expected_points_understanding

Robert Nguyen

04/03/2021

Purpose for discussion

When doing expected points, theres a few things that are doing differently compared to what is being done in other common/similar sports like NFL.

There are a few different things being done in terms of comparing the expected points methods.

One of the things talked about previously was that, one of the things that people don't seem to be doing with expected points, is putting the decision they want to evaluate into the model itself. This seems weird given the most common talked about scenario (run vs pass) has now been readily available for NFL fans for a few years now through data accessibilty in packages like nflscrapR and nflfastR

Model that people use for expected points

Question 1 - Whats up with each row in the model?

In a meeting I tried to explain some things in the write up that are different, like

- even though data is available nobody seems to add in kick/pass into expected points models
  - what they seem to do is like a pivot table and sum up EPA per play which means you lose the ability to say where on the field it might be better to change the decision
- The other weird one - this part of the question is all rows are kept in the model data frame for the multinomial fit.
  - This is the first I guess key difference (apart from different sports)
  - With the NRL data - each scoring event only features once, but with NFL that isn't the case - lets look below.

```
# https://github.com/ryurko/nflscrapR-models/blob/master/R/init_models/init_ep_fg_models.R
# https://github.com/ryurko/nflscrapR-models/tree/master/R/init_models


library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
# Access nflWAR:
# install.packages("devtools")
# devtools::install_github("ryurko/nflWAR")

library(nflWAR)
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
# Load data from 2009 to 2016 from the nflscrapR-data repository using the
# get_pbp_data() function from the nflWAR package:
pbp_data <- get_pbp_data(2009:2016)
```

```
## Warning: 6 parsing failures.
##   row            col           expected     actual
##  1244 BlockingPlayer 1/0/T/F/TRUE/FALSE J.McGraw   'https://raw.github.com/ryurko/nflscrapR-
##  2780 BlockingPlayer 1/0/T/F/TRUE/FALSE B.Williams 'https://raw.github.com/ryurko/nflscrapR-
## 16314 BlockingPlayer 1/0/T/F/TRUE/FALSE J.Hester   'https://raw.github.com/ryurko/nflscrapR-
## 21256 BlockingPlayer 1/0/T/F/TRUE/FALSE G.Hayes    'https://raw.github.com/ryurko/nflscrapR-
## 27430 BlockingPlayer 1/0/T/F/TRUE/FALSE E.Smith    'https://raw.github.com/ryurko/nflscrapR-
## ..... ............... ................... .......... ......................................
## See problems(...) for more details.
```

```
## Warning: 9 parsing failures.
##   row            col           expected     actual
##  2406 BlockingPlayer 1/0/T/F/TRUE/FALSE J.McKnight 'https://raw.github.com/ryurko/nflscrapR-
## 11418 BlockingPlayer 1/0/T/F/TRUE/FALSE T.Shaw     'https://raw.github.com/ryurko/nflscrapR-
## 18497 BlockingPlayer 1/0/T/F/TRUE/FALSE P.Bailey   'https://raw.github.com/ryurko/nflscrapR-
## 18887 BlockingPlayer 1/0/T/F/TRUE/FALSE R.Quinn    'https://raw.github.com/ryurko/nflscrapR-
## 27034 BlockingPlayer 1/0/T/F/TRUE/FALSE C.Clemons  'https://raw.github.com/ryurko/nflscrapR-
## ..... ............... ................... .......... ......................................
## See problems(...) for more details.
```

```
## Warning: 71 parsing failures.
##  row          col           expected  actual
## 1217 TwoPointConv 1/0/T/F/TRUE/FALSE Failure 'https://raw.github.com/ryurko/nflscrapR-data/
## 2263 TwoPointConv 1/0/T/F/TRUE/FALSE Success 'https://raw.github.com/ryurko/nflscrapR-data/
## 2645 TwoPointConv 1/0/T/F/TRUE/FALSE Failure 'https://raw.github.com/ryurko/nflscrapR-data/
## 3956 TwoPointConv 1/0/T/F/TRUE/FALSE Success 'https://raw.github.com/ryurko/nflscrapR-data/
## 4144 TwoPointConv 1/0/T/F/TRUE/FALSE Failure 'https://raw.github.com/ryurko/nflscrapR-data/
## .... ............ ................... ....... ........................................
## See problems(...) for more details.
```

```
## Warning: 24 parsing failures.
##  row            col           expected     actual
## 1553 BlockingPlayer 1/0/T/F/TRUE/FALSE A.Blue     'https://raw.github.com/ryurko/nflscrapR-d
## 1832 BlockingPlayer 1/0/T/F/TRUE/FALSE C.McCain   'https://raw.github.com/ryurko/nflscrapR-d
## 2790 BlockingPlayer 1/0/T/F/TRUE/FALSE J.Attaochu 'https://raw.github.com/ryurko/nflscrapR-d
```

```
## 3100 BlockingPlayer 1/0/T/F/TRUE/FALSE A.Dixon    'https://raw.github.com/ryurko/nflscrapR-
## 4578 BlockingPlayer 1/0/T/F/TRUE/FALSE T.McDonald 'https://raw.github.com/ryurko/nflscrapR-
## .... ............. ................. ......... ....................................
## See problems(...) for more details.

## Warning: 16 parsing failures.
##   row            col           expected     actual
##  5328 BlockingPlayer 1/0/T/F/TRUE/FALSE D.McCray   'https://raw.github.com/ryurko/nflscrapR-
##  5765 BlockingPlayer 1/0/T/F/TRUE/FALSE R.Jennings 'https://raw.github.com/ryurko/nflscrapR-
##  8326 DefTwoPoint    1/0/T/F/TRUE/FALSE Failure    'https://raw.github.com/ryurko/nflscrapR-
## 13954 BlockingPlayer 1/0/T/F/TRUE/FALSE M.Mauti    'https://raw.github.com/ryurko/nflscrapR-
## 14280 BlockingPlayer 1/0/T/F/TRUE/FALSE J.Johnson  'https://raw.github.com/ryurko/nflscrapR-
## ..... ............. ................. ......... ....................................
## See problems(...) for more details.

## Warning: 17 parsing failures.
##   row            col           expected   actual
##  4365 DefTwoPoint    1/0/T/F/TRUE/FALSE Success  'https://raw.github.com/ryurko/nflscrapR-da
##  6468 BlockingPlayer 1/0/T/F/TRUE/FALSE R.Okwara 'https://raw.github.com/ryurko/nflscrapR-da
##  7191 DefTwoPoint    1/0/T/F/TRUE/FALSE Failure  'https://raw.github.com/ryurko/nflscrapR-da
## 18974 BlockingPlayer 1/0/T/F/TRUE/FALSE T.McEvoy 'https://raw.github.com/ryurko/nflscrapR-da
## 22854 BlockingPlayer 1/0/T/F/TRUE/FALSE J.Allen  'https://raw.github.com/ryurko/nflscrapR-da
## ..... ............. ................. ........ ....................................
## See problems(...) for more details.
```

```r
# Remove error game from 2011 that is coded incorrectly in raw JSON data:
pbp_data <- pbp_data %>% filter(GameID != "2011121101")

nrow(pbp_data)
```

```
## [1] 362263
```

```r
# 362263

#' Define a function that takes in a play-by-play data set and returns
#' what the type of next score is and the drive number only within same half.
#' @param pbp_dataset Play-by-play dataset with the following columns:
#' sp - scoring play indicator, PlayType - what type of play, qtr - quarter
#' of the game, Drive - drive number for the play, ReturnResult - indicates
#' what happened on return type plays, posteam - indicates the possession team
#' for the play, and columns for ReturnResult, FieldGoalResult, ExPointResult,
#' TwoPointConv, DefTwoPoint, Touchdown.
#' @return Data frame with two columns: Next_Score_Half denoting the type of
#' the next scoring event occurring within the half of each play and
#' Drive_Score_Half denoting the drive number of the next scoring play.

find_game_next_score_half <- function(pbp_dataset) {

  # Which rows are the scoring plays:
  score_plays <- which(pbp_dataset$sp == 1 & pbp_dataset$PlayType != "No Play")

  # Define a helper function that takes in the current play index,
  # a vector of the scoring play indices, play-by-play data,
  # and returns the score type and drive number for the next score:
  find_next_score <- function(play_i, score_plays_i,pbp_df) {
```

```
    # Find the next score index for the current play
    # based on being the first next score index:
    next_score_i <- score_plays_i[which(score_plays_i >= play_i)[1]]

    # If next_score_i is NA (no more scores after current play)
    # or if the next score is in another half,
    # then return No_Score and the current drive number
    if (is.na(next_score_i) |
        (pbp_df$qtr[play_i] %in% c(1, 2) & pbp_df$qtr[next_score_i] %in% c(3, 4, 5)) |
        (pbp_df$qtr[play_i] %in% c(3, 4) & pbp_df$qtr[next_score_i] == 5)) {

          score_type <- "No_Score"

          # Make it the current play index
          score_drive <- pbp_df$Drive[play_i]

      # Else return the observed next score type and drive number:
    } else {

      # Store the score_drive number
      score_drive <- pbp_df$Drive[next_score_i]

      # Then check the play types to decide what to return
      # based on several types of cases for the next score:

      # 1: Return TD
      if (identical(pbp_df$ReturnResult[next_score_i], "Touchdown")) {

        # For return touchdowns the current posteam would not have
        # possession at the time of return, so it's flipped:
        if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

          score_type <- "Opp_Touchdown"

        } else {

          score_type <- "Touchdown"

        }
      } else if (identical(pbp_df$FieldGoalResult[next_score_i], "Good")) {

        # 2: Field Goal
        # Current posteam made FG
        if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

          score_type <- "Field_Goal"

          # Opponent made FG
        } else {

          score_type <- "Opp_Field_Goal"

        }
```

```
    # 3: Touchdown (returns already counted for)
  } else if (pbp_df$Touchdown[next_score_i] == 1) {

    # Current posteam TD
    if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

      score_type <- "Touchdown"

    # Opponent TD
    } else {

      score_type <- "Opp_Touchdown"

    }
    # 4: Safety (similar to returns)
  } else if (pbp_df$Safety[next_score_i] == 1) {

    if (identical(pbp_df$posteam[play_i],pbp_df$posteam[next_score_i])) {

      score_type <- "Opp_Safety"

    } else {

      score_type <- "Safety"

    }
    # 5: Extra Points
  } else if (identical(pbp_df$ExPointResult[next_score_i], "Made")) {

    # Current posteam Extra Point
    if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

      score_type <- "Extra_Point"

      # Opponent Extra Point
    } else {

      score_type <- "Opp_Extra_Point"

    }
    # 6: Two Point Conversions
  } else if (identical(pbp_df$TwoPointConv[next_score_i], "Success")) {

    # Current posteam Two Point Conversion
    if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

      score_type <- "Two_Point_Conversion"

    # Opponent Two Point Conversion
    } else {

      score_type <- "Opp_Two_Point_Conversion"
```

```
        }

        # 7: Defensive Two Point (like returns)
      } else if (identical(pbp_df$DefTwoPoint[next_score_i], "Success")) {

        if (identical(pbp_df$posteam[play_i], pbp_df$posteam[next_score_i])) {

          score_type <- "Opp_Defensive_Two_Point"

        } else {

          score_type <- "Defensive_Two_Point"

        }

        # 8: Errors of some sort so return NA (but shouldn't take place)
      } else {

        score_type <- NA

      }
    }

    return(data.frame(Next_Score_Half = score_type,
                      Drive_Score_Half = score_drive))
  }

  # Using lapply and then bind_rows is much faster than
  # using map_dfr() here:
  lapply(c(1:nrow(pbp_dataset)), find_next_score,
         score_plays_i = score_plays, pbp_df = pbp_dataset) %>%
    bind_rows() %>%
    return
}

# Apply to each game (ignore the warning messages here):
pbp_next_score_half <- map_dfr(unique(pbp_data$GameID),
                               function(x) {
                                 pbp_data %>%
                                   filter(GameID == x) %>%
                                   find_game_next_score_half()
                               })

# Join to the pbp_data:
pbp_data_next_score <- bind_cols(pbp_data, pbp_next_score_half)

# Create the EP model dataset that only includes plays with basic seven
# types of next scoring events along with the following play types:
# Field Goal, No Play, Pass, Punt, Run, Sack, Spike

pbp_ep_model_data <- pbp_data_next_score %>%
  filter(Next_Score_Half %in% c("Opp_Field_Goal", "Opp_Safety", "Opp_Touchdown",
                                "Field_Goal", "No_Score", "Safety", "Touchdown") &
```

```
        PlayType %in% c("Field Goal", "No Play", "Pass", "Punt", "Run", "Sack",
                        "Spike") & is.na(TwoPointConv) & is.na(ExPointResult) &
        !is.na(down) & !is.na(TimeSecs))

nrow(pbp_ep_model_data)
```

```
## [1] 304896
```

```
# 304805

# Now adjust and create the model variables:
pbp_ep_model_data <- pbp_ep_model_data %>%

         # Reference level should be No_Score:
  mutate(Next_Score_Half = fct_relevel(factor(Next_Score_Half), "No_Score"),

         # Create a variable that is time remaining until end of half:
         # (only working with up to 2016 data so can ignore 2017 time change)
         TimeSecs_Remaining = as.numeric(ifelse(qtr %in% c(1,2), TimeSecs - 1800,
                                        ifelse(qtr == 5, TimeSecs + 900,
                                               TimeSecs))),

         # log transform of yards to go and indicator for two minute warning:
         log_ydstogo = log(ydstogo),
         Under_TwoMinute_Warning = ifelse(TimeSecs_Remaining < 120, 1, 0),

         # Changing down into a factor variable:
         down = factor(down),

         # Calculate the drive difference between the next score drive and the
         # current play drive:
         Drive_Score_Dist = Drive_Score_Half - Drive,

         # Create a weight column based on difference in drives between play and next score:
         Drive_Score_Dist_W = (max(Drive_Score_Dist) - Drive_Score_Dist) /
           (max(Drive_Score_Dist) - min(Drive_Score_Dist)),
         # Create a weight column based on score differential:
         ScoreDiff_W = (max(abs(ScoreDiff)) - abs(ScoreDiff)) /
           (max(abs(ScoreDiff)) - min(abs(ScoreDiff))),
         # Add these weights together and scale again:
         Total_W = Drive_Score_Dist_W + ScoreDiff_W,
         Total_W_Scaled = (Total_W - min(Total_W)) /
           (max(Total_W) - min(Total_W)))
```

But what model is being used in nflscrapR and hence everyone else?

```
ep_model <- nnet::multinom(Next_Score_Half ~ TimeSecs_Remaining + yrdline100 +
                           down + log_ydstogo + GoalToGo + log_ydstogo*down +
                           yrdline100*down + GoalToGo*log_ydstogo +
                           Under_TwoMinute_Warning, data = pbp_ep_model_data,
                           weights = Total_W_Scaled, maxit = 300)
```

```
## # weights:  119 (96 variable)
## initial  value 495869.023958
## iter  10 value 383110.139468
```

```
## iter  20 value 375979.891269
## iter  30 value 366715.791415
## iter  40 value 353893.646144
## iter  50 value 352311.038338
## iter  60 value 350530.877392
## iter  70 value 337802.906677
## iter  80 value 335985.928821
## iter  90 value 333953.825511
## iter 100 value 332260.460423
## iter 110 value 332051.293559
## iter 120 value 332047.128316
## iter 130 value 332046.757661
## final  value 332046.700978
## converged
```

Lets take an example scoring event in a game and see how many rows they are in?

What we are going to look at is game_id 2009091000 which is the below game

https://www.espn.com/nfl/game?gameId=290910023

dataset we are using is the pbp_ep_model_data so question is how often does a singular scoring event appear?

In the above game, the first scoring event is a TD by PITT in the second quarter. We can see that the event being modelled is the Next_Score_Half

```
library(tidyverse)
pbp_ep_model_data%>%
  filter(GameID==2009091000)%>%
  filter(qtr==1)%>%
  select(play_id, down, time, TimeSecs, SideofField, yrdln,desc,Next_Score_Half, Total_W_Scaled)
```

```
## # A tibble: 35 x 9
##    play_id down  time  TimeSecs SideofField yrdln desc  Next_Score_Half
##      <dbl> <fct> <tim>    <dbl> <chr>       <dbl> <chr> <fct>
##  1      68 1     14:53     3593 PIT            42 (14:~ Touchdown
##  2      92 2     14:16     3556 PIT            47 (14:~ Touchdown
##  3     113 3     13:35     3515 PIT            44 (13:~ Touchdown
##  4     139 4     13:27     3507 PIT            44 (13:~ Touchdown
##  5     162 1     13:16     3496 TEN             2 (13:~ Opp_Touchdown
##  6     183 2     12:40     3460 TEN             2 (12:~ Opp_Touchdown
##  7     207 3     12:11     3431 TEN             6 (12:~ Opp_Touchdown
##  8     228 4     11:34     3394 TEN             4 (11:~ Opp_Touchdown
##  9     253 1     11:24     3384 TEN            43 (11:~ Touchdown
## 10     277 2     10:48     3348 TEN            40 (10:~ Touchdown
## # ... with 25 more rows, and 1 more variable: Total_W_Scaled <dbl>
```

So what we can see is that, even though there is no score in the first half the Touchdown by Pitts in the second quarter is assigned to each row(play)

From here expected points gets attached like so.

```
calculate_expected_points <- function(pbp_data, half_seconds_remaining,
                                      yard_line_100, down, yards_to_go,
                                      goal_to_go, td_value = 7, fg_value = 3,
                                      safety_value = 2) {
  # First assert that each of the given column names for the necessary variables
  # are in the provided play-by-play dataset:
```

```
assertthat::assert_that(all(c(half_seconds_remaining, yard_line_100, down,
                              yards_to_go, goal_to_go) %in%
                              colnames(pbp_data)),
                        msg = paste0("The provided variable names for: ",
                                     paste0(c(half_seconds_remaining, yard_line_100, down,
                                       yards_to_go, goal_to_go)[which(!(c(half_seconds_remaining, yar
                                                                          yards_to_go, goal_to_go) %i
                                                                          colnames(pbp_data)))],
                                       collapse = ", "), " are not in pbp_data!"))

# Create a copy of the dataset for the EP model:
model_pbp_data <- pbp_data

# Generate the variable names to match what is necessary for the EP model:
colnames(model_pbp_data)[which(colnames(model_pbp_data) == half_seconds_remaining)] <- "TimeSecs_Rema
colnames(model_pbp_data)[which(colnames(model_pbp_data) == yard_line_100)] <- "yrdline100"
colnames(model_pbp_data)[which(colnames(model_pbp_data) == down)] <- "down"
colnames(model_pbp_data)[which(colnames(model_pbp_data) == yards_to_go)] <- "yards_to_go"
colnames(model_pbp_data)[which(colnames(model_pbp_data) == goal_to_go)] <- "GoalToGo"

# Compute the log yards to go for the model:
model_pbp_data$log_ydstogo <- log(as.numeric(model_pbp_data$yards_to_go))
# Create the under two minute warning indicator:
model_pbp_data$Under_TwoMinute_Warning <- ifelse(model_pbp_data$TimeSecs_Remaining < 120,
                                                 1, 0)
# Convert the down variable to a factor:
model_pbp_data$down <- as.factor(model_pbp_data$down)

# Now generate the predictions from the EP model:
# First get the predictions from the base ep_model:
if (nrow(model_pbp_data) > 1) {
  base_ep_preds <- as.data.frame(predict(ep_model, newdata = model_pbp_data, type = "probs"))
} else{
  base_ep_preds <- as.data.frame(matrix(predict(ep_model, newdata = model_pbp_data, type = "probs"),
                                        ncol = 7))
}
colnames(base_ep_preds) <- c("No_Score", "Opp_Field_Goal", "Opp_Safety", "Opp_Touchdown",
                             "Field_Goal", "Safety", "Touchdown")

# Rename the columns to be consistent with the nflscrapR play-by-play datasets:
base_ep_preds <- base_ep_preds %>%
  dplyr::rename(no_score_prob = No_Score,
                opp_fg_prob = Opp_Field_Goal,
                opp_safety_prob = Opp_Safety,
                opp_td_prob = Opp_Touchdown,
                fg_prob = Field_Goal,
                safety_prob = Safety,
                td_prob = Touchdown) %>%
  # Calculate the expected points:
  dplyr::mutate(ep = (0 * no_score_prob) + (-fg_value * opp_fg_prob) +
                  (-safety_value * opp_safety_prob) +
                  (-td_value * opp_td_prob) + (fg_value * fg_prob) +
                  (safety_value * safety_prob) + (td_value * td_prob))
```

```
  # Now append to the original dataset and return:
  pbp_data %>%
    dplyr::bind_cols(base_ep_preds) %>%
    return

}
```

Then to answer the question of should run or pass?

```
pbp_data%>%
  select(ExpPts,Date, Drive, GameID,PlayType,EPA)%>%
  group_by(PlayType)%>%
  filter(PlayType %in% c("Run", "Pass"))%>%
  summarise(run_pass=mean(EPA, na.rm=TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   PlayType run_pass
##   <chr>       <dbl>
## 1 Pass       0.140
## 2 Run       -0.0625
```

Can do a whole bunch of other filters but essentially its from above.