Formula 1 Data Analysis

Jon Montgomery

Formula 1 Data Analysis

1.

In this project, we will explore the Formula 1 World Championship data from 1950 to 2024. The data set is available on kaggle. It consists of various CSV files containing tabular data regarding drivers, circuits, teams, and results ranging from the granularity of championship standings to pit stop characteristics. The data are very high quality.

I will be working with Abhishek on this project.

2.

We want to use the historical data to create a Markov Chain model to predict the performance of a driver in the next race. We want to test the effectiveness of the Markov Chain predictions against naive models to see if we can gain an edge. Here are some examples of naive models:

1. A driver's performance is random.
2. A driver's performance in the next race will be the same as their performance in the prior race.
3. A driver's performance in the next race will be equal to the average of their performance in the prior 10 races.

We will split the data into random samplings of training and test data (60% training and 40% test). Once the model is trained, we will evaluate the performance of each model against the test data using the mean absolute error metric, where the error is the difference between the predicted driver rank and the actual driver rank.

Once we have calculated the mean absolute error for each model, we will test the differences in the errors for statistical significance. The method we will use to test for statistical significance will depend on whether or not the errors are normally distributed. If the errors are normally distributed, we will compare the results with the paired t-test. If the errors are not normally distributed, we will use the Wilcoxon signed-rank test.

3.

We will use git for version control in this project. Abhishek and I will both do preliminary data cleaning and exploration and then merge our findings. I will build the markov models and most of the code infrastructure for the project. Then Abhishek will review, we will test the models, and Abhishek will compile slides for the presentation. Once we have the data, we will write the paper, dividing it between us depending on who is most familiar with each part.

For cleaning and exploration, we will evaluate each table and field to decide which variables could hold predictive power and should be included in the Markov model. We will identify missing values and handle them appropriately on a case-by-case basis. We will also identify the breakpoints between seasons with discontinuous car specifications (every 8 years or so the "formula" of the cars changes drastically). We'll divide the data randomly into testing and training data and evaluate the models. We will complete this portion of the project this weekend March 2.

I'll build the Markov Chains between March 3-7 and we will train them and do analysis over the weekend March 8-9. We will spend the rest of our time preparing the paper and the presentation.

```r
knitr::opts_chunk$set(echo = TRUE)
library(stringr)
library(ggplot2)
library(knitr)
library(unifed)
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(slider)
library(markovchain)
```

```
## Loading required package: Matrix

## Package:   markovchain
## Version:   0.10.0
## Date:      2024-11-14 00:00:02 UTC
## BugReport: https://github.com/spedygiorgio/markovchain/issues
```

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(Matrix)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(zoo)
```

```
##
## Attaching package: 'zoo'

## The following object is masked from 'package:markovchain':
##
##     is.regular

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(mltools)

set.seed(53782)
```

Markov Chain Parameters

- circuit id
- driver id
- driver's average rank over the past 10 races
- driver's most recent rank
- qualifying position
- constructor id
- constructor rank
- constructor's median pit stop time in the past 10 races
- weather ?

One row per driver per race.

We will not make predictions for the first race of the season or for drivers with fewer than 10 races. For each prediction, we only include the information that we know right before the race starts.

Read the data into dataframes

```
circuits<-
  read_csv(file.path("data", "circuits.csv"),
           col_types  = cols(
           circuitId  = col_integer(),
           circuitRef = col_character(),
           name       = col_character(),
           location   = col_character(),
           country    = col_character(),
           lat        = col_double(),
           lng        = col_double(),
           alt        = col_integer(),
           url        = col_character())) %>%
  select(-c(
    "circuitRef",
    "location",
    "country",
    "lat",
    "lng",
    "alt",
    "url")) %>%
  rename(circuitName=name)

constructor.standings<-
  read_csv(file.path("data", "constructor_standings.csv"),
           col_types = cols(
             constructorStandingsId = col_integer(),
             raceId                 = col_integer(),
             constructorId          = col_integer(),
             points                 = col_double(),
             position               = col_integer(),
             positionText           = col_character(),
             wins                   = col_integer())) %>%
  select(-c(
    "points",
    "positionText",
    "wins")) %>%
  rename(constructorStandingsPosition = position)

constructors<-
  read_csv(file.path("data", "constructors.csv"),
           col_types = cols(
             constructorId  = col_integer(),
             constructorRef = col_character(),
             name           = col_character(),
             nationality    = col_character(),
             url            = col_character())) %>%
  select(-c(
    "nationality",
    "url",
    "constructorRef")) %>%
  rename(constructorName = name)

driver.standings<-
  read_csv(file.path("data", "driver_standings.csv"),
           col_types = cols(
             driverStandingsId = col_integer(),
             raceId            = col_integer(),
             driverId          = col_integer(),
             points            = col_double(),
             position          = col_integer(),
             positionText      = col_character(),
             wins              = col_integer())) %>%
  select(-c(
    "points",
    "positionText",
    "wins")) %>%
  rename(driverStandingsPosition = position)

drivers<-
  read_csv(file.path("data", "drivers.csv"),
           col_types = cols(
             driverId =    col_integer(),
             driverRef =   col_character(),
             number =      col_character(),
             code =        col_character(),
             forename =    col_character(),
             surname =     col_character(),
             dob =         col_date(),
             nationality = col_character(),
             url =         col_character())) %>%
  select(-c(
    "number",
    "code",
    "dob",
    "nationality",
    "url",
    "driverRef"))

races<-
  read_csv(file.path("data", "races.csv"),
           col_types = cols(
             raceId =    col_integer(),
             year =      col_integer(),
             round =     col_integer(),
             circuitId = col_integer(),
             name =      col_character(),
             date =      col_date(),
             time =      col_character(),
             url =       col_character(),
             fp1_date =  col_character(),
             fp1_time =  col_character())) %>%
  select(-c(
    "year",
    "name",
    "time",
    "url",
    "fp1_date",
    "fp1_time",
    "fp2_date",
    "fp2_time",
    "fp3_date",
    "fp3_time",
    "quali_date",
    "quali_time",
    "sprint_date",
    "sprint_time"))

results<-
  read_csv(file.path("data", "results.csv"),
           col_types = cols(
             resultId =      col_integer(),
             raceId =        col_integer(),
             driverId =      col_integer(),
             constructorId = col_integer(),
             number =        col_character(),
             grid =          col_integer(),
             position =      col_character(),
             positionText =  col_character(),
             positionOrder = col_integer(),
             points =        col_double())) %>%
  select(-c(
    "number",
    "position",
    "positionText",
    "points",
    "laps",
    "time",
    "milliseconds",
    "fastestLap",
    "rank",
    "fastestLapTime",
    "fastestLapSpeed")) %>%
  rename(raceResultPosition = positionOrder, startingPosition = grid)

status<-
  read_csv(file.path("data", "status.csv"),
           col_types = cols(
             statusId = col_integer(),
             status =   col_character()
           ))
```

Creating the usable data

We only include results for drivers that have at least 10 races.

```
get_mode <- function(x) {
  unique_x <- unique(x)
  unique_x[which.max(tabulate(match(x, unique_x)))]
}

get_qualitative_race_result <- function(x) {
  return(case_when(
      x == 1 ~          "win",
      x %in% c(2, 3) ~ "podium",
      x <= 10 ~         "score",
      TRUE ~            "no score"
    ))
}
encoded_qualitative_race_result <- function(x) {
  return(case_when(
    x == "win" ~      0,
    x == "podium" ~   1,
    x == "score" ~    2,
    x == "no score" ~ 3
  ))
}
decode_qualitative_race_result <- function(x) {
  return(case_when(
    x == 0 ~ "win",
    x == 1 ~ "podium",
    x == 2 ~ "score",
    x == 3 ~ "no score"
  ))
}

# Joining the datasets
data <- results %>%
  left_join(races, by = "raceId") %>%
  left_join(circuits, by = "circuitId") %>%
  left_join(drivers, by = "driverId") %>%
  left_join(constructors, by = "constructorId") %>%
  left_join(driver.standings, by = c("raceId", "driverId")) %>%
  left_join(constructor.standings, by = c("raceId", "constructorId")) %>%
  left_join(status, by = "statusId") %>%
  arrange(driverId, date) %>% # ensure the data is ordered correctly
  group_by(driverId) %>%
  mutate(
    meanRaceResult10 = slider::slide_dbl(raceResultPosition, mean, .before = 9, .complete = TRUE),
    status = case_when(
      status == "Finished" ~         "Finished",
      grepl("\\+\\d+ Lap", status) ~ "Lapped",
      TRUE ~                         "DNF"
    ),
    raceResultQualitative = get_qualitative_race_result(raceResultPosition)
  ) %>%
  mutate(
    modeQualitativeResult10 = rollapply(raceResultQualitative, width = 10, FUN = get_mode,
                                         align = "right", fill = NA)
  ) %>%
  mutate(
    prevRaceResultPosition =           lag(raceResultPosition),
    prevDriverStandingsPosition =      lag(driverStandingsPosition),
    prevConstructorStandingsPosition = lag(constructorStandingsPosition),
    prevStatus =                       lag(status),
    prevMeanRaceResult10 =             lag(meanRaceResult10),
    prevModeQualitativeResult10 =      lag(modeQualitativeResult10),
    prevRaceResultQualitative =        lag(raceResultQualitative)
  ) %>%
  ungroup() %>%
  filter(!is.na(prevMeanRaceResult10) & !is.na(prevRaceResultPosition) & (round!=1))
data$driverStandingsPosition[is.na(data$driverStandingsPosition)] <- max(data$driverStandingsPosition, 
data$constructorStandingsPosition[is.na(data$constructorStandingsPosition)] <- max(data$constructorStan
data$prevStatus[is.na(data$prevStatus)] <- "unknown"
```

Splitting the training and testing data

```
train.index <- createDataPartition(1:nrow(data), p = 0.6, list = FALSE)
train.data <- data[train.index,]
test.data <- data[-train.index,]
```

Building the Markov Chain

```
get.trained.mc <- function(training.data.from, training.data.to) {
  # Ensure all unique states are included (as character)
  all_states <- as.character(unique(c(training.data.from, training.data.to)))

  # Create a transition table (ensuring all states are included)
  transition.table <- table(
    factor(training.data.from, levels = all_states),
    factor(training.data.to, levels = all_states)
  )

  # Convert the table to a probability matrix
  transition.matrix <- prop.table(transition.table, margin = 1)

  # Explicitly convert "table" to a numeric matrix
  transition.matrix <- matrix(as.numeric(transition.matrix),
                              nrow = length(all_states),
                              dimnames = list(all_states, all_states))

  # Create and return the Markov Chain model
  return(new("markovchain", states = all_states, transitionMatrix = transition.matrix))
}
mc.predict<-function(mc, test.data) {
  return(sapply(test.data, function(x) rmarkovchain(n = 1, object = mc, t0 = x)))
}
```

XGBoost

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:zoo':
##
##     yearmon, yearqtr

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(mltools)

# Convert categorical variables using one-hot encoding
data.xgboost <- as.data.table(data)
data.xgboost[, `:=`(
  circuitId = as.factor(circuitId),
  constructorId = as.factor(constructorId),
  prevStatus = as.factor(prevStatus),
  raceResultQualitative = encoded_qualitative_race_result(raceResultQualitative)
)]

data.onehot <- one_hot(as.data.table(data.xgboost[, c(
  "circuitId",
  "constructorId",
  "prevStatus"
)]))

# Select numerical features
data.numeric <- data.xgboost[, c(
  "startingPosition",
  "prevDriverStandingsPosition",
  "prevConstructorStandingsPosition",
  "prevRaceResultPosition",
  "prevMeanRaceResult10",
  "raceResultQualitative"
)]

# Combine numerical and one-hot encoded categorical features
```

```r
data.xgboost <- cbind(data.numeric, data.onehot)

train.data.xgboost <- data.xgboost[train.index, ]
test.data.xgboost <- data.xgboost[-train.index, ]

# Separate features and labels
train.x <- as.matrix(train.data.xgboost[, !names(train.data.xgboost) %in% "raceResultQualitative", with
train.y <- train.data.xgboost$raceResultQualitative

test.x <- as.matrix(test.data.xgboost[, !names(test.data.xgboost) %in% "raceResultQualitative", with =
test.y <- test.data.xgboost$raceResultQualitative

# Convert to XGBoost DMatrix
dtrain <- xgb.DMatrix(data = train.x, label = train.y)
dtest <- xgb.DMatrix(data = test.x, label = test.y)

params <- list(
  objective = "multi:softmax", # Multi-class classification
  num_class = length(unique(data$raceResultQualitative)), # Number of classes
  eval_metric = "mlogloss", # Multi-class log loss
  eta = 0.1, # Learning rate
  max_depth = 6, # Tree depth
  subsample = 0.8, # Sample ratio
  colsample_bytree = 0.8 # Column sample ratio
)

# Train the XGBoost model
model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  verbose = 1
)
```

```
## [1]  train-mlogloss:1.319663 test-mlogloss:1.322466
## [2]  train-mlogloss:1.262506 test-mlogloss:1.267774
## [3]  train-mlogloss:1.213810 test-mlogloss:1.221635
## [4]  train-mlogloss:1.172869 test-mlogloss:1.183186
## [5]  train-mlogloss:1.134624 test-mlogloss:1.147520
## [6]  train-mlogloss:1.102172 test-mlogloss:1.117321
## [7]  train-mlogloss:1.072786 test-mlogloss:1.090380
## [8]  train-mlogloss:1.046075 test-mlogloss:1.065842
## [9]  train-mlogloss:1.022202 test-mlogloss:1.044248
## [10] train-mlogloss:1.001316 test-mlogloss:1.025868
## [11] train-mlogloss:0.982597 test-mlogloss:1.009369
## [12] train-mlogloss:0.965233 test-mlogloss:0.994020
## [13] train-mlogloss:0.949282 test-mlogloss:0.979754
## [14] train-mlogloss:0.934625 test-mlogloss:0.966987
## [15] train-mlogloss:0.921592 test-mlogloss:0.955696
## [16] train-mlogloss:0.909349 test-mlogloss:0.945098
## [17] train-mlogloss:0.898461 test-mlogloss:0.936216
## [18] train-mlogloss:0.887997 test-mlogloss:0.927759
## [19] train-mlogloss:0.878448 test-mlogloss:0.920315
## [20] train-mlogloss:0.869628 test-mlogloss:0.913156
## [21] train-mlogloss:0.861911 test-mlogloss:0.906773
## [22] train-mlogloss:0.854113 test-mlogloss:0.901080
## [23] train-mlogloss:0.847517 test-mlogloss:0.896382
## [24] train-mlogloss:0.841030 test-mlogloss:0.891438
## [25] train-mlogloss:0.835090 test-mlogloss:0.886709
## [26] train-mlogloss:0.829517 test-mlogloss:0.882560
## [27] train-mlogloss:0.824190 test-mlogloss:0.878893
## [28] train-mlogloss:0.819310 test-mlogloss:0.875617
## [29] train-mlogloss:0.814790 test-mlogloss:0.872729
## [30] train-mlogloss:0.810090 test-mlogloss:0.869570
## [31] train-mlogloss:0.806096 test-mlogloss:0.866933
## [32] train-mlogloss:0.802205 test-mlogloss:0.864643
## [33] train-mlogloss:0.798469 test-mlogloss:0.862222
## [34] train-mlogloss:0.795413 test-mlogloss:0.860318
## [35] train-mlogloss:0.792083 test-mlogloss:0.858380
## [36] train-mlogloss:0.788838 test-mlogloss:0.856704
## [37] train-mlogloss:0.786167 test-mlogloss:0.855172
## [38] train-mlogloss:0.783414 test-mlogloss:0.853679
## [39] train-mlogloss:0.780658 test-mlogloss:0.852146
## [40] train-mlogloss:0.777972 test-mlogloss:0.850911
## [41] train-mlogloss:0.775198 test-mlogloss:0.850024
## [42] train-mlogloss:0.772363 test-mlogloss:0.848785
## [43] train-mlogloss:0.769868 test-mlogloss:0.847887
## [44] train-mlogloss:0.767641 test-mlogloss:0.846886
## [45] train-mlogloss:0.765194 test-mlogloss:0.845974
## [46] train-mlogloss:0.763096 test-mlogloss:0.845240
## [47] train-mlogloss:0.760945 test-mlogloss:0.844582
## [48] train-mlogloss:0.758840 test-mlogloss:0.843929
## [49] train-mlogloss:0.757382 test-mlogloss:0.843456
## [50] train-mlogloss:0.755620 test-mlogloss:0.842920
## [51] train-mlogloss:0.753994 test-mlogloss:0.842367
## [52] train-mlogloss:0.752007 test-mlogloss:0.842090
## [53] train-mlogloss:0.750263 test-mlogloss:0.841511
## [54] train-mlogloss:0.748596 test-mlogloss:0.841141
## [55] train-mlogloss:0.747176 test-mlogloss:0.840892
## [56] train-mlogloss:0.745568 test-mlogloss:0.840559
## [57] train-mlogloss:0.744257 test-mlogloss:0.840093
## [58] train-mlogloss:0.743077 test-mlogloss:0.839939
## [59] train-mlogloss:0.741780 test-mlogloss:0.839729
## [60] train-mlogloss:0.740667 test-mlogloss:0.839383
## [61] train-mlogloss:0.739179 test-mlogloss:0.839306
## [62] train-mlogloss:0.737578 test-mlogloss:0.839254
## [63] train-mlogloss:0.736304 test-mlogloss:0.839045
## [64] train-mlogloss:0.735088 test-mlogloss:0.838798
## [65] train-mlogloss:0.733804 test-mlogloss:0.838755
## [66] train-mlogloss:0.732442 test-mlogloss:0.838887
## [67] train-mlogloss:0.731486 test-mlogloss:0.838822
## [68] train-mlogloss:0.729987 test-mlogloss:0.838858
## [69] train-mlogloss:0.728982 test-mlogloss:0.838771
## [70] train-mlogloss:0.727591 test-mlogloss:0.838514
## [71] train-mlogloss:0.726629 test-mlogloss:0.838419
## [72] train-mlogloss:0.725470 test-mlogloss:0.838408
## [73] train-mlogloss:0.724223 test-mlogloss:0.838165
## [74] train-mlogloss:0.723208 test-mlogloss:0.838221
## [75] train-mlogloss:0.722131 test-mlogloss:0.838181
## [76] train-mlogloss:0.720799 test-mlogloss:0.837939
## [77] train-mlogloss:0.719955 test-mlogloss:0.837867
## [78] train-mlogloss:0.718793 test-mlogloss:0.837920
## [79] train-mlogloss:0.717915 test-mlogloss:0.837825
## [80] train-mlogloss:0.717067 test-mlogloss:0.837708
## [81] train-mlogloss:0.716138 test-mlogloss:0.837776
## [82] train-mlogloss:0.715652 test-mlogloss:0.837811
## [83] train-mlogloss:0.714652 test-mlogloss:0.837848
## [84] train-mlogloss:0.713569 test-mlogloss:0.837778
## [85] train-mlogloss:0.712610 test-mlogloss:0.837959
## [86] train-mlogloss:0.711552 test-mlogloss:0.838063
## [87] train-mlogloss:0.710522 test-mlogloss:0.838231
## [88] train-mlogloss:0.709530 test-mlogloss:0.838218
## [89] train-mlogloss:0.708632 test-mlogloss:0.838292
## [90] train-mlogloss:0.708042 test-mlogloss:0.838342
## [91] train-mlogloss:0.707204 test-mlogloss:0.838336
## [92] train-mlogloss:0.706453 test-mlogloss:0.838451
## [93] train-mlogloss:0.705271 test-mlogloss:0.838502
## [94] train-mlogloss:0.704178 test-mlogloss:0.838576
## [95] train-mlogloss:0.703368 test-mlogloss:0.838631
## [96] train-mlogloss:0.702478 test-mlogloss:0.838850
## [97] train-mlogloss:0.701647 test-mlogloss:0.838858
## [98] train-mlogloss:0.701055 test-mlogloss:0.838946
## [99] train-mlogloss:0.700122 test-mlogloss:0.839160
## [100]    train-mlogloss:0.699285 test-mlogloss:0.839155
```

```r
preds <- predict(model, dtest)

# Convert numeric predictions back to original categories
xgboost.predictions <- as.factor(preds)
xgboost.actual <- as.factor(test.y)
```

Naive Models

```r
naive.predict.same.as.last<-function(test.data) {
  return(test.data$prevRaceResultQualitative)
}
naive.predict.same.as.mode<-function(test.data) {
  return(test.data$prevModeQualitativeResult10)
}
naive.predict.same.as.start<-function(test.data) {
  return(sapply(test.data$startingPosition, function(x) get_qualitative_race_result(x)))
}
```

Training the Markov Chain

```r
mc <- get.trained.mc(train.data$prevRaceResultQualitative, train.data$raceResultQualitative)
```

Generate all the predictions

```
comparison.report <- function(preds, actual) {
  preds <- factor(sapply(preds, encoded_qualitative_race_result))
  actual <- factor(sapply(actual, encoded_qualitative_race_result))

  # Ensure both factors have the same levels
  levels_combined <- union(levels(preds), levels(actual))
  preds <- factor(preds, levels = levels_combined)
  actual <- factor(actual, levels = levels_combined)

  return(confusionMatrix(preds, actual))
}
markov.predictions <- mc.predict(mc, test.data$prevRaceResultQualitative)
naive.same.as.last <- naive.predict.same.as.mode(test.data)
naive.same.as.mode <- naive.predict.same.as.mode(test.data)
naive.same.as.start <- naive.predict.same.as.start(test.data)
```

Statistics for the Markov Model

Let's use the Chi-Squared test to see if the predictions are statistically correlated to the actual results.

```
print(chisq.test(table(markov.predictions, test.data$raceResultQualitative)))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(markov.predictions, test.data$raceResultQualitative)
## X-squared = 146.15, df = 9, p-value < 2.2e-16
```

```
print(comparison.report(markov.predictions, test.data$raceResultQualitative))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3
##          0   45   54  121  164
##          1   68  119  272  376
##          2  131  241  832 1307
##          3  170  345 1306 2555
##
## Overall Statistics
##
##                Accuracy : 0.4381
##                  95% CI : (0.4272, 0.449)
##     No Information Rate : 0.5431
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.0604
##
##  Mcnemar's Test P-Value : 0.5038
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity          0.108696  0.15679   0.3287   0.5804
## Specificity          0.955928  0.90255   0.6988   0.5084
## Pos Pred Value       0.117187  0.14251   0.3313   0.5839
## Neg Pred Value       0.952214  0.91198   0.6963   0.5048
## Prevalence           0.051073  0.09363   0.3122   0.5431
## Detection Rate       0.005551  0.01468   0.1026   0.3152
## Detection Prevalence 0.047372  0.10301   0.3098   0.5398
## Balanced Accuracy    0.532312  0.52967   0.5138   0.5444
```

Statistics for the Naive Same as Mode

```
print(chisq.test(table(naive.same.as.mode, test.data$raceResultQualitative)))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(naive.same.as.mode, test.data$raceResultQualitative)
## X-squared = 1868.7, df = 9, p-value < 2.2e-16
```

```
print(comparison.report(naive.same.as.mode, test.data$raceResultQualitative))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3
##          0  127   72   61   72
##          1   74  181  152  145
##          2   89  241  978  873
##          3  124  265 1340 3312
##
## Overall Statistics
##
##                Accuracy : 0.5672
##                  95% CI : (0.5564, 0.5781)
##     No Information Rate : 0.5431
##     P-Value [Acc > NIR] : 6.338e-06
##
##                   Kappa : 0.2405
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity           0.30676  0.23847   0.3864   0.7524
## Specificity           0.97335  0.94950   0.7842   0.5332
## Pos Pred Value        0.38253  0.32790   0.4484   0.6570
## Neg Pred Value        0.96308  0.92348   0.7379   0.6444
## Prevalence            0.05107  0.09363   0.3122   0.5431
## Detection Rate        0.01567  0.02233   0.1207   0.4086
## Detection Prevalence  0.04096  0.06810   0.2691   0.6219
## Balanced Accuracy     0.64006  0.59399   0.5853   0.6428
```

Statistics for Naive Same as Last

```
print(chisq.test(table(naive.same.as.last, test.data$raceResultQualitative)))
```

```
## 
##  Pearson's Chi-squared test
## 
## data:  table(naive.same.as.last, test.data$raceResultQualitative)
## X-squared = 1868.7, df = 9, p-value < 2.2e-16
```

```
print(comparison.report(naive.same.as.last, test.data$raceResultQualitative))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1    2    3
##          0  127   72   61   72
##          1   74  181  152  145
##          2   89  241  978  873
##          3  124  265 1340 3312
## 
## Overall Statistics
## 
##                Accuracy : 0.5672
##                  95% CI : (0.5564, 0.5781)
##     No Information Rate : 0.5431
##     P-Value [Acc > NIR] : 6.338e-06
## 
##                   Kappa : 0.2405
## 
##  Mcnemar's Test P-Value : < 2.2e-16
## 
## Statistics by Class:
## 
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity           0.30676  0.23847   0.3864   0.7524
## Specificity           0.97335  0.94950   0.7842   0.5332
## Pos Pred Value        0.38253  0.32790   0.4484   0.6570
## Neg Pred Value        0.96308  0.92348   0.7379   0.6444
## Prevalence            0.05107  0.09363   0.3122   0.5431
## Detection Rate        0.01567  0.02233   0.1207   0.4086
## Detection Prevalence  0.04096  0.06810   0.2691   0.6219
## Balanced Accuracy     0.64006  0.59399   0.5853   0.6428
```

Statistics for Naive Same as Starting Position

```
print(chisq.test(table(naive.same.as.start, test.data$raceResultQualitative)))
```

```
## 
##  Pearson's Chi-squared test
## 
## data:  table(naive.same.as.start, test.data$raceResultQualitative)
## X-squared = 3122.2, df = 9, p-value < 2.2e-16
```

```
print(comparison.report(naive.same.as.start, test.data$raceResultQualitative))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1    2    3
##          0  192   80   60   95
##          1  141  259  157  216
##          2   76  349 1201 1389
##          3    5   71 1113 2702
##
## Overall Statistics
##
##                Accuracy : 0.5371
##                  95% CI : (0.5262, 0.548)
##     No Information Rate : 0.5431
##     P-Value [Acc > NIR] : 0.8602
##
##                   Kappa : 0.2432
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity           0.46377  0.34124   0.4745   0.6138
## Specificity           0.96945  0.93004   0.6746   0.6790
## Pos Pred Value        0.44965  0.33506   0.3983   0.6944
## Neg Pred Value        0.97109  0.93182   0.7388   0.5967
## Prevalence            0.05107  0.09363   0.3122   0.5431
## Detection Rate        0.02369  0.03195   0.1482   0.3333
## Detection Prevalence  0.05268  0.09536   0.3719   0.4800
## Balanced Accuracy     0.71661  0.63564   0.5746   0.6464
```

Statistics for the XGBoost Model

```
print(chisq.test(table(xgboost.predictions, test.data$raceResultQualitative)))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(xgboost.predictions, test.data$raceResultQualitative)
## X-squared = 3135.1, df = 9, p-value < 2.2e-16
```

```
print(confusionMatrix(xgboost.predictions, xgboost.actual))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3
##          0  170   79   49   76
##          1  113  236  146  169
##          2   59  256 1031  698
##          3   72  188 1305 3459
##
## Overall Statistics
##
##                Accuracy : 0.604
##                  95% CI : (0.5933, 0.6147)
##     No Information Rate : 0.5431
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3109
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3
## Sensitivity           0.41063  0.31094   0.4073   0.7858
## Specificity           0.97348  0.94174   0.8183   0.5775
## Pos Pred Value        0.45455  0.35542   0.5044   0.6885
## Neg Pred Value        0.96844  0.92972   0.7526   0.6940
## Prevalence            0.05107  0.09363   0.3122   0.5431
## Detection Rate        0.02097  0.02911   0.1272   0.4267
## Detection Prevalence  0.04614  0.08191   0.2522   0.6198
## Balanced Accuracy     0.69205  0.62634   0.6128   0.6816
```

Interpreting the Confusion Matrix

A confusion matrix is a tool for evaluating the performance of a predictive model used when you have labeled categorical data. It shows the predicted category on one axis and the true category on the other. Each cell contains the number of observations for that cross section.

The Accuracy metric measures the total number of correct classifications divided by the total number of exambles:

$$Accuracy(a,p) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1 & \text{if } a_i = p_i\\ 0 & \text{else}\end{cases}$$

The confidence interval for an accuracy score is calculated via Wilson's score interval. The Wilson score interval is asymmetric, and it doesn't suffer from problems of overshoot and zero-width intervals. It can be safely employed with small samples and skewed observations.

$$z_a \approx \frac{(p-\hat{p})}{\sigma_n}$$

where z_a is the standard normal deviation for the desired confidence interval $1-a$, p is the 'true' accuracy, $hatp$ is the observed accuracy, and σ_n is the binomial sample standard deviation. The derivation of the Wilson score interval is beyond the scope of this paper.

Comparing the accuracy of our predictions

All of our models showed significant correlation between the predictions and outcomes, but which model performed best? \

Markov Model Accuracy : 0.4381
95% CI : (0.4272, 0.449)

Same as Last Accuracy : 0.5186
95% CI : (0.5077, 0.5296)

Same as Starting Position Accuracy : 0.5371
95% CI : (0.5262, 0.548)

Same as Mode Accuracy : 0.5672
95% CI : (0.5564, 0.5781)

	Model B Correct	Model B Incorrect
Model A Correct	n_{11}	n_{10}
Model A Incorrect	n_{01}	n_{00}

XGBoost Accuracy : 0.604
95% CI : (0.5933, 0.6147)

We will use Paired McNemar's Test to compare these accuracy scores because each model made predictions from the same dataset. Our null hypothesis is that there is no difference in accuracy between any of the models.

McNemar's Test is the following:

McNemar's test statistic is:

$$\chi^2 = \frac{(n_{10} - n_{01})^2}{n_{10} + n_{01}}$$

Where χ^2 has a chi-squared distribution with 1 degree of freedom.

We will perform McNemar's test on each pair of predictions from least to most accurate. If we have predictions a, b, c with $accuracy(a) > accuracy(b)$ and $accuracy(b) > accuracy(c)$, we can say that $accuracy(a) > accuracy(b) > accuracy(c)$.

Markov to Same as Last

```r
mcnemar.compare<-function(predictions.a, predictions.b, actual) {
  correct.a <- as.integer(predictions.a == actual)
  correct.b <- as.integer(predictions.b == actual)
  a <- sum(correct.a == 1 & correct.b == 1)
  b <- sum(correct.a == 1 & correct.b == 0)
  c <- sum(correct.a == 0 & correct.b == 1)
  d <- sum(correct.a == 0 & correct.b == 0)
  return(
    mcnemar.test(matrix(
      c(a, b, c, d), nrow=2, byrow=TRUE,
      dimnames=list("Model 1" = c("Correct", "Incorrect"),
                    "Model 2" = c("Correct", "Incorrect"))
  ))$p.value)
}
print(paste("Markov v Same as Last: p =",
            mcnemar.compare(markov.predictions, naive.same.as.last, test.data$raceResultQualitative)))
```

```
## [1] "Markov v Same as Last: p = 1.68642400480114e-74"
```

```r
print(paste("Same as Last v Same as Start: p =",
            mcnemar.compare(naive.same.as.last, naive.same.as.start, test.data$raceResultQualitative)))
```

```
## [1] "Same as Last v Same as Start: p = 8.95434868175451e-06"
```

```r
print(paste("Same as Start v Same as Mode: p =",
            mcnemar.compare(naive.same.as.start, naive.same.as.mode, test.data$raceResultQualitative)))
```

```
## [1] "Same as Start v Same as Mode: p = 8.95434868175451e-06"
```

```r
print(paste("Same as Mode v XGBoost: p =",
            mcnemar.compare(naive.same.as.mode, sapply(xgboost.predictions, decode_qualitative_race_resu
```

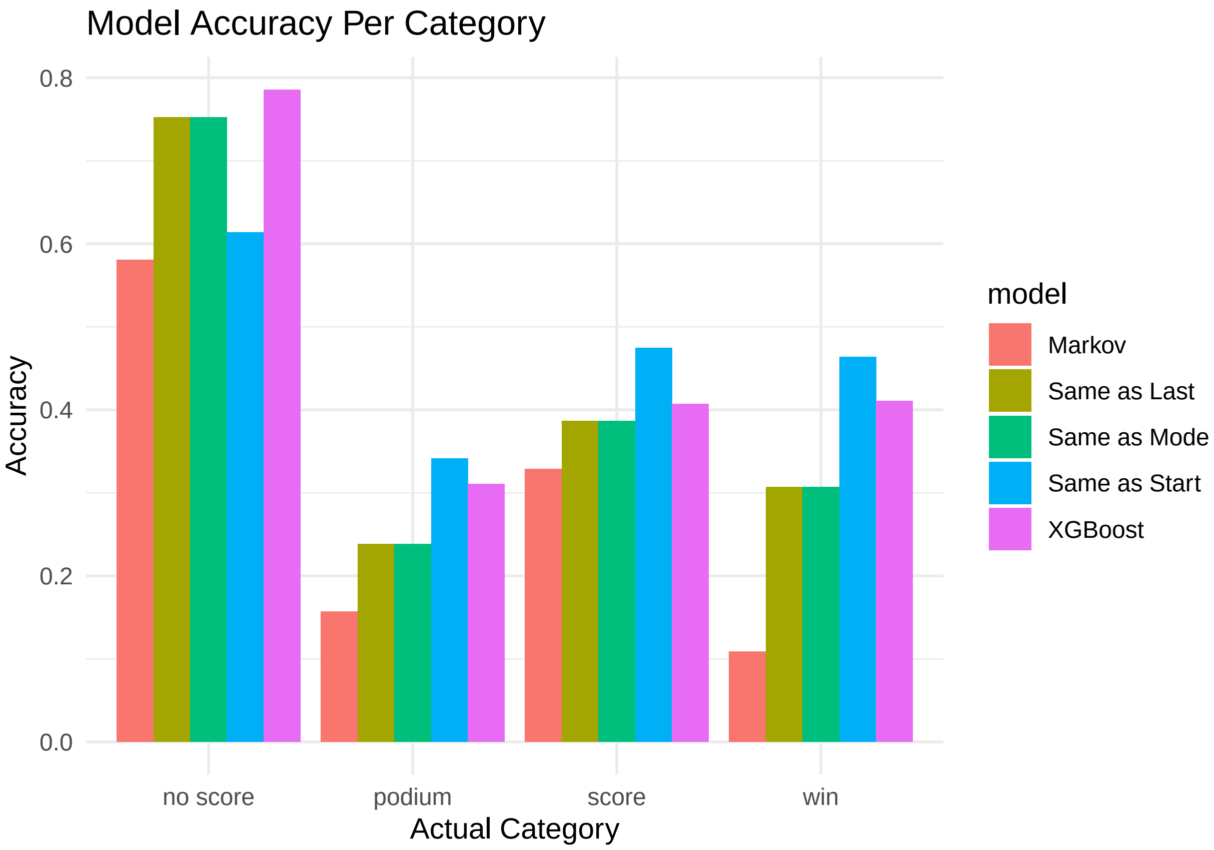
```
## [1] "Same as Mode v XGBoost: p = 8.39440175768814e-12"
```

Given a model, how does its accuracy, compare across each category?

```r
plot.category.accuracy<-function(name, pred, actual) {
  adf <- data.frame(actual, pred) %>%
    mutate(correct = actual == pred) %>%
    group_by(actual) %>%
    summarise(accuracy = mean(correct))
  return(ggplot(adf, aes(x = actual, y = accuracy, fill = actual)) +
    geom_bar(stat = "identity") +
    labs(title = paste(name, "Prediction Accuracy Per Result"),
         x = "Outcome",
         y = "Accuracy") +
    theme_minimal())
}
markov.predictions.display <- data.frame(prediction = markov.predictions, actual = test.data$raceResult
naive.same.as.last.display <- data.frame(prediction = naive.same.as.last, actual = test.data$raceResult
naive.same.as.mode.display <- data.frame(prediction = naive.same.as.mode, actual = test.data$raceResult
naive.same.as.start.display <- data.frame(prediction = naive.same.as.start, actual = test.data$raceResu
xgboost.predictions.display <- data.frame(prediction = sapply(xgboost.predictions, decode_qualitative_ra

markov.predictions.display$model  <- "Markov"
naive.same.as.last.display$model  <- "Same as Last"
naive.same.as.mode.display$model  <- "Same as Mode"
naive.same.as.start.display$model <- "Same as Start"
xgboost.predictions.display$model <- "XGBoost"

display.predictions.data <- bind_rows(
  markov.predictions.display,
  naive.same.as.last.display,
  naive.same.as.mode.display,
  naive.same.as.start.display,
  xgboost.predictions.display
)

adf <- display.predictions.data %>%
  group_by(model, actual) %>%
  mutate(correct = actual == prediction) %>%
  summarise(accuracy = mean(correct))
```

```
## `summarise()` has grouped output by 'model'. You can override using the
## `.groups` argument.
```

```r
ggplot(adf, aes(x = actual, y = accuracy, fill = model)) +
  geom_bar(stat = "identity", position = "dodge") +  # Use bar chart instead of histogram
  labs(title = "Model Accuracy Per Category", x = "Actual Category", y = "Accuracy") +
  theme_minimal()
```


Model Accuracy Per Category
Accuracy
0.0
0.2
0.4
0.6
0.8
no score
podium
score
win
Actual Category
model
Markov
Same as Last
Same as Mode
Same as Start
XGBoost