Predicting Podium Finishes in Formula 1

****

# Unsupervised Learners

Ben Ampel, Kee Han Ooi, Shruthi Murali, Stephanie Perez, Steven Ullman, Tyler Campbell

22 October 2019

Contents

[Introduction 2](#_Toc22920039)

[Problem Description 2](#_Toc22920040)

[Current State of the Domain 2](#_Toc22920041)

[Original Dataset Description 3](#_Toc22920042)

[Data Preprocessing 3](#_Toc22920043)

[Dataset Description After Preprocessing 5](#_Toc22920044)

[Intended Algorithms 6](#_Toc22920045)

[Preliminary Analysis 6](#_Toc22920046)

[Future Direction 9](#_Toc22920047)

[Appendix 12](#_Toc22920048)

## Introduction

Formula 1 is often considered the pinnacle of motorsports. With teams spending north of $400 M per race season, it is no surprise why. In addition to the monetary significance of the sport, teams rely heavily on advanced technologies and information systems. Every race weekend, roughly 2 TB of data is generated and pulled from each vehicle. It is this data component that initially drew us to the idea of exploring F1 problems (and available datasets) for our project.

## Problem Description

Throughout the Formula 1 season there is much strategization that goes on behind the scenes. Some examples of difficult decisions that teams are faced with are:

* Knowing which races to push harder in - i.e. when to prioritize vehicle preservation over performance
* Deciding when to swap engines - engines are used as long as possible, but are ideally changed before blowing and costing the driver a race
* Deciding which drivers to prioritize in a given race - e.g. have a driver sacrifice his position and/or provide a slipstream if the other driver is running a faster pace

All of the above decision scenarios could be elucidated by knowing the likelihood of race outcomes. For this reason, the team wanted to explore the possibility of predicting race finishes via traditional data-mining techniques, which would ultimately facilitate more effective team strategization.

## Current State of the Domain

There are a few examples of projects like this being attempted by others. The most prominent cases are those demonstrated on bookmaking/gambling platforms like Oddschecker[[1]](#footnote-1) and MyBookie[[2]](#footnote-2). These companies, besides just predicting race outcomes, also provide odds on fastest lap holder, pole position winner, season winner, etc. Obviously, these platforms do not solicit many details regarding the inputs and weightings that go into their algorithms.

An independent instance of this project being attempted is the blog site, F1 Predictor[[3]](#footnote-3), developed by F1 fan and data scientist, Asterios Stergioudis. The blog contains posts of predicted race outcomes as well as post-race sentiment analysis. The site appears to make use of similar data to ours, but retrieves it from Ergast Developer API[[4]](#footnote-4). The creator provides a few posts of his preprocessing techniques (feature selection in-particular), but does not go on to share which algorithm(s) he implements for his models.

As a team, we hope to distinguish our findings through high accuracy, clear documentation, and possibly the merging of peripheral datasets in Stage 3.

## Original Dataset Description

The Formula 1 dataset consists of race data from 1950 to 2017 recorded in 13 tables. The ‘circuits’ data has records of the circuit names, their locations and the coordinates. The ‘driver’ data consists of each driver’s name, date of birth, nationality and code. The ‘constructors’ table is a list of constructors and their nationalities. The ‘laptimes’ data describes each race, the associated drivers, their positions and the time taken to complete each lap. The ‘races’ data is the most descriptive of all, it provides information about a variety of elements such as the grid, points obtained, laps, rank, fastest lap attributes etc. There are also few additional tables available that have the seasons data, the qualifying results for each race, and the description of various statuses. Finally, there is a 'constructorResults' table, which can be used as our target variable when training our models.

## Data Preprocessing

We split this process into several stages: initial cleaning, data aggregation, and finally data transformation. In the initial cleaning phase, we parsed through the datasets, identifying any ill-formatted values and correcting/normalizing those values, such as eliminating special characters. The aggregation phase comprised of merging our tables into a data warehouse, and exporting to a working dataset. In the transformation phase, we eliminated useless and redundant attributes, concatenated attributes, and transformed attributes.

The data cleaning process was initially performed upon loading CSVs into a MySQL database for further analysis. Below is a sample of the raw data in some of the files, such as drivers:

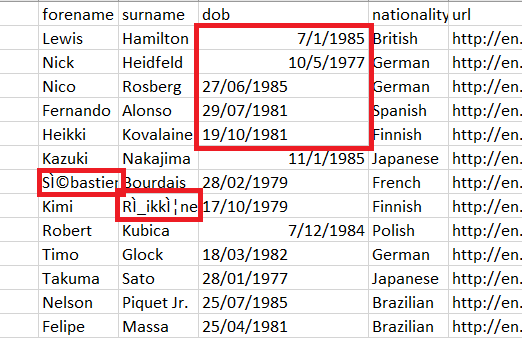


Figure 1: Raw, uncleaned data

As shown, highlighted by red boxes in the image, this data needed to be cleaned and normalized so that it could be properly imported. The image below is the change to the data so that it is normalized:

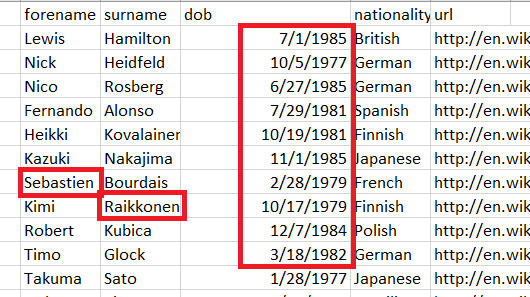


Figure 2: Cleaned data fields

The names were referenced with a master list of drivers who have competed in the F1 circuit. DOB was formatted originally as DD/MM/YYYY and was changed to MM/DD/YYYY.

#### After we cleaned the data, we moved to the aggregation phase. Below is a list of changes that were made during the MySQL merging process:

* After complete merge, 70 columns remained. That was immediately reduced to 25 based on hand selected feature selection
* Deleted *URLs* from MySQL import as they were useless to predictions
* Deleted *PositionText* column from *ConstructorStandings* as it had the same values as *Position*
* *DOB* in *drivers.csv* is not formatted properly, providing inconsistent dates across all fields. Cleaned *DOB* column for properly formatted dates
* Deleted *URL* field from numerous datasets upon import
* Driver names are inconsistent and needed formatting (Done manually with a different list of all F1 drivers)

Once we had a unified dataset, we began transforming our data. Below is a comprehensive list of changes made to our dataset:

* *PositionOrder* was reduced to 3, to limit the driver place finishers to the top 3 placewinners for each race.
* *PositionOrder* was changed to *RaceFinish*, since it signifies the place finish for the drivers per race.
* *RaceTime* had 14 distinct categories, with 12:00:00 being the value for more than 50% of the data fields. To reduce the large number of categories, *RaceTime* was binned to "Morning", "Noon", and "Afternoon", with “Morning” consisting of each race time before 12:00, “Noon” being exactly 12:00, and “Afternoon” being any race time after 12:00. After the binning process, the values were then one-hot-encoded.
* Since the *PositionOrder* was limited to only those in the top 5, Status column was removed because more than 95% of the drivers who finish in the top 5 finished the race, the other value was "+1", indicating that the driver was lapped and finished the race in the next lap behind the leading placewinners.
* *CarNumber* was removed since it is irrelevant in predicting who wins a race.
* Driver *Forename* and *Surname* is concatenated to produce a *DriverName*.
* Driver *DateOfBirth* was changed to driver age at the time of the race, and then subsequently removed.
* *FastestLap* and *LapRank* entail what lap the driver had their fastest time on, and what rank among the drivers they were. For example, a driver could have their fastest lap on lap 44 and could be in 4th place at that time. Due to the redundancy and number of categories, these two fields were removed.
* *Laps* signifies the number of laps per race per track and was removed since it is duplicated. *RaceLap* is used in place of *Laps* since it distributes lap time and other attributes incrementally for each race.
* *LapTime* was removed since it contained total race time, in addition to individual lap time, and skewed the data.

## Dataset Description After Preprocessing

The Formula 1 dataset was originally collected through the Ergast Developer API, which is an experimental web service that contains historical racing data. The data was collected from 1950 to the 2017 season and included variables such as constructors, drivers, lap times, pit stops, race date and more. It was then shared through Kaggle which is from where it was directly downloaded from for this analysis.

After preprocessing, our dataset contained 16 racing attributes and a total of 52,743 observations to use in the analysis. Our dependent variable is “raceFinish” and it indicates if the driver finished in first, second, or third place. “driverName” was another variable that was left in the dataset. This variable is important because there are some drivers that perform better than others and one of the only factors that distinguishes them from everyone else is their name. In total, there are 21 unique drivers in the dataset. Other driver characteristics such as the age of the driver at the time of the race (“ageAtRace”) and nationality are also included. Driver ages range from 19 to 43 and there is a total of 14 unique nationalities included within the data.

Race information such as “raceName”, which is a character variable naming the 25 unique races, and “raceDate”, which has race dates ranging from 2011 to 2017, also form part of the data. “RaceTime” is a binary value that indicates whether the race occurred in the morning, noon, or afternoon. The motivation behind including this variable is driven by the thought that some drivers may perform better at different times of the day, thus leading to different podium finishes.

The next few variables included in our analysis deal with pit stops. “numberOfPitStops” simply displays how many total pit stops the driver made while racing. The more pit stops the driver makes, the more time they are off the track, which can lead to a lower finishing position. The opposite is true as well, which is why it serves as a good predictor for “raceFinish”. For this variable, we had a minimum of 1 pit stop and a maximum of 6 pit stops. In addition, we included other pit stop information such as “pitStopLap” which indicates a specific pit stop and “pitStopMilliseconds” which is nothing but the length of time the driver took at the pit stop. These two variables are representative of the driver’s pit crew’s efficiency. The faster they are getting the driver out of the pit, the better the chances of the driver finishing at a higher position. For “pitStopMilliseconds” there is a minimum of 14,501ms and a maximum of 201,361ms.

Another independent variable in the data is “raceLap”, which is an integer specifying the lap in which the driver was in. In addition, “racePosition” indicates the position the driver was in at the specified lap and “lapMilliseconds” measures the length of time it took to complete that lap. These are important variables to keep in the data because they measure the overall performance of the driver during the race.

The final variable, “teamName”, is a character value that contains the name of the team in which the driver is in. A team is usually made up of two drivers which means that the driver’s performance is dependent on their teammate’s skill level. It was important to keep this variable in the data because depending on what team the driver is placed in, they may be more or less likely to finish the race in the top three. Within our dataset, there are 9 unique teams that are being analyzed.

## Intended Algorithms

A subset of our data was extracted based on top 3 podium finishers, i.e. ‘raceFinish’ = 1/2/3. We used three classification algorithms for our initial analysis - Decision Tree, Naïve Bayes and Support Vector Machines. All our algorithms were executed using R packages. We used 80% of our initial subset for training and 20% as the test data.

*Decision Tree*

Decision Tree splits a given dataset into multiple branches and leaf nodes. Each branch denotes the outcome of the classification test and each leaf node represents a classification label. The plotted result of a decision tree is easy to interpret and can handle high dimensional data. This algorithm is used for financial analysis, molecular biology, credit scoring etc. We used the R library package C50 - basic tree-based model.

*Naïve Bayes*

Naïve Bayes is based on Bayes’ theorem. The classifier predicts the probability that a given tuple belongs to a particular class. The Naïve Bayes classifier also assumes independence of all the class attributes. This classifier is widely used for its speed and high scalability. Applications of Naïve Bayes include sentiment analysis, spam filtering, text classification etc. We used the R library package e1071 - ‘naiveBayes’ function.

*Support Vector Machine*

Support Vector Machine uses kernels to calculate the distance between two observations and map the data into a higher dimension. SVM can be used to classify linear and nonlinear data. Although SVM has a longer execution time, it is the least prone to overfitting among the other algorithms. It is being used widely in applications such as object recognition, gene expression etc. We used the R library package e1701 - ‘svm’ function.

## Preliminary Analysis

After running the algorithms, we show our results in table 1. We can see that in almost every category, SVM outperforms other models. The only place it doesn't do the best in the 3 models is the F-Score for calculating finishing in first place, where decision trees did the best. F-Score is the harmonic mean of precision and recall for each class. Precision and recall are calculated through analysis of true positives, true negatives, false positives, and false negatives. which can give better analysis of your model than just pure accuracy. Tables 2, 3, and 4 give more in-depth analysis on how our model performed in each category.

*Naïve Bayes*

Performs the worst out of all our attempted algorithms in every category. We believe this is due to its simplicity and due to our predictors not having class conditional independence from each other. Naïve Bayes will most likely be dropped for following updates and model tuning.

*SVM*

This gives us our best results among our three algorithms. We believe this is due to the robustness of the algorithm and the feature space mapping. We will be using SVM moving forward for our models due to its success in this domain.

*Decision Tree*

This only underperformed SVM by a couple percentage points on the normal, baseline without tuned parameters. We believe we can employ other techniques to greatly improve our model here, which will be discussed later in the paper.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Accuracy | F-Score Class 1 | F-Score Class 2 | F-Score Class 3 |
| Naïve Bayes | 53.48% | 63.61% | 31.90% | 56.49% |
| SVM | **75.28%** | 79.60% | **69.17%** | **76.45%** |
| C5.0 Decision Tree | 73.13% | **82.18%** | 62.54% | 74.29% |

Table 1: Comparison of model accuracy and F-Score

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Naïve Bayes | | Predicted | | |  |
| 1 | 2 | 3 | Recall |
| Actual | 1 | 4219 | 397 | 551 | **.8165** |
| 2 | 2327 | 1232 | 1688 | .2348 |
| 3 | 1550 | 848 | 3011 | .5567 |
| Precision | | **.5211** | .4975 | .5735 |  |
| Model Accuracy: **53.48%** | | | | |  |

Table 2: Naïve Bayes confusion matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SVM | | Predicted | | |  |
| 1 | 2 | 3 | Recall |
| Actual | 1 | 2926 | 372 | 132 | **.8531** |
| 2 | 550 | 2278 | 615 | .6616 |
| 3 | 445 | 493 | 2735 | .7446 |
| Precision | | .7462 | .7248 | **.7855** |  |
| Model Accuracy: **75.28%** | | | | |  |

Table 3: SVM confusion matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Decision Tree | | Predicted | | |  |
| 1 | 2 | 3 | Recall |
| Actual | 1 | 3797 | 612 | 507 | .7724 |
| 2 | 393 | 2840 | 1722 | .5732 |
| 3 | 134 | 675 | 4389 | **.8444** |
| Precision | | **.8781** | .6882 | .6632 |  |
| Model Accuracy: **73.13%** | | | | |  |

Table 4: Decision Tree confusion matrix

## Future Direction

After running three different algorithms on our subset of our data, we determined that we have two different ways to improve our accuracy, precision and recall. We categorized the two ways to data based and model based, we will touch on more information on how we can use both ways to improve our analysis and prediction in the future.

**Data Based**

*Finding new datasets*

We are thinking of merging other datasets into our pre-processed Formula 1 datasets. With the new datasets we will be able to expand our analysis and find different correlations and relationships. To put this into perspective, we could find a weather dataset that shows the relevant weather information for the race track conditions on that particular day of a race. By running algorithm on it, we might be able to improve our prediction.

*Further normalization*

We are also thinking on further normalizing our dataset into more clear-cut columns and have less redundant information. Currently, we are tracking each lap and within that, we are tracking the lap milliseconds. By doing this, we have a lot of redundant data in there and we think that we can cut down on the number of redundant rows and columns going forward.

*Improve feature creation*

We are also thinking how feature creation can improve in our analysis. We have previously done some feature creation by merging multiple different relevant data into one feature and cleaning the data. We believe that we can do more feature creation by further combining different columns and attributes together to get a more relevant feature for our project.

*Reduce dimensions based on Principal Component Analysis (PCA)*

After we get new features through our feature creation, we will need to select and ensure that the feature is relevant. More features can sometimes reduce the accuracy of the models, so we are planning to use PCA to find the most relevant feature. PCA allows us to identify patterns in the data based on the correlation between features, this will help us determine which feature is most relevant to our analysis.

*Logarithmic Binning*

We are also planning to use logarithmic binning to split different sets of data into buckets. If we have a dataset which has all the independent variable in a vast range, it is hard to visualize the data. By creating buckets for the top observations, we can view the large distribution of data more easily and get a feel for our data. For example, we may have a dataset that ranges from 0 to 100,000,000, by binning them into different ranges of values like 1,000, 10,000,100,000 etc. we can have a better distribution of visualizing the data.

*Granularity change*

While we are planning to work on logarithmic binning, we are also planning to test how changing the granularity of our data set helps in our analysis. Granularity of the data show the level of detail that we want to display in our data. Low granularity might help us understand the data since it provides more information. Higher granularity allows us to get a high-level overview of our data and let us know if this data is relevant to our analysis or not.

**Model Based**

*Better tweaked models*

We believe we can further tweak our models to give us more accurate results in our next stage of our projects. We can tweak our decision tree depths in different ways to see if we do one way or the other will improve our accuracy. Moreover, we can also set learning rates in our neural network to further improve our weights to obtain a more accurate analysis. We can also run non-linear kernels on certain independent variables through SVM and see whether that improve our accuracy of our algorithm.

*Gradient Boosting*

Moving forward, we can have an optimization model of gradient boosting on our analysis by adding weak learner to minimize the loss of the model. Since the weak learner uses decision trees for gradient boosting, we can create constraints on the maximum number of layers and nodes that the weak learner has access to; hence this will preserve them to remain weak to prevent overfitting

*Cross Validation*

We thought about another way where we could improve our accuracy of our analysis by doing cross validation. We are thinking of trying out K-fold cross-validation where we could evaluate our model performance on the different subset of the training data and calculate what is the average prediction error rate. We could split our dataset into multiple subsets such as raceName, raceDate, raceTime all into one subset and raceLap, racePosition and lap milliseconds all into one subset and do a K-fold cross-validation analysis.

*Bootstrap Aggregation/Bagging*

Another model-based method that we thought about is bagging where we could use to improve on our decision tree algorithm. Decision trees are high variance algorithm, so when we change our training data, the predictions will vary by a lot. By using bagging on our decision tree, we reduce the chance of overfitting because we are combining the predictions from multiple machine learning algorithms to make our analysis more accurate.

## Appendix

TheR code used for all the three algorithms can be found below:

**Decision Tree**

> formula1 <- read.csv('CleanedFormula1.csv')

> sample\_size <- floor(0.8 \* nrow(formula1))

> seq\_len(20)

> training\_index <- sample(seq\_len(nrow(formula1)), size = sample\_size)

> train <- formula1[training\_index, ]

> test <- formula1[-training\_index, ]

> library(C50)

> train$raceFinish <- as.factor(train$raceFinish)

> test$raceFinish <- as.factor(test$raceFinish)

> predictors <- c('driverName', 'ageAtRace', 'nationality', 'raceName', 'raceDate', 'raceTime...Morning', 'raceTime...Noon', 'raceTime...Afternoon', 'numberOfPitStops', 'pitStopLap', 'pitStopMilliseconds', 'raceLap', 'racePosition', 'lapMilliseconds', 'teamName')

> model <- C5.0.default(x = train [, predictors], y = train$raceFinish)

> summary(model)

> pred <- predict(model, newdata = test)

> evaluation <- cbind(test, pred)

> head(evaluation)

> nrow (evaluation[evaluation$pred == evaluation$raceFinish, ]) / nrow(evaluation)

**Naïve Bayes**

> library(e1071)

> formula1 <- read.csv('CleanedFormula1.csv')

> summary(formula1)

> nrow(formula1[!complete.cases(formula1), ])

> sample\_size <- floor(0.8 \* nrow(formula1))

> training\_index <- sample(nrow(formula1), size = sample\_size, replace = FALSE)

> train <- formula1[training\_index,]

> test <- formula1[-training\_index,]

> formula1.model <- NaïveBayes(as.factor(raceFinish) ~ ., data = train)

> formula1.predict <- predict(formula1.model, test, type = 'class')

> results <- data.frame(actual = test[ , 'raceFinish'], predicted = formula1.predict)

> table(results)

> nrow(results[results$predicted == results$actual, ]) / nrow(results)

**Support Vector Machine**

> formula1 <- read.csv('CleanedFormula1.csv')

> summary(formula1)

> set.seed(545)

> sample\_size <- floor(0.8 \* nrow(formula1))

> training\_index <- sample(nrow(formula1), size = sample\_size)

> train <- formula1[training\_index,]

> test <- formula1[-training\_index,]

> library(e1071)

> svm\_model <- svm(as.factor(raceFinish) ~ ., data = train, method = "C-classification", kernel = "linear")

> summary (svm\_model)

> mean(results$actual == results$predicted)

1. <https://www.oddschecker.com/motorsport/formula-1> [↑](#footnote-ref-1)
2. <https://mybookie.ag/sportsbook/f1/> [↑](#footnote-ref-2)
3. <http://www.f1-predictor.com/about/> [↑](#footnote-ref-3)
4. <http://ergast.com/mrd/> [↑](#footnote-ref-4)