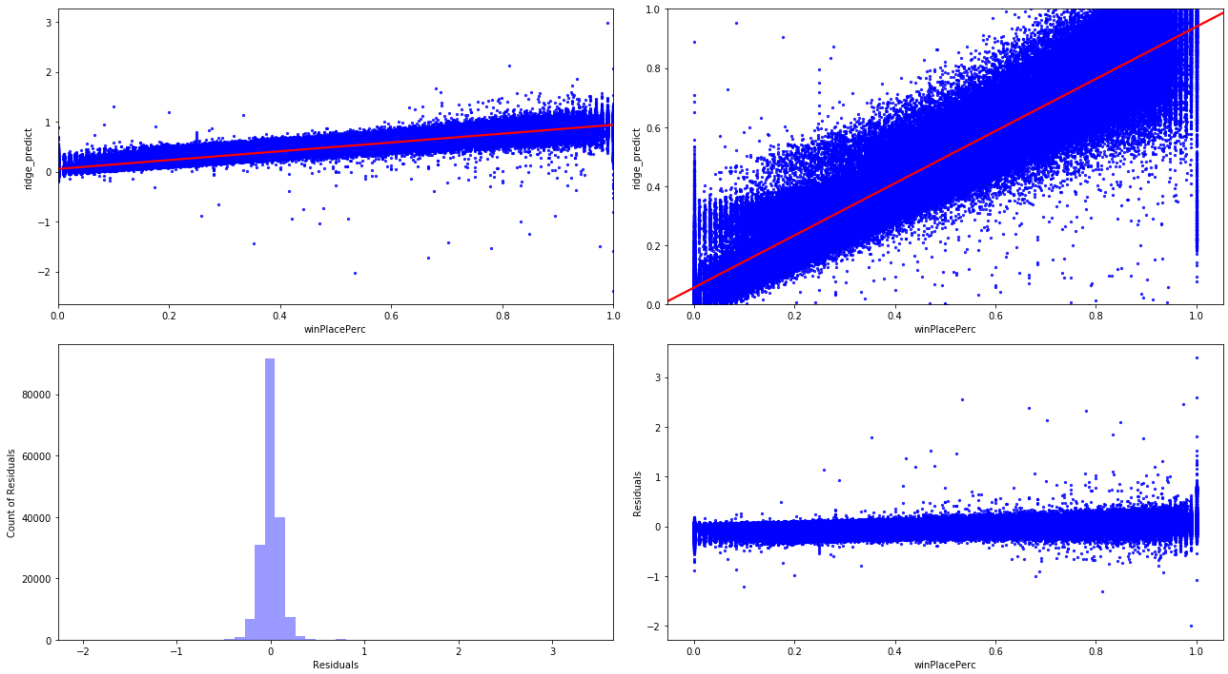


predictions of above 1 and below 0 which unfortunately does not solve the problem. I know it is not supposed to solve the problem of setting bounds on the predictions, but testing it was something worth doing

Taking a look at the coefficients of the Lasso model, it seems a new variable has the largest positive coefficient: weapons acquired at .016. Kill streaks still has the largest negative coefficient at -.066.

Ridge Regression

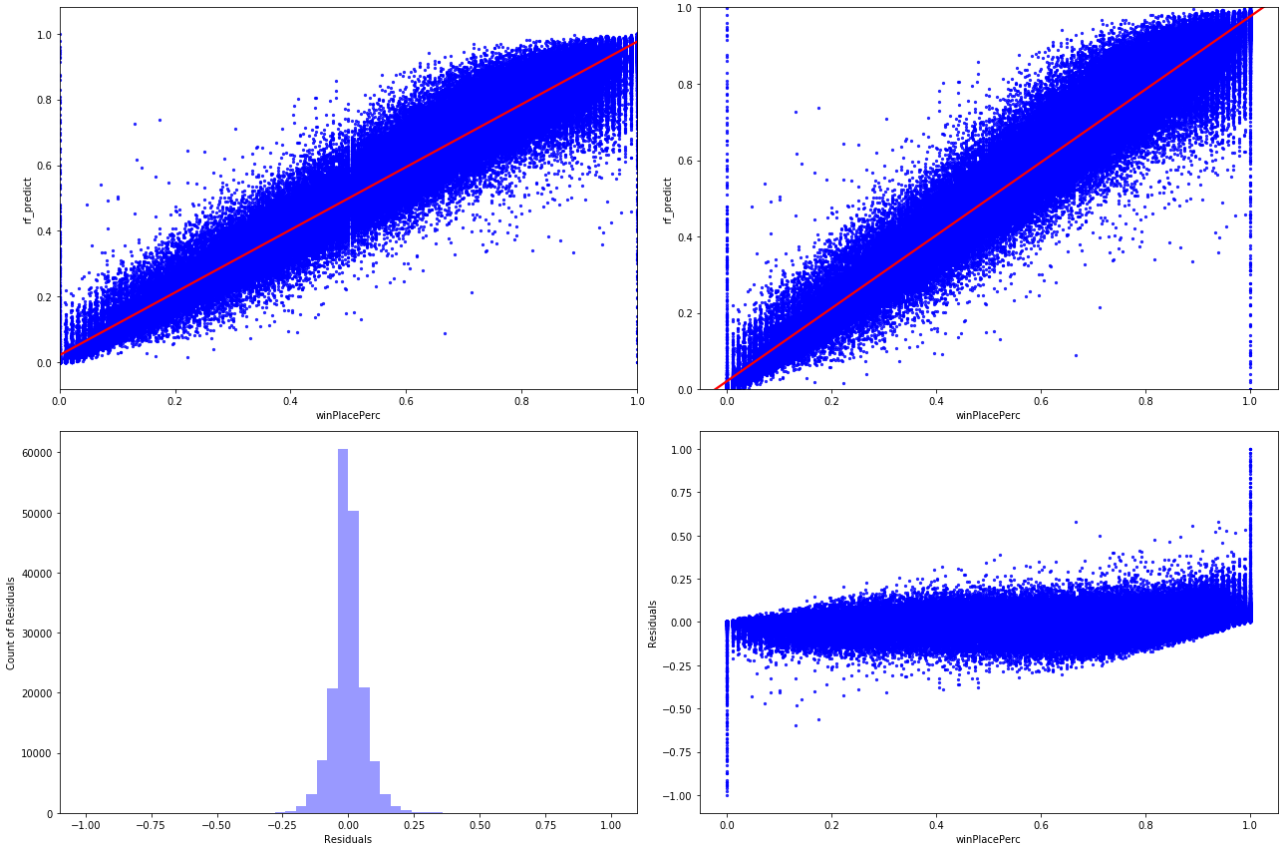
Through RIdge Regression I obtained results that are very similar to those reported by the baseline regression model of a .88 R^2 value on the training and testing data and the same largest positive and largest negative coefficients, road kills and kill streaks respectively, and looking at the graphs, still did not solve the problem of predictions going above 1 and below 0.



Finally, I took a look at the Mean Squared Error and Root Mean Squared Error for all three models, and those values backed up what the accuracy scores told us. Linear and Ridge regression were neck and neck in terms of testing set accuracy at 0.10332 RMSE for both, with Lasso regression falling just behind at 0.10983 RMSE.

Random Forest Regression

For a more in-depth approach, I decided to take a look at Random Forest regressors and see how they performed on my data. Turns out they did very well with a .95 R^2 value on the testing data after being trained with 100 trees and a max depth of 20. The graphs for Random Forest look fantastic, they do not go above 1 nor below 0, which has solved the problem I was having with linear regression. Even though the hyperparameter tuning took a while, it has paid off with the most accurate model I have gotten so far.

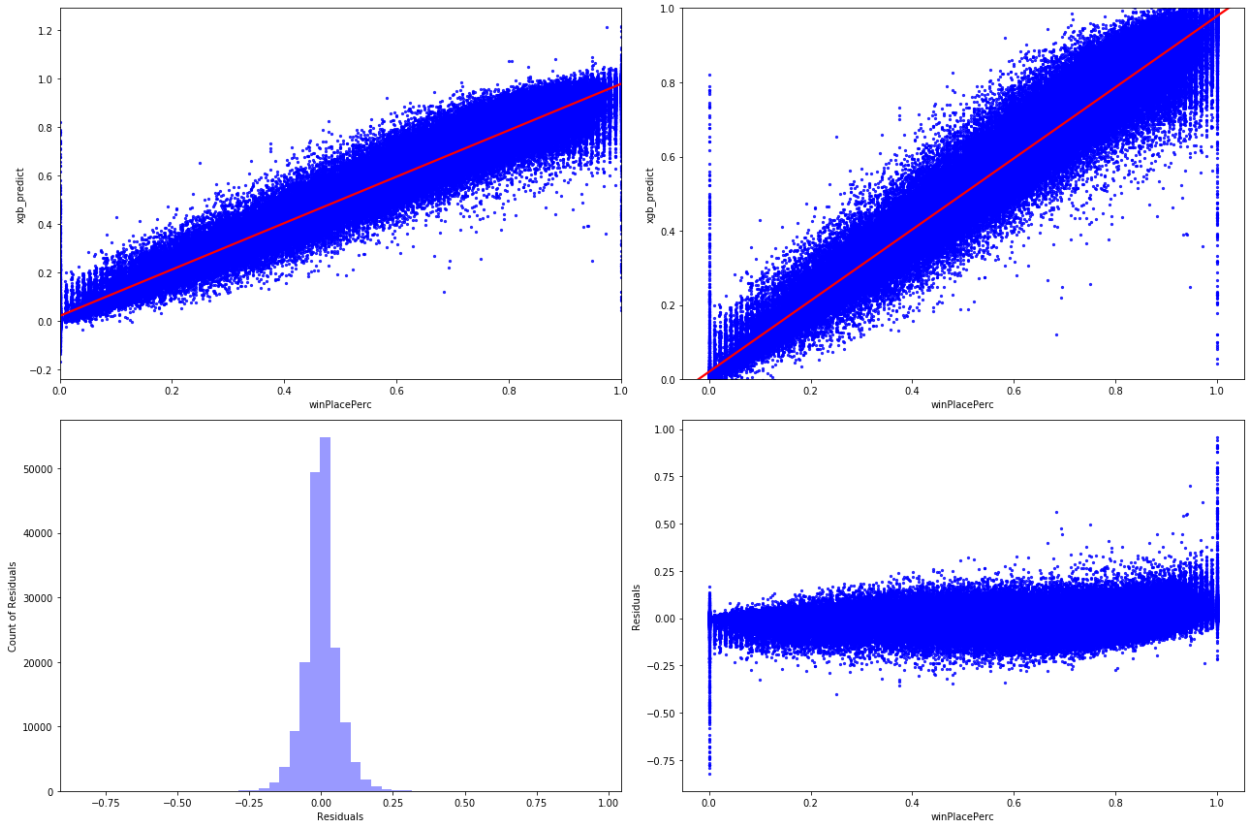


Taking a look at feature importances, it seems like this model really weighted walking distance above all else, as that was far and away the most important feature in the model at almost 80% importance. This was an interesting find as I did not think walk distance would be very important, whereas I thought kills, kill place, or damage dealt would be the most important features for these models. Next I took a look at the maximum feature parameter for Random Forest regression and tested three different max features: auto, sqrt, and log2. In a for loop I set the parameter in the model, fit, predicted, and took the score of the three different models. It turned out auto (taking the most number of features into account) gave the best R^2 value compared to the other two, as seen in the table below.

Max Features	Training R^2	Testing R^2
auto	.9836	.9565
sqrt	.9772	.9544
log2	.9772	.9542

XGBoost

XGBoost was the final method tested, with equally good results to the Random Forest regressor.



First thing is the graphs. It seems to have not taken into account the bounds on win place percentage, as there are predicted values above 1 and below 0. In that regard, the Random Forest regressor from sklearn is a better model. However, looking at the R^2 value,

XGBoost performed a little bit better than the Random Forest model with a .96 R^2 on the test data. Taking a look at feature importance next, it seemed like it fell in line with the Random Forest model in that it heavily relied on the walking distance of a player to make its predictions. At about 77% importance, it was far and away the most important feature of the data set; just like it was with the Random Forest model.

I noticed as the graphs were being made that all of these scatter plots have the same feature of large tails on them at either 0.0 or 1.0. I did some research and was not able to come to a solid conclusion as to why these tails were being produced. Potentially something to look into as future work.

Residuals

Model	95% Min Residual	95% Max Residual	Residual Range
Linear	-0.2056	0.2011	0.4067
Lasso	-0.2611	0.2727	0.5338
Ridge	-0.2056	0.2011	0.4067
Random Forest	-0.1226	0.1267	0.2493
XGBoost	-0.1189	0.1245	0.2434

Taking a look at the testing set residuals of all the models tested, it looks like XGBoost is proven as the best of the models with the smallest range of residuals. The smaller the range of residuals, the better the model performs since it is more accurate to the data. It just beats out Random Forest regression as the most accurate model, as seen here with only a .0059 point difference between the two ranges.

Conclusions

After looking at 5 different models, and coming up with predictions and coefficients for all of them, it seems like the best way to win in PUBG is to keep moving and not stay in one place for too long. In the Linear regression models, the coefficients that came out on top were road kills (Linear and Ridge regression) and weapons acquired (Lasso regression). These are all attributes of someone moving around on the map, either driving a car to kill people with or jumping from building to building to find their preferred weapon loadout. In the Random Forest regression and XGBoost, the top coefficient was walking distance for both models, with kill place coming in second for the Random Forest regressor and third behind boosts for the XGBoost model. Seeing as these were the more accurate models, this tells me that walking around and killing people are more likely to get someone to the end of the game instead of camping in a building and waiting for people to come to them.

Future Work

- Explore why walk distance is important in the Random Forest regression model and not as key in the linear regression model.
    - This could provide key insight into keeping players engaged in the game
    - If they stay in the game longer, they are more likely to finish the game with a higher placement
    - More distance walked equals more time spent in a game (for the most part)

- Further investigation should be done on the tails on the ends of the actual vs. predicted scatter plots.
    - At winPlacePerc values of 0 and 1, it appears that large tails where predictions are either far above 0 or below 1.
    - It is likely that this is a result of how the models are built from bagging, and while it does not seem to be many data points, those tails are prominent and should be explained
    - This would give additional insight to the impact when a good player has a bad day or a bad player has an amazing game

Recommendations for the Client

- Incentivise player movement
    - Players have more of a chance to win the more they find each other
    - Ways to incentivise movement:
        - Increase blue circle closing speed
        - Increase car spawn rates
        - Increase red zone frequency and lethality