University of Osnabrück

Social Data Mining

Technical Report Task 1

Wolfgang Groß
Matrikel-Nr. 957842

Ramona Leenings
Matrikel-Nr. 957842

WS 2014/2015

Content

Development Environment

In order to implement the given task, it was decided to use the software environment and programming language *R*, which was developed for statistical computing and graphical outputs. A software called *RStudio IDE* is used as development environment.

Data Preprocessing

Data Source

The data sources provided by the Uni Osnabrück are csv files containing data from the social network *Twitter*. There are three files containing data from the 12th of July to the 14th of July 2014 respectively. During this time, the FIFA world cup finals took place.

The csv files contain the following information: the unique identifier of a post, called *tweet,* the unique identifier of the corresponding user, a timestamp, values for longitude and latitude, as well as a string indicating the location of the user when making the tweet, and the content of the tweet itself. The values for longitude and latitude, as well as for the location aren't necessarily given.

The separator used in the csv files is a tab character.

Cleaning the CSV files

Irregularities occurring in the csv files. In order to work with the csv files, they needed to be preprocessed and cleaned, as there were some irregularities occurring.

First of all, the separator used (tab) wasn't exclusively used for separation purposes, but also occurred in the text of a post, that is, in the content of the tweet. When reading csv files, the libraries programmed for this task expect the separator to be used exclusively for separating the data. Also the amounts of separators are expected to equal for each line. When a separator character occurs in one of the values that it is supposed to separate, the value is splitted and interpreted as two columns, when in fact it is only one and thus producing an irregular pattern with different numbers of columns.

Also line breaking characters were used in the text content of a tweet, so invalid new lines of data were produced. Within the csv file schema every line corresponds to an instance of data. Therefore wrong

placed line breaking characters splits data that is supposed to be in one line to two lines and therefore endangers a correct interpretation of the data.

Even though libraries used for this task should have some error handling for occurrences like this, it remains unclear how exactly the data is interpreted in the end and if the resulting representation of the values is sensible. As we have a huge amount of data, this cannot be controlled manually. So the libraries for reading the csv files have to produce a reliable representation, and therefore the csv files should be cleaned from occurrences like this to be perfectly readable.

In addition, big amounts of data are processed, so every irregularity that has to be dealt with further extends the time and processing power needed, so that clean data should also be preferred regarding computational time and power.

Changes to the csv file. In order to prepare the data and to clean it from corrupted lines, some changes to the csv files were planned and implemented into a small program.

It was decided to chance the separator character from tab to semicolon. Because a semicolon is also a character used in regular text and is also part of some text character smilies, it was decided to first of all remove every semicolon from the file. This brings along a loss of information, but this was decided to be of no importance for the purposes and aims that were intended.

Afterwards the first six tabs indicating the separation of data were replaced with semicolons, so that any other tab remaining in the text could not lead to irritations.

Every line was checked in order to clean the file from corrupt data. It was checked if the line starts with a numeric value (the tweet ID) and if not, this is an indication for a misplaced line break within the tweet content and the line was removed. Afterwards it was made sure that the line splits correctly into seven columns, otherwise it was removed as well.

For the reading of the original file and the writing of the new, cleaned file a stream reader and stream writer were used. Due to the big size of the file an implementation was needed that allowed for a processing of the file line by line, in contrast to loading the whole file into the working memory.

Code Listing for the cleaning of the csv files. The program for cleaning was written in the .NET framework 4.0 with the programming language Visual Basic.NET. This was due to pragmatic reasons. We are familiar with the programming language and therefore the time needed for the implementation was small.

```
  Sub CleanCSV()
        Try
            'Input Path for original CSV files
            Dim inputpath As String = "C:/Users/mobi2/Documents/Coxi/WS1415/Social Data
Mining/Data/Raw/"

            'Output Path for new, cleaned CSV files
            Dim outputpath As String = "C:/Users/mobi2/Documents/Coxi/WS1415/Social Data
Mining/Data/Clean/"

            'Names of files that should be processed
            Dim listOfFiles As New List(Of String)
            listOfFiles.Add("13July_19xxxx.csv")
            listOfFiles.Add("numeric_20140712.csv")
            listOfFiles.Add("numeric_20140713.csv")
            listOfFiles.Add("numeric_20140714.csv")

            'Counter for the line numbers of the input CSV
            Dim line_counter As Integer = 0

            'Process each file from the filename list
            For Each file In listOfFiles

                Console.WriteLine("Processing " + file + "...")

                'Use Streamreader and Streamwriter to be able to read file line by line
                '-->prevent overloading of RAM which could be caused by loading file at
once
                Dim sr As StreamReader = New StreamReader(inputpath + file)
                Dim sw As StreamWriter = New StreamWriter(outputpath + file)

                'Read every line
                Dim line As String
                line = sr.ReadLine()
                While line IsNot Nothing

                    line_counter += 1

                    'Remove every ";" because we want to use ";" as a seperator
                    line = line.Replace(";", "")

                    'Split the line at tab with a maximum of 7 outputs
                    Dim listOfItems = line.Split(vbTab, 7, StringSplitOptions.None)

                    'If splitting was successfull and
                    If listOfItems IsNot Nothing AndAlso listOfItems.Count > 0 Then
                        'if the value of the first number is a numeric (otherwise corrupt
line)
                        If IsNumeric(listOfItems(0)) Then

                            'Build output String
```

```
                        Dim output As String = ""
                        Dim counter As Integer = 0
                        For i As Integer = 0 To listOfItems.Count - 1
                            If i < listOfItems.Count - 1 Then
                                output += listOfItems(i) + ";"
                            Else
                                output += listOfItems(i)
                            End If
                        Next

                        'Count semicolons in output
                        Dim count_of_semicolons = output.Split(";"c).Length - 1
                        'If the line is valid it must contain 6 semicolons
                        If count_of_semicolons = 6 Then
                            sw.WriteLine(output)
                        ElseIf count_of_semicolons < 6 Or count_of_semicolons > 6 Then
                            'Otherwise the program should stop
                            'so that we can analyze the line and see what happened
                            Debugger.Break()
                        End If

                    End If
                End If

                'Read next line
                line = sr.ReadLine()
            End While

            'Close Streamreader and Streamwriter
            sr.Close()
            sw.Close()

        Next

    Catch E As Exception
        Console.WriteLine(E.Message)
        Console.ReadLine()
    End Try
End Sub
```

Compressing the data into an R-specific format

R brings along an R-specific file format *Rda*, which compresses the data and can be read faster than csv files. For this purpose every csv file was read by a function called *fread* provided by the R package *data.table*. This implementation of reading the csv file is more sensitive for errors than the standard implement of R, but also much faster and as we are dealing with big data, better suited for our

purposes.[1] Afterwards the data was stores in the file format *Rda* with the help of the R function *saveRDS().*

Code Listing for loading the csv files and saving it to Rda

```
#load data into workspace and set workspace

######################
#setting environment
library(bit64)
library(data.table)
setwd("~/Documents/Coxi/WS1415/Social Data Mining/Data/Clean")
#setwd("~/Documents/programmierung/Social Data Mining All")

######################
#loading & saving data

#build in R Routines for reading CSV
#dat <- read.csv('13July_19xxxx.csv',sep='\t')
#dat <- read.csv('13July_19xxxx_with_semi2.csv',sep=';')

#Using fRead because it is much faster
#1. Reading CSV file
#2. save data in Rda file to compress it and load it faster

#2014_07_13_PART
dat_small <- fread('~/Coxi/WS1415/Social Data
Mining/Data/Clean/13July_19xxxx.csv',header=F,sep=';')
saveRDS(dat, file="~Coxi/WS1415/Social Data Mining/Data/Rda/13July_xxxx.Rda")

#2014_07_12
dat_twelve <- fread('~/Coxi/WS1415/Social Data
Mining/Data/Clean/numeric_20140712.csv',header=F,sep=';')
saveRDS(dat_twelve, file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140712.Rda")

#2014_07_13
dat_thirteen <- fread('~/Coxi/WS1415/Social Data
Mining/Data/Clean/numeric_20140713.csv',header=F,sep=';')
saveRDS(dat_thirteen, file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140713.Rda")

#2014_07_14
dat_fourteen <- fread('~/Coxi/WS1415/Social Data
Mining/Data/Clean/numeric_20140714.csv',header=F,sep=';')
saveRDS(dat_fourteen, file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140714_1.Rda")
```

[1] For a comparison of the processing speed for different R libraries see: http://statcompute.wordpress.com/2014/02/11/efficiency-of-importing-large-csv-files-in-r/.

Manipulating the Data

Calculating

The task was to calculate the number of tweets per minute and hour for each of the three days. The below-mentioned code listing shows the operations for this task. The script was executed for each of the days, with the corresponding line uncommented for the loading of correct Rda file for the day.

Plotting

The plotting of the data was done with the help of the library ggplot2, a freely available R package that extends the built-in plotting function for R. The plots for each day are shown in the next section. The below-mentioned code listing shows the R instructions used.

Code Listing for the calculations and the plotting of Task 1

```
#####################
#setting environment
library(ggplot2)
library(bit64)
library(data.table)
setwd("~/Documents/Coxi/WS1415/Social Data Mining/Data/")
#setwd("~/Documents/programmierung/Social Data Mining All")

#####################
#loading & saving data
#small_13-July_file
dat <- readRDS(file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/13July_xxxx.Rda")

#2014-07-12
#dat <- readRDS(file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140712.Rda")

#2014-07-13
#dat <- readRDS(file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140713.Rda")

#2014-07-14
#dat <- readRDS(file="~/Coxi/WS1415/Social Data
Mining/Data/Rda/numeric_20140714.Rda")

#####################
#count for each time the number of posts
count = c()
#remove the doubles form dat$Timestamp; get unique
times <- unique(dat$V3)
#table gets number of elements
count  <- data.frame(table(dat$V3))
```

```
#sum up per minute and per hour
sumMinute <- c()
for(i in 1:as.integer(length(count$Freq)/60)){sumMinute <-
append(sumMinute,sum(count$Freq[(i*60-59):(i*60)]))}
sumHour <- c()
for(k in 1:(length(sumMinute)/60)){sumHour <-
append(sumHour,sum(sumMinute[(k*60-59):(k*60)]))}

#####################
#plot
#convert Time into real time format
timeReal <- strptime(as.character(times),format="%Y%m%d%H%M%S")

df <- data.frame(timeReal,count$Freq)
ggplot(df,aes(x=timeReal,y=count$Freq)) + geom_line()
```

Interpretation of the data

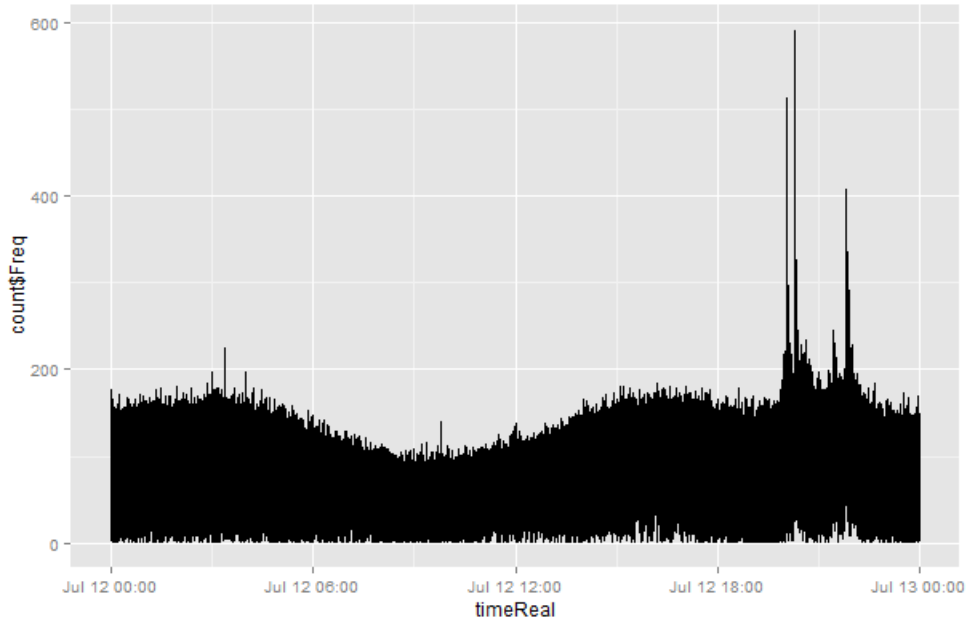

Figure 1 Plot 2014-07-12

Brasilien:Niederlande - 0:3

Anstoß: 21 Uhr GMT

Tore 3’, 17’ und 90’+1 : 21:03, 21:17 und (91+30 Halbzeit) 23:01?

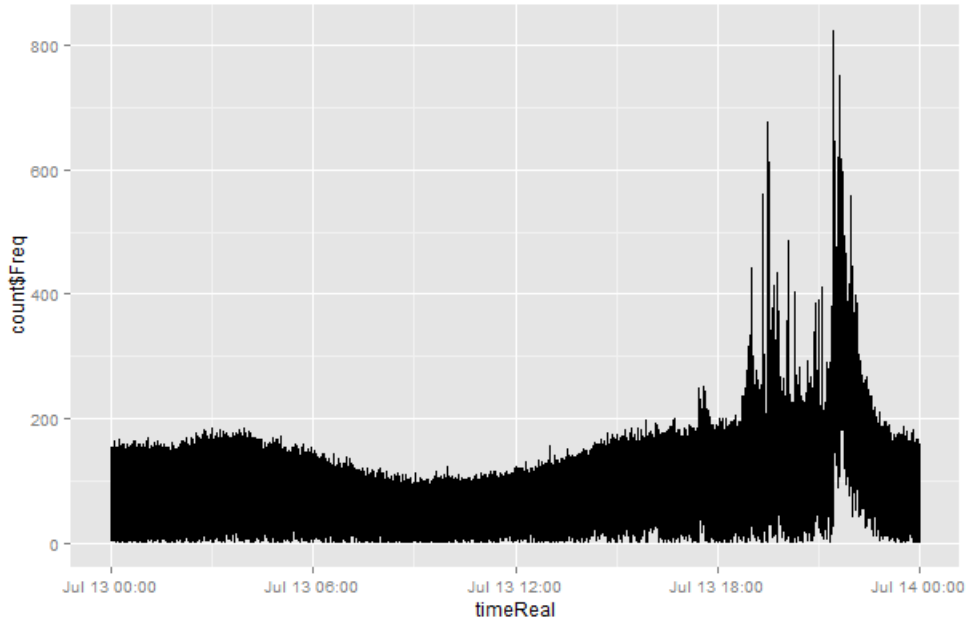

Figure 2 Plot 2014-07-13

Germany:Argentinia – 1:0

Anstoß 20 GMT, D: 21 Uhr MESZ,

Tor in 113‘, 20 Uhr + 113min +30min Halbzeit= 23:13?

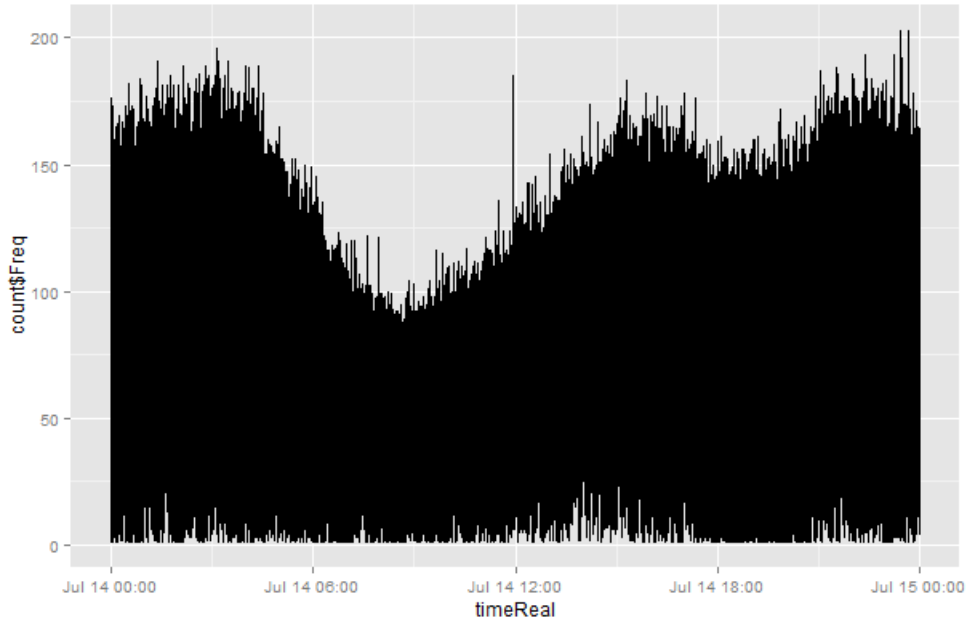

Figure 3 Plot 2014-07-14

Demonstration of Results

For different time resolutions we made a little web application with the r package shiny. The application in hosted the shinyapps server (https://wolfganggross.shinyapps.io/shiny/ .

The full code can be found on the github repository https://github.com/WolfgangGross/Social_Data_Mining .

Additional there is a presentation for this repository hosted on github created with the r package slidify. http://wolfganggross.github.io/Social_Data_Mining/ (work in progress).