sejmRP: An Information About Polish Diet - DRAFT

by Piotr Smuda, Przemysław Biecek

Abstract Open government initiatives aim to collect and expose data about functioning of public institutions, like parliament. In this article we present the **sejmRP** package that provides functions to scrap, access and analyze data from Polish Parliament as well as the infrastructure for storing the scrapped data. All applications presented here are related to data from Polish Diet's webpage but could be repeated on data from other parliaments.

Introduction

Open government is a concept of public institutions whose aim is to provide access to public information and proceedings of the government, as well as efforts to increase the transparency of public administration. The development of open government is closely linked to technological change and the availability of digital tools to facilitate communication.

For United States the availability of data is exceptional due to the Voteview database `https://voteview.polisci.ucla.edu/about` and the R package **Rvoteview**. But for many other countries hardly ever this information is released in a form, which is easy to analysis. Hence there is a need for tools that make it easier. One of these tools is **sejmRP** package, which contains functions to scrap data from Polish Diet's webpage about deputies, votings and deputies' statements from seventh to eighth term of office and insert this data into database. As is generally known databases enable facile access to data, so its analysis becomes easier.

At the end of 2015 there was a change of the party in power in Polish Diet. Since that time, many things happened in Polish politics that have been received both positively and negatively. Thanks to **sejmRP** package there is a possibility to compare present situation in Polish politics to that from four years ago. Without a doubt the results of this analysis would be very interesting.

The rest of this article has following structure. In the section ... we present general overview of the package, in the section ... we present the structure of the database in which the scrapped data is stored, in the section ... we present functions for scrapping and accessing the data and in the section ... we present selected use cases.

Structure of database

sejmRP package provides a set of functions, that can be used to scrap data from Polish Diet's webpage `http://www.sejm.gov.pl/` and insert this data into PostgreSQL database.

The database is divided into four separate tables:

- deputies,
- votings,
- votes,
- statements,

whose Entity Relationship Diagram (ERD) looks as follows:

It is important to highlight that, there is a widely available database on MI^2 Group's server. This database is updated ones a week, so you do not need to bother if it contains up-to-date data. Moreover, if you do not want to use the database from MI^2 Group's server, you can create one's own thanks to functions included into **sejmRP** package.

Functions

Functions in **sejmRP** package can be divided into three groups:

1. functions to manage the database,
2. functions to scrap data from Polish Diet's webpage and insert it to database,
3. functions to read data from the database.

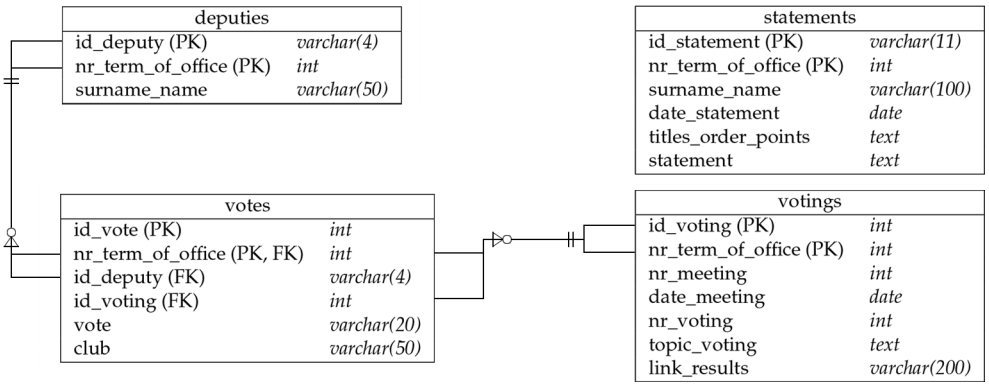

Figure 1: ERD of database.

In addition, functions from the second and the third groups are designed exclusively for each table in the database. The following table shows this devision:

subject	database	deputies table	votings table
managing functions / scraping functions	create_database remove_database	deputies_create_table deputies_update_table deputies_add_new deputies_get_data deputies_get_ids	votings_create_table votings_update_table votings_get_date votings_get_meetings_links votings_get_meetings_table votings_get_votings_links votings_get_votings_table
reading functions		get_deputies_table	get_votings_table

votes table	statements table
votes_create_table votes_update_table votes_get_clubs_links votes_get_results votes_match_deputies_ids	statements_create_table statements_update_table statements_get_statement statements_get_statements_data statements_get_statements_table
get_votes_table get_filtered_votes	get_statements_table

Table 1: List of functions for each table from database.

Database managing functions

There are two functions intended for database managment in **sejmRP** package: `create_database()` and `remove_database()`. The first function creates a database with four empty tables: *deputies*, *votings*, *votes*, *statements*; where the second removes whole database, so be careful with using it. To create the database just type in R:

```
> create_database(dbname, user, password, host)
```

where

- `dbname` is a name of database on the server,
- `user` is a username,
- `password` is a password to the database,
- `host` is an address of host or host's IP.

Scraping functions

In this group, functions can be specified for *auxiliary functions*, which scrap data and *principal functions*, which insert this data do the database. For each table there are different sets of functions, what is marked by the prefixes in the names of the functions.

In addition, there is a determined order of the principal functions call. For the first time, when the table is empty, you should use `*_create_table` and after that you should only use `*_update_table`. Differences between in the code structure of these functions are subtle but at the same time very essential. Moreover, due to connections in the database, it is important to call principal functions for each table in proper order: *deputies, votings, votes and statements*. Notice that there is also a possibility of choosing the number of term of office, so remember to scrap data firstly from seventh and then from eighth term of office.

Now let us describe briefly usage of *scraping functions*. Suppose that the database with empty tables was created, so you can start to completing it with data. First of all, as it was said, you should complete the *deputies* table. To do that use:

```
> deputies_create_table(dbname, user, password, host, nr_term_of_office = 7)
```

This function, thanks to auxiliary functions, scraps active and inactive deputies' data from Polish Diet's webpage and inserts it to the table. If you want create *deputies* table with data from eighth term of office, choose `nr_term_of_office = 8`. In the case of you want to get only deputies' data as a data frame you can try:

```
> deputies_get_data(type, nr_term_of_office = 7)
```

where `type` describes deputies' acivity, so you can choose between active and inactive deputies from seventh or eighth term of office. To update *deputies* table use:

```
> deputies_update_table(dbname, user, password, host, nr_term_of_office = 7)
```

Remember always to choose a proper value of `nr_term_of_office` variable, because otherwise you can clutter the database. After that we should complete *votings* table with:

```
> votings_create_table(dbname, user, password, host, nr_term_of_office = 7)
```

This function scraps all information about votings from votings on meetings page for `nr_term_of_office = 7` and votings on meetings page for `nr_term_of_office = 8` and inserts it to the table. If you want to update *votings* table, try:

```
> votings_update_table(dbname, user, password, host, nr_term_of_office = 7,
>         verbose = FALSE)
```

Again, you should remember to choose a proper value of `nr_term_of_office` variable. The last argument specifies if an additional info will be printed. If you are interested in extra information, you can use auxiliary functions:

```
> votings_get_meetings_table(
>        page = "http://www.sejm.gov.pl/Sejm8.nsf/agent.xsp?symbol=posglos&NrKadencji=8")
> votings_get_votings_table(page)
```

First of them enables downloading table with information about votings on meetings during eighth term of office as a data frame (the same can be done for seventh term of office). The second one does the same with votings during selected meeting.

When *votings* table is up-to-date, you need to complete *votes* table. To do that try:

```
> votes_create_table(dbname, user, password, host, nr_term_of_office = 7,
>        windows = .Platform$OS.type == 'windows')
```

The last argument specifies if you use Windows operation system. It also acts as guardian of the text encoding issue, because Polish language is encoded differently on various operating systems in R. To update *votes* table use:

```
> votes_update_table(dbname, user, password, host, nr_term_of_office = 7,
>        windows = .Platform$OS.type == "windows", verbose = FALSE)
```

If you want to know how deputies from selected club voted try:

```
> votes_get_results(page)
```

As page argument you should put page with chosen club's voting's results (example of page). As result you will get a data frame.

Finally, you should complete *statements* table with:

```
> statements_create_table(dbname, user, password, host, nr_term_of_office = 7)
```

To update *statements* table use:

```
> statements_update_table(dbname, user, password, host, nr_term_of_office = 7,
>        verbose = FALSE)
```

Like before if you are interested in extra information, you can use auxiliary functions:

```
> statements_get_statements_table(page)
> statements_get_statement(page, ...)
```

First of them enables downloading table with information about statements during chosen meeting (example of page). The second one gets statement's content (example of page with statement).

Reading functions

First of all, it should be mentioned that there are special, default values of reading functions' parameters to read data from the database from MI^2 Group's server:

- `dbname = 'sejmrp'`,
- `user = 'reader'`,
- `password = 'qux94874'`,
- `host = 'services.mini.pw.edu.pl'`.

Therefore, if you are only interested in reading data from tables, try:

```
> get_deputies_table(dbname = 'sejmrp', user = 'reader', password = 'qux94874',
>        host = 'services.mini.pw.edu.pl', sorted_by_id = TRUE,
>        windows = .Platform$OS.type == 'windows')
> get_votings_table(dbname = 'sejmrp', user = 'reader', password = 'qux94874',
>        host = 'services.mini.pw.edu.pl', sorted_by_id = TRUE,
>        windows = .Platform$OS.type == 'windows')
> get_votes_table(dbname = 'sejmrp', user = 'reader', password = 'qux94874',
>        host = 'services.mini.pw.edu.pl', sorted_by_id = TRUE,
>        windows = .Platform$OS.type == 'windows')
> get_statements_table(dbname = 'sejmrp', user = 'reader', password = 'qux94874',
>        host = 'services.mini.pw.edu.pl', sorted_by_id = TRUE,
>        windows = .Platform$OS.type == 'windows')
```

where

- `sorted_by_id` specifies if table should be sorted by id,
- `windows` specifies if you use Windows operation system.

All of these arguments are default, so usage of them comes to usage of unparametric functions like:

```
> get_deputies_table()
> get_votings_table()
> get_votes_table()
> get_statements_table()
```

Moreover, it should be said that changing `sorted_by_id` argument to `FALSE` is not recommended, because data can be unsorted and there may occur some problems during analysis.

There is also a function:

```
get_filtered_votes(dbname = 'sejmrp', user = 'reader', password = 'qux94874',
  host = 'services.mini.pw.edu.pl', windows = .Platform$OS.type == 'windows',
  clubs = character(0), dates = character(0), terms_of_office = integer(0),
  meetings = integer(0), votings = integer(0), deputies = character(0),
  topics = character(0))
```

that retrieves joined deputies, votes and votings tables with filtred data. As you see there are few possible filters:

1. `clubs` - names of clubs. This filter is a character vector with elements like for example: 'PO', 'PiS', 'SLD'. It is possible to choose more than one club.
2. `dates` - period of time. This filter is a character vector with two elements in date format 'YYYY-MM-DD', where the first describes left boundary of period and the second right boundary. It is possible to choose only one day, just try the same date as first and second element of vector.
3. `terms_of_office` - range of terms of office's numbers. This filter is a integer vector with two elements, where the first describes a left boundary of range and the second a right boundary. It is possible to choose only one term of office, just try the same number as first and second element of vector.
4. `meetings` - range of meetings' numbers. This filter is a integer vector with two elements, where the first describes a left boundary of range and the second a right boundary. It is possible to choose only one meeting, just try the same number as first and second element of vector.
5. `votings` - range of votings' numbers. This filter is a integer vector with two elements, where the first describes a left boundary of range and the second a right boundary. It is possible to choose only one voting, just try the same number as first and second element of vector.
6. `deputies` - full names of deputies. This filter is a character vector with full names of deputies in format: 'surname first_name second_name'. If you are not sure if the deputy you were thinking about has second name, try 'surname first_name' or just 'surname'. There is high probability that proper deputy will be chosen. It is possible to choose more than one deputy.
7. `topics` - text patterns. This filter is a character vector with text patterns of topics that you are interested about. Note that the votings' topics are written like sentences, so remember about case inflection of nouns and adjectives and use stems of words as patterns. For example if you want to find votings about education (in Polish: szkolnictwo) try 'szkolnictw'. It is possible to choose more than one pattern.

These filters make possible to get data in a numerous ways. For example, to find every votings of deputies from PO and PiS during 2014 year try:

```
> get_filtered_votes(clubs = c('PO', 'PiS'), dates = c('2014-01-01', '2014-12-31'))
```

or if you are only looking for votings with referendum use:

```
> get_filtered_votes(topics = "referendum")
```

Use-cases

```
> library(dplyr)
> data(votes)
> v <- c(`Za` = 5, `Przeciw` = -5, `Wstrzymał się` = 2, `Nieobecny` = 0)/10
> mat2 <- get_distance_matrix(votes[,c("surname_name", "id_voting", "vote")], weights = v)
> get_deputies_dendrogram(mat2, k=5)

> df <- votes[,c("surname_name","club")]
> df %>%
+   group_by(surname_name, club) %>%
+   summarise(n = n()) %>%
+   arrange(-n) %>%
+   group_by(surname_name) %>%
+   top_n(1) %>%
+   as.data.frame() -> clubs
> row.names(clubs) <- clubs[,1]
> clubs$club[clubs$club == "niez."] = "cross-bencher"

> get_deputies_silhouette(mat2, clubs)
> get_deputies_mds(mat2, clubs2)
```

Summary

sejmRP package provides easy-to-use tools to scrap and access open data from Polish Diet's webpage. Combining these tools with its further analysis and visualization allows to build algorithms intended

to reproducible statistical analysis and publication. The source code and user manual for the latest CRAN version of **sejmRP** package are available at the github site, where can also be found the full source code of the use-cases. Despite the fact that **sejmRP** package was built for Polish Diet's webpage, its infrastructure and similar analysis can be adapted to the case of other countries.

Piotr Smuda
Faculty of Mathematics and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw
Poland
smudap@student.mini.pw.edu.pl

Przemysław Biecek
Faculty of Mathematics and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw
Poland
przemyslaw.biecek@mini.pw.edu.pl