R package, UTDEventData *

for extracting event data from the UTD database

Dr. Patrick T. Brandt[†]
Dr. Vito D'Orazio[‡]
Hyoungah (Kate) Kim[§]

School of Economic, Political, and Policy Sciences
The University of Texas at Dallas

2019-01-29

Abstract

The R library, `UTDEventData`, provides direct access to the UTD Event Database. Using this library, researchers may query the event data stored at the UTD's API server and load them into R for subsequent analysis. This allows more sophisticated users a direct lingua franca to the data. Several methods are prepared in the library to extract and to analize the data according to users' preference. This library also provides the citation functions ont only for data tables but also the library itself.

The UTD API server contains the five different data tables; Real-time Phoenix, ICEWS, and three other Cline's Phoenix data sets. With the several searching and extracting methods in this library, political and social scientists can explore the contents of each data table and can obtain the historic and real-time data for their research. We expect that this R library provides the better evivornment in accessing to the event data for R users, so the number of studies on the large scale event data will increase in this dawing of the big data age.

Contents

[†]pbrandt@utdallas.edu
[‡]dorazio@utdallas.edu
[§]hyoungah.kim@utdallas.edu

*This is a part of the project, titled "Modernizing Political Event Data for Big Data Social Science Research" and funded by NSF RIDIR SBE-SMA-1539302.

1 Introduction

This R package allows users to extract a data set from the API Event Data server at the University of Texas at Dallas (UTD). The project of the UTDEventData R package is in progress and has been updated day by day. Your comments, feedback, and suggestions are welcome so that more user friendly methods are prepared in the library. If you have questions in using the package, please contact Kate Kim (hyoungah.kim@utdallas.edu) at UTD.

This package requires you to have an API key to access to the UTD data server. Please find the following link and fill out the form to obtain an API key: http://eventdata.utdallas.edu/signup.

You can install the package from the UTDEventData GitHub page in R with the following syntax.

```
# install the package without the vignette
devtools::install_github("KateHyoung/UTDEventData")

# install the package with the vignette
devtools::install_github("KateHyoung/UTDEventData", build_vignettes = TRUE)
```

The UTD Event Data server has the five different event data tables.

Table 1: Data Table Information

Data Table	Timeline	Further Information
Phoenix RT	Oct. 2017 - Today	OEDA
ICEWS	1995 - Oct. 2018	ICEWS Dataverse
Cline Phoenix NYT	1945 - 2005	Cline Center
Cline Phoenix FBIS	1945 - 2005	Cline Center
Cline Phoenix SWB	1979 - 2015	Cline Center

You can find the information of each data table in the specified website such as the contents of data, their entities, and attributes.

1.1 Computer Enviroments This Package Has Tested

The codes/functions in this documentation have been tested in the following versions of:

- Microsoft Open R 3.4.0
- R-3.4.3 for Windows (32/64 bit)
- OS X 10.12.2
- R-2.4.3 for Mac

2 Functions

2.1 Searching functions

The search functions help users to explore 1) the data tables in the UTD API server and 2) the attributes (variables) in a particular table. Moreover, Users can view the 100 sample data of each data table so as to check its data structure and particular values and patterns of attributes before downloading them on thier own computers.

`DataTables()` returns all data table names in the UTD server.

```
# returning all data table the server contains with entering an API key
DataTables(api_key = " ")
"'PHOENIX_RT', 'CLINE_PHOENIX_SWB', 'CLINE_PHOENIX_FBIS', 'CLINE_PHOENIX_NYT', 'ICEWS'"

# save an API key as a string value and use it so as not to repeat
# typing the key string in other functions
k <- "...api key...."
DataTables(k)
"'PHOENIX_RT', 'CLINE_PHOENIX_SWB', 'CLINE_PHOENIX_FBIS', 'CLINE_PHOENIX_NYT', 'ICEWS'"
```

`tableVar()` returns the list of variables (attributes) in a particular data table a user specified in the function. For example, the variables in the Phoenix real-time (RT) data can be obtained as shown in the following example.

```
tableVar(api_key = "...", table = "phoenix_rt")

# in the manner of using a saved API string to avoid the repeation of
# API key typing
k <- "...api key...."
tableVar(k, "Phoenix_rt")

tableVar(k, "Icews")

tableVar(k, "Cline_Phoenix_swb")
```

This function is **not** case sensitive, so a user may type either lower case or upper case of the data table name. However, the full name as returned by `Table()` should be entered in the function when applying the table name in other functions. For instance, `"cline_Phoenix_swb"` and `"CLINE_PHOENIX_SWB"` will return the same variables, but `"cline_phenix"` will return noting.

This function has another feature that a user can look up a particular variable in a data set. For example, one who may wonder the ICEWS data have a variable named "target" can type the string of the certain variable as follows;

```
tableVar(api_key = "...", table = "icews", lword = "target")

# when a user wants to know the attribute that labeled as 'target' in
# ICEWS
k <- "..api key..."
tableVar(k, table = "icews", lword = "target")
"\" Target Name\"    \" Target Sectors\", ...."
```

`previewData()` shows the 100 obsertaions of a data table as an example in order to explore the features of certain variables. Users may need to confirm the pattern to extract the correct data they are interested in. This function can be useful when building a query block with a regular expression (`returnRegExp()`) function.

```
dataSample <- previewData(api_key = " ", table_name = "PHOENIX_RT")
View(dataSample)
```

2.2 Subsetting function I

This library provides two ways of event data subsetting from the UTD server. The first way is fixed and simple as entering only countries' names and time ranges users prefer with `pullData()`. The second way is flexible and extensible in users' inputs of the events as allowing combinations of multiple data queries such as time, locations, event features, and so on.

`pullData()` returns a list of data and citaion texts.

Please confirm the format of time ranges, which is "YYYYMMDD" and note the corresponding time period of certain data in Table 1. If the given time range in the function falls outside the timeline, the function will return `list()` as its result. That means the requested data set is empty.

The country names can be either full names or the ISO-Alpha3 code. The country inputs indicate the geolocation of events such as the countries in which a particular event happened in a news article. In this inputs, please use a consistent format when you type countries names. We recommend the ISO-3 code format in order to reduce systemic errors in the function.

The function returns the citation texts & BibTeX for publication at the end of data retrieval. To avoid the printing them in your R-console, please turn off the `citation` option by choosing "FALSE" in the function.

The following code is the example usage of `pullData()`.

```
## several examples of different data tables with citation texts
k <- "api key..."
subset1 <- pullData(api_key = k, table_name = "phoenix_rt", country = list("canada",
    "China"), start = "20171101", end = "20171102", T)
# to store the data only
data <- subset1$data

# to print the citation texts
subset1$citation

# to avoid the citation texts
data <- pullData(k, "phoenix_rt", list("canada", "China"), "20171101",
    "20171102", citation = FALSE)
```

2.2.0.1 ***Note*** As illustrated, the data are retrieved by the country names and the time range a user specified. The country in a data set means the country where a particular event happened. This country could not be the same with the source or target countries of an event. A researcher should pay attention to this attribute when conducting research. This feature of each variable can be scrutinized by exploring the variables of the data set downloaded by `tableVar()` or `previewData()`.

2.2.1 Reference class for appying an API key to the functions

The package has a reference class, named `Table()` for users not to repeat the input of an API key into the searching and subsetting functions. This function works only with `DataTables()`, `tableVar()`, and `pullData()`. Some basic usages are as follows;

```
# creating an object
obj <- Table$new()
```

```
# setting an object of an API key
obj$setAPIKey("....")
obj$DataTables()  # returns the available data tables in the UTD server
obj$tableVar("cline_Phoenix_NYT")

# when a user wants to subset real-time data ('phoenix_rt) from
# 20171101 to 20171102 on MEX(Mexico)
obj$pullData("Phoenix_rt", list("MEX"), start = "20171101", end = "20171102")
```

2.3 Subsetting functions II

The second method to extracting the data from the UTD API server is more flexible and extensible in building subset queries. Some functions in this method require an API key like `pullData()` so users may apply it more frequently, but they have benefits to build user-friendly query blocks. For instance, while `pullData()` function provides fixed query options such as country names and time ranges, the method of subsetting functions II facilitates users more discretion in queries such as country names, time ranges, a dyad of actors, latitude & longitude of event locations, and source & targets indications of certain events. By combining query blocks created by the aforementioned features, users can retrieve data at their own choices.

The query blocks can be built and stored by its specific functions. To obtain data without error messages, users must acquaint each function's usage illustrated in this document or help pages in the library. The details of how individual function works together with the other functions are illustrated in this section.

The subsetting function II consists of three groups. The first method is the data requesting function, which requests the data set to the server with built query blocks: `sendQuery()`.

The second group is creating query blocks according to the user's preference. These query blocks are the basis of subsetting information such as country names, locations (latitude and longitude), time ranges, a dyad relation, and so on. Moreover, a user can use any variable (attribute) in an API data table with the regular expression function, `returnRegExp()`.

The last group of is the connective functions of query blocks: `orList()` and `andList()`. These functions play a role of a logical operator to combine the query blocks as union and intersection respectively.

2.3.1 Data request function

A data set can be retrieved with the combinations generated with query block functions. The `sendQuery()` function requests particular data to the API server. A user should input an API key, a data table, and a list of queries created by `andList()`, `orList()`, or a single query blockin the function as shown in the following example code.

The function print the citation texts of publication and BibTeX formats at the end of the extracted data as default. To avoid this printing, please turn off the option by choosing "FALSE" in the method.

`sendQuery()` returns a list of data and citation.

```
# basic usage
sendQuery(api_key = "", tabl_name = "", query = list(), citation = TRUE)

# to store the ICEWS subset in the vector of myData without the
# citation query_block is a list of the quries built by the query block
# functions illustrated in following chapter
myData <- sendQuery(api_key, "icews", query_block, citation = TRUE)
# store the data only
myData <- myData$data
```

```
# print citation texts only
myData$citation

# without the citation text
myData <- sendQuery(api_key, "icews", query_block, citation = FALSE)
```

2.3.2 Query block functions

`returnRegExp()` can be used for a special case as indicating a particular value or a pattern of the variables in the API data table. To use this function, a user must aware the variables in a certain data table. The variable list in a particular table can be found with the function of `tableVar()` in the package and the pattern is confirmed by `previewData()`. The function requires users to have an API key, and a data table name, a pattern of interest events, and a field name (attribute or variable) in the data table.

The query block functions return the lists of queries.

```
# genrate a query for all source actors that involved in governments in
# events
govQuery <- returnRegExp(api_key = " ", table_name = "phoenix_rt", pattern = "GOV",
    field = "src_agent")
# to subset the cline_phoenix_nyt data by year == 2001
nytQuery <- returnRegExp("api_key", "cline_phoenix_nyt", "2001", "year")
```

`returnCountries()` is the function that creates the list of countries. The country means the geolocation where a certain event occurred in a news article. The function requires users to specify the names of the data table and countries. The ISO-3 Code format is recommended for the country names, but full country names can work in the function. The inputs are *case-insensitive*. But we recommend using a consist way in the country name input.

```
# generating a query of the United States and Canada as a country
# restraint for real-time event data
ctr <- returnCountries(table_name = "phoenix_rt", country = list("USA",
    "CAN"))
```

`returnTimes()` is the function that generates a query to return all events between two time points. The format of typing time should be "YYYYMMDD" in the order of the start and the end points of the time range. A user must identify a data table in the function.

```
# generates a query to return all events between July 27, 1980, and
# December 10, 2004 for ICEWS data
time <- returnTimes("icews", "19800727", "20041210")
```

`returnLatLon()` returns the geo-location boundary a user specifies with latitudes and longitudes. This function does not require a data table name, but the input should be ordered by '`lat1, lat2, lon1, lon2`. They are respectively the minimum and maximum values of the latitudes and minimum and maximum values of longitudes of the boundary.

```
# generate a query with a geo-location bountry with the latitude
# between -80 and 30 and the longitude between 20 and 80
locQuery <- returnLatLon(lat1 = -80, lat2 = 30, lon1 = 20, lon2 = 80)
```

The `returnDyad()` function creates a query of particular dyads of actors in countries. In the function, a user must specify a table name and source and target countries respectively. The ISO-3 codes for country names are recommended as function inputs, but full names of countries also work in this function. These inputs are not case-sensitive.

```
# genrate a query that a source country is Syria and a target country
# is the United States
dyad <- returnDyad(table_name = , source = "SYR", target = "USA")
```

2.3.3 Connective functions

These functions play a role to connect several query blocks created by aforementioned functions. The two functions, `andList()` and `orList()`, work as logical operators, **"and"** and **"or"**, respectively.

`andList()` is the function that returns the intersection of two or more query blocks. The stored queries should be specified in the `list()` format in the function.

`orList()` returns a union of two or more query blocks. The stored queries should be specified in the `list()` format in the function as shown in the following examples.

```
# combine stored query blocks such as 'time' or 'locQuery' created
# before
and_query <- andList(query_prep = list(locQuery, time))

# subset with two or more stored query blocks such as 'locQuery' or
# 'dyad'
or_query <- orList(query_prep = list(locQuery, dyad))
```

2.3.4 Examples of subsetting II

```
# examples of subsetting functions

# creating query blocks a country constrain of 'CHN' and 'USA'
k <- "api_key"
ctr <- returnCountries("phoenix_rt", list("CHN", "USA"))

# A query of time between 2017-11-1 and 2017-11-5
time <- returnTimes("phoenix_rt", "20171101", "20171105")

# a boolean logic, or, with the two query blocks
or_query <- orList(list(ctr, time))
# request a data set to the API server with the package citation
d1 <- sendQuery(k, "phoenix_rt", or_query, TRUE)

# to view the subset
head(d1$data, 10)
View(d1)

# a boolean logic, and, with the two query blocks
and_query <- andList(list(ctr, time))
d2 <- sendQuery(k, "phoenix_rt", and_query, TRUE)

# to view the subset
head(d2$data, 10)
View(d2)

# when a user wants to extract all event in US and China with the
# events for which the source was a government actor from the Phoenix
# real-time data
```

```
rgex <- returnRegExp(k, "phoenix_rt", "GOV", "src_agent")
q <- andList(list(ctr, rgex))
data <- sendQuery(k, "phoenix_rt", q, citation = FALSE)  # no citation

# to view the data because the option for citation was off, package's
# citation was not printed.
head(data, 10)
View(data)
```

Users can create several combinations of the query blocks with the other functions such as `returnLocation()` and `returnDyad()` according to their preferences in order to obtain preferred data sets.

Please note that an issue may occur if the large size of data is extracted by `sendQuery()`. More specifically, the issue can more frequently come with the `orList()` function, so you may encounter the memory issue when using `orList()` with several query blocks. Once the issue occurs, please increase a memory size allocated to your R program and re-run the functions. If a user keeps having the issue, please consider using `pullData()` for data subsetting. `pullData()` works more efficiently than `sendQuery()`. Specific information is illustrated in the next section.

2.3.5 An error message from `sendQuery()`

The aforementioned issue in `sendQuery()` can occur when the size of requested data is greater than memory allocated to R. The error message will suggest a solution such as increasing a memory size of a user's computer with the original error.

This issue is more frequently occurred in a Windows machine because of the memory cap the R program is assigned. Once a user has the issue, `getQuerySize()` should be drawn to estimate data sizes. After comparing the size of data and machine' RAM, a user may need to increase its maximum.

please see the following examples;

```
# estimate the data size you want to extract
getQuerySize(api_key = " ", table_name = " ", query = list())

# if the error message is noted, estimate the data a user has requested
getQuerySize(k, "phoenix_rt", q)

# check your memory limit only in the Windows system
memory.limit()

# increase its size if you need
memory.size(max = 120000)
```

2.4 Data citation function

`citeData()` function returns the text of citations of data and this R library for journal publication or research documentation. The returned text contains two different type of citations; 1) a text type of citations for a user to copy and paste them to journal papers or other documents, and 2) a BibTeX type for LaTeXusers. The input for table names in this function is case-insensitive, but you should have full data table names.

```
# for the citations for Cline Phoenix Event data
citeData(table_name = "cline_Phoenix_swb")

# for the citations for UTD real-time data
```

```r
citeData(table_name = "Phoenix_rt")

# for the citations for ICEWS
citeData(table_name = "ICEWS")
```

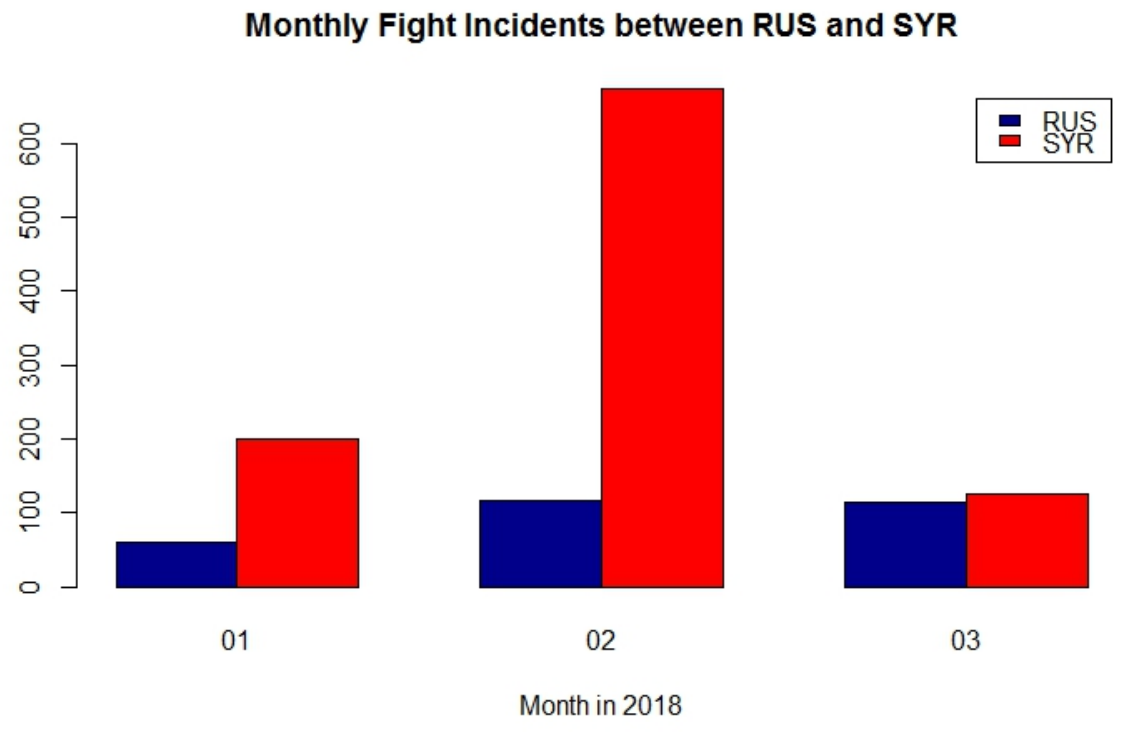
3 Usage examples

A user should prepare the R library and an API key from the UTD API server. Again, an API can be obtained on the sign-up page: http://eventdata.utdallas.edu/signup.

3.1 Example 1 - using `pullData()`

- After extracting all real-time event data between Russia and Syria from 20180101 to 20180331, draw the bar graphs by types of fights defined in the CAMEO code:

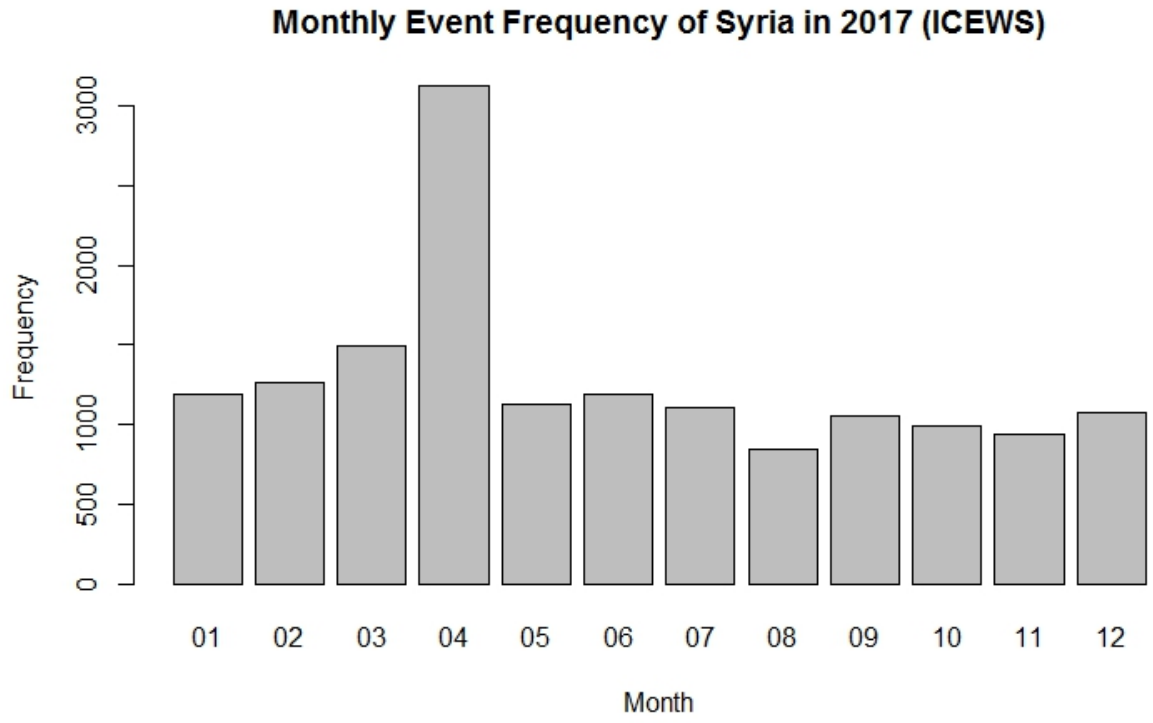
```
# Note: k <- '...provided API key'
dt <- pullData(k, "Phoenix_rt", list("RUS", "SYR"), start = "20180101",
    end = "20180331", citation = F)

## querying the fight event by CAMEO codes
Fgt <- dt[dt$code == "190" | dt$code == "191" | dt$code == "192" | dt$code ==
    "193" | dt$code == "194" | dt$code == "195" | dt$code == "1951" | dt$code ==
    "1952" | dt$code == "196", ]

Fgt <- Fgt[, 1:23]  ## removing url and oid

tb <- table(Fgt$country_code, Fgt$month)  # monthly incidents

barplot(tb, main = "Monthly Fight Incidents between RUS and SYR", col = c("darkblue",
    "red"), legend = rownames(tb), beside = TRUE, xlab = "Month in 2018")
```



3.2 Example 2 - using `sendQuery()`

- A similar data extraction to the previous example only with the functions of time and country query blocks. Please note that `sendQuery()` prints the API query syntax corresponding to user's specification. *If this syntax is thrown to your browser, you will have unstructured JSON data in the browser.*
- For extracting the data set of Russia and Syria from 1950 to 1960 from Cline Phoenix SWB and drawing the graph from the obtained data set:

```
# querying the fight event by CAMEO codes please checking the variable
# features and formats in a data set. Attribute names and formats vary
# by data tables

ctr <- returnCountries("icews", list("RUS", "SYR"))
t <- returnTimes("icews", "20170101", "20171231")
qr <- andList(list(ctr, t))
dt1 <- sendQuery(k, "icews", qr, citation = F)

dt1$`Event Date1` <- as.POSIXct(dt1$`Event Date`)
dt1$Month <- format.Date(dt1$`Event Date1`, "%m")
tab <- table(dt1$Country, dt1$Month)

barplot(tab, main = "Monthly Event Frequency of Syria in 2017 (ICEWS)",
    col = "gray", ylab = "Frequency", xlab = "Month in 2018")
```



3.3 Example 3 - using a dyad query block and `sendQuery()`

- Comparing the data set with the consistent query that Pakistan is the source and India is the target of events:

```
# creating the query of source = 'PAK' and target = 'IND' for ICEWS
query <- returnDyad("icews", "PAK", "IND")
tmp <- sendQuery(k, "icews", query, citation = F)
# the query for Phoenix_Cline_SWB
q.cline.swb <- returnDyad("cline_phoenix_swb", "PAK", "IND")
tmp.swb <- sendQuery(k, "cline_phoenix_swb", q.cline.swb, F)
# the query for Phoenix_Cline_FBIS
q.cline.fbis <- returnDyad("cline_phoenix_fbis", "PAK", "IND")
tmp.fbis <- sendQuery(k, "cline_phoenix_fbis", q.cline.fbis, F)
# the query for Phoenix_Cline_NYT
q.cline.nyt <- returnDyad("cline_phoenix_nyt", "PAK", "IND")
tmp.nyt <- sendQuery(k, "cline_phoenix_nyt", q.cline.nyt, F)
# save each observation as a data set and print it
Compare <- as.matrix(cbind(nrow(tmp), nrow(tmp.swb), nrow(tmp.fbis), nrow(tmp.nyt)))
colnames(Compare) <- c("ICEWS", "Phoenix SWB", "Phoenix FBIS", "Phoenix NYT")
xtable(Compare)
```

Table 2: The Result of Example 3

ICEWS	Phoenix SWB	Phoenix FBIS	Phoenix NYT
40731	1140	633	272

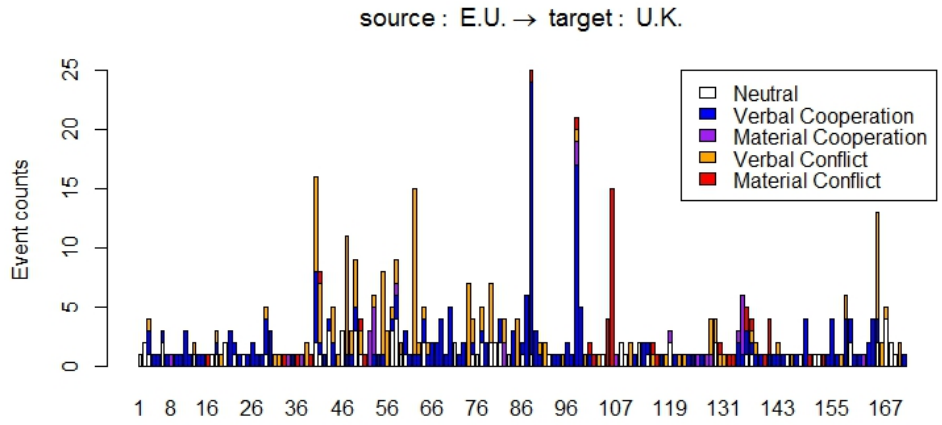
3.4 Example 4 - using a regular expression query block and `sendQuery()`

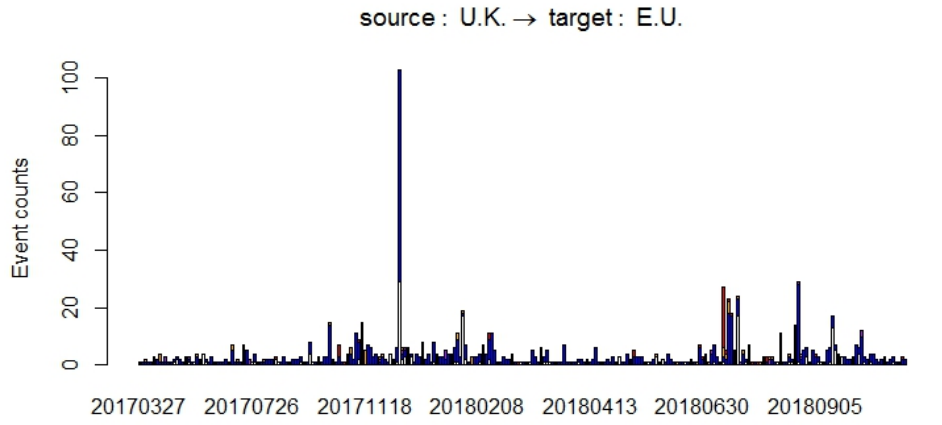
A user can extract the data by the feature of an attribute in a data table with the function of `RegExp()`.

- After extracting the data set of 'source': IGOEUREEC (EU) and 'target': United Kingdom (UK) and the one of *vice versa* from the real-time Phoenix data, create the graphs of their trends:

```
# sorce actor is EU
eu <- returnRegExp(k, "Phoenix_rt", "IGOEUREEC", "source")
# target actor is UK
uk <- returnRegExp(k, "Phoenix_rt", "GBR", "target")
dyad1 <- andList(list(eu, uk))
dd1 <- sendQuery(k, "Phoenix_rt", dyad1, F)

# source actor is UK
uk2 <- returnRegExp(k, "Phoenix_rt", "GBR", "source")
# target actor is EU
eu2 <- returnRegExp(k, "Phoenix_rt", "IGOEUREEC", "target")
dyad2 <- andList(list(eu2, uk2))
dd2 <- sendQuery(k, "Phoenix_rt", dyad2, F)

# reshaping data
EU_UK <- as.data.frame(table(dd1$date8, dd1$quad_class))
colnames(EU_UK) <- c("day", "quadclass", "count")
EU_UK <- reshape(EU_UK, idvar = "day", timevar = "quadclass", direction = "wide")
colnames(EU_UK) <- c("date", "qc0", "qc1", "qc2", "qc3", "qc4")
EU_UK <- t(EU_UK)
EU_UK <- EU_UK[-1, ]
```

```r
UK_EU <- as.data.frame(table(dd2$date8, dd2$quad_class))
colnames(UK_EU) <- c("day", "quadclass", "count")
UK_EU <- reshape(UK_EU, idvar = "day", timevar = "quadclass", direction = "wide")
colnames(UK_EU) <- c("date", "qc0", "qc1", "qc2", "qc3", "qc4")
UK_EU <- as.matrix(UK_EU)
UK_EU <- t(UK_EU)
colnames(UK_EU) <- UK_EU[1, ]
UK_EU <- UK_EU[-1, ]

# plotting
par(mfrow = c(2, 1))
barplot(EU_UK, col = c("white", "blue", "purple", "orange", "red"), ylab = "Event counts",
    main = expression(source:~E.U. %->% ~target:~U.K.))
legend("topright", c("Neutral", "Verbal Cooperation", "Material Cooperation",
    "Verbal Conflict", "Material Conflict"), fill = c("white", "blue",
    "purple", "orange", "red"))
barplot(UK_EU, col = c("white", "blue", "purple", "orange", "red"), ylab = "Event counts",
    main = expression(source:~U.K. %->% ~target:~E.U.))
```





4 Further information of the Event Data server at UTD

4.1 Real Time Event Data

The web-page of the real-time event data project
The web portal contains the general information about the real-time event data project conducted in the University of Texas at Dallas. As the portal site of the project, the basic information of the project and its outcome including related web-links are listed and organized.

4.2 Spec-real-time Server

The GitHub page for the API access on Jetstream at UTD
This GitHub page provides specific information of the direct access to the UTD server managed by Big Data Management and Analytic Lab with Mongo DB query syntax. The queries are expressed in JSON format.

4.3 Jetstream

Jetstream user guide
XSEDE user guide provides specific information of the Jetstream usage and its working system. You can explore other information of Jetstream in the linked websites.