Title: Chess Games Study & Prediction Through the Use of Web-Scraping and Regression Analysis

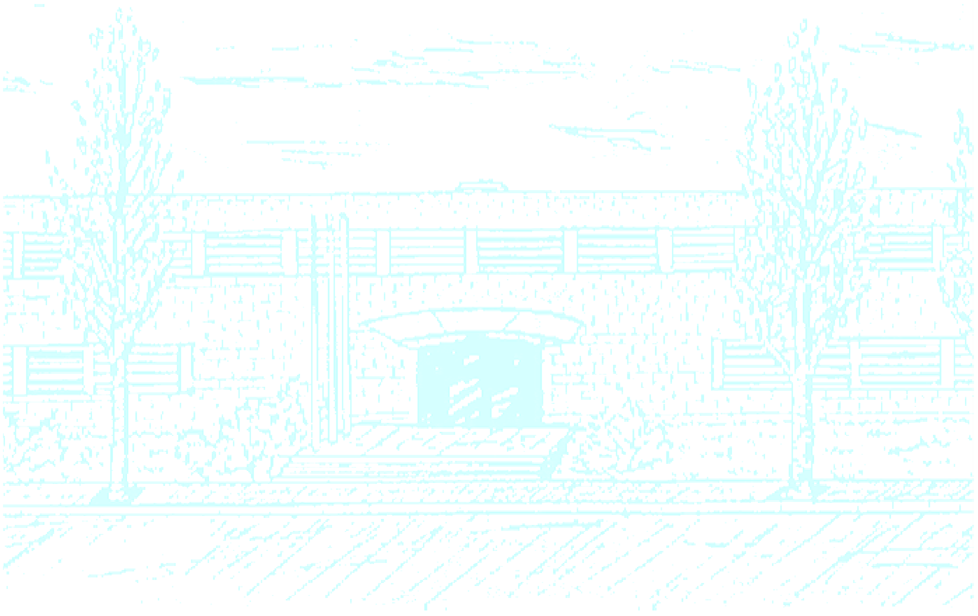
Author: Elvin Vence

Advisor: Pedro Delicado

Department: Estadística i Investigació Operativa

University: Universitat Politècnica de Catalunya

Academic year: 2015



Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

Master’s Thesis

Chess Games Study & Prediction Through the Use of Web-Scraping and Regression Analysis

Elvin Vence

Director: Pedro Delicado

Departament d'Estadística i Investigació Operativa

ABSTRACT

A study centered in using Distance-based Regression was to be done. A method in which explanatory variables are not quantitative in nature making it particularly useful for certain types of data. We searched for data that would adapt to this analysis and found a reliable set of data containing extensive information related to Chess Games in the form of a website called chesstempo.com. The highlight of this relied on the fact it was easily accessible and contained complete information for each game, information that could be translated into explanatory and dependent variables.

From this extensive set of information, Elo Ratings, the set of movements made for each game and the outcome/result of the game were selected as the main variables. We set our target to be modeling the data using the outcome of the game as the dependent variable and then predicting it. The set of movements was used as a way to track the evolution of the game and to allow us to obtain the board configuration after each movement. Distances between these configurations served as the explanatory variable for the analysis using Distance-based Regression. These distances were pair wise calculated based on the position of the pieces for both games involved to which it was then extended for every pair of games possible, thus creating the Distance Matrix. This regression method has a local variant which was also incorporated into the study. As a way to have something to compare the Distance-based Regression with, Logistic Regression was included in the study using the Elo Ratings as explanatory variable.

In order to have the data in an usable manner several steps were taken as it was initially obtained in the form of a source code. The technique called Web Scraping was used to this end in order to have a comprehensible data frame containing the relevant information for each game. The steps taken to reach this point included downloading the data, identifying the portion of the source code containing the information we were interested in, obtaining this portion of the source code, scraping the irrelevant pieces of data and finally storing it in a comprehensible data frame. Aside from the usual functions provided in R, the programming language used during this Master's Thesis to carry out the data scraping and subsequent analysis, Regular Expressions-related functions were used mainly to scrape the data. Its use could be considered intricate but key in the interest of having the data in an usable manner.

Following this a data frame replicating the key features of a chess board was created. It contained the position of each of the 32 pieces involved in a game alongside several functions to make the subsequent computations possible, including a function that would return the configuration after receiving the number of movements to be considered, used heavily as a way to create the Distance Matrix. With all the previous computations put in place, the analysis was carried out for several scenarios involving the number of movements. Finally the results obtained are interpreted and analyzed.

Key Words: Distance-based prediction, Generalized linear models, Information Storage and Retrieval, R package dbstats, Web Scraping

Mathematics Subject Classification: 62–07 62J12 68P20

INDEX

ABSTRACT III

1. **Introduction** 2

2. **Web Scraping with R & Creation of a Chess Games Data Frame** 5

2.1 Diagnostics and First Try with HTML-Related Functions *5*

2.2 Regular Expressions *15*

2.3 Creation of the Chess Games Data Frame *20*

3. **Chessboard configuration & Distance Matrix** 24

3.1 Creation of the Chessboard and its related functions *24*

3.2 Distance Matrix *35*

4. **Regression Analysis** 43

4.1 Analysis of the Chess Data using Logistic Regression *43*

4.2 Analysis of the Chess Data using Distance-based Regression *56*

5. **Conclusions** 64

APPENDIX **66**

BIBLIOGRAPHY **86**

1. INTRODUCTION

Chess, known as such in the English speaking world, is one of the main references we can find when looking for a strategy game. Its prevalence in the western and part of the eastern world as a test of strategy and intelligence remain to this day, proof of that being its worldwide popularity and status.

Its origins, as it so happens in many of today's most famous games and disciplines, are disputed but there's a growing consensus linking it to ancient Arabia and the birth of Islam. "Chess and Islam were born around the same time – Chess out of a regional need to understand complex new ideas and Islam out of Arabs' need of discipline, intelligence and peaceful community" (Shenk, 2010). Following this same line of thought it can be said that it fulfills the same role in this Master Thesis, as it serves the purpose of being the tool adopted in order to understand, analyze and use more complex ideas, being mainly in this case Web Scraping and Regression Analysis, all of which were used as a mean to model and predict the result of a Chess game.

In a slightly more detailed way, this process of modeling and predicting the outcome of a chess game would start by building a comprehensible data frame containing a large enough number of games which would be subsequently used for its analysis and prediction. An overview of this process will be given in the following paragraphs as a way to give the reader an overall view accompanying the motivations and limitations of such a study.

The development of this thesis grew out of a personal interest of delving deeper and improve in the applications related to obtaining and analyzing relatively big sets of data. After some time pondering options that would allow us to work in such a topic, it was decided we would aim at obtaining access to a large set of data which we could download, manipulate, analyze, and if the case allowed, forecast. Following after that it was decided R would be the programming language used as it is software meant for that purpose.

Once the software and the idea of the project were decided the focus turned into the data itself. Due to its accessibility and usability we settled on the chosen chess games data as it was openly available and it provided a wide array of options in order to be analyzed. The access to the data would be finally obtained through the use of a web browser's ability to show the source code of a website. Through this feature we were able to obtain a large set of information for the number of games we would decide to download. They had over 3 million games (even though in lots of cases the complete information for a game was not available) which suited us as this comfortably covered the scope we were aiming at.

The shape of this data after being taken as source code did not come in the most usable of ways; therefore in order to be properly analyzed using R, it would have to be left in better shape, this is where Web Scraping comes into play. We will delve deeper into the use of Web Scraping for this Thesis in Chapter 1, nonetheless it will be mentioned that it revolves around "the practice of gathering data through any means other than a program interacting with an API. This is most commonly accomplished by writing an automated program that queries a web server, requests data and then passes that data to extract needed information" (Mitchell, 2015).

Web Scraping allowed us to extract key information as: the Elo Rating of the chess players taking part in the game, who is the white and black player, all of the moves done in the game and other relevant data. Among the most used piece of information extracted from the website to perform our analysis was the Elo Rating, a "system of rating two-player games such as chess. [Arpad Elo] developed his formula and a chess rating system which was approved and passed at a meeting of the United States Chess Federation in St. Louis in 1960" (Ross, 2007).

At this point, having a "clean" data frame meant a significant first step but the core of the development was still to be done. The data would be modeled and predictions for known outcomes would be made in order to know its accuracy. Regression Analysis was used to reach this objective, two types of it to be precise. Distance-based Regression was our central point of analysis and parallel to this Logistic Regression was used in order to compare results against the former method. The Logistic Regression would model the outcomes with the Elo Ratings as the independent variable

Having three possible outcomes (chess games can finish as a victory for the white player, a victory for the black player or as a draw) and regular logistic regression able to model just two, we had to use the conditional probability properties to have the modeled outcome for three. In particular using the following result:
$\Pr(A \cap B) = \Pr(B | A) \cdot \Pr(A)$.

This property was also used for the analysis using Distance-Based Regression. The motivation for using the Distance-Based Regression comes from the fact that it seemed interesting to compare how predicting the outcome of a chess game based on a quantitative variable would fare against the prediction knowing a non-quantitative variable, in this case the chess board configuration. However, analyzing the outcomes using the chess board configuration meant using an uncommon explanatory variable and that is the possibility that Distance-based regression offered us: to model the data in presence of this type of variable. It solves the problem of predicting a scalar response variable from a non-quantitative predictor. Distances between all the games would be then computed and the matrix of all these combinations would serve as the variable. Therefore for the Logistic Regression the Elo Rating was the explanatory variable and for the Distance-based Regression this role was fulfilled by a Distance Matrix involving the chess board configurations of each game.

Centered on the Distance Based Regression, several cases were included. As at different points the board configuration can give different amounts of information, it was decided to have two scenarios: The board configuration after 25 and 40 movements. A local method involving the Distance-based Regression was also used as it models the data in a different way, as a way to compare it with the rest of the results.

The sample size for the modeling was of a total of 200, mostly due to time limitations in the execution of the program. Creation of a 200x200 Distance Matrix took an approximate 12-20 hrs (depending on the number of movements), later on improved to under 2 hrs, but still time consuming. Modeling a 200x200 Matrix using the Local D-BR took a big amount of time, however also being later improved as the creation of the

Distance Matrix. Therefore these limitations and sample sizes were set. In total having five sets of results: One with Logistic Regression, two with the Global Distance-based Regression and two with the Local Distance-based Regression.

These prediction vectors using the Global and Local D-BR would then be compared against the ones obtained using Logistic Regression. These results are found and analyzed in Chapter 3 and in the Conclusions.

2. WEB SCRAPING WITH R & CREATION OF A CHESS GAMES DATA FRAME

2.1 Diagnostics and First Try with HTML-Related Functions

As it became increasingly clear that it would be fairly complicated and time consuming to find a database that would have a big and representative set of Chess Games, research was begun in order to find a suitable way to "clean" the data, however it may come.

Even though knowledge in Web Scraping was relatively scarce from our side at the time, it seemed pretty clear that something similar was going to be needed to scrape the data and leave it as we needed to. Therefore a good way to begin and start getting acquainted with this process of cleaning the data was researching how to split a string into a character vector regardless of how it may come. Especially knowing that there is a huge amount of ways it could come and a predictable pattern was not always going to be available.

A course offered by Carnegie Mellon Universty in Statistical Computing (Carnegie Mellon University, 2014) was found and it became the first of several helpful sources of information in order to leave the data in a workable manner.

The main functions explained and used on the course were the 'grep' and 'strsplit' functions. These two functions can be considered the main features around Regular Expressions, explained in the R library as "a pattern that describes a set of strings". This pattern, as described, could be used, for example, in order to obtain a character vector from a string in a certain way.

Next up we will show some of the exercises presented in the course and worked in detail throughout this part's development in order to understand the use of these functions and as a consequence, to know how to scrape the data. Continuing from that, halfway through this chapter we will see how one of the functions (strsplit) was key in letting us achieve this. We present these examples as a way to show the road pursued in order to finally leave the data as we would need it, including comments done in the script in order to help the reader keep track of the development.

Grep. Search a String within a character vector:

Below, the example in R can be seen. A horizontal line at the start and end will denote the start and finish of the script and its relevant results.

After the execution and analysis of the scripts that will follow throughout the chapters, comments were made and are usually to be found below each line as to explain the essence of what was run and how it helped reach the desired objective of scraping the data.

```
> head(speech)

[1] "Fellow-Countrymen:"
[2] ""
[3] "At this second appearing to take the oath of the Presidential office there is"
[4] "less occasion for an extended address than there was at the first. Then a"
[5] "statement somewhat in detail of a course to be pursued seemed fitting and"
[6] "proper. Now, at the expiration of four years, during which public declarations"

## ‘speech’ is a variable containing a character vector separated
## by lines

> grep("God", speech)

[1] 34 35 41 45 47 54

## Returns the elements of the vector where this pattern (“God”)
## was found. We can see here in a relatively easy way, the first
## uses of Regular Expressions

> grepl("God", speech)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
…

## Says True or False depending on if the pattern is found in each
## element of the character vector

> speech[grep("God", speech)]

[1] "God, and each invokes His aid against the other. It may seem strange that any"
[2] "men should dare to ask a just God's assistance in wringing their bread from the"
[3] "offenses which, in the providence of God, must needs come, but which, having"
[4] "attributes which the believers in a living God always ascribe to Him? Fondly"
[5] "pass away. Yet, if God wills that it continue until all the wealth piled by"
[6] "God gives us to see the right, let us strive on to finish the work we are in,"

## Finds and shows the elements of the character vectors that have
## the pattern “God” within them.
```

At this point ‘grep’ and ‘grepl’ seemed like very interesting options for us to scrape the data with. They were the main choice considered but it was later dropped due to the finding of a more direct choice, particularly the one coming up: ‘strsplit’.

Strsplit. Split a string vector into a list of vectors

Continuing from the example from above in R. Same object, ‘speech’

```
> speech <- paste(speech, collapse=" ")
## Merge all of the elements of the vector, separated by a space

> speech.words <- strsplit(speech, split=" ")[[1]]
## Now we have the words in a list.
## Each element of the list should be a word as the space tab was
## used as a separator

> wc <- table(speech.words)
> wc <- sort(wc,decreasing=TRUE)
## Frequency table of the words, sorted

> head(wc,20)
## speech.words
```

the	To		and	of	that	for	Be	in	it	a	this
54	26	25	24	22	11	9	8	8	8	7	7
war	which	all	by	we	with	as	but				
7	7	6	6	6	6	5	5				

```
## We can see in the table above that the null string is the third-
## most-common word. Something we would not usually like to see

> names(wc)[3]
[1] ""

## And below we see how a comma is making us have two different
## words for the word ‘years’

> wc["years"]

years
3

> wc["years,"]

years,
1
```

These last two cases can be fixed if we learn how to work with text patterns and not just constants. These text patterns are what we previously referred to as Regular Expressions and later on we will start seeing how they were of great help to scrape the data.

But before we delve any deeper into the details of the use of the previously mentioned functions for this Master’s Thesis and how they helped in building our Chess Games data frame, we will start introducing the data found and considered manageable in order to model and analyze a set of Chess Games.

Reason for this is that throughout the content of this Master Thesis we will be following a chronological order of how each part was developed. At this point it had been found, thanks among other things to the content of the course, that if a big set of games was to be obtained in the form of a string, then through the use of Regular Expressions it could be worked into getting the information we needed from them.

At this point then, we had a partial objective: Manage to get the games in a string and once that was achieved, making use of Regular Expressions and other tools available in order to split it into the information we needed to. First though, we needed the data.

The search for the data started. We essentially needed a webpage that would allow us access to their database or some form of access that would let us to obtain this set of Chess Games. After some research was done, a very promising website was found: chesstempo.com.

Even though it wasn't in the form of a database or something concrete, we mention promising as it had a set of historical games and features that seemed to be of great use for us. As mentioned, the key information of the Chess Games in a script was needed so then it would be cut into a character vector, each character vector (game) would then represent a row in the data frame containing all of these games. Therefore with that in consideration, what looked promising of this website was the fact that in its source code all sorts of interesting information was there to be found. Among this set of information we could find:

- **Game id**
- Players involved
- **Result**
- **White player's Elo rating**
- **Black player's Elo rating**
- Date
- Number of movements
- Site where it was played
- An id for the type of opening
- **The historical set of movements**

In bold we find highlighted the values that were decisive (or heavily used) in our thesis' development. It's also worth mentioning that the other values could also be of great use for further analysis.

As a way to show how the raw data came, we leave below a sample of the source code of the website:

```
…
config.gamedb = true;
config.engineInitAtPageLoad = false;



config.gameid = 3;


config.initialGameData =
eval({"result":{"game_id":3,"white_id":99888,"black_id":14010,"whit
e":"Kaphle, Sebastian","black":"Bach,
Matthias","result":"w","elowhite":2010,"eloblack":2364,"date":"2007
.4.8","num_moves":65,"site":"Dresden GER","event":"8th ch-
Euro","round":"6","opening_id":49,"eco":"A00","opening_name":"Ander
ssen
Opening:General::","avgelo":2187,"tactic_id":null,"moves_lalg":["a2
a3","b7b5","e2e4","c8b7","b1c3","a7a6","d2d4","d7d6","g1f3","g7g6",
"f1d3","b8d7","c1g5","f8g7","e1g1","f7f6","g5e3","e7e5","d4e5","d6e
5","b2b4","g8h6","d1e2","e8g8","f1d1","h6f7","a3a4","c7c6","c3a2","
d8e7","c2c4","a6a5","c4b5","a5b4","b5c6","b7c6","d3b5","d7b8","a1c1
","e7b7","a2b4","c6e4","f3e1","e4f5","e1d3","f8d8","d3c5","b7e7","b
4d5","e7f8","d5c7","a8a5","c5b3","d8d1","c1d1","a5b5","a4b5","f8b4"
,"b3c5","g7f8","c5a6","b4a5","b5b6","b8a6","e2a6"],"canonical_id":n
ull,"duplicate_ids":null,"forced_load":false,"deleted":false,"canCa
che":null},"id":1});

    ...
```

Within this last piece of information we can find the values previously detailed. Certainly, this information would still need some scraping before it finds itself in a usable manner.

As a side note, it is worth mentioning that halfway through the development, efforts were made to contact a website so they would allow us downloading its database for personal use, though a firm no was given as an answer. Shortly after that, they sent an email warning us to stop accessing repeatedly the site to download information. Therefore, this episode somehow confirms that even though it took quite some effort to have the data from Chesstempo.com in a workable manner as you will see throughout this chapter, it was worth it as it did not seem likely that websites would grant us access o provide us with large samples of their Games Database.

Now, with the website found, we needed to access it through R, obtain the code and turn it to a string. As previously mentioned we had an idea of the tools required to make a piece of text with uncomfortable patterns into a character vector and we were now in possession of what seemed to be very reliable data which suited the objectives, thus what was missing was having the data in R and turning it into a string so it could be worked on and obtain the certain pieces of information we needed.

Reading through the contents of the course alongside most of the information online, we realized that even though the objective could be fulfilled (getting the source code in R and turning it into a string) most of the work would need to be done with Regular Expressions. As at the time experience was lacking in pattern knowledge for Regular Expressions and considering it looked fairly complicated, we decided to find other useful functions that could lighten the workload, especially the part related to Regular Expressions. We were aware that work with Regular Expressions would eventually come as it has been mentioned throughout the Thesis, however having a smaller set of text would reduce the workload and make the string more manageable.

It was through this course of action that the library called “rvest” was found. A library meant to easily scrape Web Pages and as they put it in its cran website, a package designed to “…download and then manipulate html”.

This looked promising and indeed it was. The library had several functions that had useful features that fitted our objectives. The main ones being:

- **html:** It would connect to the URL given and would download the content of the source code in HTML format
  class of the output: HTMLInternalDocument

- **html_nodes:** Extract pieces out of HTML documents using XPath and css selectors.
  class of the output: XMLNodeSet

- **html_text:** Extract text from HTML
  class of the output: Character

In possession of these three newly found functions, the way looked clearer. We would download the information with these html-related functions from R, turn it into nodes (and hopefully using CSS Selectors considerably reducing the lines of code) and finally turning it into text.

Therefore, now that we had these functions that showed us a concrete path into getting the source code into a string, we ran several examples in order to understand its use in a more complete manner and consequently applying it to our source code.

These were basic examples that were found and tried, to understand the use of the functions of the rvest library.

Next up, we will be taking a look at the first example. It is also worth mentioning that most of the examples make use of the ‘pipe’ operator, found in the “Magrittr” package. This pipe operator, as the word suggests, pipes a value onto the following expression. This could be seen as: x % > % f, using the pipe operator, rather than f(x).

Example 1:

```
library(rvest)
lego_movie <- html("http://www.imdb.com/title/tt1490017/")
## With the html function, R connects to the URL given within the
## string and then downloads the info with HTML format into the
## lego_movie variable. The class of this variable being
## “HTMLInternalDocument”

lego_movie %>%
  html_node("strong span") %>%
  html_text() %>%
  as.numeric()
[1] 7.8

## As we can see it’s all put together into the same instruction.
## The lego_movie variable (all of the source code of the URL given
## in HTML format) is piped in the function html_node as the first
## argument, the second being “strong span” which then proceeds to
## be used as an argument for html_text function and finally as an
## argument for the as.numeric function
```

After reading the lines of code and the comments below, what still could seem a bit confusing and not completely clear is the line where the lego_function variable is being introduced into the html_node function.

We could be asking ourselves: What is the role that the function is playing there and why not converting the HTML class variable into a text directly after having it as a HTML class variable?

The answer will be explained by showing also a piece of the source code as that way it becomes evident to the reader. Between lines 799 and 803 of the source code we find:

```
<div class="star-box-details"
itemtype="http://schema.org/AggregateRating" itemscope
itemprop="aggregateRating">
            Ratings:
<strong><span itemprop="ratingValue">7,8</span></strong><span
class="mellow">/<span itemprop="bestRating">10</span></span>
from <a href="ratings?ref_=tt_ov_rt"
title="219.009 IMDb users have given a weighted average vote of
7,8/10" > <span itemprop="ratingCount">219.009</span> users
</a>&nbsp;
```

When we notice what’s being highlighted in bold we see they match with the second argument of the function html_node. The reason for this is that this function receives as a first argument the HTML object and the second argument is the CSS selector, highlighted in bold above. It then returns the content found within the selectors.

The reason why we would use this function instead of directly appealing converting it to text is because if what we need is a very clear part of the code, as in the example, just a quick argument in the html_nodes (notice that there’s the html_nodes function that returns all of the matches while html_node returns just the first) could save us a considerable amount of time if we were to work with Regular Expression for 10.000 lines of source code for example. This way we extract only the information we need and if we may need some more scraping, Regular Expressions would be our best choice now.

Next up, we’ll review one more final example that helped us understanding the use of the rvest functions. Following after that it will be showed how it was done in our case with the Chess Games data.

Highlighted you will find both the relevant functions as the information extracted.

Example 2:

```
url <- "http://www.tripadvisor.com/Hotel_Review-g37209-d1762915-
Reviews-JW_Marriott_Indianapolis-Indianapolis_Indiana.html"

reviews <- url %>%
  html() %>%
  html_nodes("#REVIEWS .innerBubble")

## A similar procedure as last time takes place. The two strings
## used to be selectors as to bound the data are piped in the
## html_nodes alongside the previous outcome

id <- reviews %>%
  html_node(".quote a") %>%
  html_attr("id")
id

## Unlike the previous case, the difference here is that as the
## source code is so large, the bounding using the selectors is
## used twice with the addition of the html_attr function,
## returning directly the value associated with that attribute.

[1] "rn304421797" "rn303864139" "rn303764921" "rn303078076"
"rn303039437" "rn302967097" "rn302825109" "rn302521773"
"rn302307214" "rn302275247"
```

And finally below we'll leave an extract taken directly from the source code so we can appreciate how these functions work:

```
<div class="col2of2">
<div class="innerBubble">
<div calass="wrap">
<div class="quote isNew">
<a href="/ShowUserReviews-g37209-d1762915-r303864139-
JW_Marriott_Indianapolis-
Indianapolis_Indiana.html#CHECK_RATES_CONT"
onclick="ta.setEvtCookie('Reviews','title','',0,this.href);
ta.util.cookie.setPIDCookie('2246');"
id="rn303864139">&#x201c;<span class='noQuotes'>Incredible
Staff!</span>&#x201d;</a>
</div>
<div class="rating rev…
```

We first see with "reviews" and "innerBubble" the information is filtered into something smaller and finally with "quote" and "a" we filter it to just four lines of code. Within those four lines we identify the "id" attribute which is extracted with the html_attr function and we finally get the value we were looking for.

The extraction showed here occurs for several matches which is why in the code above when id is being printed we have a wide array of results. It's essentially all of the id's of the reviews being stored found within the selectors given, in www.tripadvisor.com for that particular hotel.

In this next part we will make use of these functions with the set of Chess Games information. (the rest of the job will be taken care of by Regular Expressions) and evaluate the quality of the outcome

The pattern we will follow will be the same as before. The code will be shown alongside a part of the source code and throughout the show of the script comments will be made about the steps being taken in case it is deemed necessary.

First try at extracting information from the webpage:

```
pagnumb <- 10
## 10 is an arbitrary number we chose just to see how algorithm
## goes

for (i in 1:pagnumb){
      pagweb <- paste("http://chesstempo.com/gamedb/game/",i,sep =
"")
      webxml <- html(pagweb)
      webxml_nodes <- html_nodes(webxml,"script")
      ##length(webxml_nodes)
      info <- html_text(webxml_nodes[[8]])
      info.tc <- textConnection(info)
      info.list <- scan(info.tc,what="list",sep="\n",quote="'\"")
      config[i] <- info.list[[18]]

}
```

In a step-by-step manner, the previous loop will be explained:

In the first line a “paste” action is being done which achieves two objectives, first it allows to create the string so then we can feed it to the html function, which in turn will connect and download the information (source code) from the website and the second objective is that, making the most of the fact that the Chess Games follow a pattern in the URL (after the slash / comes a number, changing that number we get a different game), we use the ‘i’ from the loop to change from game to game.

With the html function we then connect and download the source code. Once we have that, using the previously introduced html_nodes function we extract a piece of the source we are most interested through the use of the selector “script”. We get with this all of the pieces of the script that are found within the selector “script” (which we will see more in details below with an extract of the source code). As there are several matches, we would then need to manually find which node is the one that has the information we need. In this case, node number 8.

Extract of the webxml_nodes object. Specifically, the start of node number 8:

```
webxml_nodes

…
   this,
   true);
```

```
]]>
</script>
## Where the last “script” selector ends

[[8]]
<script>
## Where the next “script” selector begins. This set of information
## being the one we are most interested in

<![CDATA[
var config = new BoardConfig();
config.makeActive = true;
config.pieceSet = "merida";
config.pieceSize = 46;
config.squareColorClass = "-lightgrey";
config.showCoordinates = 1;

…

config.initialGameData =
eval({"result":{"game_id":11917,"white_id":195949,"black_id":33474,
"white":"Sibilio, Mario","black":"Caprio,
Guido","result":"w","elowhite":2362,"eloblack":2140,"date":"2008.4.
20","num_moves":33,"site":"Roma ITA","event":"Weekend Accademia A
Primavera","round":"5","opening_id":1416,"eco":"A01","opening_name"
:"Nimzo-Larsen Attack:Modern
Variation::","avgelo":2251,"tactic_id":null,"moves_lalg":["b2b3","e
7e5","c1b2","b8c6","e2e3","g8f6","f1b5","f8d6","b1a3","c6a5","g1f3"
…
```

Now the natural next step is to use the html_text function to turn that object of class “XMLNodeSet” into an object of class “character”. This is done in the following lines, however after doing this several unwanted letters and symbols appeared in our object. A example of this below:

```
info <- html_text(webxml_nodes[[8]])
## We save the relevant information from the source, identified as
## the eight node, in a variable called ‘info’

info

"\nvar config = new BoardConfig();\nconfig.makeActive =
true;\nconfig.pieceSet = \"merida\"
…
```

We can see that for every enter that appears in the XMLNodeSet, the characters “/n” now appear, symbolizing every line change through an enter from the source code.

This is solved with the function ‘scan’, which does something similar to the strsplit function. As the R library puts it, this function “Reads data into a vector or list from the console or file." And we do just that with our previous string. Now looking like this after being fed to the function scan:

```
[1] "var config = new BoardConfig();"

[2] "config.makeActive = true;"

…

[15] "config.gamedb = true;"

[16] "config.engineInitAtPageLoad = false;"

[17] "  config.gameid = 5;"

[18] "config.initialGameData =
eval({result:{game_id:5,white_id:66402,black_id:19562,white:Frolov,
Vitaly,black:Belekhov,
Dmitry,result:w,elowhite:2106,eloblack:1838,date:2008.2.3,num_moves
:35,site:Moscow (Russia),event:It (open)
(b),round:2,opening_id:49,eco:A00,opening_name:Anderssen
Opening:General::,avgelo:1972,tactic_id:null,moves_lalg:[a2a3,c7c5,
c2c3,c5c4,d2d4,c4d3,e2d3,b8c6,g1f3,d7d5,d3d4,c8g4,f1d3,g8f6,e1g1,e7
e6,c1f4,f8d6,f4d6,d8d6,b1d2,e8g8,d1c2,e6e5,d4e5,c6e5,f3e5,d6e5,f1e1
,e5f4,h2h3,g4h5,e1e7,f8e8,a1e1],canonical_id:null,duplicate_ids:nul
l,forced_load:false,deleted:false,canCache:null},id:1});"

…
```

We locate now the string from the character vector we are interested and we save it a variable we will be calling 'config'.

We can see that string 18 from the character vector just shown (the one partially highlighted) has all of the information we were looking for, therefore this is the string we will partially save into the "config" variable from the loop. This would be done repeatedly for a large enough sample so we could have the moves historic for a large set of games and with that the ability to create our data frame.

However, we can see that the string has information we are not interested in and also it can scraped and organized better than as it finds itself now. This is where Regular Expressions comes in and its use through the strsplit function.

2.2 Regular Expressions

Based on the knowledge found having Regular Expressions alongside other useful information, we were able to split the string 18 (stored in variable called 'config') into the several parts of information we need.

Therefore we shall see some the main points related to Regular Expressions as to present the basics of Regular Expressions to the reader and afterwards we will apply this knowledge onto what we need from our development in order to reach our planned objective.

The main features of Regular Expressions are:

- Special symbols are used in order to say what patterns will be looked for
- As we saw in the previous example with the word “God”, this pattern will be searched within the entire script, returning the position or splitting the string, depending on the function used
- Escaping is used. Meaning that for special characters that could have two meanings, a escape operator is used to say that the character coming up has a different meaning than the usual one

The content of the grammar and rules of Regular Expressions is quite large therefore we won’t present it all. We will focus on the basic definitions and some of the rules in order for the reader to understand the basic concepts before the upcoming example.

The example serves the purpose of introducing the use of regular expressions in a practical set:

Example 3:

```
> speech <- paste(speech, collapse=" ")
## We keep working with “speech”. This variable contains the text
## we showed between page 1 and page 2 of the current Chapter

## Collapsing all lines into one big string; separating them with a
## space

> speech.words1 <- strsplit(speech, split=" ")
## We save all of the words using the ‘space’ between them as
## separator


> head(sort(table(speech.words1)))
## Frequency table of the words sorted

speech.words1
 -    "the  "Woe  absorbs    accept           achieve
 1    1             1          1              1          1

## Problem is that some words have quote symbols, commas and even
## the space is considered. We are going to need Regular
## Expressions to “clean” those words


> speech.words2 <- strsplit(speech,
split="(\\s|[[:punct:]])+")[[1]]
## Detailed explanation will be below

> head(sort(table(speech.words2)))
```

```
absorbs    accept achieve against agents aid
 1         1     1      1     1         1

## We can see that using the patterns in the strsplit function, the
## words are now cleaned
```

The example from above shows us a case of how to use regular expressions for that text. We will explain a bit more in detail how that pattern works in order to get those words without any symbols, punctuation marks or others from the text. We will explain now the pattern used, from the most outer part to the most inner part.

First important thing is, the instruction after the argument ‘split’ must be within quotation marks.

Second, we see that everything is within parenthesis and at the end there is a + sign. The plus sign is considered as something called as ‘quantifiers’ which are considered greedy. Meaning they will take as many matches as they find. In this case the plus symbol denotes ‘1 or more matches’.

Third, the use of an ‘or’ being utilized in the middle of the statement. It’s shown by the following symbol ‘|’. Essentially the same symbol used in R for the logical ‘or’. In this case meaning, “match this first pattern OR the second pattern given”

Fourth, the patterns as such. At the left part of the ‘or’ symbol we find “ \\s ” and at the right part “ [[:punct]] ”. The “ \s ” means space, so every time the search finds a space it will be split. The second “ \ ” that’s placed in front symbolizes a escape. Lastly [[:punct]] means all of the punctuation marks (ASCII and non-ASCII) will be used to split the string. Therefore if we use this same pattern to split the following string: “The wasp stung John. He cried”, the pattern will find the first quotation marks and separate the string based on that. Then the spaces up to where “John” is written. Then it will find the dot, the two subsequent spaces and finally the last quotation mark. All of these symbols will be used as separators and the string would be split.

Now we will present our own dataset and analyze the pattern used and the final result after it is run.

In essence we wanted to split the following string into a character vector where every character would have <u>only</u> the information deemed interesting or of importance.
Reviewing the string once again:

```
[18] "config.initialGameData =
eval({result:{game_id:5,white_id:66402,black_id:19562,white:Frolov,
Vitaly,black:Belekhov,
Dmitry,result:w,elowhite:2106,eloblack:1838,date:2008.2.3,num_moves
:35,site:Moscow (Russia),event:It (open)
(b),round:2,opening_id:49,eco:A00,opening_name:Anderssen
Opening:General::,avgelo:1972,tactic_id:null,moves_lalg:[a2a3,c7c5,
c2c3,c5c4,d2d4,c4d3,e2d3,b8c6,g1f3,d7d5,d3d4,c8g4,f1d3,g8f6,e1g1,e7
e6,c1f4,f8d6,f4d6,d8d6,b1d2,e8g8,d1c2,e6e5,d4e5,c6e5,f3e5,d6e5,f1e1
,e5f4,h2h3,g4h5,e1e7,f8e8,a1e1],canonical_id:null,duplicate_ids:nul
l,forced_load:false,deleted:false,canCache:null},id:1});"
```

We realize that every piece of important information is preceded by some sort of title, for instance: game_id:5. We would be interested in the number 5 only and as in that case, for all cases, therefore being able to discard the title preceding it.

So we create a split pattern based on all of those titles we want to discard and use as separators, among other details. The result looking like this:

```
strsplit(config,split="game_id:|,white_id:|,black_id:|,white:|,blac
k:|,result:|,elowhite:|,eloblack:|,date:|,num_moves:|,site:|,event:
|,round:|,opening_id:|,eco:|,opening_name:|,avgelo:|,tactic_id:|,mo
ves_lalg:[[]|[]],canonical_id:|,duplicate_ids:|,forced_load:|,delet
ed:|,canCache:|},id:|})")
```

As you can see, the pattern created was done mainly by putting the titles of all the information we deemed important to split (alongside the punctuation marks we also wanted to get rid of as commas and colons) except in the most important part of the script, in the historical of movements. We can see that between the title of this set of movements and the information we needed there was a bracket, which in turn we also wanted to discard. This caused a problem because if we just wrote: |,moves_lalg:[|, R would consider this bracket as if we were opening a range (as part of the original meaning of a bracket within RegEx). After some research was done, it was recommended to put brackets that appeared literally within our string, also within brackets, which is why we have the start and finish of those brackets also within brackets.

After running it, we go through something as previously seen to something like this:

```
…
 [5] "Frolov, Vitaly"
 [6] "Belekhov, Dmitry"
 [7] "w"
 [8] "2106"
 [9] "1838"
…
"1972"
 [19] "null"
 [20]
"a2a3,c7c5,c2c3,c5c4,d2d4,c4d3,e2d3,b8c6,g1f3,d7d5,d3d4,c8g4,f1d3,g
8f6,e1g1,e7e6,c1f4,f8d6,f4d6,d8d6,b1d2,e8g8,d1c2,e6e5,d4e5,c6e5,f3e
5,d6e5,f1e1,e5f4,h2h3,g4h5,e1e7,f8e8,a1e1"
 [21] "null"
…
```

Of special interest for our analysis will be number 7 (white wins, draw or black wins), number 8, elo of the white player, number 9, elo of the black player and number 20, the historic set of movements for that game.

Now that we’ve finalized scraping the data, the procedure to save all of this information into a data frame is what should follow. Nonetheless, an improvement was found in order not to work with html-related functions and thus have a shorter and more efficient script.

This improvement was made while searching for functions that would convert directly the content of the webpage into a (very big) string. Even though this could add more work since it would require the use of Regular Expressions to find the info through thousands of lines of code, if we could find a function that would go straight from source code to string and having it somehow bounded (as we did with html_nodes) or listed so we wouldn't have to make deep changes into the regular expressions pattern already developed, it would mean an improvement.

Essentially the basis of the improvement would be on the fact that instead of downloading the source code into an object of class HTMLInternalDocument, afterwards turning it into a XMLNodeSet and finally having it as a string, we would go directly to string.

While doing this search, the library RCurl was found, which as they define it in Cran is a library that "Provides functions to allow one to compose general HTTP requests and provides convenient functions to fetch URIs, get & post forms, etc. and process the results returned by the Web server.". On the face of it, it looked as a very interesting library that could have functions that would help us get to our objective.

After some more research within the library was done, the function "getURL" was found. A very practical function defined as one that: "… downloads one or more URIs (a.k.a. URLs). It uses libcurl under the hood to perform the request and retrieve the response.". After some quick examples we realize that this is the function we needed, mostly because not only we skip the converting it to HTML step, but because no changes are needed to the strsplit.

This because after using the function 'scan', the string is made into a list and every element of the list is (approximately) one line in the source code. This way when looking at the source code and finding the lines of code we needed, we can relatively quick locate the part we are interested using this source code line and therefore limit it to the portion we are interested to use the same strsplit.

The code was adapted to the use of this function and it ended up something like this:

```
	pagnumb = 4
## 4 is a arbitrary number chosen just to see how algorithm goes

for (i in 1:pagnumb){
	pagweb <- paste("http://chesstempo.com/gamedb/game/",i,sep =
"")
	txt <- getURL(pagweb)
	info.tc <- textConnection(txt)
	info.list <- scan(info.tc,what="list",sep="\n",quote="'\"")
	config[i] <- info.list[[715]]
all_info  <-
strsplit(config,split="game_id:|,white_id:|,black_id:|,white:|,blac
k:|,result:|,elowhite:|,eloblack:|,date:|,num_moves:|,site:|,event:
|,round:|,opening_id:|,eco:|,opening_name:|,avgelo:|,tactic_id:|,mo
ves_lalg:[[]|[]],canonical_id:|,duplicate_ids:|,forced_load:|,delet
ed:|,canCache:|},id:|})")

}
```

Only six lines of code needed within the loop

2.3 Creation of the Chess Games Data Frame

Finally then, as we have the part of the information we need in a more efficient script, the next step is to store it in a comprehensible way, for example in an R data frame, that way the data will be more accessible. We say accessible because in this way we don’t have to go online and scrape the data every time we want to use it, just one loop to run it as to then having it locally.

Our focus turned now into saving all of this information in a data frame, but in doing so, several questions and challenges arise, as for instance:

- Is a data frame the best structure for our task? Compared to matrix, list, others
- How many games should we have (how many rows)
- How much information for every game should we have (column)
- If we do it for 1.000+ games, it will require R to connect 1.000+ times to the website plus all of the computations it will do within the loop. Is that doable for us and/or the computers/laptops we possess?

Related to the first question, we decided on a data frame as we saw it was the most versatile structure we could use. With matrix we would have needed only numerical values (something we couldn’t fulfill as most of the data came as a string) and a list didn’t offer as many advantages as a data frame.

The amount of information we decided on was approximately 10.000 rows. We did this because even though we knew that an analysis of 10.000 rows using Distance-based regression would be very challenging for our computers/laptops we decided on the safe side as we didn’t know for sure how many rows we would end up using. In the case of the columns, something similar was the argument for our answer; we decided to save every bit of usable information, for even though we had an idea that we would only use elo ratings, winner and historical of moves, it could have been of use later on. Therefore the expected outcome (data frame) from this should have approximate dimensions of 10000 rows and 25 columns.

For the last question facing us, the answer is that we would try it like this and in case it were too slow, a reduction of dimensions would have taken place.

Now, to start tackling the issue of saving the data in the data frame what we thought was the best option was creating a loop (something similar to what we presented earlier) and in every iteration saving the corresponding Chess Game.

We would first create the data with an empty array and the names we are planning for each column and in the loop each line would be inserted. The first lines of code for the creation of the data frame would be the following:

```
GamesDatabase <-
as.data.frame(x=array(dim=c(1,25)),stringsAsFactors=FALSE)

names(GamesDatabase)<-c("game_id", "white_id", "black_id", "white",
"black","result", "elowhite", "eloblack", "date", "num_moves",
"site", "event", "round", "opening_id" ,"eco", "opening_name",
"avgelo", "tactic_id", "moves", "canonical_id", "duplicate_ids",
"forced_load", "deleted", "canCache", "one")

> dim(GamesDatabase2)
[1]  1 25

> class(GamesDatabase)
[1] "data.frame"
```

We have the data frame for the Chess Games, in a proper format and with proper names for each column. Therefore we would now run a function where each line of code is a game and saving it as a row after it's been split.

But before we finally create the data frame with what we've achieved so far, there are four details that we would like to address. We will proceed to mention them, explain them and lastly we will show the code used to create the data frame alongside some final comments. These are:

- As not all of the matches of chesstempo.com have the historical set of movements, we will proceed to save the ones that do. The ones that don't will be recorded as NA and subsequently deleted.
- After performing the strsplit we will get a list with all the information needed. It will be "unlisted" (turned into a character vector) and the first and last element dropped as they contain no valuable information.
- The row names will be the id game number.
- 10.000 connections to chesstempo.com are quite a lot. It could happen that the wifi/internet connection being used crashes after creating most of the data frame ruining the run. Therefore samples of 1.000 will be created and saved after done and finally concatenated.

Considering these details as addressed, we will proceed to show the script used to create the data frame. It will be very similar to what we have seen throughout this chapter with the additions mentioned above.

```
samplesize<-1000
sample.total<-sample(3700000,10000)

for (i in 1:10)
{
fsample <- sample.total[( ((i-1)*samplesize)+1 ):(i*samplesize) ]

for (j in 1:samplesize)
{
```

```
pagweb <- paste("http://chesstempo.com/gamedb/game/",fsample[j],sep
= "")
hist2 <- pagweb
txt2=getURL(hist2)
info.tc2 <- textConnection(txt2)
info.list2 <- scan(info.tc2,what="list",sep="\n",quote="\"")
partx <- info.list2[[715]][1]
partx2 <- info.list2[[714]][1]

if(substring(strsplit(partx,split="eval")[[1]][2],3,7)=="resul")
 {

trial<-
strsplit(partx,split="game_id:|,white_id:|,black_id:|,white:|,black
:|,result:|,elowhite:|,eloblack:|,date:|,num_moves:|,site:|,event:|
,round:|,opening_id:|,eco:|,opening_name:|,avgelo:|,tactic_id:|,mov
es_lalg:[[]|[]],canonical_id:|,duplicate_ids:|,forced_load:|,delete
d:|,canCache:|},id:|})")

GamesDatabase[j,]<-unlist(trial)[-c(1,27)]
rownames(GamesDatabase)[j]<-paste("Game
id:",strsplit(partx2,split="gameid = |;")[[1]][2])
}
}
filename <- paste("fichero_",i,".RData",sep="")

save.image(file=filename)
}
```

We can see some friendly faces in the transcript from above, as the getURL function to download the games, as also the strsplit function to get rid of bothersome separators and split the string.

Most of the features are from the details mentioned above the code. The “if” is written so games that have no historical information are not saved. This is done through a word found in the source where if it has information it says “result” otherwise it says “error”

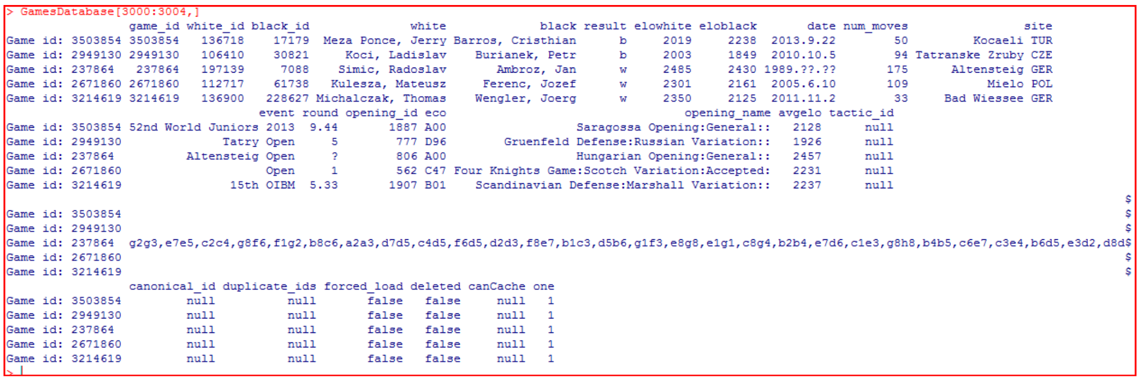
The row names are created with the games’ id extracting this piece of information using strsplit from the source code.

The samples are created taking randomly 10.000 numbers (total sample size) from 3.700.000 numbers. This is done because the website, after reviewing it, has approximately 3.700.000 games, though not all of them have the historical set of movements. Then we create a variable called “fsample” that will store the first 1.000 values which will be used in the loop and then saved into an image. After that, fsample will have the second 1.000 values and so forth. This is done to prevent losing all the data if the internet connection breaks down and the loop ends. Each data frame of 1.000 values will be saved with a different name using the iterator to avoid overwriting.

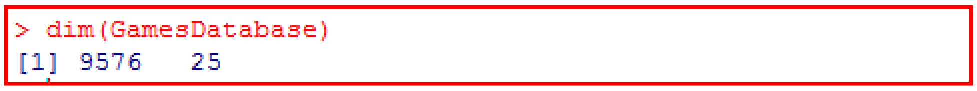
The final size of the data frame was 9576. Such a high number was reached after doing a first set of 2.000 and subsequently a second set of 10.000. This was done, because unfortunately as expected, the connection broke down after creating 2.000 rows and several tries were required to finally achieve the final size.

Lastly, all of the data frames were loaded and concatenated using rbind, the NA rows mentioned earlier deleted along with the duplicates using the duplicated function from R. Duplicates existed because even though we used the sample function so it would generate random numbers without repetition, as there were two sets (the 2.000 and the 10.000), there were cases of duplicated random numbers, hence duplicated game_id's. These cases were deleted and the final data frame was one with dimensions of 9576 x 25.

Here a sample:



```
> GamesDatabase[3000:3004,]
                 game_id white_id black_id               white             black result elowhite eloblack       date num_moves                 site
Game id: 3503854 3503854   136718    17179  Meza Ponce, Jerry Barros, Cristhian      b     2019     2238  2013.9.22        50         Kocaeli TUR
Game id: 2949130 2949130   106410    30821     Koci, Ladislav    Burianek, Petr      b     2003     1849  2010.10.5        94 Tatranske Zruby CZE
Game id: 237864   237864   197139     7088    Simic, Radoslav       Ambroz, Jan      w     2485     2430 1989.??.??       175      Altensteig GER
Game id: 2671860 2671860   112717    61738   Kulesza, Mateusz     Ferenc, Jozef      w     2301     2161  2005.6.10       109           Mielo POL
Game id: 3214619 3214619   136900   228627 Michalczak, Thomas     Wengler, Joerg      w     2350     2125  2011.11.2        33     Bad Wiessee GER
                                   event round opening_id eco                                 opening_name avgelo tactic_id
Game id: 3503854 52nd World Juniors 2013  9.44       1887 A00                  Saragossa Opening:General::   2128      null
Game id: 2949130              Tatry Open     5        777 D96        Gruenfeld Defense:Russian Variation::   1926      null
Game id: 237864          Altensteig Open     ?        806 A00                  Hungarian Opening:General::   2457      null
Game id: 2671860                    Open     1        562 C47 Four Knights Game:Scotch Variation:Accepted:   2231      null
Game id: 3214619               15th OIBM  5.33       1907 B01    Scandinavian Defense:Marshall Variation::   2237      null
                                                                                                                                                        $
Game id: 3503854                                                                                                                                        $
Game id: 2949130                                                                                                                                        $
Game id: 237864  g2g3,e7e5,c2c4,g8f6,f1g2,b8c6,a2a3,d7d5,c4d5,f6d5,d2d3,f8e7,b1c3,d5b6,g1f3,e8g8,e1g1,c8g4,b2b4,e7d6,c1e3,g8h8,b4b5,c6e7,c3e4,b6d5,e3d2,d8d$
Game id: 2671860                                                                                                                                        $
Game id: 3214619                                                                                                                                        $
                 canonical_id duplicate_ids forced_load deleted canCache one
Game id: 3503854         null          null       false   false     null   1
Game id: 2949130         null          null       false   false     null   1
Game id: 237864          null          null       false   false     null   1
Game id: 2671860         null          null       false   false     null   1
Game id: 3214619         null          null       false   false     null   1
>
```



```
> dim(GamesDatabase)
[1] 9576   25
```

3. CHESSBOARD CONFIGURATION & DISTANCE MATRIX

3.1 Creation of the Chessboard and its related functions

This chapter as the title suggests will focus on the creation of the chess board, its configuration and subsequent creation of the Distance Matrix. This Distance Matrix will look to relate two Chess Games based on a feature that will be measured between these two games and later expand it into all of the matches present in the matrix. Therefore having a matrix where the diagonal are zeros and every other entry is the distance between the two paired games

There are several requirements in order for us to actually get to that Distance Matrix, so before moving any further let's do a recap of what we have achieved so far in order to know the tools we have.

In the previous chapter we found a website from which we could download through its source code a big set of Chess Games (9.576 to be precise) that had important information within them. This information being very useful as it would help perform the analysis we have as an objective: Carry out a Distance-based Regression using this Distance Matrix, perform a prediction and compare it to the actual results. Parallel to this do the analogous procedure with Logistic Regression but using a different explanatory variable. The reason we will use a different explanatory variable is related to the nature of each Regression Analysis we are using. While the Logistic Regression makes use of a binary variable as a dependant variable and a continuous variable as an independent one, with Distance-based Regression this is not the case. We would need for this a Distance Matrix that serves as the explanatory information. This is what makes this tool so particular. It allows us to model and predict a set of values with no need of a quantitative variable, just distances between these categorical variables. Though the method will be explained a bit more in detail later.

Thus, in order for us to correctly model the data we need to be able to feed the models with the correct variables (both explanatory and dependant) so the results are reliable. After studying the data we had found, and the analyses we were to do, we reached the conclusion that we would use the result of the games as the dependant variable (what we will aim to predict) and for the Logistic Regression's analysis the Elo Rating as explanatory variable. There were other candidates considering the data we had at hand (opening_id, number of movements, who is the white player) but we decided on the Elo Rating (specifically the difference of Elo's between the players) as we considered it to be the variable that condensed the biggest amount of information of a game.

Looking back then to the tools that we have at hand, we know that as part of the data frame that we have created and whose construction we detailed in Chapter 1, we have already both pieces of information we need for the Logistic Regression Analysis: The result and the Elo Ratings. In principle almost all of the games of the data frame (if not all) should have this information available.

On the other hand though, for the Distance-based Regression, we not only need the result of the match, but also the Distance Matrix between the games that would serve as explanatory information, which we are going to have to create ourselves.

In consequence, and picking up on our initial question, what tools do we already have in order to create this matrix?

What seems to be the most fitting piece of information we have in order to create the Distance Matrix is with no doubt the historical set of movements of each match. This historical set of movements opened us a wide range of possibilities for us to create this Distance Matrix. Some examples of the uses we could have given to this historical set of movements:

- Through the set of movements analyze what pieces had been captured in that game for both players, assign a numeric value for each piece, adding a total for each player and subsequently subtract the number against the number of the same player (same player meaning white or black) from another game assigning it as the Distance.
- Count the movements (and the spaces) each piece has moved for each player and adding them, assigning a weight for each piece depending on their value. Afterwards subtract with the other game's value for the same player.

However, when delving deeper, we realize that these two ways of computing a distance between two games would not completely make the most out of this method. The reason for this is that the main point in Distance-based regression is making into a distance a variable that has no quantitative value but a high degree of information. In those cases presented earlier, we create a quantitative value to subsequently create a distance, nullifying the method's purpose as a quantitative value was already attainable for each game.

Hence, what could we have with this historical set of movements that provides us with a very important set of information, but one that it doesn't have a quantitative value? Or putting it in a different way, what could we create with this historical set of movements and the features needed to be explanatory information for each Chess Game?

For instance, if we show the following movements from a Chess Game:

b2 to b3
e7 to e5
c1 to b2
b8 to c6
e2 to e3
…

In case we have some notions of Chess, when reading it, in what do we automatically think?

It would probably be fair to say that for most people that have had some experience with Chess, the automatic mental step after reading a set of movements would be to imagine the Chess Board and locate with each movement the piece to where its movement ends, essentially the chess board configuration. This is what we will plan to do with our historical set of movements. We will track each movement on the Chess Board for each game with a function designed in R and following this course of action we will stop at a determined time (after X amount of movements) to which the function will return the chess board configuration at that particular point.

In order for us to reach the completion of this Distance Matrix, we need a historical set of movements for each game, something we have already (shown in Chapter 1). However not only this is necessary, as we have determined that an efficient way of going from the set of movements to the Distance Matrix requires us to create a chess board in which we will track the movements to a fixed point where the current state of the board will be obtained. Once in possession of the chess board configuration for each game a distance between the white player's pieces in game 1 will be measured against the white player's pieces in game 2. The same would be replicated for the black player's pieces. This will be explained more in detail later on, first showing the steps to get there (the creation of a chess board in R and its related functions) and some basic notions of chess.

We will use this opportunity to explain the basic features of the game of Chess so the reader who is unfamiliar to the game can keep on understanding the development by being introduced to the basic rules of the game and, on the opposite, the one who is familiar to the concepts already could use this as a quick review.

Next up we will show an extract taken from the World Chess Federation website that explains the most basic rules and features of the game (World Chess Federation (FIDE), 2014):

1.1 The game of chess is played between two opponents who move their pieces on a square board called a '**chessboard**'. The player with the light-coloured pieces (**White) makes the first move**, then the players move alternately, with the player with the dark-coloured pieces (Black) making the next move. A player is said to 'have the move' when his opponent's move has been 'made'.

1.2 The objective of each player is to place the opponent's king 'under attack' in such a way that the opponent has no legal move. **The player who achieves this goal is said to have 'checkmated'** the opponent's king and to have won the game. The opponent whose king has been checkmated has lost the game.

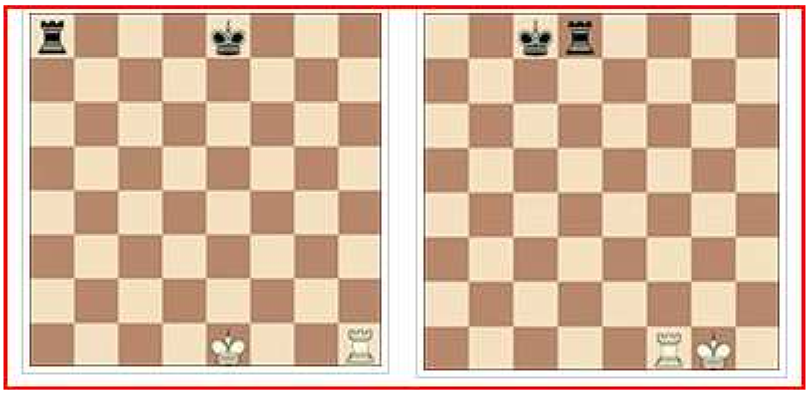
1.3 The chessboard is composed of an **8 x 8 grid of 64 equal squares** alternately light (the 'white' squares) and dark (the 'black' squares).

The rules extend way beyond this point. Nonetheless we will stop here, complementing though with a quick explanation of the pieces found in the chess game, the movement possibilities and a useful image taken also from the World Chess Federation, showing how the initial set should be placed.

Each player begins the game with:

- one king, moves a maximum of one square per turn in whichever direction the player decides.
- one queen, can cover the amount of squares the board lets it, in either a diagonal, horizontal or vertical direction. But just one direction per turn.
- two rooks, move horizontally or vertically as the maximum amount of squares of the board allows it
- two bishops, similar to the rook but diagonally
- two knights, who move in an L-shaped manner. Only piece allowed to "go over" another piece in order to reach its final square.
- eight pawns, move as the king at a maximum rate of one square per turn though only to the front. With two exceptions, the first movement can be made covering two squares and when in position of capturing another piece, then one square diagonally.

There's one last final movement called "Castling". It involves one rook and the king and it allows the king to move two steps either to its right or left with the rook doing the opposite. In the following image we will see a first and final position after doing "Castling"



In these images brought from the website of the World Chess Federation (2014) we can appreciate the Castling in more detail. In the first image we have as the pieces should be found for when they want to perform Castling. The black player will perform what's called a queen-side castling and the white player the king-side castling. In the second image the final positions are shown. It is important to mention that even though it is two movements, it is viewed as just one movement during the game. This action is included in the R function created for the movement tracking which we will see afterwards.

Finally before moving to R and its corresponding chessboard configuration, we will show the initial setting of a chess game:

The script coming up next is split into three parts.

- First part: Creation of the chessboard
- Second part: Two functions used to expand the data frame containing the chessboard-related information are introduced
- Third part: a key function to this development is shown. It allows obtaining every piece’s position (the chessboard configuration) after a set number of movements.

The **first part** is the creation of the data frame where each piece belongs to a row and where the name’s row is the piece’s name. That position (within the data frame’s row structure) remains unchanged throughout the game. The data frame will have three columns, the first column will show the piece’s position just as it is read in a chess board (letters from ‘a’ to ‘h’ and numbers from 1 to 8) and the accompanying two other columns will show this coordinate position where each column shows its corresponding value of the coordinate. Finally the chessboard has 64 squares but in our case it is the 16 pieces that will have a value assigned.

For example, an entry in the data frame for a white rook in d1 would look like this after running it in R:

```
position firstCoord secndCoord
…
rookWa      d1       4          1
…
```

The objective is that after a movement the position column updates with the new position and therefore the other two columns also update.

The complete script for this part is the following:

```
ChessGame <- data.frame(position =
c('a2','b2','c2','d2','e2','f2','g2','h2','b1','g1','a1','h1','c1',
'f1','d1','e1','a7','b7','c7','d7','e7','f7','g7','h7','b8','g8','a
8','h8','c8','f8','d8','e8'), stringsAsFactors = FALSE)
ChessGame$firstCoord <- letter2Num(ChessGame$position)
ChessGame$secndCoord <- number2Num(ChessGame$position)
rownames(ChessGame)[1] <- "pawnWa"
rownames(ChessGame)[2] <- "pawnWb"
rownames(ChessGame)[3] <- "pawnWc"
rownames(ChessGame)[4] <- "pawnWd"
rownames(ChessGame)[5] <- "pawnWe"
rownames(ChessGame)[6] <- "pawnWf"
rownames(ChessGame)[7] <- "pawnWg"
rownames(ChessGame)[8] <- "pawnWh"
rownames(ChessGame)[9] <- "knightWb"
rownames(ChessGame)[10] <- "knightWg"
rownames(ChessGame)[11] <- "rookWa"
rownames(ChessGame)[12] <- "rookWh"
rownames(ChessGame)[13] <- "bishopWc"
rownames(ChessGame)[14] <- "bshopWf"
rownames(ChessGame)[15] <- "queenW"
rownames(ChessGame)[16] <- "kingW"
rownames(ChessGame)[17] <- "pawnBa"
rownames(ChessGame)[18] <- "pawnBb"
…
rownames(ChessGame)[31] <- "queenB"
rownames(ChessGame)[32] <- "kingB"
```

The ellipsis denotes the rest of the pieces. All of the code was not shown as we felt it was redundant since it is quite repetitive. The white pieces are shown and the initial and final lines of the black pieces.

Here, when we notice the code, what we are doing is initializing what will be our Chess Game. All of the entries going into the position column of the data frame are the initial positions of a Chess Game. Consequently these positions are matched with its corresponding piece when we introduce the row names

Then we have two new columns whose values are defined through a couple of functions that feed from the position. These functions will be shown coming up next and with it the start of **this second part**. The aim of these functions is obtaining the coordinates already separate as we will need them like that later on.

The method will be the same. The script will be shown and the parts that are deemed complicated will be explained.

```
number2Num <- function(position,pos=2){
num<-as.numeric(substring(position,2,2))
return(num)
}

letter2Num <- function(position,pos=1){
  ch <- c("a","b","c","d","e","f","g","h","i")
  moves_prim <- unlist(lapply(position,
  function(tira.char){
          which( ch == substring(tira.char,pos,pos) )
                     }
                      ))
  return(moves_prim)
}
```

These two functions work in a very similar way. Its objective we quickly mentioned and it's that we want to obtain in separate columns the coordinate value of each piece. Also worth mentioning is that the part of the coordinate that has a letter will be converted into a number.

For the 'letter2Num' function we create a vector containing what we want to match to the receiving vector. So if we are going to match letters, the vector will contain the first 8 letters of the alphabet, plus an extra one we will be using for pieces that are captured, which means that when a piece is captured it will be assigned the coordinate 'i9'.

After this a variable will be created and will be fed within its computations a character vector containing all of the coordinates. Within this variable, the "lapply" function is the predominant feature. This variable is then returned and contains all of the alphabetic part of the coordinates converted into a number.

For the numbers is much simpler, taking just the second part of the coordinate as it is already a number.

We have now a chessboard configured in such a way that it contains the position each piece finds itself into. But that is only the start, since now we need to be able to track how any set of movements could change (and will change) this configuration. With this latest point we start **the third part** of this script

Therefore, in order for this board to update based on new information, we will evidently need the complete information for this update and also up to which point will the function use the information. In other words, and more applied to our case, we will need to receive the set of movements and the number of moves that we will consider before it stops and returns the current configuration of the board.

Next up the code of the function used and below the script the comments made.

```
chessconfig <- function(moves_vector, moves_nr)
{
      ## Cut the movement vector into the number of movements
      ## received by the function using the commas as separator
      moves_pred <- substring(moves_vector, 1, (moves_nr*5)-1)
      moves_ind <- strsplit(moves_pred,split=",")

      for (i in 1:moves_nr){
            for (j in 1:32){
                  ## Splitting each movement in start and finish
                  moves_prim <- substring(moves_ind[[1]][i],1,2)
                  moves_segu <- substring(moves_ind[[1]][i],3,4)

                  ## Castling/Enroque
                  if( ChessGame$position[32]=='e8' &
moves_prim=='e8' & moves_segu=='c8' )
                        {ChessGame$position[27]='d8'}
                  else if(ChessGame$position[32]=='e8' &
moves_prim=='e8' & moves_segu=='g8' )
```

```
                       {ChessGame$position[28]='f8'}
                 else if(ChessGame$position[16]=='e1' &
moves_prim=='e1' & moves_segu=='c1' )
                       {ChessGame$position[11]='d1'}
                 else if(ChessGame$position[16]=='e1' &
moves_prim=='e1' & moves_segu=='g1' )
                       {ChessGame$position[12]='f1'}

                 ## Piece-by-Piece Comparison and subsequent
                 ## assigning

      if(moves_segu==ChessGame$position[j]){ChessGame$position[j]<-
"i9"}

      if(moves_prim==ChessGame$position[j]){ChessGame$position[j]<-
moves_segu}
                       }
                       }
           ChessGame$firstCoord <- letter2Num(ChessGame$position)
           ChessGame$secndCoord <- number2Num(ChessGame$position)
           ChessGame

}
```

The first argument the function receives (moves_vector) will be the set of movements for that game, which we have already extracted in Chapter 1. The second argument (moves_nr) will be a number given by us which denotes up until what point we want to update the chessboard. The numbers we will be using are: 25 and 40. We selected these numbers as the average of a Chess Game length in our data frame is approximately 80 movements and we wanted to have enough information to do a prediction without overstepping. Below a small sample of the script showing the average length of a Game in our data frame

```
> mean(nchar(GamesDatabase$moves))
[1] 396.7045
## Considering that each movement in the vector is 4 characters
## long plus the comma, by dividing it by 5 we get approximately 80
## movements per game.
```

In order for it to be clear, a movement, as you will see in the example below, will be when a piece is moved, regardless of whose turn is it.

The movements' vector comes complete, from the first movement to the last which means we need to cut it and discard the movements we will not use, at least in this case.

An example of the movements' vector we have is next up:

e2e4,e7e6,d2d4,d7d5,b1d2,c7c5,g1f3,g8f6,e4d5,e6d5,f1b5,c8d7,b5d7,b8d7,e1g1,f8e7,d4c5,d7c5,f1e1,e8g8,d2f1,c5e4,c1e3,a8c8,e3d4,e7c5,f1e3,d8d6,d1d3,f8d8,a1d1,d6b6,d3b3,b6b3,a2b3,h7h5,c2c3,a7a5,e3f5,g8f8,f3e5,g7g6,d4c5,e4c5,f5d4,d8d6,d1a1,b7b6,a1d

1,c8e8,f2f3,f6d7,d4b5,d6e6,e5d7,c5d7,e1e6,f7e6,c3c4,d7f6,c4d5,e6d5,b5c3,e8d8,g1f2,f8e7,f2e3,e7d6,e3d4, ...

This is exactly how a movement's vector looks like taken directly from the data frame made in Chapter 1. In this case it has more than 60 movements so let's imagine we only need the first 20.

What we do in the script is using the substring function (needs a string, a starting point and an ending point) to cut until the point we need. We notice that each movement is exactly four digits long plus a comma long so we multiply the number of movements times 5 and we subtract 1. This last subtraction would discard the last comma. After performing this operation we would end up with something like this:

e2e4,e7e6,d2d4,d7d5,b1d2,c7c5,g1f3,g8f6,e4d5,e6d5,f1b5,c8d7,b5d7,b8d7,e1g1,f8e7,d4c5,d7c5,f1e1,e8g8

Now to have it in a more usable manner it will be split into the twenty different movements using the strsplit function as we can see in the script above. After that we will obtain a list with twenty values.

After having this we see in the script that a double loop is created. The first loop is done to cover all of the movements and the second is done to cover all of the chessboard's pieces, hence the number 32.

Consequently we take the movement, for example "d7d5", we split it into two, considering the first two digits as the first part of the movement (current position of the piece) and the second two digits the final part of the movement (end position of the piece).

Subsequently we perform a piece-by-piece comparison to check which piece was located in the initial part of the movement, we locate it and finally we assign the final part of the movement as its current location. This is done also considering the Castling movement which is the first 'if' after the two loops.

The Castling movement is also considered when analyzing the next step for a piece. The reason why we will consider a Castling movement differently is because all of the information related to it is not present in the movement. It's a 2-in-1 movement and just one of the movements is shown. Therefore we need to identify when we are dealing with Castling and perform the movement in the pieces partaking in this operation.

As we will see in an example, the movement that is shown in the source code (and thus in our script) is the one that the King is doing. This is fortunate for us because this is a unique movement for the King to be doing, which means that every time we spot the King doing this, we will know for sure that it is Castling. Reason for it is that in Castling the King moves two squares, when his maximum length of movement is one. Thus knowing that if he moves two squares is because he is taking part in Castling alongside the Rook. Even more recognizable if he is doing it from his initial position (the only place he is allowed to do Castling).

An example of how the King’s Castling movement is the one being recorded in the source code, we can see it in the string of movements shown above and again here: (e2e4,e7e6,d2d4,d7d5,b1d2,c7c5,g1f3,g8f6,e4d5,e6d5,f1b5,c8d7,b5d7,b8d7,e1g1,f8e7, d4c5,d7c5,f1e1,**e8g8**).

If we notice the twentieth movement we can see it is a castling. It’s the King’s first movement and he ends up two squares to his left. However the Rook’s movement is not recorded thus we need to do it ourselves.

We create an ‘if’ statement where we try to find what type of Castling is occurring. We do this because depending on the type of Castling the Rook will move one way or the other. We will show the part of the script where this occurs so the reader has a chance to get a glimpse of this once again:

```
if( ChessGame$position[32]=='e8' & moves_prim=='e8' &
moves_segu=='c8' )
                              {ChessGame$position[27]='d8'}
                      else if(ChessGame$position[32]=='e8' &
moves_prim=='e8' & moves_segu=='g8' )
                              {ChessGame$position[28]='f8'}
                      else if(ChessGame$position[16]=='e1' &
moves_prim=='e1' & moves_segu=='c1' )
                              {ChessGame$position[11]='d1'}
                      else if(ChessGame$position[16]=='e1' &
moves_prim=='e1' & moves_segu=='g1' )
                              {ChessGame$position[12]='f1'}
```

The ‘if’ is divided into four scenarios and depending on what type of movement the King is doing, the end position assigned to which of the four rooks will vary.

Once we have this solved we will show the last part of the function. It deals with the assignment once we have identified what piece is being moved.

```
if(moves_segu==ChessGame$position[j]){ChessGame$position[j]<-"i9"}

if(moves_prim==ChessGame$position[j]){ChessGame$position[j]<-
moves_segu}
```

The first line deals with when a piece has been captured. It checks if the second part of the movement is the current state of any piece (meaning it’s been captured by another) and in case it finds one, it will assign the ‘i9’ value to it. A made up value by us to identify when a piece is in a captured state.

In the second line is where the true assignment occurs. It compares the movement extracted from the vector to each piece. When one has been found it overwrites the end position over the initial one.

Finally in the script the two other columns are updated as we have our position column with the proper values after the comparison and assignment.

Before considering this part of the chapter related to the chessboard configuration as ended, we will show a script that also creates a chessboard in order to perform some chess analytics. In this case a different procedure is carried out, mainly as it creates classes for the board and pieces, unlike in our case.

```
…

ChessGame <- function(f=NULL) {
   o <- list()
   class(o) <- "ChessGame"
   if (!is.null(f))
          loadPGN(o, f)
   o$board <- ChessBoard()
   o$whitePieces <- array(o$board[, 1:2])
   o$blackPieces <- array(o$board[, 7:8])
   o$current_position <- 0
   return(o)
}

loadPGN <- function(game, f) {
        pat  <-  "\\[(.*) \".*\"\\]"
        e <- readLines(f)
        meta <- e[grep(pat, e)]
        a <- getMeta(meta)
        moves <- e[grep(pat, e, invert=TRUE)]
        moves <- moves[grep("^$", moves, invert=TRUE)]
        getMoves(moves)
}
…
```

The above pasted script was found in the website R-Bloggers (2011) published by user called ‘enguyen’ who mentions performs data mining and analysis of chess games. In order to carry out his analyses he first creates the Chess Games as shown above alongside other features not shown due to its size.

He first creates different functions for different levels involved in the game as: the pieces, the board and the game, unlike in our case where we dealt with it by creating a data frame. Also for each function he creates a class of that object. In the script we see the ChessGame function, which in essence analyzes if what is received is not a null (presumably a website) and if it’s actually not null a function called loadPGN is called. This function we can see downloads the content of this website by using readLines and then making use of Regular Expressions in order to “clean” it.

It is interesting to see the use of Regular Expressions and Web Scraping in order to obtain the relevant information from each match, denoting in this way an apparently common method between this solution and ours.

Now that we have shown an overview of how other people might approach the problem of Chess Game analytics and created the chessboard and a function that allows us to extract the configuration at any point we want from the game, we will introduce the

Distance Matrix, a matrix that computes distances between the chess games and by doing so, making use of what has been done so far in this chapter: the chessboard and its related functions.

3.2 Distance Matrix

In the most general way what the Distance Matrix will look to attain is to pair two Chess Games and to calculate a distance between them, then with the help of a loop, replicate this computation to all of the possible pairings between the Chess Games that will be studied. In the end having a matrix where each crossing of a row and a column denotes a distance between those two games.

We have mildly explained what this Matrix looks to achieve, now though there is the question mark surrounding the 'how'. We know that will compute a distance between two games and turn that into a matrix, but how is this distance going to be computed?

First we would need to cover one of the technical aspects of the distance. By this we mean which distance we will use. We gathered the three most known distances which are the L1, the L2 and the L-Infinite, whose formulas we will see below assuming a distance between a point in space and the origin:

$$L1\ Distance = (|x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n + y_n|)$$

$$L2\ Distance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

$$L\infty\ Distance = \max\{|x_1 - y_1|, |x_2 - y_2|, \dots, |x_n - y_n|\}$$

We can see in Figure 2.1 and 2.2, taken from Wikimedia Commons (2015), how they show us two graphical examples of these distances.

In Figure 2.2 we see how the unitary distance from the origin for each metric is shown and in Figure 2.1 the green denotes the L2 Distance, the sum of all the fragments of the blue, yellow or red line is the L1 Distance and one of the two red lines being the L-Infinity distance (as it's the maximum between the distances from either dimension).

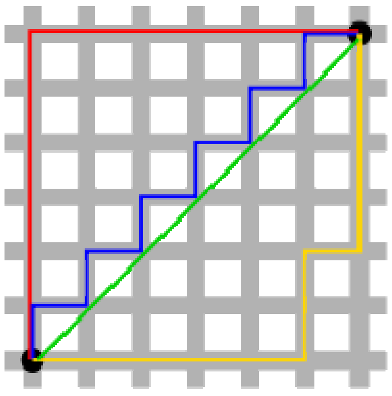


Fig 2.1

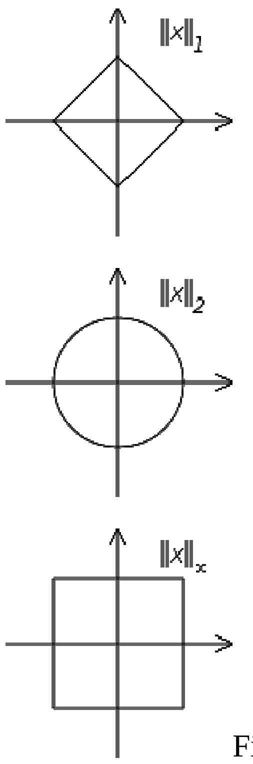


Fig 2.2

After some analysis we realized that by using a different distance than the Euclidean it was hard to tell beforehand if it would help in achieving better results, so we decided to work with the most common distance, the L2.

After settling the Distance we would use, now was the time to decide what we were going to use to create a distance between two games or two configurations.

For this case, several possibilities were analyzed:

- Measuring the Euclidean distance of a piece's position in game 1 (for instance) against the same piece's position in game 2. Doing this for the 32 pieces and then adding them all up.

- Something similar as in the previous case but separating the bishops, for example, into white bishops and black bishops. Thus having two 2x2 matrices. Then applying a procedure where we take the minimum and we discard the two pieces involved and we repeat, storing these minimum distances.

From these two possibilities we discard the first one as at some point it can occur that the two same pieces (but from different games) end up having two very distant distances and it wouldn't be a very reliable way of measuring it as there is no need to do so in such a fixed manner.

As the first option seems very static and not so reliable, we decide to go for the second option that provides some more flexibility and could be seen as more reasonable as it takes the minimum distance between two pieces alike.

Now we have the essence of what we would need to start coding the script for this Distance Matrix. We know with which distance we are going to work and we know how we are going to do it (at least theoretically). Next up the script for its creation will be shown and its explanation.

Due to the big size of the code related to the Distance Matrix, we will show the most relevant parts of it. In order to see the complete script used to create the Distance Matrix, we invite the reader to the Appendix where this part of the development can be found complete.

The matrix was created with a 200x200 size. The reason for this is that the Distance Matrix was meant to be done for several cases (after 25 and 40 movements as with our case) and having bigger dimension required a big time investment, mostly during the first tries as the function was not working efficiently yet.

We will start with the **initialization of the matrix and the computations done for the pawns**. Also the creation of the test data frame and the test sample will be shown

```
dimPrueba<-200
Distancias25<-matrix(data=0,nrow=dimPrueba,ncol=dimPrueba)

hmm<-40

lngth<-dim(GamesDatabaseUpd)[1]

## ----------------------
set.seed(456)

randSample <- sample(lngth,200)

GamesDatabaseUpd2<-GamesDatabaseUpd[randSample,]
## Test Sample

## ----------------------
set.seed(987)

randSample2<-sample(lngth,200)

GamesDatabaseUpd3<-GamesDatabaseUpd[randSample2,]
## Main Sample


for (j in 1:dimPrueba  )
{game_j_hmm<- chessconfig(GamesDatabaseUpd3$moves[j],hmm)

for (k in 1: dimPrueba )
{game_k_hmm<- chessconfig(GamesDatabaseUpd3$moves[k],hmm)

if(j==k)
{Distancias25[j,k]<-0}
## We would not consider this ‘if statement’ when
## working with the Distance Matrix between the 2 games
## as the diagonal won’t be 0
```

```
else
{

pnsBcsJg1<-game_j_hmm[1:8,]
pnsNgsJg1<-game_j_hmm[17:24,]
pnsBcsJg2<-game_k_hmm[1:8,]
pnsNgsJg2<-game_k_hmm[17:24,]

coordsBcs1<-as.numeric(paste(pnsBcsJg1$firstCoord,
pnsBcsJg1$secndCoord, sep=""))
coordsBcs2<-as.numeric(paste(pnsBcsJg2$firstCoord,
pnsBcsJg2$secndCoord, sep=""))
coordsNgs1<-as.numeric(paste(pnsNgsJg1$firstCoord,
pnsNgsJg1$secndCoord, sep=""))
coordsNgs2<-as.numeric(paste(pnsNgsJg2$firstCoord,
pnsNgsJg2$secndCoord, sep=""))

. . .
```

The first lines are basic. We create the matrix with a certain amount of rows and columns, in this case 200.

After that using two set seeds we create what we will use as sample and test sample. The Distance Matrix of the sample will be used to create the model and subsequently the distance matrix between the sample and the test sample will be used to predict.

Two loops are then created to compute and assign the values. It could be seen graphically as one moving through the columns and other moving through the rows.

In the line below as to not waste effort, since we know that two games will have the exact same configuration when they are the same, we will know that the distance between them will be zero. Thus the 'if' statement is created denoting this possibility and assigning a value for when we detect it, otherwise we do the normal procedure where the distance is computed (else case).

This procedure of distance computing we will start with the pawns.

The first four lines are where it can be seen we make use of the first part of this chapter related to the chessboard configuration and the function associated to it, 'chessconfig()'. An object is called (game_h_hmm or game_k_hmm) which makes use of the previously mentioned function. This function then obtains the board configuration after a set amount of games, in this case denoted by the variable 'hmm'.

As seen, the pawns are separated into four categories, the white pawns from game 1, black pawns from game 1, white pawns from game 2 and black pawns from game 2. In the first case we take the first eight values as they represent the white pawns from the chessboard configuration and the black pawns are located in the rows between the 17th and 24th. In the following two lines we do it with the other chess game.

Afterwards the two numerical coordinates are combined and coerced into a numeric form. This step of making sure of having the coordinates in a numeric form is done as we will compute the distances based on these values.

Next part will deal with the computation of the distances between all possible combinations.

```
vectorDB<-
round(
ifelse(
(substring(expand.grid(coordsBcs1, coordsBcs2)[,1],1,1)=='9' &
substring(expand.grid(coordsBcs1, coordsBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsBcs1, coordsBcs2)[,2],1,1)=='9' &
substring(expand.grid(coordsBcs1, coordsBcs2)[,1],1,1)!='9')
      , 17 ,
      sqrt(
(as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,2],1,1)))^
2+
(as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,2],2,2)))^
2)),2)

vectorDN<-
round(
ifelse(
(substring(expand.grid(coordsNgs1, coordsNgs2)[,1],1,1)=='9' &
substring(expand.grid(coordsNgs1, coordsNgs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsNgs1, coordsNgs2)[,2],1,1)=='9' &
substring(expand.grid(coordsNgs1, coordsNgs2)[,1],1,1)!='9')
      , 17 ,
       sqrt(
(as.numeric(substring(expand.grid(coordsNgs1,coordsNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsNgs1,coordsNgs2)[,2],1,1)))^
2+
(as.numeric(substring(expand.grid(coordsNgs1,coordsNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsNgs1,coordsNgs2)[,2],2,2)))^
2)),2)
```

It is split into parts, first one for the white pawns and second for the black pawns.

It is important to highlight the use of the function expand.grid which serves to create all possible combinations between the pieces involved.

An example in a simpler context below will make clearer how this function works:

```
> al<-c("c8","b7","e5")
> mn<-c("c3","b2","e1")
> expand.grid(al,mn)
  Var1 Var2
1   c8   c3
2   b7   c3
3   e5   c3
4   c8   b2
5   b7   b2
```

```
6   e5   b2
7   c8   e1
8   b7   e1
9   e5   e1
```

This is important as it will save us the bother of creating a loop to replicate all possible combinations within this type of piece. This way we have all possible combinations already listed so we can proceed to calculate the distance. Also worth highlighting that once we have the vector in such a shape as printed above (after expand.grip) we could recognize the first dimension of the first piece as the first digit of the element in the first row, the second dimension of the first piece as the second element in the first row and so forth.

Before calculating the distances though, if we notice we have an 'ifelse' statement when we start, this is done due to the captured pieces. It makes sense to have a standard distance between a captured and a playing piece as the position of the playing piece should not affect the outcome, Therefore we will identify the times a captured piece and a non-captured piece are being matched and subsequently assign a standard distance which will be the approximate length of the diagonal of a chessboard (11,35) multiplied by 1,5, resulting in 17 approximately, all of this if we consider 1 the length of a square. After doing so we can proceed to compute the Euclidean distance of the rest of the pieces.

As we can see we have every other possible combination within white and black pieces (each on its side) already done. Now we will compute the Euclidean distance using the help of the substring function. We use it as we need to subtract the two coordinates belonging to the first dimension and the two coordinates belonging to the second dimension as part of the procedure required in the formula.

Following this, we round up to two decimals of all the computed numbers and they are stored into a vector. This vector will contain all of the possible distances between all of the white pawns of game 1 against game 2. The same is done for the black pawns.

In this next part we will see this vector turned into a matrix and then taking the minimum values off of this matrix to create the final vector that will contain the (minimum) distances between the 8 white pawns of game 1 vs the 8 white pawns of game 2, alongside the black pawns in a different vector.

```
matrixDN<-matrix(vectorDN, nrow=8, ncol=8, byrow=FALSE)
matrixDB<-matrix(vectorDB, nrow=8, ncol=8, byrow=FALSE)

vectorDistanciaPeonesBlancos<-rep(0,8)
vectorDistanciaPeonesNegros<-rep(0,8)

for (i in 1:8)
{
      if(length(matrixDB)==1 || length(matrixDN)==1)
      {
            vectorDistanciaPeonesBlancos[i]<-matrixDB
            vectorDistanciaPeonesNegros[i]<-matrixDN
      }
      else
      {

            vectorDistanciaPeonesBlancos[i]<-min(apply(matrixDB, 2,
            min))
            matrixDB<-matrixDB[-which(apply(matrixDB, 1,
            min)==min(matrixDB))[1],
            -which(apply(matrixDB, 2, min)==min(matrixDB))[1]]
            vectorDistanciaPeonesNegros[i]<-min(apply(matrixDN, 2,
            min))
            matrixDN<-matrixDN[-which(apply(matrixDN, 1,
            min)==min(matrixDN))[1],
            -which(apply(matrixDN, 2, min)==min(matrixDN))[1]]
      }

}
```

We first convert this vector of distances into a matrix. Afterwards a loop is created. The aim of this loop is to locate the minimum distance of the matrix, as it would represent two pawns closest to each other from different games, and to store it in a vector. Next we would identify the pawns involved, delete their corresponding row and column and proceed again until all the matrix has been gone over completely.

In case the minimum distance happens in more than one case, the first one is the one used. We repeat this the same amount of times as pawns are there for every color and we end up with a vector that stores the minimum distances between the pawns of two matches.

This is done also for the knights, rooks and bishops with the same procedure. Only change is that instead of having an 8x8 matrix, it is done over a 2x2 matrix.

The only case where it slightly changes is for the queen and king. As there isn’t a matrix since there is only one queen and king by color by game, the assignment is done directly with no need of loops as we will see here:

```
vectorDistanciaReyNegras<-
round(
      ifelse(
(substring(expand.grid(coordsReyNgs1, coordsReyNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsReyNgs1, coordsReyNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],1,1)!='9')
      ,17,
       sqrt(
(as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,2],2,2)))^2))
      ,2)
```

The assignment to the final vector (vectorDistanciaReyNegras, in this case for the black king) will be done just after doing the calculations as there is no need to take a minimum.

Finally, the distance calculation is done. It will be a weighted sum between all of the vectors. The weights will be used based on what was found in RBloggers (2015) in an article about Chess Analytics called in the world of chess as: Chess Piece Relative Value with the following values: (♟:1, ♞:3, ♝:3, ♜:5, and ♛:9). Pawn is 1, knight and bishop 3, rook is 5 and queen is 9. The king doesn't have a weight as this is done to see how a player compares against its rival, ignoring the King's position. As our case is different we assign the King (due to its importance) a higher value than every other piece, being this 12.
Therefore obtaining one final value that represents the distance between these two games. Finally, this is replicated for the amount of times needed, and when done the Distance Matrix is created.

There will be 2 distances matrix. One calculated for the chessboard configuration after 25 movements and the second one after 40 movements.

Each of these matrices will serve to analyze the Distance-based regression from different perspectives, as the growth of the number of movements should represent more reliable information.

4. REGRESSION ANALYSIS

4.1 Analysis of the Chess Data using Logistic Regression

The final chapter has been reached and with it the methods to model and predict using the structures and data we have obtained so far.

We have built a Games Database containing, among many other things, the result of the game (white wins, black wins or draw), a chess board configuration alongside other functions and most of this in order to reach a Distance Matrix.

Two conditions were imposed onto the Games Database. One condition was done from a necessity that the game had to fulfill in order for the analysis to be coherent and another from an assumption we made in order to make the predictions a bit more challenging.

The first condition was made as we needed games from the Database that would have more than 40 movements. As we would need the same games for the two matrices, it would have not made sense to have a game with incomplete information for one of the cases, hence a minimum of 40 movements was deemed a constraint for all games selected. In second place we considered that we needed to limit the Elo ratings. The Elo Ratings as such were not a problem but the subtraction of white Elo Rating minus black Elo Rating, when too big in absolute value, could have been too decisive in terms of predicting.

Once the constraints were put in place, the number of rows of the Games Database was reduced from 9576 to 2645 and subsequently two random samples of 200 numbers were taken from those 2645 to create two new data frames of 200 rows each. The first one was used to model the Regression being studied and the second one was the one we predicted. We will call this second sample, the one used to see how model predicted, as the test sample. One seed number was used for each sample.

With the Games Database reduced and the Distance Matrix created based on those 200 games, the pieces were put in place to start the analysis.

We will first tackle the Logistic Regression Analysis, also giving an overview of how the model fits the whole data frame and in this way confirming that for a big elo difference the prediction is less challenging. Continuing from this last part, the focus would change to the Distance-Based Regression. After the review and explanation of the methods, a comparison of the results will follow.

We will start then with the Logistic Regression Analysis. As we know, Logisitc Regression allows us to model a dependant variable, which will be categorical, based on explanatory variables. It is based on the notion that when we need a value as a function of x between 1 and 0, linear functions can’t provide that as they are unbounded unlike the logit function:

$$\frac{1}{1+e^{-(\beta_0+x\cdot\beta)}}$$

As in our case we have three different outcomes: white wins, black wins or draw, a straight-forward use of the logistic regression assuming the dependent variable as a binary one was not possible. We had to look for alternatives in order to be able to model and predict the outcome of the game.

Two candidates were the possible solutions for this. We could use the ordinal logistic regression or the logistic regression using a binary variable as outcome (white vs. non-white) and then within the non-white repeating the regression for draw vs. black.

The ordered logistic regression is in essence a logistic regression with the distinctive feature that its possible outcomes follow an order. If for instance you are looking to model the difficulty level that a course might have (1 to 4, from the easiest to the hardest) based on variables as for instance, how many people have approved, the amount of exams, among other possible variables, this could be modeled with the ordered logit as the dependent variable clearly follows an order.

The alternative to this however was using the regular logistic regression for binary dependent variables with the help of the properties of conditional probability.

In our case we would first build the model for the probability of white victories vs. non-white victories branding the white victories as 1 within the outcome vector and the non-white victories as 0. Once this is done, we would take the non-white victory cases and model within it black vs. draw and after that thanks to the properties previously mentioned, we would be able to compute the probability for a black victory for all cases. More detail for the modeling of black victories below:

$$\begin{aligned} Pr(B) = Pr\ (B \cap (B \vee D)) &= \boldsymbol{Pr(B|B \vee D)} * Pr(B \vee D) \\ &= Pr(B|B \vee D) * (1 - Pr(W)) \end{aligned}$$

🟥 What we would model in the second part after computing white vs non-white.

Finally to obtain the probability of draws we would subtract 1 minus the probability of white victories and minus the probability of black victories.

Technically they were both valid options and options with which we could obtain reliable results. Nonetheless we settled for the second option (regular Logistic Regression) as we realized it was our best option in order to be coherent. This, because for Distance-based Regression working with three outcomes is not possible, unlike with the Ordinal Logistic Regression. Also it is worth mentioning that even though this option would be less direct and would require a bit more work, we could work with it better as we had more knowledge about the regular Logistic Regression than about the Ordinal Logistic Regression.

Once this was settled we ran the tests for the whole data frame. We will show the results as a way to see how the model fits the whole data and why we decided to force the data frame to fulfill the two conditions previously mentioned.

First we attached three new columns into the data frame. The first one would be the difference of Elos (white minus black), the second one would be white minus black but in absolute value. This is not used in the analysis but it is used as a tool to work with the data frame with some extractions of data. And the third one would be a Boolean that would have TRUE as a value when it was a white victory and false if not. We will start by showing the script and results.

```
elodiff <- abs(as.numeric(GamesDatabase$elowhite)-
as.numeric(GamesDatabase$eloblack))
## Creation of the new column containing the difference in absolute
## value of the Elos

elodiff2 <- as.numeric(GamesDatabase$elowhite)-
as.numeric(GamesDatabase$eloblack)
## Creation of the new column containing the difference of Elos.
## The one we will mainly be using

GamesDatabase <- cbind(GamesDatabase, elodiff2)
GamesDatabase <- cbind(GamesDatabase, elodiff)
## Attaching these 2 new column into the data frame

GamesDatabase$white <- (GamesDatabase$result == 'w')
## Creation of the third column. The Booleans showing True when
## it’s a white victory

logrefw <- glm(white ~ elodiff2, family = "binomial", data =
GamesDatabase)
## Fitting of the model using the difference of Elos

plot(elodiff2,white)
## Plotting the Booleans based on the Elo difference

points(na.omit(elodiff2),logrefw$fitted.values,col=6)
## Fitted values of the model based on the Elo difference
```

The first part of the script we already mentioned: the creation and binding of the three new columns. In the second part we model the data through the use of the glm function and afterwards plot the results.

The outcome plot we will show below with a short explanation for every line in the chart:

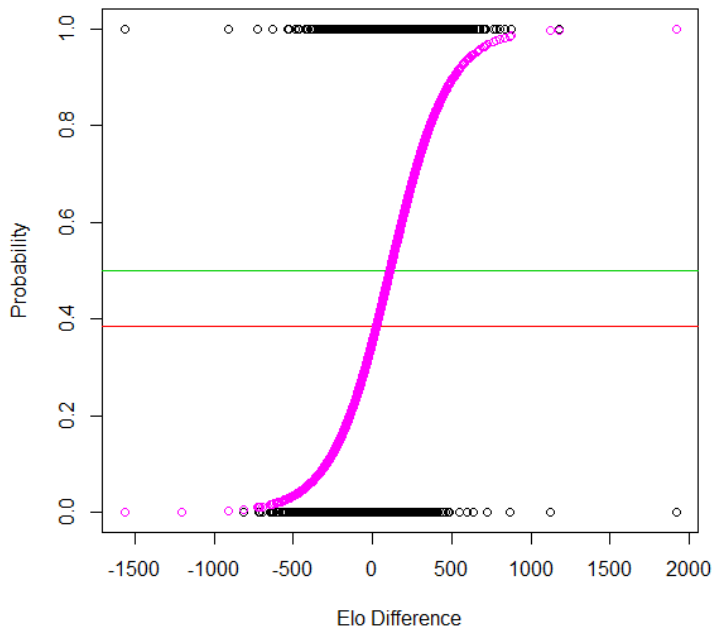


Image 3.1: Graphical representation of the Logistic Regression's fitted values based on the game's Elo Difference between the white and black player (purple curve) using two reference lines, 0.5 (green line) and proportion of white victories (red line)

More in detail about the plot:

- The **black** points found only in the 1 and 0 horizontal lines are all the Boolean values from the newly added column (GamesDatabase$white).

- The red line is the proportion of white victories within the whole set.

- The green line marks 0.5. Values above it are considered to be a white victory and below either a draw or a black victory.

- The purple line is the fitted values from the regression when the explanatory variable is the elo difference without the absolute value. It is shown here how the model considers a big (bigger than zero) difference to be clearly linked to a white victory (Y=1) and viceversa.

After seeing some important features of the data and model in the plot, below the summary of the regressions will be shown.

```
> summary(logrefw)
## Elo difference as explanatory variable

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.5976650  0.0247667  -24.13   <2e-16 ***
```

```
elodiff2     0.0054617  0.0001638   33.34   <2e-16 ***
```

From this summary we have highlighted in bold several of the things we consider relevant.

First off when we observe the coefficients we see that we have a considerably small p-value therefore able to reject the null hypothesis that the coefficients have no effect over the dependent variable.

Second, we see a considerably small coefficient for elodiff2. This is just a mere observation to mention it’s due to the fact that the y-axis has numbers from the range 0-1 as we know, while the x-axis has the majority of values within the range -1500,1500. Hence, a small coefficient appears as a way to compensate.

Third point and final is related to the value of the coefficient that accompanies no variable. We will see what would be the Elo Difference required to cross into the white victory part or better said the minimum Elo Difference needed to predict a white victory. When we substitute the two coefficients alongside a Y value of 0,5 (denoting this way the minimum probability required to predict a white victory) and find the x associated with this value, we find the value 109,5. This meaning that a white victory will be predicted starting at this value.

Now we will show some results after predicting based on a large difference of Elos. This will help understand why we decided to crop the sample by imposing one of the constraints.

```
> trtr<-na.omit(GamesDatabase[elodiff>500,])
## We take from the data frame the rows that have a valid elo
## difference and bigger than 500. elodiff is the difference in
## absolute value

> trtrl<-
logrefw$fitted.values[which(na.omit(GamesDatabase$elodiff)>500)]
## We select the corresponding fitted values from the model with
## the same features as above. We use the model with elodiff2 as
## explanatory variable.

> trtrp<-apply(cbind(trtrl,1-trtrl),1,which.max)
## we create a vector rating as 1 or 2 every probability from
## above. 1 when a white victory is predicted, 2 otherwise

> table(trtr$result,trtrp)
   trtrp
      1    2
  b   2    42
  d   5    6
  w   57   6
## We run a table to compare the true results against the predicted
## results.
```

As we have seen in the comments below each line, we first select the rows that are higher in elo difference than 500, same with the fitted values. We then mark with 1 or 2 the one with the higher probability of occurring and later we do a table.

The interpretation of the table is the following: 1, the prediction of the white victory is done 64 times (2 + 5 + 57) of which 57 were correct (89% accurate). On the other side, 2 predicts when it's a draw or black victory. This is done 54 times (42 + 6 + 6) of which, 48 are correct (89% accurate). We can see here that for a big difference in Elo values, the Elo difference is quite good as an explanatory variable to model.

With an Elo difference greater than 250 the results were the following:

```
     1          2
  b  52         435
  d  82         125
  w  537        63
```

White victories prediction accuracy:	537 / 671 (80 %)
Non white victories prediction accuracy:	560 / 623 (90 %)

The percentage of accuracy is slightly lower compared to before but still high.

Considering then, that a large difference between the Elo values might be too decisive information when predicting, especially if these are results that will be compared against a Regression Analysis based on the board configuration which might not be so informative as the Elo difference, we decided to settle for the threshold of a Elo difference of 75. Imposing an Elo difference of 75 as a condition would reduce the sample to a third but we would still have a relatively big data frame if we know that we will need maximum 500 elements (due to computer capabilities when computing the Distance Matrix and its regressions, though we later settled for 200). Therefore we chose the 75 threshold for the Elo difference as it would guarantee a large enough size sample and relevant information without giving too much of it as an Elo difference of 500 will do.

Below the size of the sample for the case where elo difference is smaller or equal to 75 (approximately a third) and the results when predicting with the whole sample

```
> sum(na.omit(GamesDatabase$elodiff<=75))
[1] 3008
## Size of the sample when elo difference smaller or equal to 75

## And the table below, prediction accuracy when predicting for the
## whole sample
         1    2
  b      287  2199
  d      522  2320
  w      1508 1792
## White victories accuracy with the whole sample (65 %)
## Non white victories accuracy with the whole sample (71 %)
```

As we have covered now in relative detail the decision to crop the initial data frame and why, we will go into the Logistic Regression Analysis and the results needed to compare against the Distance-based Regression. Most of the script shown above was also used in this next part so we may skip part of it.

As mentioned in the beginning, first the modeling is done based on the first sample of size 200 and then the predictions are made for the sample called test sample. Following the first part where it's done for the white vs non-white victories we will go into the second part, containing the modeling of the elements of the sample that are non-white victories. Finally after this the table of results will be shown in a similar manner as above. A table comparing true results vs. predicted, incorporating the use of the conditional probability properties.

```
plot(GamesDatabaseUpd2$elodiff2,GamesDatabaseUpd2$white, xlim=c(-
100,100) , xlab='Elo Difference', ylab='Probability')
## GamesDatabaseUpd2 is the test sample with number of rows 200.
## These appear in black in the plot below

logrefw3 <- glm(white ~ elodiff2, family = "binomial", data =
GamesDatabaseUpd3)
jj<-predict(logrefw3, GamesDatabaseUpd2, type="response")
points(GamesDatabaseUpd2$elodiff2,jj, col=3)
## object ‘jj’ is where predictions for white vs non-white victory
## are stored. logrefw3 was created using the first sample
## and now in ‘jj’ we predict for the second sample.

## In general, plotting of fitted points based on Elo difference.
## In green in plot below

abline(h=sum(GamesDatabaseUpd2$white)/length(GamesDatabaseUpd2$whit
e),col=2)
## Proportion of white victories within the test sample. Red line

abline(h=0.5, col=4)
## Marks 0.5. Above this point a white victory would be predicted
## against a non-white victory
```

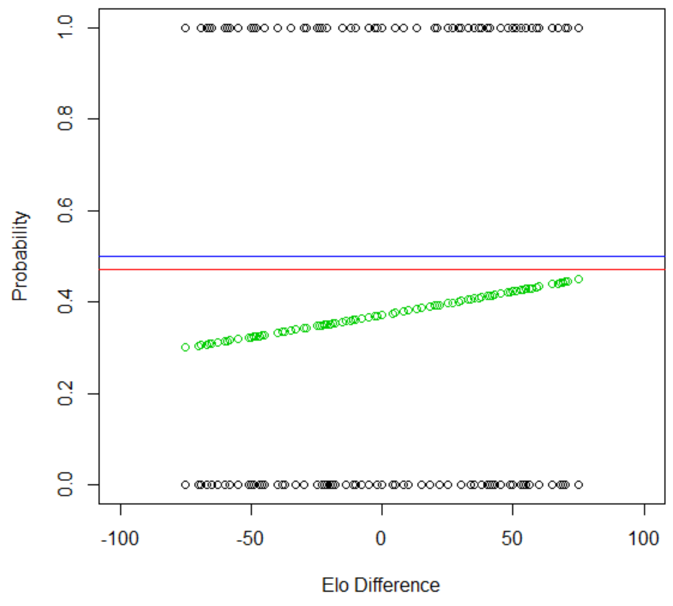


Image 3.2: Graphical Representation of the prediction of the test sample (greem) using two reference lines, the 0.5 mark (blue) and the proportion of white victories for the test sample (red)

If we notice when we used the whole sample (Image 3.1), the green line behaves similar for the range of the x-axis considered (-75, 75) to this case.

Coming up, the creation of the model for non-white victories is shown.

```
DrawBlackUpd3 <- (GamesDatabaseUpd3$result == 'd' |
GamesDatabaseUpd3$result == 'b')
## Boolean that identifies when it’s either a draw or a black
## victory

## GamesDatabaseUpd3 is the sample in which we are doing
## our model. On the other hand GamesDatabaseUpd2 is the test
## sample, on which we will be performing our predictions.

DrawBlackUpd3.df <- GamesDatabaseUpd3[DrawBlackUpd3,]
## We extract the values from the data frame where those booleans
## are TRUE

DrawBlackUpd3.df$black <- (DrawBlackUpd3.df$result=='b')
## We create a Boolean to brand when entry is a black victory

logrefdb3 <- glm(black ~ elodiff2, family = "binomial", data =
DrawBlackUpd3.df)
## We model the data based on cases where they are only black
## victories or draws

kk<- predict(logrefdb3, GamesDatabaseUpd2, type="response")
```

```
plot(GamesDatabaseUpd2$elodiff2),kk,, xlab='Elo Difference',
ylab='Probability',ylim=c(0,1),col=4)

points(na.omit(GamesDatabaseUpd2$elodiff2),(1-jj),col=6)
```

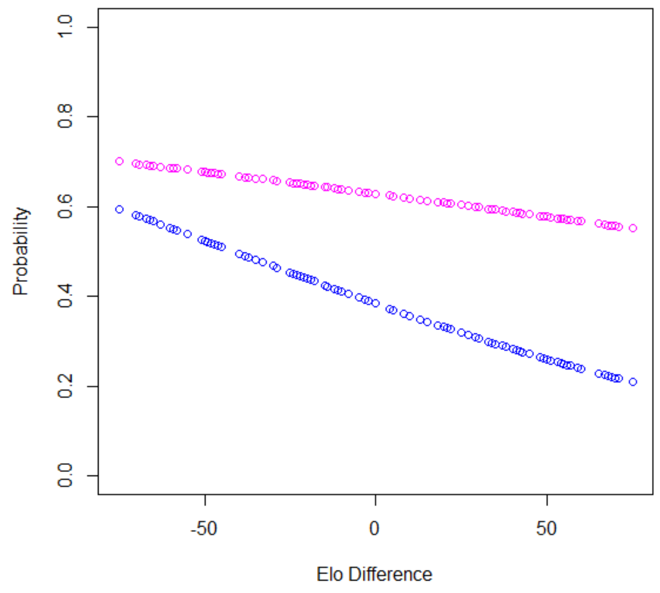


Image 3.3: Probability of a black victory for the test sample assuming it is either a draw or a black victory (blue line) vs. probability of either a draw or a black victory (purple line)

In the script above a new data frame is created to model the non-white victory entries. We first brand as true with Booleans when it's a black victory or draw to subsequently extract the rows where this happens. When this is done we create a new Boolean to determine if it's a black victory, this value being the one we will use when modeling it as a Logistic Regression. Finally we plot the results.

The purple curve is the probability of the result being either a draw or a black victory from the previous case while the blue one is the probability of being a black victory knowing it's either a draw or a black victory. It was chosen to show both of them as a way to graphically compare. Both cases shown are the predictions for the test sample.

Now the last part of the script related to Logistic Regression and the results afterwards. During this next part is where the conditional probability properties are used.

```
## Predicting with the previously created model the sample test.
Some of this
## was previously used for the chart
jj <-predict(logrefw3, GamesDatabaseUpd2, type="response")
kk <- predict(logrefdb3, GamesDatabaseUpd2, type="response")

pr.ww <- jj
pr.bb <- kk*(1-jj)
pr.dd <- 1-(pr.ww+pr.bb)

pred.result2 <- unlist(apply(cbind(pr.bb,pr.dd,pr.ww),1,which.max))
table(GamesDatabaseUpd2$result, pred.result2)

pred.result2
>
      1     2     3
  b   14    25    3
  d   13    40    11
  w   17    62    15

plot(GamesDatabaseUpd2$elodiff2,           pr.ww,           xlim=c(-
80,80),ylim=c(0,0.7))
points(GamesDatabaseUpd2$elodiff2, pr.bb, col=3)
points(GamesDatabaseUpd2$elodiff2, pr.dd, col=4)

boxplot(pr.ww~GamesDatabaseUpd2$result)
boxplot(pr.dd~GamesDatabaseUpd2$result)
boxplot(pr.bb~GamesDatabaseUpd2$result)
```

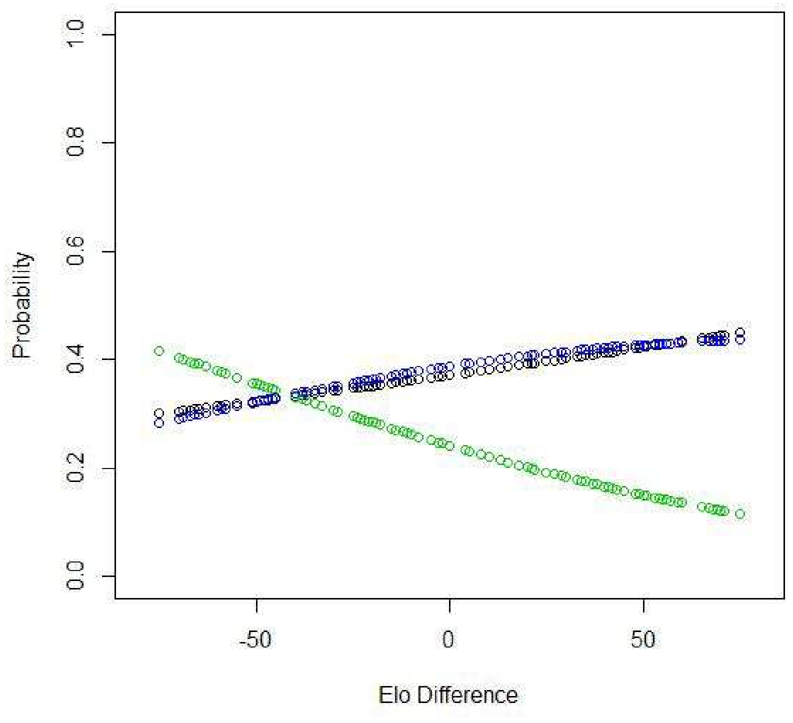


Image 3.4: Probability of a black victory (green line) vs. probability of a draw (blue line) vs. probability of a white victory (black line). All done for the test sample

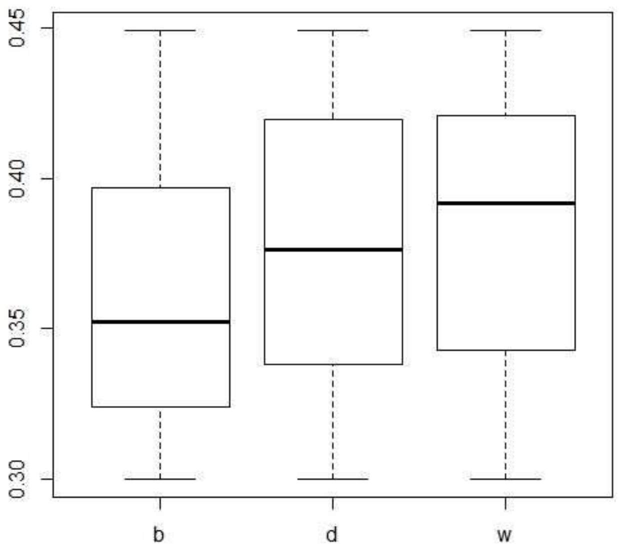


Image 3.5: White victory probability based on actual results

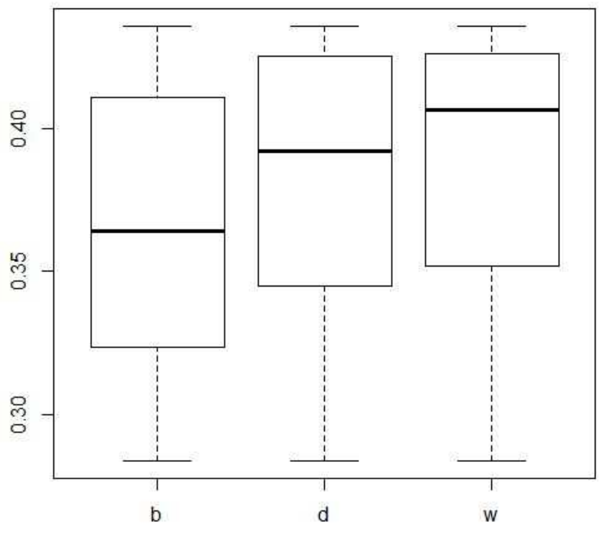


Image 3.6: Draw victory probability based on actual results

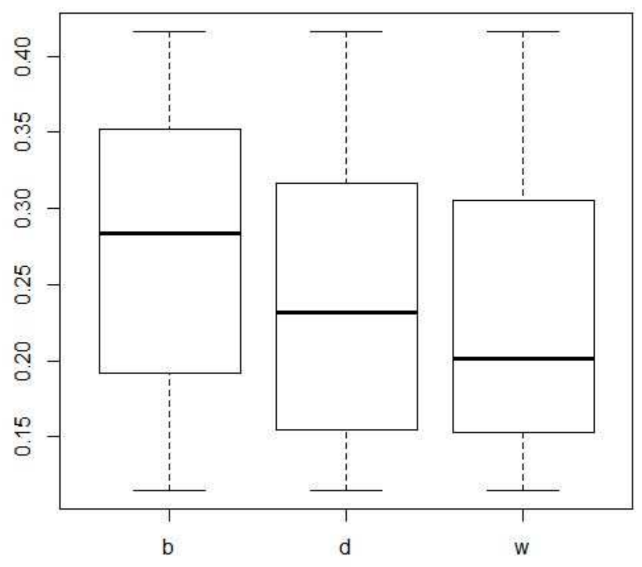


Image 3.7: Black victory probability based on actual results

From the past Script the most relevant thing we can extract is the table. Below the results examined in more detail as they will be relevant for the subsequent comparisons:

From a total of 200 predictions
- Black victory was predicted 44 times. From that a successful 14 times. Meaning 32% accuracy
- White victory was predicted 29 times. From that a successful 15 times. Meaning 52% accuracy
- Draw was predicted 127 times. From that a successful 40 times. Meaning 31,5% accuracy
- A total 34,5% accuracy

From those results we can see how in this case, the model predicts more draws. It essentially predicts a black victory when the Elo difference is between -75 and -40, seeing it approximately in the image 3.4, predicts a draw and between -40 and 50 and a white victory between 50 and 75

We have obtained then our first set of results for our data using Logistic Regression. The next step is then to obtain results in this format and under the same conditions but using Distance-based Regression in order to be able to compare results.

Lastly though, we will show results using the Logistic Regression when predicting for the whole sample using a similar method as before. We will use a sample for the model and a test sample for the predictions. The difference is that we will use 80% of the complete sample for the modeling and we will predict the results for the other 20%. This modeling will be done considering the whole sample, not just within an elo difference of [-75,75]

A random 80% of the white victories were extracted, along an 80% of the black and the draws as well. The model was done based on this sample and with it the results were predicted for the test sample

After this process we will see the table where the accuracy of the results will be shown. Even though they will not be used to compare against the Distance-based Regression, some of these results will be mentioned in the conclusions so we feel it is convenient to introduce them before.

When predicting for the test sample

	1	2	3
b	253	123	121
d	134	191	243
w	74	112	474

Prediction accuracy: 918 / 1725 (53%)

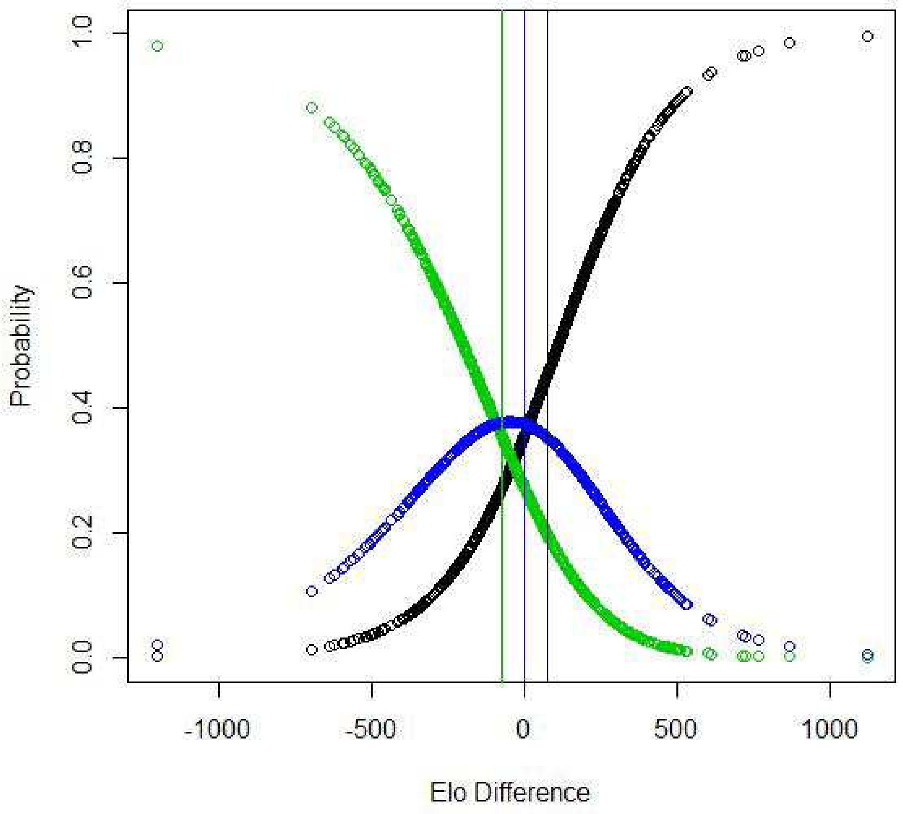


Image 3.8: Probability of a black victory (green line) vs. probability of a draw (blue line) vs. probability of a white victory (black line) using three reference lines located in -75, 0, 75. All the predictions done for the test sample.

Black curve represents the white victories, green curve black victories and blue curve the draws. There are also three vertical lines for $x = -75$, $x = 0$ and $x = 75$. It can be seen how the black victories are predicted until approximately -75, the draws between approximately -75 and 75 and the white victories from there onwards.

As a last comment from this section, tests were run using first black victories vs non-black victories and the model constructed based on that. The results differed very little, therefore the decision to run first the white vs non-white victory has no real effects on the results compared to starting with black victories vs non-black victories.

4.2 Analysis of the Chess Data using Distance-based Regression

The main purpose of the Distance-based Regression is to be able to model the data we have using unconventional explanatory variables. This meaning that these explanatory variables may differ to what is usually used. In our case it will be the already computed Distance Matrix.

We will use 2 different Distance Matrices. Each Distance Matrix has a dimension of 200x200 and in each entry we see the distance between the cross formed by the game found in the row against the game found in the column. In the diagonal of the matrix all values will be zero as it is the distance between two equal games. One of the Distances Matrix will be after 25 movements and the other after 40 movements.

The Distance between two games was created by having a weighted sum of the Euclidean distances between the corresponding pieces of those two games. After this computation is done for the two games involved, it would be replicated for the entire matrix.

The reason why there are two different matrices is because as the configuration of the board plays a major role in the outcome of the model and we would also like to see how it affects the ability to predict the outcome as the amount of information grows, it was decided that we would use several cases in order to get a better look at this differences. The cases to be used would be the board configuration after 25 movements and after 40.

A definition and important result from the Distance-based Regression is the following:
Let $\Omega = \{ [1], [2], \dots, [n]\}$ be a set of n individuals, a matrix D with Distances between those individuals is available and y, a $n \, x$ 1 response vector. It is known that with the use of Multidimensional Scaling a Euclidean Configuration X is attainable containing the coordinate matrix. Through this Euclidean Configuration a regression can be performed to estimate the values of y.

However a key point proved by Cuadras, shows that $\hat{y}$, does not depend on the choice of X. Therefore allowing us to go straight from the Distance Matrix to its estimate.
(Cuadras, 1989)

The Distance-based Regression has two variants: The Global Distance-based Regression and the Local version. A comparison of how each of the methods work will be given in the next paragraph as both will be used throughout this section.

The main difference between these two methods is that with the Global Distance-based Regression the fitting is done by using the results vector and the Distance Matrix, to which this model could then predict every point within the range defined for the explanatory variable using the parameters found. On the other hand, with the Local Distance-based Regression, the fitting is done for each point according to a delimited neighborhood, making it more flexible, though slower. This means that when a prediction is made for the Global method, the explanatory variable is substituted in the model and the solution is found. For the Local Model, when calculating the prediction, a new fitting takes place considering the neighborhood of the new addition and subsequently is the prediction calculated, taking a slightly bigger amount of time.

These methods were developed as a way to be able to model a real-valued variable based on a non real-valued variable which is a problem often found in different scenarios related to applied mathematics. Distances would then be measured between these non real-valued observations in this way obtaining the construction of explanatory information that would allow us to model. However, in order to measure distances a metric has to be selected to which we know that the results of the model will depend on the selection of the metric. (Arenas, Cuadras. 2002)

In order to apply the previous concept for our case, the function dbglm() was used. This function comes from a library in R called "dbstats". The main arguments needed are the Distance Matrix and the results vector (0's and 1's) which applied onto the previously given definition would represent the response variable Y and Δ the $n \times n$ matrix, whose entries are the squared distances. This would allow us to model the data, however when meaning to predict an outcome, another argument is needed. This will be the distances between the variable we are meaning to predict and all of the entries used to model the data. This will be heavily used in our model as we will first model the data using a sample and then we predict the new outcomes using the test sample.

We will now proceed to explain the obtaining of results for this part. We should be able to obtain 4 sets of results. These 4 results separated depending on the amount of movements considered for that board configuration and the method used. Therefore the results would consist of:

- actual results vs prediction made with **global** method based on **25 movements**
- actual results vs prediction made with **global** method based on **40 movements**
- actual results vs prediction made with **local** method based on **25 movements**
- actual results vs prediction made with **local** method based on **40 movements**

In order to get to the results, first we will naturally go through the script used to get there alongside some explanations of the most relevant parts of it. It is important to mention that the Distance Matrix calculated in Chapter 2 will be heavily used in the following parts of the script. We will show the main points of the script for one of the cases, mention where the changes were made in order to run the other and finally show the results.

A quick explanation before we go through it. The dependent variable will be '1' in case it is a victory for the white player and '0' if it is a victory for the black player or a draw, this being the first step when modeling it. Then to separate a black victory from a draw, we will make use of the properties of the conditional probability as we did for the Logistic Regression case.

```
library("dbstats")

predictions25<-rep(0,200)
## The vector where we will store the predictions of white vs. non-
## white

Distancias25 <- Distancias25 ^ 2
DistEntre2.25 <- DistEntre2.25 ^ 2
## As one of the possibilities for using the model is by feeding it
## with a D2 Matrix, we turn them into a squared Distance Matrix.

## One of them represents the Distance Matrix between the games
## themselves (Distancias25) used to model the data and the other
## one for predicting

class(Distancias25)<-"D2"
class(DistEntre2.25)<-“D2”
## The class as well

y<-as.numeric(GamesDatabaseUpd3$result=='w')
## our dependent variable

## ---------------------------------Data Modeling

dbglmDss25   <-   dbglm(Distancias25,y,family   =   binomial(link   =
"logit"), method="rel.gvar")
## We then model the data using the Distance Matrix created within
## the games of GamesDatabaseUpd3 and its white vs non-white
## results

for(i in 1:200){
predictions25[i]<-predict(dbglmDss25, DistEntre2.25[i,],
type.var="D2", type.pred="response")
}
## After modeling, we predict the results. As in the case with the
## logistic regression shown above, the explanatory variable
## contains the information of the test sample.
## This Distance Matrix now contains the distances between the
## sample and the test sample.
```

It is quite straight forward the procedure being done above. Modeling the data to then predict the results for another set of data, the test sample. A point to be highlighted though, is that DistEntre2.25 (unlike Distancias25) is a Distance Matrix that contains all the distances between the sample and the test sample. Distancias25 however contains the distances between the matches of the sample themselves, making the diagonal zero-valued as it would be the distance between two equal matches. The former is used to predict and the latter to model the data.

Now we have the predictions of white victory vs non-white victory but the predictions need to be more specific, we need to know when the prediction is being done for a draw or for a black victory. In the following script, predictions for these matches are done.

```
DraBla25<-(GamesDatabaseUpd3$result=='b'                          |
GamesDatabaseUpd3$result=='d')
## Boolean that assigns true when it’s a draw or a black victory

Bistancias25 <- Distancias25[DraBla25,DraBla25]
## We take the cases from the Distance Matrix that its results are
## either a draw or a black victory

DraBla25.df<-GamesDatabaseUpd3[DraBla25,]
## Same scenario as above, but the values extracted come from the
## Data frame.

DraBla25.df$black <- (DraBla25.df$result=='b')
## Within this data, a true is assigned when the victory goes to
## the black player

yy<-as.numeric(DraBla25.df$black)

class(Bistancias25)<-"D2"

predicsions25<-rep(0,200)
## vector where the black victory predictions (assuming it’s either
## a draw or a black victory) will be stored.

## ---------------------------------- Second part
dbglmBss25  <-  dbglm(Bistancias25,yy,family  =  binomial(link  =
"logit"), method="rel.gvar")
## The data is modeled for the black victory vs draw cases

for(i in 1:200){
predicsions25[i]<-predict(dbglmBss25,    DistEntre2.25[i,DraBla25],
type.var="D2", type.pred="response")}
## We predict for the test sample. We see that not all columns of
## the matrix are taken. Just the ones that had a true Boolean for
## cases where the victorious player was the  black or it was a
## draw
```

We can see that it is very similar to the last part where we did the predictions for the whole matrix considering white victories vs other results. One of the things that changes is that, as we are not working with the whole matrix, just the cases where there is a draw or a black victory, the variable being fed to the predict function suffers a small change. As there is a conditional being assumed related to taking only black victories and draws, we then take these corresponding cases from the Distance Matrix between the samples. All of the rows are taken as those represent the predictions we need.

Doing a quick recap, at this point we have:

- Predictions of the probability of a white victory using the whole Distance Matrix (200x200)
- Predictions of the probability of a black victory, knowing it is either a draw or a black victory using the corresponding part of the Distance Matrix and then predicting for the 200 cases.

Seeing this, we would still need to know the probability of a black victory without being conditioned to a draw or a black victory. There, as with the part using the Logistic Regression, we will use the properties of the conditional probability. The computations of the script below alongside a summary table:

```
jd25<-predictions25

kd25<-predicsions25*(1-predictions25)

dd25<-1-(jd25+kd25)


pred.result25                                            <-
unlist(apply(cbind(kd25,dd25,jd25),1,which.max))

TableDist25<-table(GamesDatabaseUpd2$result, pred.result25)
```

Table1:

	b.p	d.p	w.p
b	8	20	14
d	15	28	21
w	21	37	36

This would be our first set of results using Distance-based Regression. It represents the results for the global method after 25 movements. The interpretation of these results follows as this: the rows represent the real results, so the total of the first row for instance would give you the total black victories, 42. The second row (draws) would have 64 and the third one (white victories) 94. The columns would be the guesses of the model, which would mean that column number one (b.p) is the black victory guesses with 44 of them (8 of them correct), the second column (d.p, draws) had 85 guesses (28 correct) and finally the third column (w.p) had 71 guesses (36 correct). The correct predictions would be the matching of the guess with the actual result, thus the diagonal. In this case obtaining a total 72 correct predictions.

Now we will present the rest of the results, including the global Distance-based Regression based on the Distance Matrix after 40 movements and using the local method. The script used for this part is similar to what is done before with the only change being the variable that contains the information about the number of movements. From 25 to 40, and in the second case modeling it using the local method.

For the local method case, the changes are done every time the data is modeled. In these cases instead of using the “dbglm” function, we will use the “ldbglm” function. As mentioned previously the local method can wok with two distance matrices. One of them represents the matrix that has the distances for all elements within a set neighborhood. We will use the default parameters set by the function which takes the neighborhood distance as the first quartile of all the distances and then takes the corresponding elements from the Distance Matrix that are within that distance from the element/chess game. Below the results:

Table1:

	b.p	d.p	w.p
b	**8**	20	14
d	15	**28**	21
w	21	37	**36**

- Global DB-GLM after 25 moves: 72 predicted /200 (36%)

36 / 94	predicted white victories / total white victories	(38 %)
28 / 64	predicted draws / total draws	(44 %)
8 / 42	predicted black victories / total black victories	(19 %)

We start with one previously seen, as to have them all lined up

Table 2:

	b.p	d.p	w.p
b	**6**	7	29
d	5	**11**	48
w	8	7	**79**

- Global DB-GLM after 40 moves: 86 predicted /200 (43%)

79 / 94	predicted white victories / total white victories	(84 %)
11 / 64	predicted draws / total draws	(17 %)
6 / 42	predicted black victories / total black victories	(14 %)

Table 3:

	b.p	d.p	w.p
b	**15**	14	13
d	16	**30**	18
w	21	35	**38**

- Local DB-GLM after 25 moves: predicted 83 / 200 (41,5%)

38 / 94	predicted white victories / total white victories	(40 %)
30 / 64	predicted draws / total draws	(47 %)
15 / 42	predicted black victories / total black victories	(36 %)

Table 4:

	b.p	d.p	w.p
b	**9**	17	16
d	6	**36**	22
w	11	48	**35**

- Local DB-GLM after 40 moves: 80 predicted / 200 (40 %)

35 / 94	predicted white victories / total white victories	(37 %)
36 / 64	predicted draws / total draws	(56 %)
9 / 42	predicted black victories / total black victories	(21 %)

And to recap, the results of the Logistic Regression

	b.d	d.d	w.d
b	**14**	25	3
d	13	**40**	11
w	17	62	**15**

- Logistic Regression: 69 predicted / 100 (34.5%)

15 / 94	predicted white victories / total white victories	(16 %)
40 / 64	predicted draws / actual draws	(63 %)
14 / 42	predicted black victories / actual black victories	(33 %)

The first point that we can take as highlight of the results is the fact that using the **Distance-based Regression, the results appear to be more accurate** with all of the Distance-based Regressions having a better accuracy, this improvement ranging between 1,5 and 9 percent compared to the Logistic Regression.

Within the Distance-based Regression we are able to find the second highlight. Using the **local method, the results are slightly better.** But also equally interesting as the fact that it has better results is the fact that it seems to **provide more stable accuracy** rates. What we mean by this is that when we for instance take the best accuracy rate among the three possible outcomes for one set of results (draw predictions in Table 1 with 44%) and its worst (black predictions in the same Table 1 with 19%) we see that the difference between its best and worst accuracy is 23 (44 – 19 = 25). When we do this for the 4 possible sets of results of Distance-based Regression and compare global vs local we find the following:

- The difference in accuracy between its best and worst rate in the global method after 25 movements was 25 (44% - 19%) vs 11 (47% - 36%) for the local method.

- In the case for after 40 movements the difference in the global was 70 (84% - 14%) vs 35 (56% - 21%) in the local.

In both cases the difference is equal or more than the double. This increase in the difference also happens when we go from 25 movements to 40 movements. One possible explanation is that when using the local method we consider just a neighborhood when modeling and predicting while on the other case the whole sample is considered, making it more susceptible to sets of outliers. While a set of extreme values would only affect themselves in the prediction using the local case, for the global case it would affect all of the predictions.

One last highlight seen is **the improvement found when predicting after 40 movements** compared to predicting after 25. On average the prediction accuracy for predicting after 40 movements is 41,5% while for predicting after 25 is 38,7%.

Finally, before going to the conclusions we will make one observation we deem to be important.

It was mentioned in Section 3.1 that starting to model the data with white vs non-white compared to black vs. non-black made little difference when looking at the results. The same does not apply here. We mention this is as it was proved with 4 different scenarios using the global method (2 for after 25 movements and 2 for after 40 movements) and in all four cases there were relevant changes to the results when starting with white victories compared to starting with black victories.

In general, starting with a particular outcome would favor it in total of guesses made. Also, we used this extra set of results to validate the three highlights mentioned previously and they generally repeat themselves. All of the tables were not shown as to not overwhelm the reader with endless results tables, showing instead the most significant ones.

5. CONCLUSIONS

An end-to-end process was carried out in which a set of Chess Games was found, scraped and stored into a data frame of approximate dimensions of 10.000 x 25 using mainly tools related with Web Scraping and Regular Expressions. Subsequently, key information was taken out from this data frame as we were interested in doing a comparison considering several Regression methods. Mainly the Logistic Regression and Distance-based Regression using its two variants, Global and Local.

After extracting key information and performing the Logistic Regression using the results of the Chess Game and the Difference of Elos between the players as explanatory variable, a Distance Matrix was created. This was done as the explanatory variable required for the Distance-based Regression is a Distance Matrix. At this point we can mention that the workload related to the Distance-based Regression is considerably bigger as it requires firstly to identify how to measure the distances (and what metric to use) between the variables and subsequently computing this Distance Matrix.

After doing so and computing the predictions, these were compared against each other in order to see its accuracy. When arriving at this point we were able to observe that with the conditions imposed, the Distance-based Regression obtains in general a better accuracy than the Logistic, even though the improvement not being quite as significant as expected. As we can see when using a sample size of 200 the accuracy percentages moved within the range of 36 and 43 percent whilst on the other hand when using Logistic Regression the accuracy percentage was 34,5 %. It should be highlighted however that when expanding the Elo Difference and removing the condition forcing it to be smaller than 75 in absolute value, the predictions reached a 53% accuracy.

The results accuracy of the Distance-based Regression could be considered improvable, especially when we take into account the extra effort involved when computing the Distance Matrix as mentioned previously. Nonetheless it’s exactly in this point where we would like to focus. This extra load of work comes with a certain set of decisions and parameters that the developer has to assume and it is here where most of the improvements can be made. We mean by this cases as deciding the sample size, how to measure two games (metric and method as such), how much to weight each piece, how to measure a captured against a non-captured piece and how many movements to consider before stopping.

The improvement seen when increasing the number of movements was visible in both scenarios (local and global), with the exception that with the local method the improvements seemed to be more balanced, not depending as much on having a big success in one outcome at the expense of other outcomes. On the other hand though, with the Logistic Regression the core of the success came when increasing the sample size and removing the conditions. This could be substantiated by the fact that when running the Logistic Regression for the whole data frame, the Elo Difference (explanatory variable) can provide much more information combined with the fact that the modeling is done for a sample size of approximately 7600, raising the percentage accuracy to 53% from 34,5%. Unfortunately running the Distance-based Regression for the whole sample is unfeasible at the time being due to computer and time limitations.

When creating the Distance Matrix several decisions were made. These were made following what seemed reasonable and not too extreme combined with basic notions of chess. However there might be possibilities to obtain better results by changing how to measure the distance between two games or changing the weights of the pieces. For example in our case a weight of 9 was assigned to the queen as it was the standard weight considered, combined with a distance of 17 when a piece is captured (50% more than the diagonal of a chess game, the maximum distance) would give us 153 distance units when measuring two games where one has a captured queen. This compared to a bishop with weight 3 that is 6 distance units away would give us 18 distance units for two games that have the bishops 6 distance units away. It is almost ten times as much and it could be considered too much, though on the other hand a captured queen contains a considerable amount of information that the other case doesn't.

Tuning the parameters as to reach more reasonable differences and increasing the sample size could be key features in obtaining a better accuracy for the Distance-based Regression method using this set of data.

In general and as a way to conclude it can be said as a first comment that the use of Web Scraping and Regular Expressions provided us with the possibility of obtaining the data in a way that is reachable for anyone with internet access, making these tools very useful and highly recommendable. It requires some work, especially depending on the state in which the data finds itself, but the usefulness of obtaining data from scratch only with the use of R is high.

As a way to analyze the data, the Logistic Regression provided us with some very acceptable results when conditions were not imposed. We say this based on the fact that when not limiting the data to an Elo Difference of [-75,+75] the Logistic Regression predicts with a 53% accuracy. Considering that we only take one explanatory variable of several that could be taken, it is shown that there is margin for improvement in the predictions when using Logistic Regression.

Finally the Distance-base Regression gave comparatively similar accuracy results when compared to an average of both the Logistic Regression results, but also showing room for improvement. It can be said that Distance-based Regression is a very a powerful tool to perform Regressions on data samples that contain non-quantitative explanatory variables. It is important to consider though, that it might require some increases in sample size when possible and adaptations to the parameters down the line in order to achieve higher accuracy when used to predict.

APPENDIX

The core of the script used will be shown below.

It will be presented in chronological order as it was developed and titled with the Chapter it finds itself in the Content of the Thesis

<u>CHAPTER 1</u>

– WEB SCRAPING & DATABASE CONSTRUCTION

```
sample.total <- sample(3700000, 10000)
samplesize <- 1000

GamesDatabase <-
as.data.frame(data="",array(dim=c(20,25)),stringsAsFactors=FALSE)

names(GamesDatabase) <-c("game_id", "white_id", "black_id",
"white", "black","result", "elowhite", "eloblack", "date",
"num_moves", "site", "event", "round", "opening_id","eco",
"opening_name", "avgelo", "tactic_id", "moves",
"canonical_id", "duplicate_ids", "forced_load", "deleted",
"canCache", "one")

library(RCurl)

for (i in 1:10)
{

fsample <- sample.total[( ((i-1)*samplesize)+1)
:(i*samplesize) ]

for (j in 1:samplesize)
{

pagweb <- paste("http://chesstempo.com/gamedb/game/",fsample[j],sep
= "")
hist2 <- pagweb
txt2=getURL(hist2)
info.tc2 <- textConnection(txt2)
info.list2 <- scan(info.tc2,what="list",sep="\n",quote="\"")
partx <- info.list2[[715]][1]
partx2 <- info.list2[[714]][1]

if(substring(strsplit(partx,split="eval")[[1]][2],3,7)=="resul")
{

trial<-
strsplit(partx,split="game_id:|,white_id:|,black_id:|,white:|,black
:|,result:|,elowhite:|,eloblack:|,date:|,num_moves:|,site:|,event:|
,round:|,opening_id:|,eco:|,opening_name:|,avgelo:|,tactic_id:|,mov
es_lalg:[[]|[]],canonical_id:|,duplicate_ids:|,forced_load:|,delete
d:|,canCache:|},id:|})")

GamesDatabase[j,]<-unlist(trial)[-c(1,27)]
```

```
rownames(GamesDatabase)[j]<-paste("Game
id:",strsplit(partx2,split="gameid = |;")[[1]][2])
}
}
filename <- paste("fichero_",i,".RData",sep="")
save.image(file=filename)
}
```

CHAPTER 2.

- CHESS BOARD CONFIGURATION AND ITS RELATED FUNCTIONS

```
init.chess.game <- function(){
ChessGame <- data.frame(position =
c('a2','b2','c2','d2','e2','f2','g2','h2','b1','g1','a1','h1','c1',
'f1','d1','e1','a7','b7','c7','d7','e7','f7','g7','h7','b8','g8','a
8','h8','c8','f8','d8','e8'), stringsAsFactors = FALSE)

ChessGame$firstCoord <- letter2Num(ChessGame$position)
ChessGame$secndCoord <- number2Num(ChessGame$position)

rownames(ChessGame)[1] <- "pawnWa"
rownames(ChessGame)[2] <- "pawnWb"
rownames(ChessGame)[3] <- "pawnWc"
rownames(ChessGame)[4] <- "pawnWd"
rownames(ChessGame)[5] <- "pawnWe"
rownames(ChessGame)[6] <- "pawnWf"
rownames(ChessGame)[7] <- "pawnWg"
rownames(ChessGame)[8] <- "pawnWh"
rownames(ChessGame)[9] <- "knightWb"
rownames(ChessGame)[10] <- "knightWg"
rownames(ChessGame)[11] <- "rookWa"
rownames(ChessGame)[12] <- "rookWh"
rownames(ChessGame)[13] <- "bishopWc"
rownames(ChessGame)[14] <- "bishopWf"
rownames(ChessGame)[15] <- "queenW"
rownames(ChessGame)[16] <- "kingW"
rownames(ChessGame)[17] <- "pawnBa"
rownames(ChessGame)[18] <- "pawnBb"
rownames(ChessGame)[19] <- "pawnBc"
rownames(ChessGame)[20] <- "pawnBd"
rownames(ChessGame)[21] <- "pawnBe"
rownames(ChessGame)[22] <- "pawnBf"
rownames(ChessGame)[23] <- "pawnBg"
rownames(ChessGame)[24] <- "pawnBh"
rownames(ChessGame)[25] <- "knightBb"
rownames(ChessGame)[26] <- "knightBg"
rownames(ChessGame)[27] <- "rookBa"
rownames(ChessGame)[28] <- "rookBh"
rownames(ChessGame)[29] <- "bishopBc"
rownames(ChessGame)[30] <- "bishopBf"
rownames(ChessGame)[31] <- "queenB"
rownames(ChessGame)[32] <- "kingB"
return(ChessGame)}

letter2Num <- function(position,pos=1)
{
```

```
ch <- c("a","b","c","d","e","f","g","h","i")
moves_prim <- unlist(lapply(position, function(tira.char)
{
which( ch == substring(tira.char,pos,pos) )
}
))
return(moves_prim)
}


number2Num <- function(position,pos=2)
{
num<-as.numeric(substring(position,2,2))
return(num)
}



chessconfig <- function(moves_vector, moves_nr)

{
ChessGame<-init.chess.game()
## Cutting the movement vector in the number set by the element
## sent and afterwards separate them by commas

      moves_pred <- substring(moves_vector, 1, (moves_nr*5)-1)
moves_ind <- strsplit(moves_pred,split=",")

      for (i in 1:moves_nr)
{

            for (j in 1:32)
{
## Splitting each string in starting position and end
## position

                  moves_prim <- substring(moves_ind[[1]][i],1,2)
moves_segu <- substring(moves_ind[[1]][i],3,4)

## Castling/Enroque

if( ChessGame$position[32]=='e8' & moves_prim=='e8' &
moves_segu=='c8' )
{
ChessGame$position[27] <- 'd8'
}

else if(ChessGame$position[32]=='e8' & moves_prim=='e8' &
moves_segu=='g8' )
{
ChessGame$position[28] <- 'f8'
}

else if(ChessGame$position[16]=='e1' & moves_prim=='e1' &
moves_segu=='c1' )
{
ChessGame$position[11] <- 'd1'
}

else if(ChessGame$position[16]=='e1' & moves_prim=='e1' &
moves_segu=='g1' )
{
ChessGame$position[12] <-'f1'
}

## Piece-by-Piece comparison and subsequent
## assignment


if(moves_segu==ChessGame$position[j])
{
ChessGame$position[j] <-"i9"
}


if(moves_prim==ChessGame$position[j])
{
ChessGame$position[j]<-moves_segu
}

            }

      }

            ChessGame$firstCoord <- letter2Num(ChessGame$position)
            ChessGame$secndCoord <- number2Num(ChessGame$position)
            ChessGame



}
```

- DISTANCE MATRIX

```
## There were 4 200x200 Distance Matrices created.

## 2 measured the distances between the games of the sample itself,
## which would be modeled afterwards using Distance-based
## Regression. 1 was done after 25 movements and 1 after 40

## The other 2 matrices measured the distances between one game and
## the other. This was done as it is required when using the
## predict function, to give these distances as argument

## We will show here the creation of 1 of the 4 matrices, and
## comment on the parts where changes were made to obtain the other
## 3. This is done because each Distance Matrix script would take
## 10 pages and the changes between them are minimal


dimPrueba<-200

hmm<-40
## Two of the matrices used as value 25, the other two 40
```

```
GamesDatabase$elodffs<-abs(as.numeric(GamesDatabase$elowhite)-
as.numeric(GamesDatabase$eloblack) )

GamesDatabaseUpd<-
na.omit(GamesDatabase[GamesDatabase$elodffs<=75,])

GamesDatabaseUpd<-
GamesDatabaseUpd[nchar(GamesDatabaseUpd$moves)>=199,]

## We make a copy of the Games Database with games that have
## minimum 40 movements and an Elo Difference of minimum 75


Distancias25<-matrix(data=0,nrow=dimPrueba,ncol=dimPrueba)

lngth<-dim(GamesDatabaseUpd)[1]

## ----------------------
set.seed(456)

randSample<-sample(lngth,200)

GamesDatabaseUpd2<-GamesDatabaseUpd[randSample,]
## Test Sample

## ----------------------
set.seed(987)

randSample2<-sample(lngth,200)

GamesDatabaseUpd3<-GamesDatabaseUpd[randSample2,]
## Main Sample

## The procedure to assign values to the matrix starts below

for (j in 1:dimPrueba )
{game_j_hmm<- chessconfig(GamesDatabaseUpd3$moves[j],hmm)

for (k in 1:dimPrueba )
{game_k_hmm<- chessconfig(GamesDatabaseUpd3$moves[k],hmm)

if(j==k)
{Distancias25[j,k]<-0}
## We would not consider this ‘if statement’ when
## working with the Distance Matrix between the 2 games ## as the
diagonal won’t be 0

else
{

pnsBcsJg1<-game_j_hmm[1:8,]
pnsNgsJg1<-game_j_hmm[17:24,]
pnsBcsJg2<-game_k_hmm[1:8,]
pnsNgsJg2<-game_k_hmm[17:24,]

coordsBcs1<-as.numeric(paste(pnsBcsJg1$firstCoord,
pnsBcsJg1$secndCoord, sep=""))
```

```
coordsBcs2<-as.numeric(paste(pnsBcsJg2$firstCoord,
pnsBcsJg2$secndCoord, sep=""))
coordsNgs1<-as.numeric(paste(pnsNgsJg1$firstCoord,
pnsNgsJg1$secndCoord, sep=""))
coordsNgs2<-as.numeric(paste(pnsNgsJg2$firstCoord,
pnsNgsJg2$secndCoord, sep=""))

vectorDB<-
round(
ifelse(
(substring(expand.grid(coordsBcs1, coordsBcs2)[,1],1,1)=='9' &
substring(expand.grid(coordsBcs1, coordsBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsBcs1, coordsBcs2)[,2],1,1)=='9' &
substring(expand.grid(coordsBcs1, coordsBcs2)[,1],1,1)!='9')
,17,

sqrt(
(as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,2],1,1)))^
2+
(as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsBcs1,coordsBcs2)[,2],2,2)))^
2) ),2)

vectorDN<-
round(
ifelse(
(substring(expand.grid(coordsNgs1, coordsNgs2)[,1],1,1)=='9' &
substring(expand.grid(coordsNgs1, coordsNgs2)[,2],1,1)!='9')

|

(substring(expand.grid(coordsNgs1, coordsNgs2)[,2],1,1)=='9' &
substring(expand.grid(coordsNgs1, coordsNgs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsNgs1,
coordsNgs2)[,1],1,1))-as.numeric(substring(expand.grid(coordsNgs1,
coordsNgs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsNgs1,
coordsNgs2)[,1],2,2))-as.numeric(substring(expand.grid(coordsNgs1,
coordsNgs2)[,2],2,2)))^2))
,2)


matrixDN<-matrix(vectorDN, nrow=8, ncol=8, byrow=FALSE)
matrixDB<-matrix(vectorDB, nrow=8, ncol=8, byrow=FALSE)

vectorDistanciaPeonesBlancos<-rep(0,8)
vectorDistanciaPeonesNegros<-rep(0,8)

for (i in 1:8)
{
      if(length(matrixDB)==1 || length(matrixDN)==1)
      {
            vectorDistanciaPeonesBlancos[i]<-matrixDB
            vectorDistanciaPeonesNegros[i]<-matrixDN
      }
      else
      {

vectorDistanciaPeonesBlancos[i]<-min(apply(matrixDB, 2, min))
matrixDB<-matrixDB[-which(apply(matrixDB, 1,
min)==min(matrixDB))[1],
            -which(apply(matrixDB, 2, min)==min(matrixDB))[1]]
vectorDistanciaPeonesNegros[i]<-min(apply(matrixDN, 2, min))
matrixDN<-matrixDN[-which(apply(matrixDN, 1,
min)==min(matrixDN))[1],
            -which(apply(matrixDN, 2, min)==min(matrixDN))[1]]
      }

}


##--------------------------------------------------------------------

trsBcsJg1<-game_j_hmm[11:12,]
trsNgsJg1<-game_j_hmm[27:28,]
trsBcsJg2<-game_k_hmm[11:12,]
trsNgsJg2<-game_k_hmm[27:28,]

coordsTrsBcs1<-as.numeric(paste(trsBcsJg1$firstCoord,
trsBcsJg1$secndCoord, sep=""))
coordsTrsBcs2<-as.numeric(paste(trsBcsJg2$firstCoord,
trsBcsJg2$secndCoord, sep=""))
coordsTrsNgs1<-as.numeric(paste(trsNgsJg1$firstCoord,
trsNgsJg1$secndCoord, sep=""))
coordsTrsNgs2<-as.numeric(paste(trsNgsJg2$firstCoord,
trsNgsJg2$secndCoord, sep=""))

vectorDTB<-
round(
ifelse(
(substring(expand.grid(coordsTrsBcs1,coordsTrsBcs2)[,1],1,1)=='9'
&
substring(expand.grid(coordsTrsBcs1, coordsTrsBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsTrsBcs1, coordsTrsBcs2)[,2],1,1)=='9'
& substring(expand.grid(coordsTrsBcs1,
coordsTrsBcs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsTrsBcs1,
coordsTrsBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsTrsBcs1,
coordsTrsBcs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsTrsBcs1,
coordsTrsBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsTrsBcs1,
coordsTrsBcs2)[,2],2,2)))^2))
,2)

vectorDTN<-
round(
ifelse(
(substring(expand.grid(coordsTrsNgs1, coordsTrsNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsTrsNgs1, coordsTrsNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,1],1,1)!='9')
      ,17,
       sqrt(
(as.numeric(substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsTrsNgs1,
coordsTrsNgs2)[,2],2,2)))^2))
,2)

matrixDTN<-matrix(vectorDTN, nrow=2, ncol=2, byrow=FALSE)
matrixDTB<-matrix(vectorDTB, nrow=2, ncol=2, byrow=FALSE)


vectorDistanciaTorresBlancos <-rep(0,2)
vectorDistanciaTorresNegros <-rep(0,2)

for (i in 1:2)
{
      if(length(matrixDTB)==1 || length(matrixDTN)==1)
      {
            vectorDistanciaTorresBlancos[i]<-matrixDTB
            vectorDistanciaTorresNegros[i]<-matrixDTN
      }
      else
      {

vectorDistanciaTorresBlancos[i]<-min(apply(matrixDTB, 2, min))
matrixDTB<-matrixDTB[-which(apply(matrixDTB, 1,
min)==min(matrixDTB))[1],
-which(apply(matrixDTB, 2, min)==min(matrixDTB))[1]]
vectorDistanciaTorresNegros[i]<-min(apply(matrixDTN, 2, min))
matrixDTN<-matrixDTN[-which(apply(matrixDTN, 1,
min)==min(matrixDTN))[1],
-which(apply(matrixDTN, 2, min)==min(matrixDTN))[1]]
      }

}



##-------------------------------------------------------------------

cbsBcsJg1<-game_j_hmm[9:10,]
cbsNgsJg1<-game_j_hmm[25:26,]
cbsBcsJg2<-game_k_hmm[9:10,]
cbsNgsJg2<-game_k_hmm[25:26,]
```

```
coordsCbsBcs1<-as.numeric(paste(cbsBcsJg1$firstCoord,
cbsBcsJg1$secndCoord, sep=""))
coordsCbsBcs2<-as.numeric(paste(cbsBcsJg2$firstCoord,
cbsBcsJg2$secndCoord, sep=""))
coordsCbsNgs1<-as.numeric(paste(cbsNgsJg1$firstCoord,
cbsNgsJg1$secndCoord, sep=""))
coordsCbsNgs2<-as.numeric(paste(cbsNgsJg2$firstCoord,
cbsNgsJg2$secndCoord, sep=""))

vectorDCB<-
round(
ifelse(
(substring(expand.grid(coordsCbsBcs1, coordsCbsBcs2)[,1],1,1)=='9'
& substring(expand.grid(coordsCbsBcs1,
coordsCbsBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsCbsBcs1, coordsCbsBcs2)[,2],1,1)=='9'
& substring(expand.grid(coordsCbsBcs1,
coordsCbsBcs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsCbsBcs1,
coordsCbsBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsCbsBcs1,coordsCbsBcs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsCbsBcs1,
coordsCbsBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsCbsBcs1,coordsCbsBcs2)[,2],2
,2)))^2)
      )
,2)

vectorDCN<-
round(
ifelse(
(substring(expand.grid(coordsCbsNgs1, coordsCbsNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsCbsNgs1,
coordsCbsNgs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsCbsNgs1, coordsCbsNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsCbsNgs1,
coordsCbsNgs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsCbsNgs1,
coordsCbsNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsCbsNgs1,coordsCbsNgs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsCbsNgs1,
coordsCbsNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsCbsNgs1,coordsCbsNgs2)[,2],2
,2)))^2)
      )
,2)

matrixDCN<-matrix(vectorDCN, nrow=2, ncol=2, byrow=FALSE)
matrixDCB<-matrix(vectorDCB, nrow=2, ncol=2, byrow=FALSE)
```

```
vectorDistanciaCaballsBlancos<-rep(0,2)
vectorDistanciaCaballsNegros<-rep(0,2)

for (i in 1:2)
{
      if(length(matrixDCB)==1 || length(matrixDCN)==1)
      {
            vectorDistanciaCaballsBlancos[i]<-matrixDCB
            vectorDistanciaCaballsNegros[i]<-matrixDCN
      }
      else
      {

vectorDistanciaCaballsBlancos[i]<-min(apply(matrixDCB, 2, min))
matrixDCB<-matrixDCB[-which(apply(matrixDCB, 1,
min)==min(matrixDCB))[1],
            -which(apply(matrixDCB, 2, min)==min(matrixDCB))[1]]
vectorDistanciaCaballsNegros[i]<-min(apply(matrixDCN, 2, min))
matrixDCN<-matrixDCN[-which(apply(matrixDCN, 1,
min)==min(matrixDCN))[1],
            -which(apply(matrixDCN, 2, min)==min(matrixDCN))[1]]
      }

}

##------------------------------------------------------------------

alfBcsJg1<-game_j_hmm[13:14,]
alfNgsJg1<-game_j_hmm[29:30,]
alfBcsJg2<-game_k_hmm[13:14,]
alfNgsJg2<-game_k_hmm[29:30,]

coordsAlfBcs1<-as.numeric(paste(alfBcsJg1$firstCoord,
alfBcsJg1$secndCoord, sep=""))
coordsAlfBcs2<-as.numeric(paste(alfBcsJg2$firstCoord,
alfBcsJg2$secndCoord, sep=""))
coordsAlfNgs1<-as.numeric(paste(alfNgsJg1$firstCoord,
alfNgsJg1$secndCoord, sep=""))
coordsAlfNgs2<-as.numeric(paste(alfNgsJg2$firstCoord,
alfNgsJg2$secndCoord, sep=""))

vectorDAB<-
round(
      ifelse(
(substring(expand.grid(coordsAlfBcs1, coordsAlfBcs2)[,1],1,1)=='9'
& substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsAlfBcs1, coordsAlfBcs2)[,2],1,1)=='9'
& substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsAlfBcs1,
coordsAlfBcs2)[,2],2,2)))^2)
     )
,2)

vectorDAN<-
round(
ifelse(
(substring(expand.grid(coordsAlfNgs1, coordsAlfNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,2],1,1)!='9') |
(substring(expand.grid(coordsAlfNgs1, coordsAlfNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,2],1,1)))^2+
(as.numeric(substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsAlfNgs1,
coordsAlfNgs2)[,2],2,2)))^2)
     )
,2)


matrixDAN<-matrix(vectorDAN, nrow=2, ncol=2, byrow=FALSE)
matrixDAB<-matrix(vectorDAB, nrow=2, ncol=2, byrow=FALSE)


vectorDistanciaAlfilBlancos<-rep(0,2)
vectorDistanciaAlfilNegros<-rep(0,2)

for (i in 1:2)
{
      if(length(matrixDAB)==1 || length(matrixDAN)==1)
      {
            vectorDistanciaAlfilBlancos[i]<-matrixDAB
            vectorDistanciaAlfilNegros[i]<-matrixDAN
      }
      else
      {

vectorDistanciaAlfilBlancos[i]<-min(apply(matrixDAB, 2, min))
matrixDAB<-matrixDAB[-which(apply(matrixDAB, 1,
min)==min(matrixDAB))[1],
            -which(apply(matrixDAB, 2, min)==min(matrixDAB))[1]]
vectorDistanciaAlfilNegros[i]<-min(apply(matrixDAN, 2, min))
matrixDAN<-matrixDAN[-which(apply(matrixDAN, 1,
min)==min(matrixDAN))[1],
            -which(apply(matrixDAN, 2, min)==min(matrixDAN))[1]]
      }

}
```

```
##-------------------------------------------------------------------
rnaBcsJg1<-game_j_hmm[15,]
rnaNgsJg1<-game_j_hmm[31,]
rnaBcsJg2<-game_k_hmm[15,]
rnaNgsJg2<-game_k_hmm[31,]


coordsRnaBcs1<-as.numeric(paste(rnaBcsJg1$firstCoord,
rnaBcsJg1$secndCoord, sep=""))
coordsRnaBcs2<-as.numeric(paste(rnaBcsJg2$firstCoord,
rnaBcsJg2$secndCoord, sep=""))
coordsRnaNgs1<-as.numeric(paste(rnaNgsJg1$firstCoord,
rnaNgsJg1$secndCoord, sep=""))
coordsRnaNgs2<-as.numeric(paste(rnaNgsJg2$firstCoord,
rnaNgsJg2$secndCoord, sep=""))


vectorDistanciaReinaBlancas<-
round(
      ifelse(
(substring(expand.grid(coordsRnaBcs1, coordsRnaBcs2)[,1],1,1)=='9'
& substring(expand.grid(coordsRnaBcs1,
coordsRnaBcs2)[,2],1,1)!='9')
|
(substring(expand.grid(coordsRnaBcs1, coordsRnaBcs2)[,2],1,1)=='9'
& substring(expand.grid(coordsRnaBcs1,
coordsRnaBcs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsRnaBcs1,
coordsRnaBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsRnaBcs1,coordsRnaBcs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsRnaBcs1,
coordsRnaBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsRnaBcs1,coordsRnaBcs2)[,2],2
,2)))^2)
       )
,2)

vectorDistanciaReinaNegras<-
round(
      ifelse(
(substring(expand.grid(coordsRnaNgs1, coordsRnaNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsRnaNgs1,
coordsRnaNgs2)[,2],1,1)!='9') |
(substring(expand.grid(coordsRnaNgs1, coordsRnaNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsRnaNgs1,
coordsRnaNgs2)[,1],1,1)!='9')
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsRnaNgs1,
coordsRnaNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsRnaNgs1,coordsRnaNgs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsRnaNgs1,
coordsRnaNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsRnaNgs1,coordsRnaNgs2)[,2],2
,2)))^2)
      )
,2)



##-----------------------------------------------------------------

reyBcsJg1<-game_j_hmm[16,]
reyNgsJg1<-game_j_hmm[32,]
reyBcsJg2<-game_k_hmm[16,]
reyNgsJg2<-game_k_hmm[32,]


coordsReyBcs1<-as.numeric(paste(reyBcsJg1$firstCoord,
reyBcsJg1$secndCoord, sep=""))
coordsReyBcs2<-as.numeric(paste(reyBcsJg2$firstCoord,
reyBcsJg2$secndCoord, sep=""))
coordsReyNgs1<-as.numeric(paste(reyNgsJg1$firstCoord,
reyNgsJg1$secndCoord, sep=""))
coordsReyNgs2<-as.numeric(paste(reyNgsJg2$firstCoord,
reyNgsJg2$secndCoord, sep=""))


vectorDistanciaReyBlancas<-
round(
      ifelse(
(substring(expand.grid(coordsReyBcs1, coordsReyBcs2)[,1],1,1)=='9'
& substring(expand.grid(coordsReyBcs1,
coordsReyBcs2)[,2],1,1)!='9') |
(substring(expand.grid(coordsReyBcs1, coordsReyBcs2)[,2],1,1)=='9'
& substring(expand.grid(coordsReyBcs1,
coordsReyBcs2)[,1],1,1)!='9')
      ,17,
       sqrt(
(as.numeric(substring(expand.grid(coordsReyBcs1,
coordsReyBcs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsReyBcs1,coordsReyBcs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsReyBcs1,
coordsReyBcs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsReyBcs1,coordsReyBcs2)[,2],2
,2)))^2)
      )
,2)

vectorDistanciaReyNegras<-
round(
      ifelse(
(substring(expand.grid(coordsReyNgs1, coordsReyNgs2)[,1],1,1)=='9'
& substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,2],1,1)!='9') |
(substring(expand.grid(coordsReyNgs1, coordsReyNgs2)[,2],1,1)=='9'
& substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],1,1)!='9')
```

```
      ,17,
      sqrt(
(as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],1,1))-
as.numeric(substring(expand.grid(coordsReyNgs1,coordsReyNgs2)[,2],1
,1)))^2+
(as.numeric(substring(expand.grid(coordsReyNgs1,
coordsReyNgs2)[,1],2,2))-
as.numeric(substring(expand.grid(coordsReyNgs1,coordsReyNgs2)[,2],2
,2)))^2)
      )
,2)



Distancias25[j,k]<-
sum(vectorDistanciaPeonesBlancos)+sum(vectorDistanciaPeonesNegros)+
sum(3*vectorDistanciaCaballsBlancos)+sum(3*vectorDistanciaCaballsNe
gros)+
sum(5*vectorDistanciaTorresBlancos)+sum(5*vectorDistanciaTorresNegr
os)+
sum(3*vectorDistanciaAlfilBlancos)+sum(3*vectorDistanciaAlfilNegros
)+
sum(9*vectorDistanciaReinaBlancas)+sum(9*vectorDistanciaReinaNegras
)+
sum(12*vectorDistanciaReyBlancas)+sum(12*vectorDistanciaReyNegras)

}

}

}
```

CHAPTER 3

– REGRESSION ANALYSIS

– LOGISTIC REGRESSION

```
## It will be split into 2. The first part will be related to the
## Logistic Regression where
## both the sample and the test sample have a size of 200

## The second part will be related to the sample containing 80%
## of the GamesDatabase and the test sample contains the other 20%.
## As both parts are very similar, from the second part we will
## only show the creation of the samples.

elodiff <- abs(as.numeric(GamesDatabaseUpd3$elowhite)-
as.numeric(GamesDatabaseUpd3$eloblack))
```

```
elodiff2 <- as.numeric(GamesDatabaseUpd3$elowhite)-
as.numeric(GamesDatabaseUpd3$eloblack)
GamesDatabaseUpd3 <- cbind(GamesDatabaseUpd3, elodiff2)
GamesDatabaseUpd3 <- cbind(GamesDatabaseUpd3, elodiff)

elodiff <- abs(as.numeric(GamesDatabaseUpd2$elowhite)-
as.numeric(GamesDatabaseUpd2$eloblack))
elodiff2 <- as.numeric(GamesDatabaseUpd2$elowhite)-
as.numeric(GamesDatabaseUpd2$eloblack)
GamesDatabaseUpd2 <- cbind(GamesDatabaseUpd2, elodiff2)
GamesDatabaseUpd2 <- cbind(GamesDatabaseUpd2, elodiff)

GamesDatabaseUpd2$white <- (GamesDatabaseUpd2$result == 'w')
GamesDatabaseUpd3$white <- (GamesDatabaseUpd3$result == 'w')
logrefw3 <- glm(white ~ elodiff2, family = "binomial", data =
GamesDatabaseUpd3)
## We model GamesDatabaseUpd3

attach(GamesDatabaseUpd3)

jj<-predict(logrefw3, GamesDatabaseUpd2, type="response")
## object ‘jj’ is where predictions for white vs non-white victory
## samples are made.
plot(GamesDatabaseUpd2$elodiff2,jj, col=3, xlim=c(-75,75),
ylim=c(0,1))
abline(h=sum(GamesDatabaseUpd2$white)/length(GamesDatabaseUpd2$whit
e),col=2)
## Proportion of white victories within the test sample.
abline(h=0.5, col=4)


## We start the modeling for non-white victories

DrawBlackUpd3 <- (GamesDatabaseUpd3$result == 'd' |
GamesDatabaseUpd3$result == 'b')

DrawBlackUpd3.df <- GamesDatabaseUpd3[DrawBlackUpd3,]
DrawBlackUpd3.df$black <- (DrawBlackUpd3.df$result=='b')

logrefdb3 <- glm(black ~ elodiff2, family = "binomial", data =
DrawBlackUpd3.df)
## We model the black victories against the draws. From the first
## sample

kk <- predict(logrefdb3, GamesDatabaseUpd2, type="response")
## We predict the data for the test sample

plot(na.omit(GamesDatabaseUpd2$elodiff2),kk,, xlab='Elo
Difference',
ylab='Probability',ylim=c(0,1),col=4)
points(na.omit(GamesDatabaseUpd2$elodiff2),(1-jj),col=6)
## We plot the results. Probability of black victory against a draw
## and 1 minus the probability of a white victory from the previous
## prediction.

## Predicting the test sample based on the models created
jj <-predict(logrefw3, GamesDatabaseUpd2, type="response")
kk <- predict(logrefdb3, GamesDatabaseUpd2, type="response")
```

```
pr.ww <- jj
pr.bb <- kk*(1-jj)
pr.dd <- 1-(pr.ww+pr.bb)
## Conditional Probability Properties

plot(GamesDatabaseUpd2$elodiff2, pr.ww, xlim=c(-80,80),ylim=c(0,1),
xlab='Elo Difference', ylab='Probability')
points(GamesDatabaseUpd2$elodiff2, pr.bb, col=3)
points(GamesDatabaseUpd2$elodiff2, pr.dd, col=4)

boxplot(pr.ww~GamesDatabaseUpd2$result)
boxplot(pr.dd~GamesDatabaseUpd2$result)
boxplot(pr.bb~GamesDatabaseUpd2$result)

pred.result2 <- unlist(apply(cbind(pr.bb,pr.dd,pr.ww),1,which.max))
table(GamesDatabaseUpd2$result, pred.result2)


## Second part where we create a sample with 80% of white
## victories, 80%
## of draws and 80% of black victories

GamesDatabaseClean<-
GamesDatabase[which(GamesDatabase$elowhite!='null' &
GamesDatabase$eloblack!='null'),]

## From the complete database  we will only take the ones that have
## a valid
## Elo Difference as this will be our explanatory variable

GamesDatabaseClean<-
GamesDatabaseClean[which(GamesDatabaseClean$result=='w' |
GamesDatabaseClean$result=='b' | GamesDatabaseClean$result=='d'),]
## And also valid results. Our dependent variable

set.seed(987)

## 80 per cent of the number of each result
ex_b<-round(0.8*sum(GamesDatabaseClean=='b'))
ex_d<-round(0.8*sum(GamesDatabaseClean=='d'))
ex_w<-round(0.8*sum(GamesDatabaseClean=='w'))

## Total of cases per result
ex_cb<-sum(GamesDatabaseClean=='b')
ex_cd<-sum(GamesDatabaseClean=='d')
ex_cw<-sum(GamesDatabaseClean=='w')

## Taking note of where each result is located
whch_cb<-which(GamesDatabaseClean$result=='b')
whch_cd<-which(GamesDatabaseClean$result=='d')
whch_cw<-which(GamesDatabaseClean$result=='w')

## Making a random sample taking the 80% of results per case
samp_b<-sample(ex_cb, ex_b)
samp_d<-sample(ex_cd, ex_d)
samp_w<-sample(ex_cw, ex_w)
```

```
## Taking an 80% of the numbers of the GD per each result
whch_b<-whch_cb[samp_b]
whch_d<-whch_cd[samp_d]
whch_w<-whch_cw[samp_w]

## Taking the other 20%
whch_ib<-whch_cb[-samp_b]
whch_id<-whch_cd[-samp_d]
whch_iw<-whch_cw[-samp_w]

## Making a Data frame with those selected numbers
GamesDatabaseCleanA<-GamesDatabaseClean[c(whch_b,whch_d,whch_w),]
GamesDatabaseCleanB<-
GamesDatabaseClean[c(whch_ib,whch_id,whch_iw),]

elodiff <- abs(as.numeric(GamesDatabaseCleanA$elowhite)-
as.numeric(GamesDatabaseCleanA$eloblack))
elodiff2 <- as.numeric(GamesDatabaseCleanA$elowhite)-
as.numeric(GamesDatabaseCleanA$eloblack)
GamesDatabaseCleanA <- cbind(GamesDatabaseCleanA, elodiff2)
GamesDatabaseCleanA <- cbind(GamesDatabaseCleanA, elodiff)

elodiff <- abs(as.numeric(GamesDatabaseCleanB$elowhite)-
as.numeric(GamesDatabaseCleanB$eloblack))
elodiff2 <- as.numeric(GamesDatabaseCleanB$elowhite)-
as.numeric(GamesDatabaseCleanB$eloblack)
GamesDatabaseCleanB <- cbind(GamesDatabaseCleanB, elodiff2)
GamesDatabaseCleanB <- cbind(GamesDatabaseCleanB, elodiff)

## After this, the procedure would be as in the last case.
## The Model is created for GamesDatabaseCleanA and predicted for
## the test sample
```

– DISTANCE-BASED REGRESSION

```
## As in the past parts of the Appendix, we will have different
## scenarios for this last
## part, specifically 2. The DBR done after 25 movements and after
## 40 movements.
## We will show one of the methods and explain where would the
## changes take place.

library("dbstats")

predictions25<-rep(0,200)
predictions40<-rep(0,200)

Distancias25_D2<-Distancias25^2+t(Distancias25^2)
Distancias40_D2<-Distancias40^2+t(Distancias40^2)

DistEntre2y3.25_D2<-DistEntre2y3.25^2
DistEntre2y3_D2<-DistEntre2y3^2

## We coerce the classes
class(Distancias25_D2)<-"D2"
class(Distancias40_D2)<-"D2"
class(DistEntre2y3.25_D2)<-"D2"
class(DistEntre2y3_D2)<-"D2"

y <-as.numeric(GamesDatabaseUpd3$result=='w')

## ---------------------------------Modeling starts
dbglmDss25 <- dbglm(Distancias25_D2,y,family = binomial(link =
"logit"), method="rel.gvar")

dbglmDss40 <- dbglm(Distancias40_D2,y,family = binomial(link =
"logit"), method="rel.gvar")

for(i in 1:200){
predictions25[i]<-predict(dbglmDss25, DistEntre2y3.25_D2[i,],
type.var="D2", type.pred="response")}

for(i in 1:200){
predictions40[i]<-predict(dbglmDss40, DistEntre2y3_D2[i,],
type.var="D2", type.pred="response")}

## ---------------------------------Second Part
DraBla25<-(GamesDatabaseUpd3$result=='b' |
GamesDatabaseUpd3$result=='d')

Bistancias25_D2 <- Distancias25_D2[DraBla25,DraBla25]
Bistancias40_D2 <- Distancias40_D2[DraBla25,DraBla25]

DraBla25.df <-GamesDatabaseUpd3[DraBla25,]

DraBla25.df$black <- (DraBla25.df$result=='b')

yy<-as.numeric(DraBla25.df$black)

class(Bistancias25_D2)<-"D2"
class(Bistancias40_D2)<-"D2"

predicsions25<-rep(0,200)
predicsions40<-rep(0,200)

dbglmBss25 <- dbglm(Bistancias25_D2,yy,family = binomial(link =
"logit"), method="rel.gvar")

dbglmBss40 <- dbglm(Bistancias40_D2,yy,family = binomial(link =
"logit"), method="rel.gvar")

for(i in 1:200){
predicsions25[i]<-predict(dbglmBss25, DistEntre2y3.25_D2[i,
DraBla25], type.var="D2", type.pred="response")}

for(i in 1:200){
predicsions40[i]<-predict(dbglmBss40, DistEntre2y3_D2[i, DraBla25],
type.var="D2", type.pred="response")}

jd25<-predictions25
jd40<-predictions40

kd25<-predicsions25*(1-predictions25)
kd40<-predicsions40*(1-predictions40)
```

```
dd25<-1-(jd25+kd25)
dd40<-1-(jd40+kd40)


pred.result25 <- unlist(apply(cbind(kd25,dd25,jd25),1,which.max))
TableDist25<-table(GamesDatabaseUpd2$result, pred.result25)

pred.result40 <- unlist(apply(cbind(kd40,dd40,jd40),1,which.max))
TableDist40<-table(GamesDatabaseUpd2$result, pred.result40)

## ---------- Local Method

predictionsL25<-rep(0,200)
predictionsL40<-rep(0,200)

y<-(GamesDatabaseUpd3$result=='w')
y<-as.numeric(y)

ldbglmDss25<-ldbglm(Distancias25_D2,y=y,family = binomial(link =
"logit"),
method.h="user.h", rel.gvar=0.75)

ldbglmDss40<-ldbglm(Distancias40_D2,y=y,family = binomial(link =
"logit"),
method.h="user.h", rel.gvar=0.75)

for(i in 1:200){
predictionsL25[i]<-predict(ldbglmDss25, DistEntre2y3.25_D2[i,],
type.var="D2", type.pred="response")}

for(i in 1:200){
predictionsL40[i]<-predict(ldbglmDss40, DistEntre2y3_D2[i,],
type.var="D2", type.pred="response")}

## ----------------------Black Draw Part

DraBla25<-(GamesDatabaseUpd3$result=='b' |
GamesDatabaseUpd3$result=='d')

Bistancias25_D2 <- Distancias25_D2[DraBla25,DraBla25]
Bistancias40_D2 <- Distancias40_D2[DraBla25,DraBla25]

DraBla25.df<-GamesDatabaseUpd3[DraBla25,]

DraBla25.df$black <- (DraBla25.df$result=='b')

yy<-as.numeric(DraBla25.df$black)

class(Bistancias25_D2)<-"D2"
class(Bistancias40_D2)<-"D2"

predicsionsL25<-rep(0,200)
predicsionsL40<-rep(0,200)

ldbglmBss25 <- ldbglm(Bistancias25_D2,y=yy,family = binomial(link =
"logit"),
method.h="user.h", rel.gvar=0.75)
```

```
ldbglmBss40 <- ldbglm(Bistancias40_D2,y=yy,family = binomial(link =
"logit"),
method.h="user.h", rel.gvar=0.75)


for(i in 1:200){
predicsionsL25[i] <-predict(ldbglmBss25, DistEntre2y3.25_D2[i,
DraBla25], type.var="D2", type.pred="response")}

for(i in 1:200){
predicsionsL40[i] <-predict(ldbglmBss40, DistEntre2y3_D2[i,
DraBla25], type.var="D2", type.pred="response")}

jd25<-as.numeric(predictionsL25)
jd40<-as.numeric(predictionsL40)

kd25<-as.numeric(predicsionsL25)*(1-jd25)
kd40<-as.numeric(predicsionsL40)*(1-jd40)

dd25<-1-(jd25+kd25)
dd40<-1-(jd40+kd40)


pred.resultL25 <- unlist(apply(cbind(kd25,dd25,jd25),1,which.max))
TableDistL25 <-table(GamesDatabaseUpd2$result, pred.resultL25)

pred.resultL40 <- unlist(apply(cbind(kd40,dd40,jd40),1,which.max))
TableDistL40 <-table(GamesDatabaseUpd2$result, pred.resultL40)
```

BIBLIOGRAPHY

ARENAS, C., & CUADRAS, M. (2002). Recent statistical methods based on distances. Contributions to Science, 2 (2), 183-191.

BAATH R. (2015) Big Data and Chess Follow-up: Predictive Piece Values Over the Course of a Game [Online] Publishable Stuff. Available from: http://www.sumsar.net/blog/2015/06/big-data-and-chess-followup/

BOJ, E., CABALLE, A., DELICADO, P., FORTIANA, J., (2013). Dbstats: Distanced-based statistics. R package version 1.0.4

CARNEGIE MELLON UNIVERSITY (2014) Statistical Computing. [Online] Available from: http://www.stat.cmu.edu/~cshalizi/statcomp/14/

CUADRAS, C. M. (1989). Distance analysis in discrimination and classification using both continuous and categorical variables. Statistical data analysis and inference, 459-473.

MITCHELL, R. (2015). Web Scraping with Python: Collecting Data from the Modern Web. O'Reilly Media, Inc.

RBLOGGERS (2011) First Attempt at Chess Data Mining [Online] Available from: http://www.r-bloggers.com/first-attempt-at-chess-data-mining/

ROSS, D. (2007) Arpad Elo and the Elo Rating System. [Online] Available from: http://en.chessbase.com/post/arpad-elo-and-the-elo-rating-system

RSTUDIO BLOG (2014) RVest: Easy Web Scraping with R [Online] Available from:

http://blog.rstudio.org/2014/11/24/rvest-easy-web-scraping-with-r/

SHENK, D. (2007) The Immortal Game: A History of Chess. Reprint Edition. Anchor Publisher.

WIKIMEDIA COMMONS (2015) Illustrations of unit circles in different p-norms [Online] Available from: https://commons.wikimedia.org/wiki/File:Vector_norms.svg

WORLD CHESS FEDERATION (2014) Laws of Chess, [Online] Available from: https://www.fide.com/component/handbook/?id=171&view=article