Analysis of Rocket League Replay Metrics: Machine Learning Classification and "Game Performance" Equation

Andrew J. Otis

Ritchie School of Engineering and Computer Science, University of Denver

Data Science Tools II

Professor Donald Dalton (Don)

June 2, 2023

Introduction

Analysis of Rocket League Replay Metrics: Machine Learning Classification and "Game Performance" Equation

Purpose of the Project

Sports analytics is nothing new, but what is relatively new in the industry of Professional Athletics would be E-sports, bringing a wide variety of data. Every video game will have data in common, but there's also a lot of data unique to each individual game (*e.g. baseball vs basketball*). I will be comparing various classification algorithms to see which one is most appropriate to the data.

Significance of Project

As seen on T.V. sports analytics are a big part of what makes up the "experience". Insights from game data can be advantageous to all those involved, from players to officials, and even game developers.

Research Question(s)

- What Machine Learning models would be appropriate for predicting whether a player is mvp or not?
- Can an equation be derived to calculate what will be called "game performance" to predict whether a player is mvp or not?

Description of the Dataset

The dataset consists of multiple web scrapes from the site https://ballchasing.com [3]. Any player with an online account from a supported platform can post metrics of their game, this includes professional and casual games. The features of the data set include almost every data type, including location data and columns that have multiple categories for a single response. In essence the main dataset has the dimensions (201 rows x 136 columns).

Data Preprocessing

Data Preparation

The data was scraped from via opensource API where various columns had responses that is a list containing a list of dictionaries [[{dic}, {dic2}, {dic3}, …]]. First, the data pull was for a single game's replay metrics,

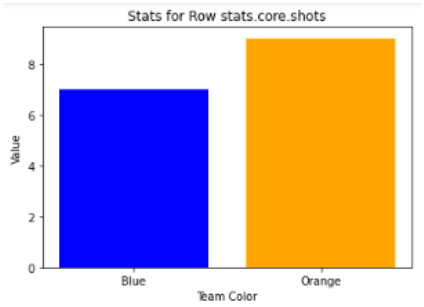
building up to a more complex set of functions that perform the same task, but for the replay metrics of multiple games, from which a single data frame is constructed. Where it made sense, null values were set to zero or removed completely. For instance, overtime in some rows had a null response when in reality all it was that these players were part of a match that never made it to overtime. The predictor variable's column was Label Encoded since the analysis is a binary classification problem and sklearn ML models don't like strings as response variables. This was the bulk of transformations performed, additional cleaning operations done on the train and test set, since by the time the player data is isolated, new columns have popped up due to some columns having a response with multiple categories.

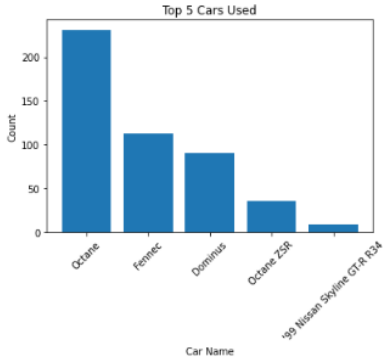
Exploratory Data Analysis

First, descriptive statistics of all the data's numeric columns were collected, then general team statistics for each game were isolated and then explored via an indexable list of tables. These tables contained some stats any sports fan would be familiar with as well as data unique to Rocket League itself. Then from its own column consisting of a list of dictionaries, player data is extracted and viewable in its own data frame.

Visualization

A bar plot is constructed from the general team stats for a non-table visual as well as the player stats to create another bar plot to determine which care was utilized the most





Data Splitting

A train and test set were created from the main dataset using a 70-30% split ratio.

Model Building and Evaluation

Model Building

Data assumptions for a logistic classification model were checked. The assumption for a binary output variable has already been met so first, a scatterplot of all possible input variables against the output variable to visualize if any features had a linear relationship with the output variable. Then a check for independence of observation with a correlation matrix was done. Which is then followed by a check of Multicollinearity using a statsmodels package.

Check Assumption of Linearity Between Features and Predictor

```
import seaborn as sns

# Concatenate X and y_encoded into a single DataFrame
data = pd.concat([X, pd.Series(y_encoded, name='encoded_mvp')], axis=1)

# Iterate over each independent variable
for column in X.columns:
    # create a scatter plot with a logistic regression line
    sns.lmplot(x=column, y='encoded_mvp', data=data, logistic=True, height=5)
    plt.title(f"Relationship between {column} and Log-Odds of MVP")
    plt.show()
```

Check Assumption Independence of Observations

```
# Calculate the correlation matrix
correlation_matrix = X.corr()

# Set the threshold for correlation
threshold = 0.8

# Print the highly correlated columns
highly_correlated_cols = set()
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            col1 = correlation_matrix.columns[i]
            col2 = correlation_matrix.columns[j]
            highly_correlated_cols.add((col1, col2))

# Print the highly correlated columns
for col1, col2 in highly_correlated_cols:
    print(f"{col1} and {col2} are highly correlated.")

# Add a space between print statements
print()

# Plot the correlation matrix as a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")

# Show the plot
plt.show()
```

Check Assumption of MultiColinearity Between Independent Variables

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Remove null and None values from X
X_cleaned = X.dropna()

# Create a DataFrame with the independent variables
independent_variables = X_cleaned

# Calculate the VIF for each independent variable
vif = pd.DataFrame()
vif["Variable"] = independent_variables.columns
vif["VIF"] = [variance_inflation_factor(independent_variables.values, i) for i i

# Filter variables with VIF greater than or equal to 5
high_vif_variables = vif[vif["VIF"] >= 5]

# Filter variables with VIF less than 5
low_vif_variables = vif[vif["VIF"] < 5]

# Print the variables with high VIF
print("Variables with VIF greater than or equal to 5:")
print(high_vif_variables)

# Print the variables with low VIF
print("Variables with VIF less than 5:")
print(low_vif_variables)
```

Finally, the models are trained with the training data and then used to make predictions on test sets.

Model Selection and Optimization

Hyperparameters for each model are adjusted to see if the classification model would perform better than default parameters. In one case, tuning resulted in a model performing worse.

Model Evaluation

```
---Logistic Classifier Accuracy w/ default parameters---
Accuracy on training set: 0.7756756756756756
Accuracy on test set: 0.7975460122699386

---Logistic Classifier Accuracy w/ optimum parameters---
Accuracy on training set w/ optimum parameters: 0.9945945945945946
Accuracy on test set w/ optimum parameters: 0.8679245283018868

---Random Forest Classifier Accuracy w/ optimum parameters---
Accuracy on training set w/ optimum parameters: 0.6459459459459459
Accuracy on test set w/ optimum parameters: 0.5398773006134969

---Decision Tree Accuracy w/ default parameters---
Accuracy on training set: 0.7756756756756756
Accuracy on test set: 0.7975460122699386

---Decision Tree Accuracy w/ optimum parameters---
Accuracy on training set w/ optimum parameters: 0.7756756756756756
Accuracy on test set w/ optimum parameters: 0.7975460122699386
```

Conclusion

Concluding Thoughts

The equation for game performance was determined as follows, resulting in an accuracy score of about 54%.

$$game\ performance\ = 0.9 * assists + 0.1 * goals\_against\_while\_last\_defender$$

In conclusion, it was determined that the decision tree model was best suited for working with the data set and goals of the analysis. This is due to the decision tree being the only model that did not overfit the data

The random forest and logistic models did not perform well, indicated by overfitting the data, but the decision tree model had a decent performance on new data and the model did not overfit the data.

Lessons Learned

However, the process of model building allowed me to identify features that affected the prediction the most so the efforts were not fruitless. Additionally, when collecting data from a source that updates frequently, it is important to consider how that might affect one's analysis because new data that was not originally considered could be introduced in place of data scraped the day before

Recommendations

It is recommended that the data be saved at various stages of the program so that an individual would not have to re-scrape the data in order to run the analysis. This not only saves time but maintains consistency in findings as the model is built and tested due to the source site updating whenever someone decides to post their replay metrics.

References

Image

1. Nickolas Carroll, (2021, March 17), How to Watch Rocket League, esports.missouri.edu. https://esports.missouri.edu/how-to-watch-rocket-league/
2. lethamyr.com, (n.d.) Top Down Rocket League, lethamyr.com. https://lethamyr.com/mymaps/top-down-rocket-league

Online

3. Ballchasing.(n.d.). Ballchasing.com. Retrieved May 20, 2023, from https://ballchasing.com
4. Ballchasing.(n.d.). API Documentation. Ballchasing.com. Retrieved May 20, 2023, from https://ballchasing.com/doc/api
5. Ballchasing.(n.d.). Frequently Asked Questions (FAQ). Ballchasing.com. Retrieved May 20, 2023, from https://ballchasing.com/doc/faq
6. Hockey-Graphs. (2016, July 13). Measuring Single-Game Productivity: An introduction to Game Score. Hockey-Graphs. Retrieved May 20, 2023, from https://hockey-graphs.com/2016/07/13/measuring-single-game-productivity-an-introduction-to-game-score/

Course

7. Dalton. (2023). Data Science Tools II [COMP4448]. University of Denver.