Chess betting odds

Alberto Santini
Monte dei Paschi di Siena

Outline

- What is chess-bet?
- Chess rating, expected score, team events, odds.
- Anatomy of chess-bet app.
- How to build a Shiny app.
- A few hours (!?) later...

What is chess-bet?

chess-bet is a web app, implementing a model to calculate the odds for the players and the teams in chess events.

With R and Shiny.

Credits to Peter Zhdanov

Team events: beating the bookmakers?!

Chess rating and expected score

The Elo rating system is a method for calculating the relative skill levels of players in two-player games such as chess. It is named after its creator Arpad Elo, a Hungarian-born American physics professor.

Expected score:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}.$$

source: http://en.wikipedia.org/wiki/Elo_rating_system

Team events and odds

Usually there are two teams composed of four chess players each.

This is typical for the Chess Olympiad, World Team Chess Championship and other important events.

How high is the probability that team A will win? How probable is a draw? Team B's victory?

Anatomy of chess-bet app

1. Calculate the rating differences on each board and find out the expected score of the player.
2. Find out the expected probability of a win, draw and loss in each game.
3. Repeat steps 1 and 2 for all the four boards in the team.
4. Then calculate the probabilities of all the outcomes of Team 1’s victories (and draws) and add them up.

Model details - I

Score

```
EloExpectedScore <- function(rating1, rating2) {
    1 / (1 + 10 ^ ((rating2 - rating1) / 400))
}
```

Game probability

```
getProb <- function(score, winPerc, drawPerc) {
    x <- score / (winPerc + 0.5 * drawPerc)

    win <- winPerc * x
    draw <- (score - win) * 2
    loss <- 1 - win - draw

    c(win, draw, loss)
}
```

Model details - II

Player profile

```
getPlayerProfile <- function(player) {
    fideRatingsUrl <- "http://ratings.fide.com/search.phtml?search="

    playerUrl <- paste(fideRatingsUrl, """, player, """, sep="")
    tables <- readHTMLTable(playerUrl)

    table <- tables[[1]]
    card <- getIntegerfromFactor(table[6, 1])
    player <- as.character(table[6, 2])
    rating <- getIntegerfromFactor(table[6, 7])
    ...
```

Model detail - III

Player stats

```
getPlayersStats <- function(card1, card2) {
  fideCardStatsUrl <- "http://ratings.fide.com/chess_statistics.phtml?event="

  playersRating <- c(card1$rating, card2$rating)
  weakerPlayerColumn <- match(min(playersRating), playersRating)

  playersCards <- c(card1$card, card2$card)
  playerCard <- playersCards[weakerPlayerColumn]

  statsUrl <- paste(fideCardStatsUrl, playerCard, sep="")
  doc <- htmlParse(statsUrl)
  tableNodes = getNodeSet(doc, "//table")

  table <- readHTMLTable(tableNodes[[6]])
  whiteWin <- getIntegerfromFactor(table[2, ])
  whiteDraw <- getIntegerfromFactor(table[3, ])
  whiteLoss <- getIntegerfromFactor(table[4, ])
  ...
```

Model details - IV

Team probabilities

```
getTeamProbs <- function(gamesProbs) {
    results <- permutations(n=3, r=4, v=c(1, 0.5, 0), repeats.allowed=TRUE)
    teamResults <- rowSums(results)
    ...
    teamWins <- results[teamResults > 2, ]
    teamDraws <- results[teamResults == 2, ]
    ...
    teamWinProb = sum(apply(teamWins, 1, function(outcomes) {
        probs[1, outcomes[1]] *
        probs[2, outcomes[2]] *
        probs[3, outcomes[3]] *
        probs[4, outcomes[4]]
    }))
    ...
```

How to build a Shiny app

A Shiny app combines the statistical power of R with the simplicity of a web page.

Shiny provides a fast and convenient way to build user-interfaces without HTML or JavaScript knowledge, but some applications will inevitably require more flexibility. For this type of application, you can define your user interface directly in HTML.

Shiny details - html

HTML UI with JavaScript templating

```
<div id="game1"></div>
...
<script type="text/x-template" id="profiles-and-game-probs">
<div class="row">
    <div class="span4">
        <form class="form-inline">
            <input type="text" name="<%= playerA %>"
                value="<%= playerAValue %>" />
            <span id="<%= profileARating %>"
                class="shiny-text-output badge badge-info">Loading...
            </span>
            <br>
            <span id="<%= profileACard %>"
                class="shiny-text-output label">Loading...
            </span>
```

Shiny details - server.R

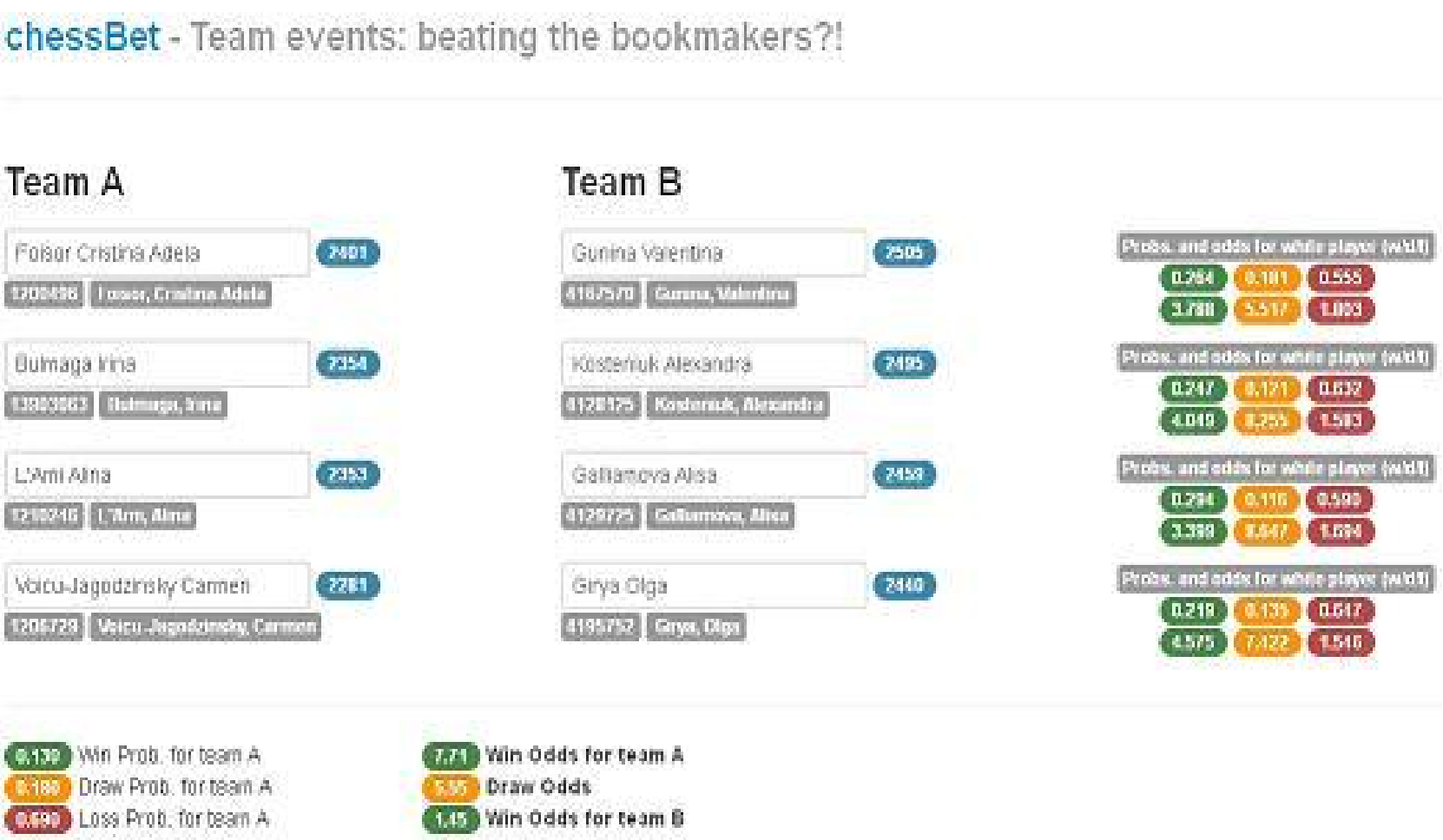
There are three kinds of objects in reactive programming: reactive sources, reactive conductors, and reactive endpoints.

```
shinyServer(function(input, output) {

  game1PlayerAProfile <- reactive ({
      getPlayerProfile(input$playerA1)
  })
  game1PlayerBProfile <- reactive ({
      getPlayerProfile(input$playerB1)
  })
  game1Stats <- reactive ({
      getPlayersStats(game1PlayerAProfile(), game1PlayerBProfile())
  })
...
```

Shiny details - server.R again

```
output_names <- rbind(output_names, c("profileA1", "game1PlayerAProfile",
    "profileB1", "game1PlayerBProfile", "game1", "game1Stats"))
...
apply(output_names, 1, function (out_name) {
    local({
        info_names <- c("Card", "Player", "Rating")
        my_out_name <- out_name[1]
        for (info_name in info_names) {
            local({
                my_info_name <- info_name
                end_out_name <- paste(my_out_name, my_info_name, sep = "")

                output[[end_out_name]] <<- renderText({
                    my_profile <- get(out_name[2])()
                    my_profile[[tolower(my_info_name)]]
                })
...
```

A few hours (?!) later...
From snippet to web app



Thanks!

Alberto Santini
https://github.com/albertosantini

http://glimmer.rstudio.com/icebox/chessBet/
https://github.com/albertosantini/chess-bet