# F1 Race Outcome Prediction

Joshua Lee Nicholas Pfeifer

## Table of contents



## 1 STAT4255 Final Project: F1 Race Podium Forecasting

The Formula 1 dataset we used has 72 features where each observation represents the results of 1 driver in a specific race. For example, if a race has 20 participating drivers, then there will be 20 records associated with that specific race. Many of the features are related to characteristics of a track such as the average speed, the maximum speed, the longest straight length, the number of corners, etc. Other features tell information about the track conditions such as temperature, humidity, precipitation, and windspeed. Lastly, some of the most important features have to do with how a driver or team has performed in previous races. These features convey performance through how many points or wins a driver or team has accumulated in a span of prior races.

The main question we wanted to tackle was determining what training size is best for predicting podium (top 3) finishes for each driver. To accomplish this we had to fit models for various training sizes (also called windows) of prior races and then compare how well they performed in classifying an observation as a podium finish or not in the next race. This question was especially interesting because in previous analyses, overfitting to the training data was a common source of error. F1 teams are capable of significantly upgrading their cars from race to race which can be difficult for models to adapt to. The aim of this question was to pinpoint

the window size for which the training data is most most relevant. A model with a window size of just a handful of races (say 2 or 3) may not have enough information to make accurate predictions and put too much weight in a small sample of results. On the other hand, a model with a large window size (15 or 20 races) could be basing its predictions off of noisy data that is no longer representative of the current competition. We wanted to know what window size is the "sweet spot" for making the most accurate predictions.

## 2 Preliminary Methods and Approach

There were many approaches that we considered in attempting to make the most accurate predictions.

The first step of the project was creating a function that could accept a specified method, and return the results of training on data windows of size up to a specified value of $n$ (See our code for more specific details of this implementation models_by_window.py). This function was successful and laid the foundation for comparing different variable selection and modeling processes. The function works by iterating over every combination of round, year, and $n$ selected for the fitting procedure, and then trains an XGBoosting model over the $n$ previous races to predict podium finish outcomes on the next race in sequence. After predictions are made for each race, the function summarizes performance metrics for each value of $n$. The predictions are classified by assigning the drivers with the 3 highest probabilities derived from the model to the podium class and the rest to the non-podium class. The metric that we thought was most indicative of performance was the average F1 score. F1 score is calculated by $\frac{2*P*R}{P+R}$ where P is the precision (the accuracy of podium predictions) and R is recall (the true positive rate). An F1 score greater than 0.5 indicates that the model is better than random guessing. F1 score was better than using accuracy since accuracy would naturally be pretty high for every model with the worse case of getting 14/17 non-podium predictions correct.

Once we had the ability to see how F1 score varied for each value of n we immediately saw a trend starting to form. As n increased, F1 score would generally increase up to a certain point. After reaching that point, larger values of $n$ typically yielded the same or lower F1 scores. These were encouraging results as it appeared there was a "sweet spot" value of $n$.

The next step was determining what type of model and feature selection was the best.

The first type of model that we fit was a logistic regression model with manually selected features. This approach did not yield high F1 scores. One issue that we ran into was encoding interaction terms.

We also experimented with decision trees. The decision trees showed some promising results and could generate high F1 scores for individual races. Unfortunately, the decision trees were prone to overfitting which limited their effectiveness.

At this point we wanted to try out automatic feature selection. This was done using XGBoost. The thought here was also that the models could benefit from interaction features implemented by XGBoost. However, this approach did not perform well either. After fitting XGBoost over a series of data window sizes, we found that a model which included all of the features typically performed worse relative to a model with far fewer features. This indicated that features were not adequately being selected, since their effect could have been ignored in a full model if they were.

Additionally, we observed higher F1 scores after implementing synthetic minority oversampling or SMOTE. Up to this point, our classification problem had been imbalanced. Only 3/20 or 15% of results were positive (podium) classifications for each race. This means that our models were biased toward negative (non-podium) classifications. By using SMOTE, the minority (positive) classifications are randomly resampled from the minority distribution until the number of positive and negative classifications are more balanced. This balanced classification technique improved performance by 15-20% in F1 score.

## 3 Selected Approach and Methodology

This process ultimately led us to our final approach which was using an xg boosting regressor for predicting the classification value (0 or 1 - the result was the mean value of 0 and 1 for each region selected by the regressor - which amounts to the probability of a positive classification). In hindsight, we could have also used an XGB classifier with predict probabilities to rank the top 3 most likely podium finishers and classify podium finishes in that way. Then, we manually selected features by running models for each feature one at a time, checking if they improved the average F1 score of predictions over the 2024 race result data in addition to features which had performed best previously. The features we ultimately settled on are listed in the results section of this report.

Additionally, we utilized a variation of SMOTE (SMOTE-NC) to improve the balance of the training data distribution. This helped to improve the performance of our methods as seen by the resulting 20% improvement in F1 scores compared to not using this technique.

Our experimental evaluations led us to determine that 6 to 8 races was the optimal number of races to train on for predicting the results of the next race in sequence. Additional information about these methods can be seen in the results section.

## 4 Experimental Results

**Parameterizing the training data window size**

In order to validate our approach, we performed several experiments over the data. First, we thoroughly investigated the performance of the XGBoost model over 2024 F1 race data (races spanning from Bahrain 2024 to Qatar 2024) for a fixed selection of features, namely:

- The proportion of points in the standings belonging to each driver before a given race
- Whether or not a driver drove for McLaren
- Whether or not a driver drove for RedBull
- Whether or not a driver was Charles Leclerc
- The maximum track speed for a given circuit
- The number of previous constructor wins
- The number of podiums scored by a given driver over the previous 2, 3, and 4 races

Using these features, we fitted XGBoost models over data window sizes of 2 to 11 rounds. Because XGBoost's fitting algorithm is stochastic, we needed to perform multiple runs to generate a confidence interval over the true F1 score of the model on the data. From a series of initial runs, we determined that 7 races was the strongest data window size, and then ran 15 iterations to estimate the standard errors of this estimate. Using CLT, we can approximate the distribution of mean F1 scores for training on 7 races of data by a normal distribution (as long as the sample size is sufficiently large).

Namely, for training an XGBoosing model over data from the previous 7 races, we observe an average F1 score of 0.536 over the next race in sequence (for all races from Bahrain to Qatar). The standard deviation of average F1 scores achieved by these models was 0.0072, and so a 95% confidence interval for an estimate of the average F1 score achieved by XGBoost trainged over the prior 7 races of data in 2024 was

$$[0.5224, 0.5506]$$

This is a nice bound, since it means that our XGBoost model typically performs well for race data windows of size 7 (better than random guessing). To see that seven is the best choice, the results for window sizes from 2 to 11 can be seen below:

```
import pandas as pd

res = pd.read_csv("../code/experiments/results_window_sample.csv")
res
```

| | n | Accuracy | F1 Score | Acc_Max | Acc_Min | F1_Max | F1_Min |
|---|---|---|---|---|---|---|---|
| 0 | 2.0 | 0.837594 | 0.460317 | 1.0 | 0.700000 | 1.000000 | 0.000000 |
| 1 | 3.0 | 0.842356 | 0.476190 | 1.0 | 0.700000 | 1.000000 | 0.000000 |
| 2 | 4.0 | 0.870927 | 0.571429 | 1.0 | 0.700000 | 1.000000 | 0.000000 |
| 3 | 5.0 | 0.856642 | 0.523810 | 1.0 | 0.789474 | 1.000000 | 0.333333 |

| | n | Accuracy | F1 Score | Acc_Max | Acc_Min | F1_Max | F1_Min |
|---|---|---|---|---|---|---|---|
| 4 | 6.0 | 0.866416 | 0.555556 | 1.0 | 0.800000 | 1.000000 | 0.333333 |
| 5 | 7.0 | 0.856642 | 0.523810 | 0.9 | 0.789474 | 0.666667 | 0.333333 |
| 6 | 8.0 | 0.847118 | 0.492063 | 1.0 | 0.789474 | 1.000000 | 0.333333 |
| 7 | 9.0 | 0.837594 | 0.460317 | 0.9 | 0.700000 | 0.666667 | 0.000000 |
| 8 | 10.0 | 0.832832 | 0.444444 | 0.9 | 0.700000 | 0.666667 | 0.000000 |
| 9 | 11.0 | 0.813784 | 0.380952 | 0.9 | 0.700000 | 0.666667 | 0.000000 |
| 10 | 12.0 | 0.823308 | 0.412698 | 0.9 | 0.700000 | 0.666667 | 0.000000 |

**The utility of track features**

On their own, track features are not explicitly useful. The reason for this is that for a given track the track features will be the same for all drivers. As such, if we do not identify the drivers in question, the podium outcomes will be randomly associated with the track features themselves. Thus, we include the driver and constructor information to interact with these variables to achieve improved performance outcomes.

Namely, we consider McLaren and Red Bull as teams and Charles Leclerc as a individual driver for interaction with track features. To determine if these features make a meaningful performance contribution, we compare baseline models which do and do not include track features. Unfortunately, we find that often times including track features actually diminished performance on the whole. Perhaps the reason for this was that different track features are differentially useful across a given race season, and so the average F1 score fitted on models for features that are not useful over all possible windows adds unecessary noise and diminishes performance.

**The utility of scaled point distributions**

To determine if scaling the point distribution was useful or not, we compared the performance of XGboost models fitted only on raw previous driver point totals and scaled previous driver point totals (driver scaled points = driver total points / number of points awarded to all drivers).

Below you can see the comparative performance of XGboost models trained over data from 2 to 9 of the previous races on the next race in sequence when scaled and unscaled driver point totals are given:

***Raw (Unscaled) point totals***

```
unscaled_pts = pd.read_csv("../code/experiments/un-scaled_points_sample.csv")
unscaled_pts
```

| | n | Accuracy | F1 Score | Acc_Max | Acc_Min | F1_Max | F1_Min |
|---|---|---|---|---|---|---|---|
| 0 | 2.0 | 0.775439 | 0.253968 | 0.9 | 0.684211 | 0.666667 | 0.0 |
| 1 | 3.0 | 0.785213 | 0.285714 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 2 | 4.0 | 0.794987 | 0.317460 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 3 | 5.0 | 0.799749 | 0.333333 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 4 | 6.0 | 0.799499 | 0.333333 | 1.0 | 0.700000 | 1.000000 | 0.0 |
| 5 | 7.0 | 0.785213 | 0.285714 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 6 | 8.0 | 0.780702 | 0.269841 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 7 | 9.0 | 0.771178 | 0.238095 | 0.9 | 0.700000 | 0.666667 | 0.0 |
| 8 | 10.0 | 0.780702 | 0.269841 | 0.9 | 0.700000 | 0.666667 | 0.0 |

***Scaled point totals***

```
scaled_pts = pd.read_csv("../code/experiments/scaled_points_sample.csv")
scaled_pts
```

| | n | Accuracy | F1 Score | Acc_Max | Acc_Min | F1_Max | F1_Min |
|---|---|---|---|---|---|---|---|
| 0 | 2.0 | 0.794737 | 0.317460 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 1 | 3.0 | 0.813784 | 0.380952 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 2 | 4.0 | 0.818546 | 0.396825 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 3 | 5.0 | 0.813784 | 0.380952 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 4 | 6.0 | 0.809023 | 0.365079 | 1.0 | 0.7 | 1.000000 | 0.0 |
| 5 | 7.0 | 0.813784 | 0.380952 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 6 | 8.0 | 0.828070 | 0.428571 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 7 | 9.0 | 0.799499 | 0.333333 | 0.9 | 0.7 | 0.666667 | 0.0 |
| 8 | 10.0 | 0.823308 | 0.412698 | 0.9 | 0.7 | 0.666667 | 0.0 |

As you can see, F1 scores improve substantially when using a scaled point distribution, indicating that this change was predictively beneficial. All of these methods and results can be replicated by cloning our repository: f1-fantasy-analysis

## 5 Conclusions

The results which we have gathered confirm that our hypothesis was correct. It can be observed that changing the amount of training data (changing n) has a differential effect on the performance of a model on the test data. In the context of Formula 1, this makes sense because we tend to see teams and drivers go through highs and lows throughout a season. Most of the time, the podium is not static from race to race.

We found that medium sized training windows (6 to 8 races) perfom better than small or large sized windows. The interpretation for this finding could be that a small training window is lacking suffient data while a large training window has too much data, some of which is no longer relevant.

In the future there are several parts of the analysis that could be improved. Firstly, getting an automated feature selection process working would eliminate the need for manual feature selection. This is important because it can assist with determining the optimal set of features for each race that we are training over. If each race has its own tailored features, model performance should increase. At the same time there is a risk of overfitting, but that is yet to be determined in future analyses. Additionally, there are countless more features that could be added such as qualifying results.