Rocket League 3v3 Win/Loss Prediction on Tertiary Stats

By Nick Hazard

Obtaining the Data

- Third Party website, Ballchasing.com, aggregates Rocket League (RL) data and provides an public API
- Queried the Ballchasing API for 100,000 RL games
- Resulting 600,000 x 63 dataframe

```
(599706, 63)
```

Predicting Wins

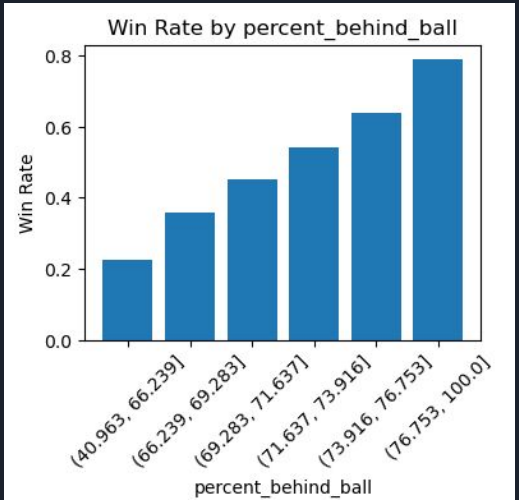
- Is it possible to predict RL game outcomes via only tertiary stats?
- How accurate can a model be?
- Examples of NON-tertiary stats:
  - Goals
  - Shots
  - Saves
  - Assists
- Example of tertiary stats:
  - Boost Used per Minute (BPM)
  - Boost Collected per Minute (BCPM)
  - Percent Time Spent in Offensive Third
  - Percent Time Spent Behind the Ball

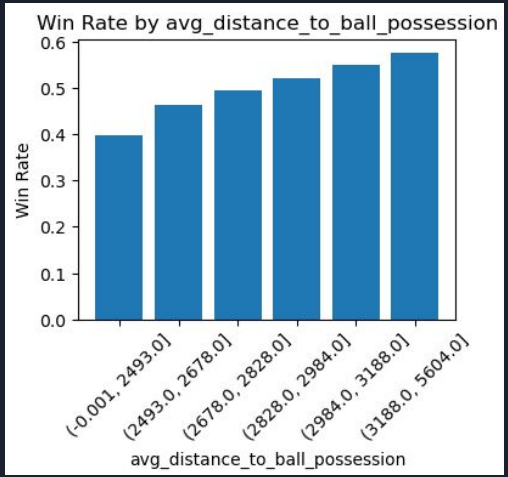
```
feature_cols = ['bpm', 'bcpm', 'amount_stolen', 'percent_zero_boost',
       'percent_full_boost', 'avg_speed', 'percent_boost_speed', 'percent_high_air',
       'avg_distance_to_mates', 'percent_offensive_third',
       'percent_neutral_third', 'percent_behind_ball', 'avg_distance_to_ball_possession']
```

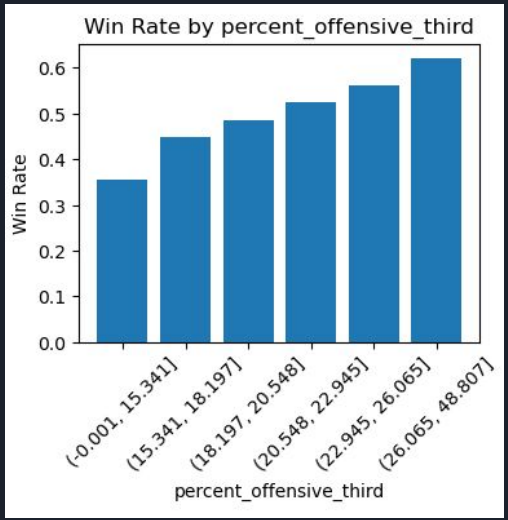
EDA Insights

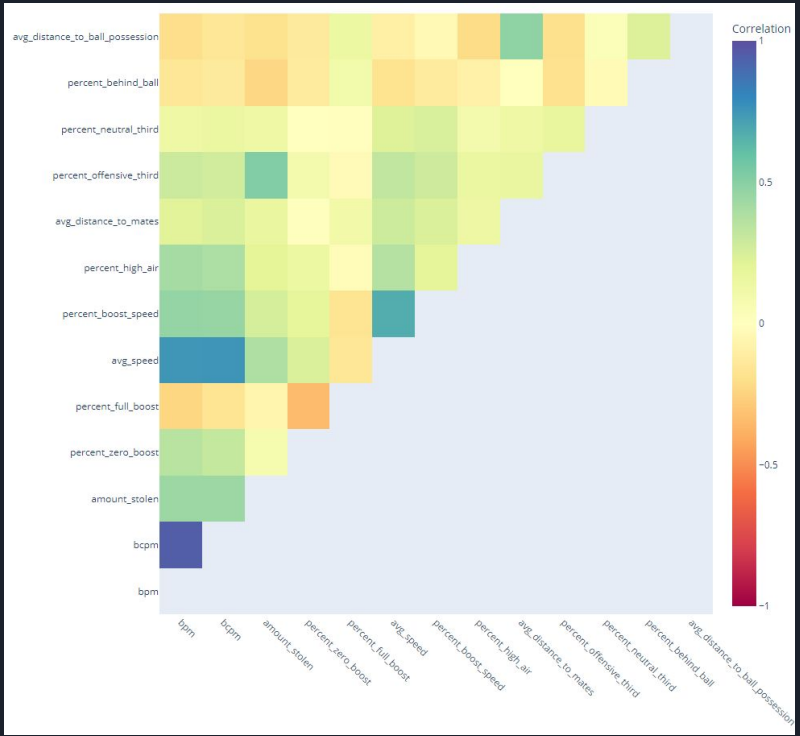
- Using the version of stats with percentages instead of straight numeric values is valuable due to the possibility of forfeits in RL games
- Lots of highly UNcorrelated stats, such as boost stolen, boost overfill, count collected big/small, etc.

Most highly correlated stat to wins was ‘percent_behind_ball’




Win Rate by avg_distance_to_ball_possession
Win Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
(-0.001, 2493.0]
(2493.0, 2678.0]
(2678.0, 2828.0]
(2828.0, 2984.0]
(2984.0, 3188.0]
(3188.0, 5604.0]
avg_distance_to_ball_possession


Win Rate by percent_offensive_third
Win Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
(-0.001, 15.341]
(15.341, 18.197]
(18.197, 20.548]
(20.548, 22.945]
(22.945, 26.065]
(26.065, 48.807]
percent_offensive_third


Correlation
1
0.5
0
−0.5
−1
avg_distance_to_ball_possession
percent_behind_ball
percent_neutral_third
percent_offensive_third
avg_distance_to_mates
percent_high_air
percent_boost_speed
avg_speed
percent_full_boost
percent_zero_boost
amount_stolen
bcpm
bpm

Modeling

- Used 5 different ML models:
    - Logistic Regression
    - Random Forest
    - XGBoost
    - LightGBM
    - CatBoost
- LightGBM performed the best and was quick to train
- NN and SVM were both not any better at predicting outcomes and took much longer to train

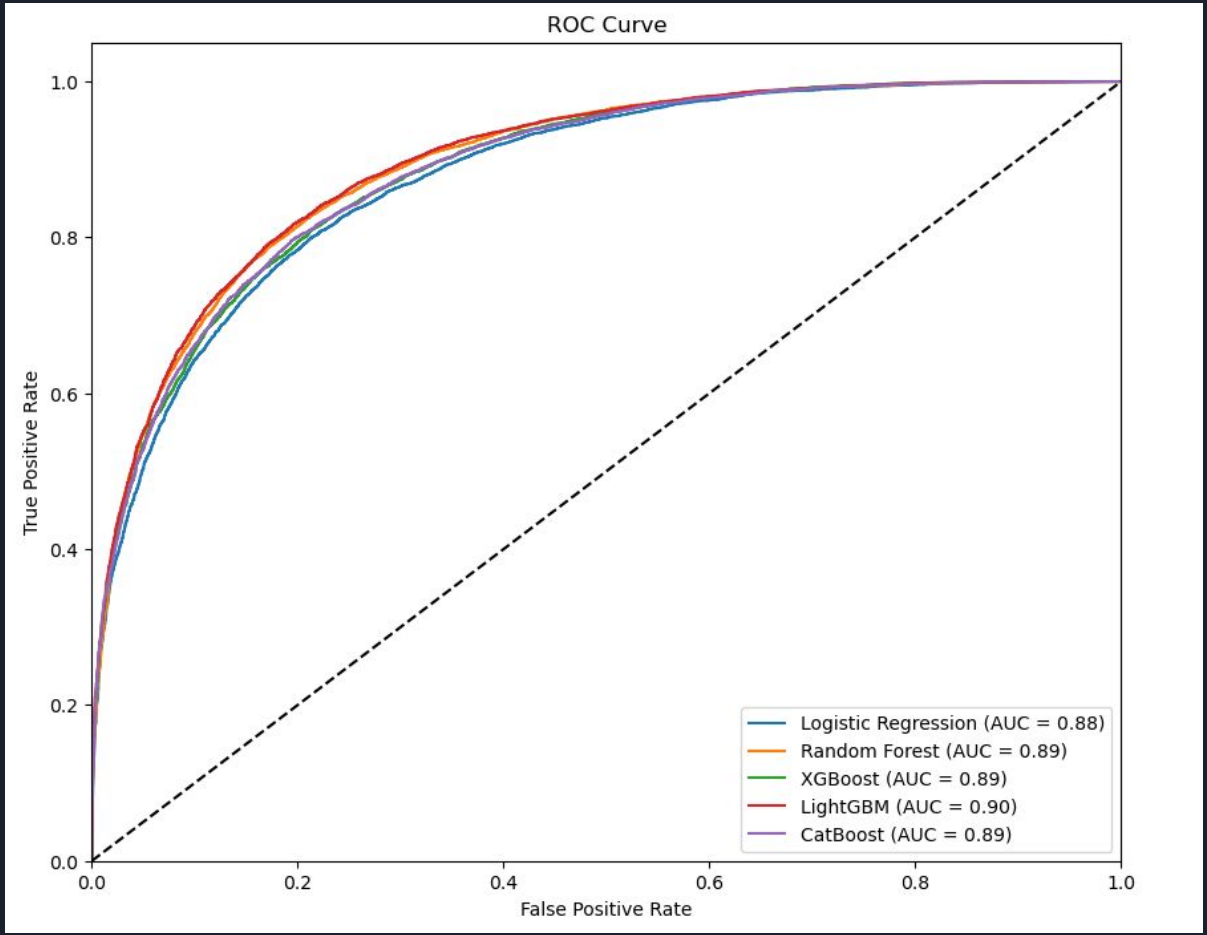
```
LightGBM Results:
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 300, 'num_leaves': 127}
Best Cross-validation Score: 0.8101
Validation Score: 0.8142
Test Score: 0.8096
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81     10112
           1       0.81      0.81      0.81      9879

    accuracy                           0.81     19991
   macro avg       0.81      0.81      0.81     19991
weighted avg       0.81      0.81      0.81     19991

Confusion Matrix:
[[8207 1905]
 [1902 7977]]
```

```
Logistic Regression Results:
Best Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
Best Cross-validation Score: 0.7955
Validation Score: 0.7948
Test Score: 0.7926
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     10112
           1       0.79      0.79      0.79      9879

    accuracy                           0.79     19991
   macro avg       0.79      0.79      0.79     19991
weighted avg       0.79      0.79      0.79     19991

Confusion Matrix:
[[8025 2087]
 [2060 7819]]
```

```
XGBoost Results:
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}
Best Cross-validation Score: 0.7999
Validation Score: 0.8006
Test Score: 0.7972
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80     10112
           1       0.79      0.79      0.79      9879

    accuracy                           0.80     19991
   macro avg       0.80      0.80      0.80     19991
weighted avg       0.80      0.80      0.80     19991

Confusion Matrix:
[[8086 2026]
 [2028 7851]]
```

```
Random Forest Results:
Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 300}
Best Cross-validation Score: 0.8056
Validation Score: 0.8102
Test Score: 0.8071
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81     10112
           1       0.81      0.80      0.80      9879

    accuracy                           0.81     19991
   macro avg       0.81      0.81      0.81     19991
weighted avg       0.81      0.81      0.81     19991

Confusion Matrix:
[[8210 1902]
 [1954 7925]]
```

```
CatBoost Results:
Best Parameters: {'depth': 8, 'iterations': 300, 'learning_rate': 0.1}
Best Cross-validation Score: 0.8018
Validation Score: 0.8033
Test Score: 0.8005
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80     10112
           1       0.80      0.80      0.80      9879

    accuracy                           0.80     19991
   macro avg       0.80      0.80      0.80     19991
weighted avg       0.80      0.80      0.80     19991

Confusion Matrix:
[[8097 2015]
 [1973 7906]]
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (AUC = 0.88)
Random Forest (AUC = 0.89)
XGBoost (AUC = 0.89)
LightGBM (AUC = 0.90)
CatBoost (AUC = 0.89)

Conclusion & Next Steps

- Although unable to increase model accuracy much, training time was greatly reduced with similar results
- Possible upper limit to how well these sort of tertiary stats are able to predict game outcomes
- Real world implementation:
  - Wrapping the LightGBM model in a basic web app
  - Going to give this tool to 9Moons commentators so they can input players stats and discuss the model's prediction for more talking points
  - Apply what I have learned to a new problem with more real-world use cases