Momentum Modeling

Data Acquisition

Data Wrangling

```
finalMatchSet1 <- dplyr::bind_rows(wimbledon$round7$match1$set1)
finalMatchSet2 <- dplyr::bind_rows(wimbledon$round7$match1$set2)
finalMatchSet3 <- dplyr::bind_rows(wimbledon$round7$match1$set3)
finalMatchSet4 <- dplyr::bind_rows(wimbledon$round7$match1$set4)
finalMatchSet5 <- dplyr::bind_rows(wimbledon$round7$match1$set5)
```

```
wimbledonData <- read.csv("Data/Wimbledon_featured_matches.csv")
wimbledonMatches <- split(wimbledonData, wimbledonData$match_id)
```

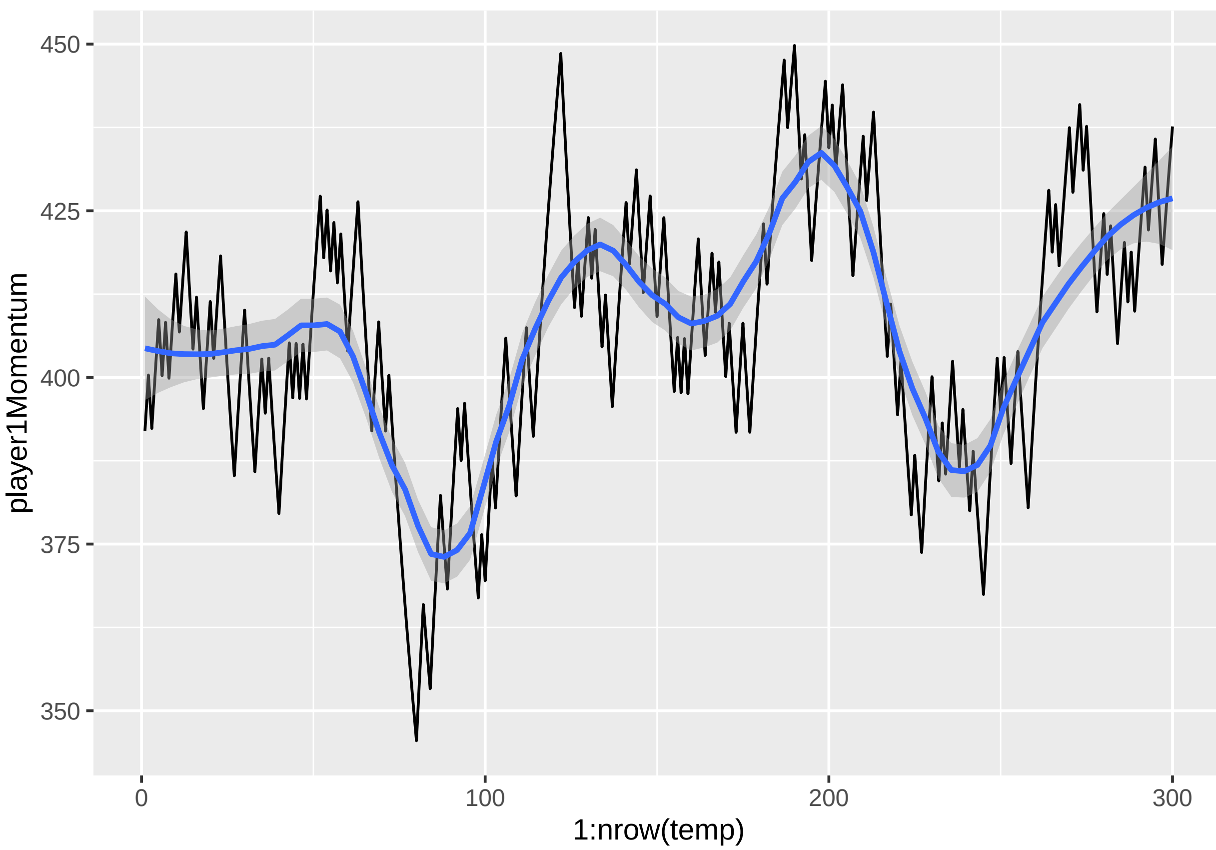
```
eloRating <- function(player1Rating, player2Rating, kFactor, gameOutcome) {
    # Expected win probability
    player1WinProbability <- (1.0) / (1 + 10^((player2Rating - player1Rating) / 400))
    player2WinProbability <- (1.0) / (1 + 10^((player1Rating - player2Rating) / 400))

    # Rating update
    player1NewRating <- 0
    player2NewRating <- 0

    if (gameOutcome == 1) # Player 1 wins
    {
        player1NewRating <- player1Rating + kFactor * (1 - player1WinProbability)
        player2NewRating <- player2Rating + kFactor * (0 - player2WinProbability)
    }
    else if (gameOutcome == 2) # Player 2 wins
    {
        player1NewRating <- player1Rating + kFactor * (0 - player1WinProbability)
        player2NewRating <- player2Rating + kFactor * (1 - player2WinProbability)
    }

    newRatings <- list(player1NewRating, player2NewRating)
    return(newRatings)
}

computeMomentumRating <- function(pointSet) {
    playerMomentum <- matrix(0, nrow = nrow(pointSet), ncol = 2)
    kFactor <- 16 # Elo rating movement strength

    for (point in 1:nrow(pointSet))
    {
        if (point == 1)
        {
            playerMomentum[point, 1:2] <- unlist(eloRating(400, 400, kFactor, pointSet$point_victor[point]))
        }
        else
        {
            playerMomentum[point, 1:2] <- unlist(eloRating(playerMomentum[point - 1, 1], playerMomentum[point
        }
    }
```

```r
    playerMomentum <- as.data.frame(playerMomentum)
    names(playerMomentum) <- c("player1Momentum", "player2Momentum")

    pointSet <- cbind(pointSet, playerMomentum)
    return(pointSet)
}
```

```r
for (match in 1:length(wimbledonMatches)) {
    wimbledonMatches[[match]] <- computeMomentumRating(wimbledonMatches[[match]])
}
```

```r
temp <- wimbledonMatches[[1]]
# ggplot(as.data.frame(lowess(1:nrow(temp), temp$momentumDelta, f = 0.1))) +
#   geom_point(aes(1:nrow(temp), temp$player1Momentum)) +
#   geom_point(aes(1:nrow(temp), temp$player2Momentum)) +
#   geom_point(aes(1:nrow(temp), temp$x))
#

ggplot(temp) +
    # geom_line(aes(1:nrow(temp), momentumDelta)) +
    # geom_smooth(aes(1:nrow(temp), momentumDelta), span = 0.3)
    geom_line(aes(1:nrow(temp), player1Momentum)) +
  geom_smooth(aes(1:nrow(temp), player1Momentum), span = .3)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```r
    # geom_smooth(aes(1:nrow(temp), player1Momentum, color = "Carlos Alcaraz"), span = 0.2) +
    # geom_smooth(aes(1:nrow(temp), player2Momentum, color = "Nicolas Jarry"), span = 0.2)
```

```r
momentum_delta <- lowess(temp$player1Momentum, f = .3)$y |> diff()
has_momentum <- function(x, span, threshold) {
  momentum_delta <- lowess(x$player1Momentum, f = span)$y |> diff()
  c("none",case_when(
```

```r
    momentum_delta >= threshold ~ "p1",
    momentum_delta <= -threshold ~ "p2",
    .default = "none"

  )) |> factor()
}

pv <- wimbledonData$point_victor - 1

score <- 2*(pv-.5)
next5 <- numeric(length(pv))
for (i in 1:length(pv)) {
  ind <- (i + 1):(min(i+5, length(pv)))
  next5[i] <- sum(score[ind])
}
t <- 1:(length(pv)-5)
trn_momentum <- function(w) {
  momentum_indicator <- lapply(wimbledonMatches, has_momentum,
                              span = w[1], threshold = w[2]) |> unlist(use.names = F)
  if (length(levels(momentum_indicator)) == 1) {
    1e6
  } else {
      mod <- lm(next5[t] ~ momentum_indicator[t])
      AIC(mod)
  }

}

w1 <- c(.3,.3)
best <- optim(w1, trn_momentum, method = "L-BFGS-B", lower=.2,
             control = list(trace = 1))
```

```
## final  value 32132.506302
## converged
```

```r
wimbledonData$momentum_indicator <- lapply(wimbledonMatches, has_momentum,
                              span = best$par[1], threshold = best$par[2]) |> unlist(use.names = F)
t2 <- 2:nrow(wimbledonData)
pred_momentum <- nnet::multinom(momentum_indicator[t2] ~ p1_winner[t2-1] + p2_winner[t2-1] +
                                  p1_ace[t2-1] + p2_ace[t2-1] +
                                  p1_unf_err[t2-1] + p2_unf_err[t2-1] +
                                  p1_double_fault[t2-1] + p2_double_fault[t2-1], data = wimbledonData)
```

```
## # weights:  30 (18 variable)
## initial  value 8001.193298
## iter  10 value 7831.296851
## iter  20 value 7753.881738
## final  value 7749.907683
## converged
```

```r
summary(pred_momentum)
```

```
## Call:
## nnet::multinom(formula = momentum_indicator[t2] ~ p1_winner[t2 -
##     1] + p2_winner[t2 - 1] + p1_ace[t2 - 1] + p2_ace[t2 - 1] +
##     p1_unf_err[t2 - 1] + p2_unf_err[t2 - 1] + p1_double_fault[t2 -
##     1] + p2_double_fault[t2 - 1], data = wimbledonData)
##
## Coefficients:
##    (Intercept) p1_winner[t2 - 1] p2_winner[t2 - 1] p1_ace[t2 - 1]
## p1  -0.4180959         0.0208557       -0.05898175      0.1228196
```

```
## p2  -0.6705399        -0.1061796        0.33946523      0.1728480
##    p2_ace[t2 - 1] p1_unf_err[t2 - 1] p2_unf_err[t2 - 1] p1_double_fault[t2 - 1]
## p1     -0.1554099        -0.06629171          0.2696848              -0.3845397
## p2     -0.4266437         0.39469567          0.1654658              -0.3249069
##    p2_double_fault[t2 - 1]
## p1             -0.05232305
## p2             -0.03651468
##
## Std. Errors:
##    (Intercept) p1_winner[t2 - 1] p2_winner[t2 - 1] p1_ace[t2 - 1]
## p1  0.04311706        0.08789969        0.09452994      0.1459657
## p2  0.04670137        0.09832432        0.09308404      0.1623213
##    p2_ace[t2 - 1] p1_unf_err[t2 - 1] p2_unf_err[t2 - 1] p1_double_fault[t2 - 1]
## p1      0.1584019         0.09831675          0.0898109               0.2497900
## p2      0.1614541         0.09528263          0.0990780               0.2289962
##    p2_double_fault[t2 - 1]
## p1               0.2158971
## p2               0.2388849
##
## Residual Deviance: 15499.82
## AIC: 15535.82
```

Just using point victor to train the thresholds ends up just setting everything lowest, so I'm training on predicting the point difference of the next 5 points. lmk if you think of anything better.

Modified Elo Rating

Algorithm